Scheduling Tasks with Preemptions and Precedence Constraints on Identical Parallel Machines to Minimise the Maximum Lateness

by Samuel James Walker

Supervisor: Yakov Zinder

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

University of Technology, Sydney – 2016

CERTIFICATE OF ORIGINAL AUTHORSHIP

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signature of Student:

Date:

Many thanks to my supervisor Yakov Zinder, and to my parents.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

This work examines in depth a collection of related scheduling problems. The problems considered all relate to the processing of tasks on machines over time, with the goal of minimising some function of the tasks' completion times. The tasks being scheduled are subject to preemptions, that is they may be interrupted during their processing and resumed later on the same or a different set of machines. The tasks are subject to precedence constraints, meaning some tasks cannot be processed until others have been completed. The machines on which the tasks are processed are identical, meaning they have the same speed and functionality, and parallel, in contrast to scheduling models where tasks must be processed on multiple machines sequentially. This is a thoroughly investigated area of research in the literature.

Due to the computational complexity of the problems considered, approximation algorithms for these problems are presented and are analysed in several ways. Several well known results are expanded to apply to more general scheduling problems, and a completely new class of results is described. In particular, the solvable cases of some approximation algorithms – i.e. the situations under which the algorithms provide an optimal solution to the considered problem – are examined in more detail than heretofore. Bounds on the worst case of the considered algorithms are also presented, as are results regarding the computational complexity of certain subproblems.

The goal of this work is to advance the understanding of this focused but significant area of research, and present a wide variety of results relating to it.

Chapter 1

Introduction

This thesis examines a collection of related problems from the field of scheduling theory: the processing of a number of jobs on machines over time. More specifically, there is a set N of jobs / tasks that must be processed over time, each with a specific target of processing time required for completion. These tasks may be, for example, units of parallel processing on a supercomputer or different jobs on a construction site. There are a number of machines that can process these tasks over time, limited to processing only one task at any given point in time. Real world examples of these include teams of labourers and threads in a CPU. In some cases, the goal is simply to find a feasible schedule that results in the completion of all tasks in N; more commonly the goal is to minimise some function of the completion times of the individual tasks, known as the objective function.

A schedule is an assignment of tasks to machines over time: in particular each assignment of a task to a set of machines takes place for an interval of non-zero length, and there is a finite number of such assignments made in the course of a schedule. Schedules generally begin at time 0, with tasks being scheduled only for non-negative times.

Scheduling theory encompasses many different scheduling problems, only a few of which are considered in this thesis. A scheduling problem specifies the general rules by which schedules are constructed and compared, e.g. the types of constraints under which tasks are processed and the objective function which is to be minimised. For each scheduling problem, there are (usually) infinitely many instances of that problem. Each instance is a specific realisation of the parameters of the problem: it includes all information required to schedule the tasks and measure the quality of the resulting schedule via the objective function. For each instance of a problem, there are (usually) infinitely many valid schedules; the goal is generally to produce a schedule which minimises the objective function for the given instance of the given problem. This delineation of problems, instances and schedules is particularly im-

portant in this thesis: unlike many other works on this subject, it is often necessary in this thesis to compare and contrast schedules for different instances of the same problem.

1.1 Scheduling Theory

Since there are many different scheduling models propounded within scheduling theory, a system of notation, known as three-field notation, has been developed to specify scheduling problems. In particular, the problem is represented as $\alpha|\beta|\gamma$, where α, β and γ take on different values depending on the problem.

The first field, α, describes the machines used to process the tasks. The problems considered in this thesis relate to identical parallel machines, denoted $\alpha = P$. This means that there is some number m of machines, and tasks may be processed on any one of them (or, in some scheduling problems, more than one), without it mattering precisely which.

Sometimes P appears on its own, indicating there is some specific number of machines that is part of the description of each instance of the problem – different instances can have different values of m. Other times P appears with a number, indicating that the considered problem is for a specific number of machines, e.g. the much investigated case of $P2$: two identical parallel machines. Sometimes, there may be an essentially unlimited number of parallel machines, which is indicated by $P\infty$. The cases of $P|\beta|\gamma$ and $Pm|\beta|\gamma$ for specific m are examined in this thesis.

A generalisation of this machine environment is $\alpha = Q$ (or $\alpha = Qm$ for some m). This refers to uniform machines: the machines differ only by their speed, with each machine i having a speed s_i, and each machine processing tasks at a rate proportional to its speed. Although this thesis does not consider uniform machines, some results derived for uniform machines are relevant to this thesis, and appear in the literature survey.

If there is only one machine it is not necessary to specify "P" or "Q" in the α field: in this case $\alpha = 1$.

The second field, β, describes the nature of the tasks that must be processed. It may contain any number of elements describing how the tasks are defined and constrained. It may be empty, indicating the default assumptions are in place. The default assumptions are that each task must be processed for an amount of time particular to that task in order to be completed, may be processed on any machine at any time until completion, may only be processed on one machine at a time and

may not be interrupted while being processed. Values of β considered in this thesis are:

- *pmtn*: Preemptions are available, that is the processing of tasks may be interrupted, and they may be resumed later on the same or a different set of machines. This is sometimes represented as *prmp* in the literature.

- $int-pmtn$: Preemptions are available, but only at integer points in time. This notation is unique to this thesis.

- r_j: The problem involves release dates, that is that each task j can only begin processing at time r_j. When all release dates are zero, this reduces to the problem with no release dates.

- *prec*: This indicates that precedence constraints are in effect. This means that a binary relation $\rightarrow$ on the set of tasks N exists which is irreflexive, antisymmetric and transitive, that it is, in other words, a strict (or irreflexive) partial order. For two tasks j and g, if $j \rightarrow g$ this means that task g cannot begin processing until task j has been completed. If neither $j \rightarrow g$ nor $g \rightarrow j$ then j and g are said to be independent. It will often be convenient to consider sets of tasks in which all tasks are pairwise independent: for the sake of simplicity, the term "pairwise" will be omitted when referring to such sets. To use the terminology of order theory, a partial order in which all elements (tasks) are independent will be referred to as an antichain.

- $p_j = 1$: The case of unit execution times (UET). While normally the amount of time a task j must be processed until it is completed, p_j, may be any positive real number, in this case it is constrained to be one for all tasks. This is the case used to represent all situations where the tasks have equal processing times, as they may be assumed to be one without loss of generality.

- c_{jg}: The problem is subject to communication delays, i.e. if there are two tasks j and g such that $j \rightarrow g$, and they are processed on different machines, task g cannot begin processing until c_{jg} time units after the completion of task j.

- $c_{jg} = 1$: All communication delays (between pairs of tasks that are not independent) are equal to one.

- u_j: Tasks may be processed on multiple machines up to a given limit for each task. This notation is unique to this thesis. The combination of *pmtn* and u_j has been denoted by var, δ_j in [78] but is not used in this thesis for the sake

of consistency of notation. Tasks that may be processed on multiple machines subject to preemptions at any time are referred to as "malleable tasks."

This thesis focuses primarily on scheduling models with preemptions and precedence constraints, with or without malleable tasks, i.e. $\beta = prec, pmtn$ or $\beta = prec, pmtn, u_j$, or integer preemptions and precedence constraints, i.e. $\beta = prec, int - pmtn$.

The last field, γ, specifies the objective function of the problem; it will usually contain only one entry. For the most part, this thesis is concerned with $\gamma = L_{max}$: the maximum lateness. The maximum lateness is the largest amount by which the completion time of a task exceeds that task's due date. This value may be negative. Also considered is the problem where $\gamma = C_{max}$: the makespan. The makespan is simply the largest completion time among all tasks. Note that if all due dates are zero, the maximum lateness reduces to the makespan criterion.

Subsection 7.1.1 considers the objective of total weighted tardiness: the weighted sum of the tardiness of all tasks (denoted $\gamma = \sum w_j T_j$), where the tardiness of a task is zero if it is completed before its due date and is otherwise equal to the difference between the task's completion time and its due date. This has received some attention in the literature – see [2], [6], [38] and [81]. If all due dates are equal to zero, this reduces to the total weighted completion time criterion (denoted $\gamma = \sum w_j C_j$). If all weights are equal, these objective functions are denoted $\sum T_j$ and $\sum C_j$.

Finally, let U_j be equal to zero if task j is completed on time, and one otherwise. The criterion $\sum U_j$ is the total number of tardy jobs, and $\sum w_j U_j$ is the total weighted number of tardy jobs.

One case addressed by this thesis is the $P|prec, pmtn|L_{max}$ problem. This is the problem of scheduling tasks with preemptions and precedence constraints on identical parallel machines with the goal of minimising the maximum lateness. This problem has received significant attention in the literature, for example [55], [71] and [105].

Another case addressed by this thesis is a generalisation of the $P|prec, pmtn|L_{max}$ problem. Specifically, malleable tasks have been given significant attention in recent years – see [24], [26], [42], [51], [52], [56], [61], [80], [93], [97], [98] and [99] for recent work in this area – largely due to their application to the field of parallel processing. As stated earlier, malleable tasks are tasks which may be processed on more than

one machine at a time, where the rate of processing is some function of the number of machines on which the task is being processed. The specific model of malleable tasks considered in this thesis defines the rate of processing of a task in an interval of time as *proportional* to the number of machines on which it is processed. In order to study the effect of this parallelism on the scheduling problem, this problem defines, for each task j, a value $1 \leq u_j \leq m$ such that j may be processed on no more than u_j machines simultaneously. This problem will be denoted $P|prec, pmtn, u_j|L_{max}$, and reduces to the $P|prec, pmtn|L_{max}$ problem when all $u_j = 1$.

1.2 Problems Considered

For the sake of rigour and clarity, a formal definition of the $P|prec, pmtn, u_j|L_{max}$ problem follows. There is a set of tasks N, where each task is denoted by a natural number hereafter referred to as its task index, and where $|N| = n$. There exists a number of identical parallel machines m. For each task j, there is

- a positive real number p_j that specifies the amount of processing required by j in order to be completed,
- a real number d_j (which can be positive, negative or zero) specifying the due date of j and
- a natural number $u_j \in \{1, 2, \ldots, m\}$ specifying the degree of parallelism allowed for the task j.

There is also a partial order $\rightarrow$ on N specifying the precedence constraints.

In summary, an instance of the considered problem may be represented by a tuple

$$(m, \rightarrow, \{(j, p_j) : \ j \in N\}, \{(j, d_j) : \ j \in N\}, \{(j, u_j) : \ j \in N\}) \tag{1.1}$$

which encodes all of the information specific to that instance of the general problem.

A valid schedule σ is a vector function $\sigma_j(t)$ specifying the number of machines on which task j is processed at time t. It is defined for all $j \in N$ and $t \in \mathbb{R}$. This function can take on only non-negative integer values, is piecewise constant on closed intervals, and has only a finite number of points of discontinuity.* Each

*This definition neatly sidesteps a technical difficulty: if a schedule was defined as an assignment of tasks to machines, a collection of problem instances could be imagined in which m and u_j are exponential in n. If specifying a schedule required each machine be considered separately, a schedule could not be constructed in polynomial time even in the absence of precedence constraints. Since all practical problem instances have reasonably small values of m, the avoidance of this issue using the definition of a schedule above has no practical consequences.

machine can process at most one task at a time, meaning $\sum_{j \in N} \sigma_j(t) \leq m$ for all $t \geq 0$, and similarly each task j can be processed on at most u_j machines at a time, so $\sigma_j(t) \leq u_j$ for all $t \geq 0$. For the sake of simplicity of notation, these constraints are not applied at the points of discontinuity of σ.* Each task j must receive p_j processing time in the schedule, i.e.

$$\int_0^\infty \sigma_j(t)dt = p_j.$$

A final constraint on a valid schedule is defined by the precedence constraints: for two tasks j and g such that $j \to g$, if $\sigma_j(t) > 0$ for any time $t \geq 0$ then $\sigma_g(t') = 0$ for all $t' < t$.

The completion time of a task j in schedule σ is the supremum of all t for which $\sigma_j(t) > 0$, and is denoted $C_j(\sigma)$, i.e.

$$C_j(\sigma) = \sup_{t \in \mathbb{R}} \{t : \sigma_j(t) \neq 0\}.$$

A task j is said to be available at time t in schedule σ if $C_j(\sigma) > t$ and $C_g(\sigma) \leq t$ for all $g \in N$ such that $g \to j$. The lateness $L_j(\sigma)$ of a task j in schedule σ may be defined as $L_j(\sigma) = C_j(\sigma) - d_j$. Let

$$C_{max}(\sigma) = \max_{j \in N} C_j(\sigma) \qquad \text{and} \qquad L_{max}(\sigma) = \max_{j \in N} L_j(\sigma)$$

denote the makespan and maximum lateness respectively. The goal of the problem is to find, for a given instance, a schedule σ such that $L_{max}(\sigma)$ is minimised. Such a schedule is known as an optimal schedule for that instance. For the sake of consistency, this thesis will generally use the following notation to denote different kinds of schedules:

- optimal schedules, i.e. schedules which minimise the maximum lateness for a given problem instance, will be denoted ω,
- schedules that are derived from some given approximation algorithm will be denoted η, and
- arbitrary schedules with no particular properties or origins will be denoted σ.

Thus is the $P|prec, pmtn, u_j|L_{max}$ problem defined. The special case where $u_j = 1$ for all $j \in N$ is denoted $P|prec, pmtn|L_{max}$ and is also considered in this thesis.

*This exception is intended to simplify the notation used to represent intervals in time during which tasks are processed – it can now be said that tasks are processed during closed intervals without the need to consider the finitely many points in time where these intervals overlap.

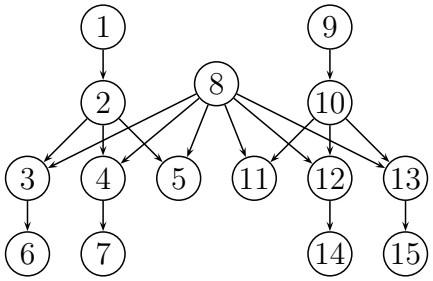


Figure 1.1: The precedence constraints of an example instance.

Several terms that are in common usage, and which will be used throughout this work, will now be defined. Task j is said to be a predecessor of task g, and to precede task g, if and only if $j \rightarrow g$; g is said to be a successor of j. Because $\rightarrow$ is irreflexive, $j \not\rightarrow j$. Within this work the set of successors of a task j will be denoted K_j, and the set of predecessors will be denoted J_j. Task g may be said to be an immediate successor of j, and j and immediate predecessor of g, if and only if $g \in K_j$ and there does not exist $q \in K_j \cap J_g$. The set of immediate successors and predecessors of a task j will be known as $\tilde{K}_j$ and $\tilde{J}_j$ respectively. Any of these sets may be empty. For the sake of simplicity of notation, for any set $S \subseteq N$, the union $\cup_{j \in S} K_j$ will be denoted by $K(S)$ and the union $\cup_{j \in S} J_j$ will be denoted by $J(S)$.

An instance of problem $P|prec, pmtn, u_j|L_{max}$ is defined below. This example will be used throughout the remainder of this thesis to illustrate different algorithms and issues.

Example 1.1 *Consider an instance of the problem $P|prec, pmtn, u_j|L_{max}$ for which $m = 3$ and $N = \{1, 2, \cdots, 15\}$; hence, $n = 15$. Figure 1.1 is a graphical representation of the partial order of the considered problem instance, specifically the transitive reduction of $\rightarrow$. Each task is labelled with its task index. Table 1.1 specifies the values of p_j, u_j and d_j for all of the tasks.*

As a matter of practicality, it is often not possible to divide the processing of a task arbitrarily – often tasks can only be divided at certain regular points. Without loss of generality it may be assumed that these regular points are integers. These integer preemptions have been considered in a number of papers, for example [109] and [6]. A modification of the problem $P|prec, pmtn|L_{max}$ to include integer preemptions – denoted $P|prec, int - pmtn|L_{max}$ – is considered in Chapter 8 of this thesis.

Tasks	p_j	u_j	d_j
$1, 9$	6	2	6
$2, 10$	6	1	11
8	3	3	7
$3, 4, 12, 13$	2	1	15
$5, 11$	2	2	11
$6, 7, 14, 15$	6	2	15

Table 1.1: The parameters of an example instance.

1.3 Contributions of this Thesis

This thesis presents a number of new results, all of which are related to the scheduling of tasks subject to precedence constraints and preemptions on parallel machines with the goal of minimising the maximum lateness.

Chapter 3 deals with the computational complexity of a number of different scheduling problems. Although the general problems considered in this thesis are NP-hard, the examination of the computational complexity of subproblems has been an area of considerable research – see Sections 2.1 and 2.2 for an overview. Chapter 3 is partly a very focused literature review and partly a summary of results obtained in this thesis.

Eight different subproblems, each characterised by a special structure of precedence constraints, i.e. the precedence constraints come from a particular class of partial orders, and each of which has received some attention in the literature, are considered. The centrepiece of this chapter is Table 3.2 which provides, where known,

- the computational complexity of the $P|prec, pmtn|\gamma$ problem for the given class of precedence constraints, where $\gamma \in \{C_{max}, L_{max}\}$; and
- two measurements of the rate of growth, in terms of n, of the number of partial orders of the given class on n elements.

While many of the results in Table 3.2 originate from the scheduling theory and order theory literature, some were derived by the author in completion of this thesis. Of these, some follow from results derived later in this thesis, while others are presented in Section 3.3. Specifically,

- Theorem 3.1 establishes a fairly simple result about the rate of growth, in terms of n, of a particular class of partial orders on n elements; and

- Theorem 3.2 establishes the NP-hardness of a particular subproblem, using a non-trivial adaptation to the problem with preemptions and arbitrary processing times of a result presented in [118].

Although no broad conclusion is drawn, this chapter is a step along the road to discovering what specific structures of precedence constraints make a scheduling problem NP-hard.

Given that the general problems considered in this thesis are NP-hard, the study of approximation algorithms makes up a large portion of the results. Chapter 4 presents the basis for many of the results to follow. In particular,

- Section 4.1 presents three families $\mathfrak{P}$, $\mathfrak{F} \subset \mathfrak{P}$ and $\mathfrak{S} \subset \mathfrak{P}$ of approximation algorithms for the $P|prec, pmtn|L_{max}$ problem and its generalisation $P|prec, pmtn, u_j|L_{max}$, all of which include adaptations of the algorithms described in [105], [55] and [120] to the case of malleable tasks, along with infinitely many as yet undiscovered algorithms;

- Section 4.2 describes the logic common to all algorithms in family $\mathfrak{P}$;

- Section 4.3 describes four specific approximation algorithms (the Brucker-Garey-Johnson Algorithm, the Garey-Johnson Algorithm, the Zinder-Roper Algorithm and the Optimal Recursive Algorithm) that have received attention in the literature, all of which are in family $\mathfrak{F} \cap \mathfrak{S}$, also establishing certain simple properties that they have; and

- Section 4.4 establishes a common structure of the schedules produced by these algorithms.

Although much of Chapter 4 is merely establishing a basis for further study in later chapters, there is one result original to this author that establishes a generalisation of a common result proven for individual algorithms covered in many different papers. Specifically, [105], [55] and [120] introduce, in different forms, some kind of modification of the due dates d_j and then subsequently prove that these modifications leave the maximum lateness unaffected. Lemma 4.1 in Section 4.2 provides a general criterion for how the due dates can be modified without changing the maximum lateness of any schedule. This generalisation provides much insight into results established in a wide selection of papers. Much of the material of this chapter was presented at the 9th Workshop on Models and Algorithms for Planning and Scheduling Problems [124], in collaboration with Yakov Zinder.

Chapter 5 concerns the examination of bounds on the performance, also called performance guarantees, of the approximation algorithms described in the previous chapter. Specifically, the goal is to find an upper bound on $L_{max}(\eta)$, where η is a schedule produced by the considered algorithm, in terms of some parameters of the problem instance and $L_{max}(\omega)$, where ω is an optimal schedule for the problem instance. Results of this kind have been the subject of a great deal of study (see Section 2.3 for more information) due to their utility when the algorithms are put to use: they guarantee certain limits on how badly the algorithms can perform so that it is known how closely the resulting schedules approximate optimal schedules.

Theorem 5.2 proves a bound on the performance of the Brucker-Garey-Johnson Algorithm, which is proven tight in Theorem 5.6. These results, originally presented in [124], are a generalisation to the problem with malleable tasks of one of the main results of [105]. While [105] proves the tightness of the bound only in certain places, Theorem 5.2 provides a slight improvement in this bound which allows it to be proven tight everywhere in Theorem 5.6.

Corollary 5.3 provides a bound corresponding to that obtained in [120] for the Brucker-Garey-Johnson Algorithm, but generalised to the case of malleable tasks.

Theorem 5.14 proves a bound on the performance of the Garey-Johnson Algorithm that is similar in form to the bound proved in Theorem 5.2 for the Brucker-Garey-Johnson Algorithm. Although the two bounds appear similar, Theorem 5.14 uses techniques similar to those found in [55]; these techniques are not used in the proof of Theorem 5.2.

The bound established in Theorem 5.14 is proven tight in Theorem 5.28, which is the centrepiece of the chapter. Specifically, Theorem 5.28 proves that no algorithm from family $\mathfrak{F} \cap \mathfrak{S}$ can have a better performance guarantee, proving the tightness of the bound. This result has strong implications for the future of research into approximation algorithms for the $P|prec, pmtn|L_{max}$ and $P|prec, pmtn, u_j|L_{max}$ problems: if a better performance guarantee than that established in Theorem 5.14 is desired then an algorithm outside family $\mathfrak{F}$ must be sought. In essence this demonstrates the limitations of this much studies class of approximation algorithms. A corollary of this result, Corollary 5.30, demonstrates that no algorithm in $\mathfrak{F}$ can solve the much studied three machine problem, $P3|prec, pmtn, u_j|L_{max}$. This work was originally presented in the 11th Workshop on Models and Algorithms for Planning and Scheduling Problems [116], in collaboration with Yakov Zinder and Claire Hanen.

Theorem 5.32 is a similar result for the family $\mathfrak{S}$ of algorithms, although no algorithm is known that achieves the bound provided. Similarly to Theorem 5.28, Corollary 5.33 demonstrates that no algorithm from $\mathfrak{S}$ can solve even the makespan problem on three machines. Again, this has important implications for the direction

of future research.

Corollary 5.16 establishes a bound for the Garey-Johnson Algorithm that is analogous to that established for the Brucker-Garey-Johnson Algorithm in Corollary 5.3.

Finally, Theorem 5.49 establishes a bound on the performance of the Zinder-Roper Algorithm that is a generalisation – with a slight improvement – of the bound proved in [120] to the case of malleable tasks. It is similar in form to the bounds established in Corollary 5.3 and Corollary 5.16. This result was originally presented by the author in the 8th International Conference on Optimization: Techniques and Applications [122] in collaboration with Yakov Zinder.

Chapter 6 is concerned with a completely unprecedented concept: the domain of an algorithm. There has been much research on the subject of the solvable cases of approximation algorithms, see Section 2.2 for a survey. One thing that has never been done, to the knowledge of the author, is to specify precisely which partial orders will and will not guarantee optimality for an approximation algorithms.

Formally stated, the domain $\mathcal{D}^{\mathscr{A}}$ of an approximation algorithm $\mathscr{A}$ is the class of all partial orders S such that,

1. if $S \in \mathcal{D}^{\mathscr{A}}$ all problem instances with precedence constraints S will be solved by $\mathscr{A}$ and

2. if $S \notin \mathcal{D}^{\mathscr{A}}$ there exists a problem instance with precedence constraints S that is not solved by $\mathscr{A}$.

As an example, it has long been known that the Brucker-Garey-Johnson Algorithm solves all problem instances where the precedence constraints take the form of an in-tree, but it was not known for which other partial orders this is true. This thesis not only describes a broader class of partial orders within the domain of the Brucker-Garey-Johnson Algorithm, it further demonstrates that no other partial orders are in this domain. This complete description of the domain of an algorithm has implications for the future of research: not only does it describe a broad class of solvable cases, it also demonstrates that no further solvable cases exist, aiding researchers in their search for further results.

The first major result of Chapter 6 is a complete description of the domain $\mathcal{D}^{\mathscr{BGJ}}$ of the Brucker-Garey-Johnson Algorithm for the $P|prec, pmtn|L_{max}$ problem. The class of partial orders is simple enough to be described in terms of thirty prohibited subgraphs depicted in Figure 6.2 and Figure 6.3. That this condition is sufficient to guarantee optimality is proven in Theorem 6.6, while necessity is proven in Theorem

6.16. No equivalent finding is known to the author, excepting two more results also presented in this chapter. These results were originally published by the author, in collaboration with Yakov Zinder, in Algorithms and Computation [114], after presentation at the 24th International Symposium on Algorithms and Computation.

The second major result of this chapter is even more surprising. The domain $\mathcal{D}^{\mathscr{ZR}}$ of the Zinder-Roper Algorithm for the $P|prec, pmtn|L_{max}$ problem is described, and sufficiency is proven in Theorem 6.23. However, Theorem 6.75 proves that having precedence constraints in the domain of the Zinder-Roper Algorithm is not merely a necessary condition of optimality for the Zinder-Roper Algorithm itself, but it is also a necessary condition for optimality for all algorithms in the family $\mathfrak{F}$ mentioned earlier. Another way of saying this is that, for every partial order $P \notin \mathcal{D}^{\mathscr{ZR}}$, every algorithm in $\mathfrak{F}$ will produce a non-optimal schedule for some instance of $P|prec, pmtn|L_{max}$ where P specifies the precedence constraints.

This result in some ways resembles Theorem 5.28: rather than analysing a single algorithm, the limits of a broad family of algorithms is discovered. Again, this has important implications for future research. It is often desired to find new classes of precedence constraints for which an algorithm can be guaranteed to produce an optimal schedule. Theorem 6.23 finds such a class, much broader than anything already known. Lest researchers expend fruitless effort searching for more solvable cases, Theorem 6.75 demonstrates that no more exist. Furthermore, researchers have often sought new algorithms, more powerful than those already known. Theorem 6.75 demonstrates that a broad family of algorithms can yield no more solvable cases, and that the search for better algorithms must be directed beyond the scope of this family.

Finally, the domain $\mathcal{D}^{\mathscr{GJ}}$ of the Garey-Johnson Algorithm for the $P|prec, pmtn|L_{max}$ problem is described. Sufficiency for optimality is proven in Theorem 6.80 and necessity is proven in Theorem 6.105. It is shown that

$$\mathcal{D}^{\mathscr{BGJ}} \subset \mathcal{D}^{\mathscr{GJ}} \subset \mathcal{D}^{\mathscr{ZR}}.$$

Chapter 7 focuses on an algorithm for solving the $P|prec, pmtn, u_j|L_{max}$ problem, with the goal of obtaining insight into what makes a problem easy or difficult to solve. The overall problem is decomposed into a (generally non-polynomial) number of linear programs. Theorem 7.2 demonstrates that the linear program is valid while Theorem 7.5 proves the validity of a certain kind of simplification of the linear program. As noted in Subsection 7.1.1, the approach considered in this chapter

can be adapted to the objective of minimising total weighted tardiness, instead of the maximum lateness. This suggests a new family of approximation algorithms not bound by the limitations on $\mathfrak{F}$, $\mathfrak{S}$ or $\mathfrak{F} \cap \mathfrak{S}$ proved earlier in this thesis. This approach was presented at the ASOR Recent Advances 2015 [113], and was prepared in collaboration with Yakov Zinder.

Finally, Chapter 8 works to reproduce some of the preceding results for the case of integer preemptions, i.e. the $P|prec, int - pmtn|L_{max}$ problem. Despite how similar this problem is to the $P|prec, pmtn|L_{max}$ problem this generalisation is decidedly non-trivial, requiring a novel adaptation of Algorithm P to the field of integer preemptions, which allows the insights of previous chapters to be adapted for this case. In particular, performance guarantees are established for

- the Brucker-Garey-Johnson Algorithm, in Theorem 8.16 and Corollary 8.17;
- the Garey-Johnson Algorithm, in Theorem 8.18 and Corollary 8.19; and
- the Zinder-Roper Algorithm, in Theorem 8.20.

The results in this chapter, developed in collaboration with Yakov Zinder, were presented at the 25th conference of the European Chapter on Combinatorial Optimization, and have been accepted for publication in Discrete Applied Mathematics [123].

The peer-reviewed papers published in the course of this thesis are:

- Scheduling Flexible Multiprocessor Tasks on Parallel Machines [124];
- The Solvable Cases of a Scheduling Algorithm [114]; and
- Algorithms for Scheduling with Integer Preemptions on Parallel Machines to Minimize the Maximum Lateness [123].

The presentations given in the course of this thesis are:

- [124] was originally presented at the 9th Workshop on Models and Algorithms for Planning and Scheduling Problems;
- [114] was originally presented at the 24th International Symposium on Algorithms and Computation;

- [123] was originally presented at the 25th conference of the European Chapter on Combinatorial Optimization;
- [113] was presented at the ASOR Recent Advances 2015 conference;
- [116] was presented at the 11th Workshop on Models and Algorithms for Planning and Scheduling Problems; and
- [122] was presented at the 8th International Conference on Optimization: Techniques and Applications.

Chapter 2

Literature Review

Several books ([20], [78] and [89]) are recommended to gain a general understanding of the subject of scheduling theory, and they are cited in this work from time to time. For more specific information, the literature review is divided into separate sections based on the subject areas:

- Section 2.1 deals with proofs of NP-hardness,
- Section 2.2 is concerned with subproblems that are amenable to a polynomial-time solution,
- Section 2.3 studies the establishment of bounds on the performance of approximation algorithms
- Section 2.4 examines the usage of Branch and Bound techniques in scheduling, and
- Section 2.5 examines the theoretical background of partial orders and relates this to the precedence constraints of scheduling problems.

Each section begins with a general summary of the literature in this area, which is followed by an in-depth examination of certain papers of particular relevance to this thesis.

2.1 NP-Hardness

A general overview of computational complexity results is given by [22], a website set up by Peter Brucker and Sigrid Knust to amalgamate existing results about the computational complexity of scheduling problems. It contains reduction graphs for the different scheduling problems, along with lists of maximal polynomially solvable, minimal NP-hard and minimal and maximal open problems. Of particular use to this

thesis is the page on identical parallel machines with preemptions. A more detailed survey of the complexity of scheduling problems can be found in [50], and another in [77]. Two useful references for complexity theory in general are [48] and [64], which prove the NP-hardness of a number of problems by describing polynomial reductions from other NP-hard problems. Some early parallel processing complexity results can be found in [37], although these are not, strictly speaking, malleable tasks.

Two important kinds of NP-hard problems exist: weakly and strongly NP-hard. The distinction is that strongly NP-hard problems remain NP-hard even when their numerical parameters are bounded above by a polynomial in the length of the input, while weakly NP-hard problems can be solved in polynomial time if such a limit is imposed. Another way of putting it is that weakly NP-hard problems become polynomial when the numerical parameters in their input are given in unary rather than binary – see [47] for more information. Unless stated otherwise, all NP-hard problems considered in this thesis are strongly NP-hard.

Most of the problems considered in this thesis are NP-hard, see [110] and [22] for example. Indeed, [110] demonstrated that the $P|prec, pmtn, p_j = 1|C_{max}$ problem is NP-hard. Chapter 3 examines the computational complexity of a number of scheduling problems, with particular attention to the structure of the precedence constraints. For a good resource on this subject, see [94].

Many results have established that the complexity of a scheduling problem is sensitive to the number of machines available for processing. Several examples are given below.

A flowshop consists of a series of stages, and each task must be processed on the first stage, then on the second stage then on the third stage and so on. For the case where there are two stages, each consisting of a single machine, [62] demonstrated that minimising the makespan can be achieved in polynomial time in the absence of precedence constraints, regardless of whether preemptions are allowed. In contrast, if at least one stage has more than one machine was proven in [58] to be strongly NP-Hard, again regardless of whether preemptions are allowed.

In [5] and [12] it was proven that $1|prec, pmtn, r_j|L_{max}$ can be solved in polynomial time. However, in [110] it was demonstrated that the problem $P|prec, pmtn, p_j = 1|C_{max}$ is strongly NP-Hard, i.e. that allowing an arbitrary number of machines makes the polynomially solvable problem $1|prec, pmtn, r_j|L_{max}$ non-polynomial, even if the input is restricted so that all $r_j = d_j = 0$ and all $p_j = 1$.

Similarly, in [4] it was established that $1|pmtn, r_j|\sum C_j$ can be solved in polynomial time, while in [39] it was established that $P2|pmtn, r_j|\sum C_j$ is weakly NP-Hard.

Even the addition of a single machine converts a polynomially solvable problem into an NP-Hard problem.

The paper [21] considers the problem of $P|prec, p_j = 1|L_{max}$, i.e. the problem of scheduling unit execution time tasks subject to precedence constraints on identical parallel machines with the goal of minimising the maximum lateness.

The specific kinds of precedence constraints considered are in-trees and out-trees – as is noted in [71] these problems are essentially the same as the problem of scheduling tasks constrained by in-forests and out-forests. This latter formulation provides a somewhat simpler definition: an in-forest (respectively out-forest) is a partial order in which no task has more than one successor (respectively predecessor). An in-tree is an in-forest with a single element with no successor, and an out-tree is defined analogously

First is considered the case of in-trees. An algorithm is described that provides an optimal solution to any instance of $P|prec, p_j = 1|L_{max}$ where the precedence constraints take the form of an in-tree. See Section 2.2 for more information.

Second is considered the case of out-trees. The problem $P|out-tree, p_j = 1|L_{max}$, i.e. where the precedence constraints are in the form of an out-tree, is proven to be NP-complete by a reduction from the well known decision problem NODE COVER. In this problem an undirected graph G and a natural number k are supplied as input; the required output is "true" if there exists a set $\hat{V}$ composed of k vertices of G such that every edge of G has at least one endpoint in $\hat{V}$, and "false" otherwise.

For any given instance of NODE COVER an instance of $P|out-tree, p_j = 1|L_{max}$ is created according to a procedure which guarantees that, if and only if there is a node cover $\hat{V}$ of k vertices of G, a schedule exists for the instance of $P|out-tree, p_j = 1|L_{max}$ such that all tasks are completed on time.*

A similar result for $P|out-tree, pmtn|L_{max}$ was proven by J. K. Lenstra, but remains unpublished (see [22]).

In [118] another complexity result for the $P|prec, p_j = 1|C_{max}$ problem is presented: specifically, if the precedence constraints have the form of a bipartite order, the scheduling problem is NP-hard. This is achieved by a reduction of the problem from the clique problem, extending work originally found in [76].

The general idea behind the proof is, given an undirected graph G and a natural number k, an instance of $P|prec, p_j = 1|C_{max}$ will be defined such that the optimal

*A precise description of the problem is too complicated for inclusion in a survey of the literature. A summary is included here to give the general idea.

makespan $C_{max}(\omega)$ is equal to three if there exists a clique (a complete subgraph) of k nodes in G, and is strictly greater than three otherwise. A task is introduced for each node of G and each edge of G. All "node" tasks are independent of one another, as are all "edge" tasks; however, each node task precedes precisely those edge tasks that correspond to the edges to which the node is connected in G. Additional tasks are introduced which, as a consequence of the precedence constraints between them, are forced to have specific completion times in any schedule σ such that $C_{max}(\sigma) = 3$. These tasks constrain when the node and edge tasks can be processed.

There are only enough free machines during the time interval $[0, 1]$ to process precisely k tasks. If these tasks correspond to a clique in G, $\binom{k}{2}$ edge tasks will become available upon the completion of the k node tasks. If they do not correspond to a clique, fewer edge tasks will become available, forcing more edge tasks to be processed at a later time; not all of these tasks can be completed early enough to allow $C_{max}(\sigma) = 3$. Hence, if the problem instance generated from G and k can be solved so can the clique problem, which is known to be NP-hard.

See Section 3.3 for more information, where this proof is adapted to the problem $P|bipartite, pmtn|C_{max}$.

The paper [40] addresses the complexity of minimising the makespan and "mean flow time" – another term for the total completion time – under a particular class of precedence constraints known as chains. There are three key results.

- It is proven that, for each fixed $m > 1$, the problem $Pm|chains|C_{max}$ is NP-hard. This is accomplished by a reduction from the 3-PARTITION problem.

- It is proven that, for each fixed $m > 1$, the problem $Pm|chains|\sum C_j$ is NP-hard. This also employs a reduction from the 3-PARTITION problem. This result extends several existing results established in [101].

- It is proven that preemptions do not allow a smaller total weighted completion time $\sum w_j C_j$ when the precedence constraints take the form of chains. In other words, an optimal schedule for the $Pm|chains|\sum w_j C_j$ problem is also an optimal schedule for the $Pm|chains, pmtn|\sum w_j C_j$ problem. This extends a result in [81] which proved the same result in the case without precedence constraints – a special case of the problem with chains.

When the latter two results are combined, they show that the $Pm|chains, pmtn|\sum C_j$ problem is also NP-hard.

2.2 Polynomially Solvable Cases

As has been stated in the previous section, most of the problems considered in this thesis are NP-hard. However, the discovery of subproblems of interest that are amenable to a polynomial solution is an area of continuing research. See [67] for a survey of simple polynomial algorithms for scheduling problems in the general area of research of this thesis. Results of this sort are relevant to Chapter 3, which examines the computational complexity of a number of scheduling problems, and Chapter 6, which examines in detail the solvable cases of certain polynomial-time algorithms.

In [81] several simple scheduling problems are considered, but the most important for subsequent work is $P|pmtn|C_{max}$, that is the problem of scheduling tasks subject to preemptions on identical parallel machines with the goal of minimising the makespan.

This problem has a very simple solution: the optimal makespan is

$$C_{max}(\omega) = \Delta = \max\left\{\max_{j\in N} p_j, \frac{1}{m}\sum_{j\in N} p_j\right\}.$$

It is simple to see that this is a lower bound on the makespan.

This paper presents a straightforward algorithm for constructing such a schedule, known as McNaughton's Algorithm.

McNaughton's Algorithm

1. Set $\tau := 0$. Select an arbitrary machine on which to process tasks.
2. If all tasks in N have been considered, terminate the algorithm. Otherwise, select any $j \in N$ that has not already been considered.
3. If $\tau + p_j < \Delta$, process j continuously during the time interval $[\tau, \tau + p_j]$ on the currently selected machine, set $\tau := \tau + p_j$ and go to Step 2. Otherwise, process j continuously during the time interval $[\tau, \Delta]$ and continue to Step 4.
4. Select an arbitrary machine (which has not already been used) on which to process tasks. Process j continuously during the time interval $[0, p_j - (\Delta - \tau)]$ on the newly selected machine, set

   $$\tau := p_j - (\Delta - \tau)$$

   and go to Step 2.

Note that, in Step 4 of this algorithm it may be the case that $p_j = \Delta - \tau$. If so, the interval $[0, p_j - (\Delta - \tau)]$ has zero length.

This algorithm is used as a subroutine in other scheduling algorithms, see for example [71] and [105]. Given its importance it is useful to note the algorithm's low computational complexity: the tasks and machines may be selected arbitrarily, so the only processing this algorithm performs is the computation to schedule the one (in Step 3) or two (in Step 3 and Step 4) segments of each task; since there is only simple arithmetic used to schedule these segments the algorithm takes $O(n)$ time to complete.

As stated in Section 2.1, [21] describes a polynomial algorithm that solves the $P|in-tree, p_j = 1|L_{max}$ problem. It builds on work in a previous paper [60] that examined the corresponding makespan problem using the so-called "critical path method." This is the origin of the Brucker-Garey-Johnson Algorithm, which is examined in several other papers considered in this literature review. Derivatives of this algorithm have been employed to solve other problems, and they have continued to be a subject of research in recent decades – see for example [103] and [104].

The algorithm is based on the observation that the processing of a task is constrained not just by its own due date but also by the due dates of any tasks it precedes. Consequently, the due dates are modified for all tasks j (as is the case for several other papers considered here) by setting $d'_j = d_j$ if $K_j = \emptyset$ and

$$d'_j = \min \left\{ d_j, \min_{g \in K_j} d'_g - 1 \right\}$$

otherwise. Since this recalculation depends on the modified due dates of the successors it must be completed in a particular order, specifically the reverse of any linear extension of the precedence constraints. It is simple to prove that, for all schedules σ, $L_{max}(\sigma)$ is unchanged by the replacement of the original due dates by these modified due dates.

Once this is done the tasks are scheduled by the well known list scheduling algorithm. This algorithm constructs a list of tasks in ascending order of their modified due dates and iterates over integer points in time in ascending order, at each point scheduling the earliest tasks on the list that can be scheduled during that time.

The optimality of the resultant schedule η is established as follows. Select among all tasks j such that

$$C_j(\eta) - d'_j = L_{max}(\eta)$$

a task x with the smallest completion time.

Firstly, suppose that

- all tasks j with $C_j(\eta) < C_x(\eta)$ have $d'_j \leq d'_x$ and
- there are no idle machines during the time interval $[0, C_x(\eta) - 1]$.

By this assumption, and because the modified due dates do not change the maximum lateness, the schedule must be optimal.

On the other hand, if there is an idle machine or a task j with $d'_j > d'_x$ processed during $[0, C_x(\eta) - 1]$, select the smallest t such that there is no idle machine and no task j with $d'_j > d'_x$ processed during $[t, C_x(\eta) - 1]$. By the list scheduling algorithm and the fact that $d'_j < d'_g$ if $j \to g$, there must be some tasks $z_1, \ldots, z_k$ with

$$C_{z_1}(\eta) = \cdots = C_{z_k}(\eta) = t$$

and task x and all tasks j such that $t < C_j(\eta) < C_x(\eta)$ are each preceded by at least one of $z_1, \ldots, z_k$. Since $k < m$, by the definition of in-trees, there must be fewer than m tasks processed at any one time during $[t, C_x(\eta) - 1]$, and so the definition of t implies $t = C_x(\eta) - 1$. Hence, for any i for which $z_i \to x$,

$$L_{max}(\eta) \geq C_{z_i}(\eta) - d'_{z_i} \geq C_{z_i}(\eta) - d'_x + 1 = C_x(\eta) - d'_x = L_{max}(\eta)$$

which contradicts the selection of task x.

Similarly, [45] considers the problem $P2|prec, p_j = 1|L_{max}$, i.e. the problem of scheduling precedence constrained unit execution time tasks on two parallel machines with the goal of minimising the maximum lateness. This is the origin of the Garey-Johnson Algorithm, which is examined in several other papers described in this thesis. Derivatives of this algorithm for use with other problems are still an active area of research, see [112] and [10] for example.

This paper proceeds in much the same manner as [21], in that it describes an algorithm that modifies the due dates of the tasks in such a way that the maximum lateness remains unchanged. While in [21] the modified due dates took into account only the due date of each task's most urgent successor, the algorithm described in this paper considers the total processing time of all successors, as well as all of their due dates. Specifically, the modified due dates are defined for all tasks j by setting

$d'_j = d_j$ if $K_j = \emptyset$ and

$$d'_j = \min\left\{d_j, \min_{g \in K_j}\left(d'_g - \left\lceil \frac{1}{2}\left|\{q : q \in K_j \wedge d'_q \leq d'_g\}\right|\right\rceil\right)\right\}. \tag{2.1}$$

As before, these values must be assigned according the reverse order of any linear extension of the precedence constraints. Also as before, $L_{max}(\sigma)$ is unchanged by the replacement of the original due dates by these modified due dates for all schedules σ. The list scheduling algorithm from [21] is then used to schedule all tasks according to these modified due dates.

The optimality of the resultant schedule η is established as follows. Select task x as before, and if η is not optimal, select time t as before as well. By the list scheduling algorithm and the fact that $d'_j < d'_g$ if $j \to g$, there must be a task z with $C_z(\eta) = t$ that precedes task x and all tasks j such that $t < C_j(\eta) < C_x(\eta)$. In this case (2.1) implies

$$L_{max}(\eta) \geq C_z(\eta) - d'_z \geq C_z(\eta) - d'_x + \left\lceil \frac{1}{2}\left|\{q : q \in K_z \wedge d'_q \leq d'_x\}\right|\right\rceil$$

$$\geq C_z(\eta) - d'_x + \left\lceil \frac{1}{2}(1 + 2(C_x(\eta) - 1 - C_z(\eta)))\right\rceil = C_x(\eta) - d'_x = L_{max}(\eta)$$

which contradicts the selection of task x.

As a consequence of this, any problem instance where the precedence constraints are such that there are at most three independent tasks can be solved in polynomial time: if $m \leq 2$ the Garey-Johnson Algorithm will construct an optimal schedule, and if $m \geq 3$ all tasks can be processed as soon as they become available, meaning any list schedule will be optimal.

It is also proven that the problem of minimising the number of tasks completed after their due dates when processed on a single machine, denoted $1|prec, p_j = 1|\sum U_j$, is NP-hard.

In a similar manner to [45], [46] deals with the $P2|prec, p_j = 1, r_j|L_{max}$ problem, i.e. the problem of scheduling precedence constrained unit execution time tasks with release dates on two parallel machines with the goal of minimising the maximum lateness.

The algorithm modifies the due dates of the tasks using the fact that, for any

schedule σ in which no tasks are tardy,

$$C_j(\sigma) \leq d - \left\lceil \frac{1}{2} \left| \{g : g \neq j \wedge d_g \leq d \wedge (g \in K_j \vee r_g \geq r)\} \right| \right\rceil$$

for all $r_j \leq r \leq d_j \leq d$.

This result is the origin of a new algorithm, the optimality of which is demonstrated using reasoning similar to [21] and [45].

The chapter [71] of Deterministic and Stochastic Scheduling is devoted to solving three problems involving the preemptive scheduling of precedence constrained jobs on (not necessarily identical) parallel machines with the goal of minimising the maximum lateness:

- the in-tree problem, denoted $P|in-tree, pmtn|L_{max}$* and which is equivalent to $P|out-tree, pmtn, r_j|C_{max}$, the preemptive analog of the problem addressed in [21];
- the two machine problem with equal release dates, denoted $Q2|prec, pmtn|L_{max}$ and which is equivalent to $Q2|prec, pmtn, r_j|C_{max}$, the preemptive analog of the problem addressed in [45]; and
- the general two machine problem, denoted $Q2|prec, pmtn, r_j|L_{max}$, involving release dates, the preemptive analog of the problem addressed in [46].

As such it uses techniques very similar to those in the above cited papers to achieve these ends: tasks are given modified due dates d'_j and, defining $p_j(t)$ to be the processing time of task j remaining at time t, a greedy algorithm is developed to process tasks j with the highest value of $p_j(t) - d'_j$ first. This can be viewed as a preemptive version of the list scheduling algorithm used in [21], [45] and [46] to schedule unit execution time non-preemptive tasks. It is also a more complex algorithm to achieve the same purpose as the algorithm in [105].

The due date modifications used are also analogs of those found in [21], [45] and [46]. In essence, this chapter translates several well known results for unit execution time tasks to tasks with arbitrary processing times subject to preemptions. It is observed that there are many similarities between these two scheduling problems.

What will be termed the Zinder-Roper Algorithm is introduced in [119]. The

*Technically this paper considers the precedence constraints to be in the form of an in-forest, but this is an unimportant distinction.

algorithm is designed for the $P|prec, p_j = 1|L_{max}$ problem, although derivatives have been used for other problems, see [9] and [11] for example.

Its mechanism is very similar to [21] and [45]. Rather than modify the due dates of the tasks, however, it computes μ_j for each task j. It then uses the list scheduling algorithm – with the tasks in the list in descending order of μ_j rather than ascending order of d'_j – to construct the schedule. This superficial difference does not change the nature of the algorithm: each μ_j can be viewed as $d_{max} - d'_j$, where

$$d_{max} = \max_{j \in N} d_j. \tag{2.2}$$

This means that all μ_j are non-negative, which is useful for the purposes of illustration as these values can be viewed as "delivery times" or "tails," but they otherwise may be viewed as another representation of modified due dates.

For all tasks j for which $K_j \neq \emptyset$, define an auxiliary problem instance φ_j which consists only of the tasks in K_j. The number of machines in problem instance φ_j is the same as in the primary problem instance considered for scheduling, as are all due dates of the tasks in K_j. The precedence constraints between all tasks in K_j are preserved in φ_j.

The calculation of μ_j recursively invokes the scheduling algorithm itself: for each $j \in N$ such that $K_j = \emptyset$, $\mu_j = d_{max} - d_j$, otherwise

$$\mu_j = \max\left\{d_{max} - d_j, G_\mu\left(\eta^j\right)\right\}$$

where

$$G_\mu(\sigma) = \max_{j \in N}\left(C_j(\sigma) + \mu_j\right)$$

and η^j is the schedule constructed by applying the list scheduling algorithm to the auxiliary problem instance φ_j.

Analogously, values λ_j are calculated for each $j \in N$ such that $K_j = \emptyset$, $\lambda_j = d_{max} - d_j$, otherwise

$$\lambda_j = \max\left\{d_{max} - d_j, G_\lambda\left(\omega^j\right)\right\}$$

where

$$G_\lambda(\sigma) = \max_{j \in N}\left(C_j(\sigma) + \lambda_j\right)$$

and ω^j is an optimal schedule for the auxiliary problem instance φ_j. Since finding these optimal solutions is NP-hard, this is not a practical algorithm. It is simply designed to aid the analysis of the schedule produced using the μ algorithm. In this thesis, this algorithm will be known as the Optimal Recursive Algorithm.

It is proven that $G_\lambda(\sigma) = L_{max}(\sigma) + d_{max}$ for all schedules σ, using similar reasoning to the proofs that the due date modification procedures in [21] and [45] leave the maximum lateness unchanged. However, it may be the case that $G_\mu(\sigma) > L_{max}(\sigma) + d_{max}$: the potential for non-optimal schedules η^j creates this possibility.

It is proven that the μ algorithm produces an optimal schedule for the $P2|prec, p_j = 1|L_{max}$ problem using induction on the number of tasks n. Specifically it is trivially true in the case of $n = 1$. It is then assumed true for all $n \leq k$. Then it is noted that, in the case where $n = k + 1$, $|K_j| \leq k$ implying $\mu_j = \lambda_j$ for all $j \in N$. The optimality of the schedule is then proven in a similar manner to the proof of the optimality of the Garey-Johnson Algorithm in [45].

This paper also presents a performance guarantee for this algorithm, see Section 2.3 for further details.

A version of the Zinder-Roper Algorithm (originally specified in [119] for the $P|prec, p_j = 1|L_{max}$ problem) as applied to the $P|prec, pmtn|L_{max}$ problem is considered in [120]. Specifically, this paper adapts the μ assignment algorithm from [119] to the case of tasks with preemptions and arbitrary processing times, and uses the scheduling algorithm from [105] to construct a schedule on this basis.

It is then proven that the resulting algorithm is optimal for the case of two machines as well as the case where the precedence constraints are in the form of in-trees, using reasoning similar to that used in [45] and [21] respectively.

A performance guarantee for this algorithm is also presented, see Section 2.3 for further details.

The paper [100] is rather short, and is based on existing work in [88] for the case of unit execution time, non-preemptive tasks, and similar to a later paper [3] on unit execution time, non-preemptive tasks with unit communication delays. It has been generalised somewhat in [13], [86] and [35].

Considered is the problem $P|prec, pmtn|C_{max}$ with the additional restriction that the precedence constraints must take the form of an interval order, a particular class of partial orders. The paper describes an algorithm for solving this problem in polynomial time.

First a set is defined:

$$[N]^{\leq m} = \{B : B \subseteq N \wedge |B| \leq m \wedge B \text{ antichain}\}.$$

This is the set of all antichains (sets of task that do not precede one another)

with cardinality not greater than m. A partial order $<$ is defined on $[N]^{\leq m}$ for $B, B' \in [N]^{\leq m}$ such that $B < B'$ if and only if there exists $b \in B$ and $b' \in B'$ such that $b \to b'$. The existence of this partial order is a consequence of the definition of an interval order. Once this has been done, Linear Program 2.1 is solved.

$$\begin{aligned} \min \quad & \sum_{B \in [N]^{\leq m}} x_B & \\ \text{s.t.} \quad & \sum_{j \in B \in [N]^{\leq m}} x_B = p_j & \forall j \in N \\ & x_B \geq 0 & \forall B \in [N]^{\leq m} \end{aligned}$$

Linear Program 2.1: The linear program used to determine the length of individual intervals while scheduling interval ordered tasks.

The variables x_B indicate how long an interval the tasks in B are processed on. These intervals may be arranged according to any linear extension of $<$. This method of using a linear program to solve a simple (but non-trivial) scheduling problem is similar to the method found in [73], which solves the problem of scheduling preemptive tasks without precedence constraints on any fixed number m of machines, where each task j has a different processing time $p_{j,i}$ on each machine i, with the goal of minimising the maximum lateness.

Both upper and lower bounds on the optimal makespan for the $P|prec, pmtn|C_{max}$ problem are presented in [35]. However, what is most relevant to this thesis is that, in the process of providing these bounds, an optimal algorithm for interval orders is presented, and unlike the formulation in [100] it is polynomial even when m is not fixed.

The method used is similar in concept to that employed in [100]: the tasks are broken up into antichains and then scheduled in intervals determined by a linear program. There are two key differences. The first is that only maximal antichains, i.e. antichains to which no additional independent task can be added, are considered. The second is that the tasks might be scheduled for different amounts of time in a given interval, requiring that McNaughton's algorithm be used.

In [70] another class of precedence constraints, known as series-parallel, is considered. It is proven that although $1|prec|\sum w_j C_j$ is (weakly) NP-Hard, it can be solved in polynomial time if the precedence constraints take the form of a series-parallel partial order. The problem $P2||\sum w_j C_j$ is known to be (weakly) NP-Hard

(see [48]), and the $P2|prec|\sum w_jC_j$ problem has been addressed using integer programming (see [95] for example). For more information on scheduling in the presence of series-parallel precedence constraints, see [43], [115], [1], [83] and [102]. A more general problem is analysed in [59].

2.3 Performance Guarantees

As stated in Section 2.1, most of the problems considered in this thesis are NP-hard. However, approximation algorithms are an area of considerable study; [44] and [68] are two well-known early examples. This section is devoted to the establishment of bounds on the performance of approximation algorithms, in particular approximation algorithms for minimising the makespan or maximum lateness. These results take the form of upper bounds on the objective function (maximum lateness, makespan etc.) of a schedule η – produced by a particular approximation algorithm – in terms of the optimal value of the objective function, and perhaps some other parameters of the problem instance.

Two kinds of bounds appear most often: ratio bounds and difference bounds. Ratio bounds are upper bounds on

$$\frac{C_{max}(\eta)}{C_{max}(\omega)} \qquad \text{or} \qquad \frac{L_{max}(\eta)+d_{max}}{L_{max}(\omega)+d_{max}}$$

where d_{max} is defined as in (2.2). Some of the earliest performance guarantees are of this type. For example, it was proven in [49] (published in 1969) that $\frac{C_{max}(\eta)}{C_{max}(\omega)} \leq 2-\frac{1}{m}$ for a simple list scheduling algorithm applied to the problem $P|prec|C_{max}$. When considering the maximum lateness, the criterion $\frac{L_{max}(\eta)}{L_{max}(\omega)}$ because the maximum lateness can be negative or zero. Since ratio bounds only makes sense if the numerator and denominator are both guaranteed to be positive, the largest due date is added to both terms. This kind of criterion has been used, for example, in [119].

Because it is often difficult to prove the tightness of such bounds, ratio bounds may take the form

$$C_{max}(\eta) \leq \delta_1 C_{max}(\omega)+\delta_2 \qquad \text{or} \qquad L_{max}(\eta) \leq \delta_1 L_{max}(\omega)+(\delta_1-1)d_{max}+\delta_2$$

for various values of δ_1 and δ_2.

A bound of this form for was obtained in [27] and [28] for the algorithm described

in [60] and was proven to be asymptotically tight; specifically

$$\frac{C_{max}(\eta)}{C_{max}(\omega)} \leq \begin{cases} \frac{4}{3} & \text{if } m = 2 \\ 2 - \frac{1}{m-1} & \text{if } m \geq 3 \end{cases}.$$

In [26] a new algorithm is presented for the minimisation of the makespan for a more general model of malleable tasks than is considered in this thesis. The upper bound on $\frac{C_{max}(\eta)}{C_{max}(\omega)}$ obtained is 3.42 in general and 2.96 given certain assumptions. For a similar problem, [42] provides an approximation algorithm where the upper bound is 2 in general, but less for any specific value of m.

Difference bounds are upper bounds on

$$C_{max}(\eta) - C_{max}(\omega) \qquad \text{or} \qquad L_{max}(\eta) - L_{max}(\omega),$$

where η is the schedule produced by the approximation algorithm and ω is an optimal schedule. These bounds are often given in terms of the number of machines m and the length l of the longest chain of tasks in the problem instance. For the purposes of this definition, a chain is defined as a set of tasks $S \subseteq N$ such that, for all $j \in S$ and $g \in S \setminus \{j\}$, either $j \to g$ or $g \to j$. The length of a chain S for the $P|prec, pmtn, u_j|L_{max}$ problem is defined to be

$$\sum_{j \in S} \frac{p_j}{u_j}.$$

For the $P|prec, pmtn|L_{max}$ problem, i.e. the problem with non-malleable tasks, this reduces to the sum of processing times for all tasks in the chain. Difference bounds are generally easier to prove tight than ratio bounds, which is one reason for their use. Additionally, as will be seen in Chapter 5 of this thesis, it is simple to transform a difference bound into a ratio bound, so it may be that difference bounds are in some sense more appropriate measures of an approximation algorithm's quality.

A bound of this form the algorithm described in [60] for the $P|prec, p_j = 1|C_{max}$ problem was obtained in [106] and proven to be tight; specifically

$$C_{max}(\eta) - C_{max}(\omega) \leq \frac{n}{m(m-1)} + \frac{l(m-3)}{m-1}.$$

In [75] a linear programming method for determining a difference bound for the problem $P|prec, r_j|L_{max}$ is described.

Both ratio and difference bounds are derived in Chapter 5 of this thesis. One additional fact should be noted: often approximation algorithms for NP-hard problems

arise from polynomial algorithms designed to optimally solve certain subproblem. This is the case with many of the algorithms referenced in this section. Recent work on performance guarantees includes [29], [57], [30], [8] and [96].

The paper [105] presents two difference-type worst case performance guarantees, one for the Brucker-Garey-Johnson Algorithm (originally specified in [21] for a special case of the $P|prec, p_j = 1|L_{max}$ problem) and the other for its preemptive counterpart (described in [71] for the $P|prec, pmtn|L_{max}$ problem).

Once these modified due dates have been computed, they are supplied to an algorithm which will construct a schedule. For the $P|prec, p_j = 1|L_{max}$ problem this algorithm is the list scheduling algorithm used in [21], as described previously. For the $P|prec, pmtn|L_{max}$ problem a greedy algorithm, similar to but simpler than that described in [71], schedules tasks in intervals, giving priority to tasks with the largest value of $p_j(t, \sigma) - d_j$, where $p_j(t, \sigma)$ is the remaining processing time of task j at time t in schedule σ.

Both bounds are very similar, and the proof of both rely on a similar procedure. In particular, an important task, denoted x in the preemptive case, is selected such that

$$C_x(\eta) - d'_x = L_{max}(\eta).$$

After the selection of this task, all tasks less urgent, i.e. all tasks j such that for all times $0 \leq t \leq C_j(\eta)$

$$t + p_j(t, \eta) - d'_j < L_{max}(\eta)$$

may be disregarded. Intervals in time are divided into complete (wherein all machines process "important" tasks at all times) and incomplete. It follows from the manner in which the due dates are modified that a lower bound on $L_{max}(\omega)$ is given by the sum of all important processing divided by m, plus a term for the priority of task x.

Another lemma establishes the existence of a chain of important tasks extending through the length of all incomplete intervals. Combining these observations gives

$$L_{max}(\eta) - L_{max}(\omega) \leq \frac{m-1}{m} l \tag{2.3}$$

for the case of non-preemptive unit execution time tasks and

$$L_{max}(\eta) - L_{max}(\omega) \leq \frac{m-1}{m} (l - p_{min}) \tag{2.4}$$

for the case of preemptive tasks, where

$$p_{min} = \min_{j \in N} p_j.$$

Both (2.3) and (2.4) are proven to be tight, although the latter is proven tight only for integer values of $\frac{l}{p_{min}}$, and in fact is not tight for $l < 2p_{min}$.

An additional bound for the case of non-preemptive unit execution time tasks is presented:

$$L_{max}(\eta) - L_{max}(\omega) \leq \begin{cases} \min\{\frac{n-l}{m} - 1, \frac{m-1}{m} l\}, & \text{if } n - l \geq m \\ 0 & \text{otherwise} \end{cases}.$$

Presented in [55] are two ratio-type performance guarantees for the preemptive counterpart of Garey-Johnson Algorithm, originally specified in [45] for the $P2|prec, p_j = 1|L_{max}$ problem, adapted to the $Q2|prec, pmtn|L_{max}$ problem in [71]. The problem considered is $P|prec, pmtn|L_{max}$.

The notation for the algorithm differs from that in [45] and [71] because, rather than modifying the due dates, it introduces values μ_j for each task j, a notation also used in [119]. Nonetheless, the algorithm does not differ is any essential way from that described in [71]: the constant term d_{max} is simply added to all modified due dates. The notation $Q_{max}(\sigma) = \max_{j \in N}(C_j(\sigma) + \mu_j)$ is defined to simplify exposition, and it is then demonstrated that

$$Q_{max}(\sigma) = \max_{j \in N}(C_j(\sigma) + \mu_j) = \max_{j \in N}(C_j(\sigma) - d_j) + d_{max} = L_{max}(\sigma) + d_{max}$$

for all schedules σ, where d_{max} is defined as in (2.2). This implies that minimising Q_{max} will also minimise the maximum lateness, and indeed that these two criteria are equivalent. The algorithm used to construct the schedule once these values have been computed is equivalent to that in [105].

The construction of the bounds is achieved in a similar manner to that in [105], i.e. an important task x is selected and then all processing that is at least as urgent is considered, dividing time intervals into complete and incomplete as before.

Two key results are used in the construction of the bounds:

- a lower bound on an optimal schedule is obtained from the total time taken up by "important" processing as in [105] and
- a similar form of lower bound is obtained from the amount by which the μ's

of tasks are affected by their successors.

A linear combination of these two bounds eliminates all extraneous variables, giving

$$\frac{L_{max}(\eta) + d_{max}}{L_{max}(\omega) + d_{max}} \leq \left(1 + \left(1 - \frac{2}{m}\right) \frac{K_{max} - m}{K_{max} - 2}\right)$$

where K_{max} is the maximum between m and the largest number of tasks scheduled in a single iteration by the schedule construction algorithm. Further reasoning leads to

$$\frac{L_{max}(\eta) + d_{max}}{L_{max}(\omega) + d_{max}} \leq 2 - \frac{3}{m+1}.$$

Neither of these bounds is proven to be tight, but it is proven that there exist problem instances for which

$$\frac{L_{max}(\eta) + d_{max}}{L_{max}(\omega) + d_{max}} \geq 2 - \frac{2}{\sqrt{m}+1}.$$

As stated in the previous section, [119] describes an algorithm for scheduling nonpreemptive unit execution time tasks on the basis of recursively calculated values μ_j for all $j \in N$.

Similarly to [105], a task x is selected such that

$$C_x(\eta) + \mu_x = G_\mu(\eta)$$

and this task is used as a cutoff for the priority of "important" tasks. Time intervals are again divided into complete and incomplete, but for the Zinder-Roper Algorithm it is possible to prove that at least two important tasks are processed during every interval.

It is then demonstrated that

$$L_{max}(\eta) \leq \left(2 - \frac{2}{m}\right) L_{max}(\omega) + \left(1 - \frac{2}{m}\right) d_{max} - r(m)$$

where

$$r(m) = \begin{cases} \frac{m-3}{m} & \text{if } m \text{ odd,} \\ \frac{m-2}{m} & \text{if } m \text{ even.} \end{cases}$$

This bound is proven tight. This result is similar to that proven in [54] for the Garey-Johnson Algorithm. It is remarked that, for the case where all $d_j = 0$, i.e. the makespan, this bound reduces to a well known bound, derived in [68] and [14],

for another algorithm, originally presented in [32].

As stated in the Section 2.2, [120] describes an algorithm for the $P|prec, pmtn|L_{max}$ problem on the basis of recursively calculated values μ_j for all $j \in N$.

Using a procedure similar to [119] it is proven that at least two "important" tasks are processed in each interval, and consequently it is proven that

$$L_{max}(\eta) \leq \left(2 - \frac{2}{m}\right) L_{max}(\omega) + \left(1 - \frac{2}{m}\right) d_{max}.$$

Examined in [117] is the $P|prec, p_j = 1|L_{max}$ problem, i.e. the problem of scheduling nonpreemptive unit execution time tasks, subject to precedence constraints, on parallel machines with the goal of minimising the maximum lateness. Considered is a general family $\mathfrak{Q}$ of algorithms, based on the list scheduling algorithm described earlier.

Algorithms in $\mathfrak{Q}$ generate a partial order on the set N of all tasks. The list which is used to create the schedule is any linear extension of this partial order. This is significantly more general than previously considered applications of the list scheduling algorithm, accepts not a partial order but rather a list of real numbers, one for each task, and sorts the tasks in order of the values.* There is one further requirement for the membership of a given algorithm in the considered family, although it is too technical to give rigorously in a survey of the literature.†

With the definition of this family of algorithms, two interesting results are derived. The first is that, for every algorithm $\mathscr{A} \in \mathfrak{Q}$, there exists a constant γ such that the performance guarantee

$$L_{max}\left(\eta^{\mathscr{A}}\right) \leq \gamma L_{max}\left(\omega\right) + (\gamma - 1) d_{max}$$

holds, where $\eta^{\mathscr{A}}$ is the schedule generated by the algorithm $\mathscr{A}$ and ω is an optimal schedule. Defining $\gamma(\mathscr{A})$ to be the smallest value of γ for which this performance guarantee holds, it is then proven that the Zinder-Roper Algorithm, as defined in

*Tasks ordered in this manner can be said to be subject to a "weak order," defined in Section 3.1.

†It can be summarised as the requirement that an algorithm will produce consistent partial orders when applied to certain subinstances of the original problem instance.

[119], has the smallest value of $\gamma(\mathscr{A})$ among all $\mathscr{A} \in \mathfrak{Q}$, specifically

$$\gamma(\mathscr{LR}) = 2 - \frac{2}{m}$$

as obtained in [119].

2.4 Branch and Bound

The general technique known as branch and bound, first proposed in [69], is widely regarded (see [74], [79] and [31]) as one of the most effective methods for solving NP-Hard problems. Although this thesis does not directly address branch and bound methods for solving the various NP-Hard problems considered, it does make a substantial contribution that in the future could be used in such methods – see Chapter 7. The purpose of this section of the literature review it to provide an overview of the kinds of methods recently examined in the literature, for the purpose of providing a better idea of the state of the art.

For the most part, recent results in this area have dealt with non-preemptive scheduling, and that is what this section of the literature review will focus on. Chapter 7 describes a process by which a problem with preemptive tasks can be transformed into a combinatorial optimisation over a set of linear programs, expressing the problem in a form suited to a branch and bound algorithm.

In two well known papers, [90] and [91], the notion of dominance of one segment of the search space over another. More specifically, it is often possible to prove that there always exists an optimal schedule with a particular property, allowing the search space to be reduced substantially. In [108] a generalised concept based on these two papers, known as the dynamic programming dominance property, is described. The paper considers three related non-preemptive scheduling problems, in which the branch and bound search space is based on the order in which tasks are processed. It is noted that, given a subset $S \subseteq N$ of tasks, and given two partial schedules σ^1 and σ^2 of the tasks in S, if

- the total contribution to the objective function from σ_1 does not exceed that of σ_2, and

- the total processing time used on each machine for σ_1 does not exceed the corresponding value for σ_2, then

the node in the branch and bound scheme represented by σ^2 is dominated by σ_1, and need not be considered. Essentially, if an optimal schedule can be constructed out of σ^2 then one can also be constructed out of σ_1. This reduction of the search space can dramatically improve the performance of the branch and bound procedure.

It is then demonstrated that in many cases only a small subset of pairs of nodes need to be checked for dominance. Computational experiments are performed which demonstrate a large performance boost from the application of these results, although the amount is dependent on both the search strategy employed and the problem considered.

The problem $1|r_j|\sum U_j$ is studied in [34]. The addition of preemptions makes this problem polynomial (see [72]) as does the removal of release times (see [84]) but the problem considered in this paper is NP-Hard (see [77]).

The branch and bound process employed in [34] adapts a concept introduced in [33]: the master sequence. Using an analysis of dominance similar to what is found in [108], the possible orderings of tasks are limited substantially while still guaranteeing that an optimal schedule can be found. The results are then encoded (in polynomial time) in a "master sequence" of tasks, with length at most $\frac{n(n+1)}{2}$; this sequence has the property that at least one optimal ordering of tasks is contained in it.

The master sequence is used as the basis for the branch and bound procedure, and allows quick computation of several dominance rules established in the paper. Computational experiments are then performed and the results are compared (favourably) to those obtained by a different branch and bound approach in [7].

An interesting hybrid approach was developed in [121]. In this paper the search space over which the branch and bound algorithm operates is not the entire set of feasible schedules; rather, it is the set of task priorities used in an approximation algorithm akin to those proposed in [21], [45] and [119].

More specifically, the problem considered is a modification of the $P|prec, p_j = 1 |L_{max}$ problem to include release times (lower limits on the time at which tasks can begin processing) and unit communication delays (when a task j is completed its successors must wait an additional unit of time if they are processed on any machine other than the one on which j was processed). This problem is known as $P|prec, r_j, p_j = 1, c_{jg} = 1|L_{max}$. Each task j is given a priority μ_j, and then the tasks are scheduled using a simple list scheduling algorithm. Once this has been

done, the schedule produced is used to select a set of tasks on which to branch – it is proven that at least one task in the selected set can be given a larger value of μ without worsening the objective function – and then different subproblems are considered based on the selection of which task in the set will have an increased value of μ. Each subproblem results in a new schedule generated by the approximation algorithm. It is proven that there is some possible selection of μ's that results in an optimal schedule, demonstrating that the branch and bound method will eventually minimise the objective function if executed to completion.

This method benefits from the power of a customised approximation algorithm to help select initial solutions and to determine key tasks on which to branch, and demonstrates a useful new direction for new research.

2.5 Order Theory

Order theory is the study of partial orders and related structures. Two common subjects of study in this field, both of which are relevant to this thesis, are the properties of specific classes of partial orders, and the enumeration of the number of elements in such classes. These subjects are relevant because the precedence constraints of a problem instance can be represented as a partial order. Indeed, some classes of partial orders have arisen naturally from the study of scheduling problems. Several of the polynomially solvable special cases of NP-hard scheduling problems are defined in terms of a particular class of precedence constraints, e.g. in-trees in [21] and interval orders in [35]. In [85], the class of quasi-interval orders was defined and characterised, and an algorithm was presented to solve the $P|prec, p_j = 1|C_{max}$ in polynomial time in the case where the precedence constraints take the form of a quasi-interval order. In [25] it was proven that the Coffman-Graham Algorithm (see [32]) optimally solves a related problem for an even broader class of precedence constraints they termed overinterval orders.

The intersection between order theory and scheduling theory is explored in depth in Chapter 3, and can be seen throughout this thesis. Two general works covering the enumeration of partial orders are [41] and [87]. For more information on different classes of partial orders, see [82].

The paper [92] examines the relationship between binary relations on labelled and unlabelled elements.

It is often desired to find an asymptotic formula for the number A_n of some class of binary relation on n unlabelled elements. Usually it is easier to find a formula for

the number B_n of the structure on n labelled elements. In situations such as this a common solution is to transform the asymptotic formula for B_n into a formula for A_n by proving bounds on the number of automorphisms of the labelled structures. Usually this is done on a case-by-case basis, but this paper provides a general result for a broad class of problems.

The main lemma, on which the entire paper is based, states that if there is an infinite class of finite labelled structures defined by a binary relation which

1. is closed under substructures and isomorphisms and

2. for some $c > 0$, arbitrary d and two functions $\zeta(n) \in o(n)$ and $\xi(n) \in o(n)$

$$cn^2 + dn + \zeta(n) < \log_2 B_n < cn^2 + dn + \xi(n),$$

then there exists some constant s such that

$$A_n \leq \frac{B_n}{n!}\left(1 + \frac{s}{2^{cn}}\right).$$

This result may be used to prove several existing results (on the number of graphs for example) as well as some previously unknown results. One result original to this paper is that,

$$A_n \sim \frac{1}{n!}\sum_{i=1}^{n}\sum_{j=1}^{n-i}\binom{n}{i}\binom{n-i}{j}(2^i-1)^j(2^j-1)^{n-i-j},$$

where A_n is the number of partial orders on n unlabelled elements. This follows from the main lemma and the well known Kleitman-Rothschild Theorem, originally proven in [66].

The Kleitman-Rothschild Theorem provided an asymptotic formula for the number B_n of partial orders on n labelled elements. The paper [18] sets out to prove the same result in a much simpler way while also providing an asymptotic formula for the number of linear extensions E_n of partial orders on n labelled elements.

The first theorem states that for some constant $c > 1$

$$B_n = (1 + O(c^{-n}))\sum_{s=0}^{n}\binom{n}{s}2^{(s+1)(n-s)}.$$

The second theorem is a rather technical result but leads to the corollary that

$$E_n = C_{(n \bmod 2)} \left\lfloor \frac{n}{2} \right\rfloor ! \left\lceil \frac{n}{2} \right\rceil ! n 2^{-\frac{n}{2}} \left(1 + O\left(\frac{1}{n}\right)\right),$$

where

$$C_0 \approx 5.041445 \qquad \text{and} \qquad C_1 \approx 5.041419.$$

That is, E_n is asymptotic to a slightly different function depending on whether n is even or odd.

In [19] a new method for enumerating partially ordered sets (posets) is described, and then applied to the enumeration of isomorphically distinct partial orders on up to sixteen elements, as well as of partial orders on up to eighteen labelled elements.

Previously, attempts at the enumeration of partial orders proceeded by generating many partial orders on labelled elements, and then testing them for isomorphism. Since this testing is very time consuming, a different method was used in this paper: canonical construction paths, which guarantee that precisely one partial order will be generated for each isomorphism class. The concept had been invented previously but had not yet been applied to the enumeration of partial orders.

The idea is to construct posets on n elements from posets on $n-1$ elements by adding a single element in a canonical way. More specifically, a collection of labelled posets, called candidates, containing at least one element from each isomorphism class of posets is created by the addition of this single element. For each candidate a "canonical orbit" of elements under the automorphism group of the candidate is selected and the candidate is accepted if and only if the candidate was constructed by the addition of an element in the canonical orbit.

After describing the algorithm, the paper presents a large number of tables, containing counts of isomorphically distinct and labelled posets on different numbers of elements with a number of different properties.

The paper [16] provides asymptotic results on the number of partial orders that have an upper bound on their width. The details defy summary here, but there are three key results. Firstly it is proven that the number of labelled partial orders of width at most two is

$$n! \frac{4^n}{n^{\frac{3}{2}}} \frac{8}{25\sqrt{\pi}} (1 + o(1)).$$

Of interest for this thesis is the second result, that the number of unlabelled partial

orders of width at most two is

$$n!\frac{4^n}{n^{\frac{3}{2}}}\frac{4(2-\sqrt{3})}{3\sqrt{\pi}}(1+o(1)).$$

Together the above results are curious, as described in the quote below.

> Combining the two results, we see that the mean number of automorphisms of a labelled width 2 partial order [on n elements] tends to
>
> $$\frac{25}{6}(2-\sqrt{3}) \approx 1.11645, \text{ as } n \to \infty.$$
>
> This is perhaps a little surprising at first sight, since the most familiar random structures either have a trivial automorphism group, or have very many automorphisms, almost surely.

The third result is that the number B_n^k of labelled partial orders of width at most k on n elements obeys

$$D_1(k)n!4^{n(k-1)}n^{-C_1k^2} \leq B_n^k \leq D_2(k)n!4^{n(k-1)}n^{-C_2k^2}$$

for some positive constants C_1 and C_2 and some functions $D_1(k)$ and $D_2(k)$.

Given an initial collection S of finite posets, consider the class of posets that can be built from them using the operations of series composition (ordinal sum) and parallel composition (disjoint union). These posets are known as series-parallel. In [111] a method is described for recognising them in linear time.

If S contains a partial order on only one element then the resultant class is known as series-parallel. The goal of [107] is to present an asymptotic formula for the number A_n of unlabelled posets of this type of class of partial orders on n elements.

A formula for the generating function

$$F(x) = \sum_{n=0}^{\infty} A_n x^n$$

is presented, and the particular case of series parallel posets is analysed. It is discovered that $A_n \sim Cn^{-\frac{3}{2}}\alpha^{-n}$ where α is the unique positive root of

$$F(x) - \frac{1+\sqrt{5}}{2}$$

and

$$C = \sqrt{\frac{1}{\pi(3\sqrt{5}-5)}\left[\frac{\alpha}{1-\alpha} + \sum_{k=2}^{\infty} \alpha^k F'(\alpha^k)\left(1 - \frac{1}{F(\alpha^k)^2}\right)\right]}.$$

This asymptotic formula has the same form as that for in-trees derived in [87].

Chapter 3

Computational Complexity

It is well known that $P|prec, pmtn|C_{max}$ and $P|prec, pmtn|L_{max}$ are both NP-hard – see for example [110]. However, there are some subproblems, noted in [71] for example, that are amenable to solution by polynomial algorithms. This chapter considers the computational complexity of particular kinds of subproblems: those with restricted classes of precedence constraints that can be described by prohibited subgraphs. It is in part a very specific survey of the literature, but it also includes results derived in this thesis. For a survey of the existing state of the art, see [94].

3.1 Classes of Partial Orders

A partial order (also known as a partially ordered set or poset) P can be rigorously defined as a binary relation over a set $N(P)$* that is irreflexive, antisymmetric and transitive. The elements of $N(P)$ will also be referred to as the elements of the partial order P. Sometimes partial orders are said to be reflexive but this issue does not change their fundamental structure. They may also be considered to be specific kinds of digraphs, and will often be illustrated as such in this thesis.

It is important to distinguish between partial orders on labelled and unlabelled elements. As defined above, the elements of $N(P)$ are part of the definition of a partial order P. A new partial order may be derived from P simply by changing the elements of the set $N(P)$, i.e. by changing the "labels" of the elements. This renumbering of elements does not change the underlying structure of the partial order, with which this thesis is concerned. Hence the notion of poset isomorphism is introduced: two partial orders P and Q are isomorphic if and only if there exists a bijection f from $N(P)$ to $N(Q)$ with the property that $x\,P\,y$ if and only if $f(x)\,Q\,f(y)$. Poset isomorphism is an equivalence relation; hence the isomorphism

*It can be assumed that this is a set of natural numbers, but the specific elements of $N(P)$ are unimportant for the purposes of this thesis.

class of a partial order P can be defined as the class of all partial orders isomorphic to P. Such an isomorphism class can be viewed as a poset on unlabelled elements, in comparison to the earlier definition of a poset on labelled elements. Unless otherwise noted, all partial orders considered in this thesis are partial orders on labelled elements.

An induced suborder Q of P is defined as a partial order such that $N(Q) \subseteq N(P)$ and, for all $x \in N(Q)$ and $y \in N(Q)$, $x \, Q \, y$ if and only if $x \, P \, y$; Q is said to be the suborder of P induced by $N(Q)$. No other kind of suborder will be considered in this thesis so, to simplify notation, all suborders referred to in this work are induced suborders. For any two partial orders P and Q, P may be said to be Q-free if P does not contain any subgraph that is isomorphic to Q. The classes of precedence constraints examined in this chapter are classes of Q-free partial orders for different values of Q.

If Q is a partial order on zero, one or two elements the class of Q-free graphs is rather trivial:

- if Q is the unique partial order on zero elements, no partial orders are Q-free;
- if Q is a partial order on a single element, only the partial order on zero elements is Q-free;
- if Q is a partial order on two elements x and y, for each $n \geq 0$ there is only one poset on n unlabelled elements that is Q-free:
  - if $x \not Q \, y$ and $y \not Q \, x$ then the class of Q-free partial orders is the class of chains,
  - if $x \, Q \, y$ or $y \, Q \, x$ then the class of Q-free partial orders is the class of antichains.

As a consequence of the uninteresting nature of these classes of partial orders, only posets on three or more elements will be considered. There are five posets on three unlabelled elements, as depicted in Figure 3.1.

- The width of a partial order P is the largest cardinality $|S|$ among sets $S \subseteq N(P)$ such that $x \not P \, y$ for all $\{x, y\} \subseteq S$, i.e. among sets $S \subseteq N(P)$ for which the corresponding subgraph of P is an antichain. The class of A-free partial orders is the class of partial orders with width at most two.
- The height of a partial order P is the largest cardinality $|S|$ among sets $S \subseteq N(P)$ such that $x \, P \, y$ or $y \, P \, x$ for all $\{x, y\} \subseteq S$, i.e. among sets $S \subseteq N(P)$ for which the corresponding suborder of P is a chain. The class of B-free partial

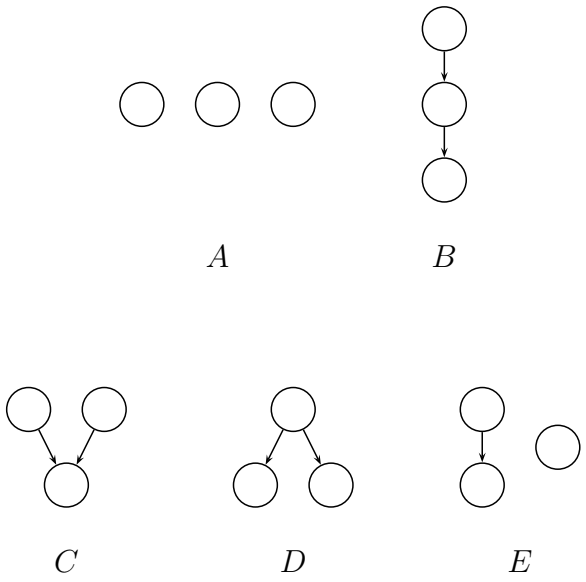


Figure 3.1: Partial orders on three elements.

orders is the class of partial orders with height at most two. Such partial orders P are also known as bipartite orders, because $N(P)$ may be partitioned into two sets S_1 and S_2 such that if $x \, P \, y$ then $x \in S_1$ and $y \in S_2$.

- Partial orders that are C-free are known as out-forests, composed of one or more out-trees.
- Partial orders that are D-free are known as in-forests, composed of one or more in-trees.
- Partial orders P that are E-free are known as weak orders. This means that $N(P)$ may be partitioned into one or more sets $S_1, S_2, \cdots$ such that, for any $x \in S_a$ and $y \in S_b$,
  - $a < b$ implies $x \, P \, y$ and
  - $a = b$ implies $x \not P \, y$.

There are sixteen posets on four unlabelled elements. Only some of them are referred to in this chapter, and the prohibited subgraphs defining these classes of interest appear in Figure 3.2.

- Partial orders that are F-free have width at most three.

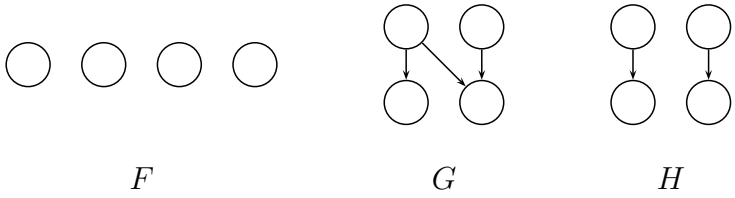


Figure 3.2: Partial orders on four elements.

- Partial orders that are G-free are known as series-parallel orders. These may be defined recursively, as noted in Section 2.5. Suppose there are two partial orders: P_1 and P_2 where $N(P_1) \cap N(P_2) = \emptyset$. Define the series composition of P_1 onto P_2 as being the poset P with $N(P) = N(P_1) \cup N(P_2)$ such that

  - $x \in N(P_1)$ and $y \in N(P_1)$ implies $x \, P \, y$ if and only if $x \, P_1 \, y$,
  - $x \in N(P_2)$ and $y \in N(P_2)$ implies $x \, P \, y$ if and only if $x \, P_2 \, y$ and
  - $x \in N(P_1)$ and $y \in N(P_2)$ implies $x \, P \, y$.

  Define the parallel composition of P_1 and P_2 as being the poset P with $N(P) = N(P_1) \cup N(P_2)$ such that

  - $x \in N(P_1)$ and $y \in N(P_1)$ implies $x \, P \, y$ if and only if $x \, P_1 \, y$,
  - $x \in N(P_2)$ and $y \in N(P_2)$ implies $x \, P \, y$ if and only if $x \, P_2 \, y$ and
  - $x \in N(P_1)$ and $y \in N(P_2)$ implies $x \not P \, y$.

  A poset on one element is defined to be series-parallel. A series composition of one series-parallel poset onto another is series-parallel. A parallel composition of two series-parallel posets is series-parallel.

- Partial orders that are H-free are known as interval orders. These are so named because they arise from a particular ordering of open intervals on the real number line. Suppose there is a set of intervals

  $$S = \{(l_1, r_1), (l_2, r_2), \cdots\}.$$

  An interval order P is constructed from this by defining $N(P)$ with a one-to-one correspondence with the intervals in S so that, for $x \in N(P)$ and $y \in N(P)$,

– if $r_x < l_y$ then $x \, P \, y$,

– if $r_y < l_x$ then $y \, P \, x$ and

– $x \not P \, y$ and $y \not P \, x$ otherwise.

These eight classes of partial orders will be examined in the following section.

3.2 Results

To the author's knowledge, no previous research has attempted, in an organised fashion, to compare

- the computational complexity of a scheduling problem subject to a particular class of precedence constraints to
- the number of partial orders in the given class.

Since the eight considered classes of partial orders are infinite, they will be considered on the basis of how rapidly the number of partial orders on a given number of elements grows.

Enumerating partial orders is well known to be difficult: [19], for example, describes the significant effort expended in enumerating all isomorphism classes of partial orders on up to sixteen elements. Random generation of partial orders is also difficult: see [15]. However, there are asymptotic results regarding the number of partial orders on a given number of elements in a particular class.

An asymptotic formula for the total number of partial orders on a given number of elements is known. In [18] an asymptotic formula for the number of posets on n labelled elements is given. This result is used in [92] to give a figure for the number of posets on n unlabelled elements, i.e. the number of isomorphism classes of partial orders on n elements. A somewhat simplified version of this formula for the approximate number A_n of posets over n unlabelled elements is

$$A_n \sim \frac{\phi_{(n \bmod 2)}}{n!} 2^{\frac{(n+1)^2}{4}} \binom{n}{\lfloor \frac{n}{2} \rfloor} \sim \frac{\psi_{(n \bmod 2)}}{n! \sqrt{n}} 2^{\frac{n^2+6n+1}{4}}$$

where

$$\phi_0 \approx 2.1289312 \qquad \text{and} \qquad \phi_1 \approx 2.1289368$$

and $\psi_x = \phi_x \sqrt{\frac{2}{\pi}}$. Notice that there is a very slightly different constant coefficient for odd and even n.

Similar results are available for some of the classes considered, but in other cases more crude measurements are necessary. In particular, [17] separates classes

Family	Asymptotic Number
I	$B_n = n!(c + o(1))^n$ for some constant c
II	$n^{c_1 n} \leq B_n \leq n^{c_2 n}$ for some $c_2 \geq c_1 > 1$
III	$B_n = n^{\omega(n)}$ and $B_n = 2^{o(n^2)}$
IV	$B_n = 2^{\frac{n^2}{4}+o(n^2)}$

Table 3.1: The four poset class families.

of posets into different families based on the rate of growth of the number B_n of labelled posets on n elements as n increases. The families are described in Table 3.1. Rather surprisingly, all classes of posets defined by the exclusion of a given suborder are members of one of these four families as long as the prohibited suborder is on at least three elements. Note that the $\omega(n)$ notation in this context indicates that if $f(n) = \omega(g(n))$ then

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0.$$

Table 3.2 is the centrepiece of this chapter, summarising the body of existing knowledge, along with some new results original to this thesis. The scheduling problems considered take the form of $P|\mathcal{Q}, pmtn|\gamma$, where

- $\mathcal{Q}$ indicates the presence of a particular class $\mathcal{Q}$ of precedence constraints, each of which is considered in a row of Table 3.2, and

- γ is either C_{max} or L_{max}, considered in the second and third columns of Table 3.2 respectively.

Of the sixteen scheduling problems examined in this chapter, the computational complexity of all except $P|series-parallel, pmtn|C_{max}$ and $P|interval\ order, pmtn|L_{max}$ are known. Some of the results follow directly from the fact that $P|\mathcal{Q}, pmtn|C_{max}$ is a particular case of $P|\mathcal{Q}, pmtn|L_{max}$. Others are well known results, reviewed in Section 2.1 and Section 2.2 of this thesis. Three complexity results in Table 3.2 are original to this thesis. All results marked "NP-hard" are in fact strongly NP-hard.

The fourth column from the left gives an estimate of the number A_n of partial orders of the given class $\mathcal{Q}$ on n unlabelled elements, or equivalently, the number of isomorphism classes of partial orders in $\mathcal{Q}$ on n labelled elements. The rightmost column delineates which family (as defined in Table 3.1) of partial order classes $\mathcal{Q}$ belongs to.

Class ($\mathcal{Q}$)	**Complexity** (C_{max})	**Complexity** (L_{max})	**Asymptotic Number**	**Family** [17]
Weak Order	P*	P†	2^{n-1}‡	I
In-Forest	P	P [71]	$1.300 \times n^{-\frac{3}{2}} \times 2.956^n$ [87]	I
Out-Forest	P [71]	NP-hard [22]	$1.300 \times n^{-\frac{3}{2}} \times 2.956^n$ [87]	I
Width At Most Two	P	P [71]	$0.202 \times n^{-\frac{3}{2}} \times 4^n$ [16]	I
Series-Parallel	Unknown	NP-hard§	$0.229 \times n^{-\frac{3}{2}} \times 4.621^n$ [107]	I
Width At Most Three	P	P [71]	¶	I
Interval Order	P [35]	Unknown	**	II
Bipartite	NP-hard††	NP-hard	$\frac{\psi_{(1-(n \bmod 2))}}{n!\sqrt{n}} 2^{\frac{n^2}{4}+n}$ [18] [92]	IV

Table 3.2: Computational complexity for $P|\mathcal{Q}, pmtn|\gamma$ with $\gamma \in \{C_{max}, L_{max}\}$.

*Follows from the fact that the problem $P|$*weak order*,$pmtn|L_{max}$ is in P, and the fact that $P|\mathcal{Q}, pmtn|C_{max}$ is a special case of $P|\mathcal{Q}, pmtn|L_{max}$ for any class $\mathcal{Q}$ of partial orders. Citations of this kind are omitted from the remainder of this table.

†Follows from Corollary 6.19 and Theorem 6.23 later in this thesis.

‡Proved in Theorem 3.1.

§Follows from the fact that every out-forest is also series-parallel.

¶To the author's knowledge there is no good estimate of this asymptotic growth rate. In [16] it is established that the number B_n of labelled posets on n elements with width at most three obeys $C_1 n! 16^n n^{-C_2} \leq B_n \leq C_3 n! 16^n n^{-C_4}$ for some positive constants C_1, C_2, C_3 and C_4. This makes it easily the fastest growing class in family I among the eight classes examined in this chapter.

**To the author's knowledge the best estimate of this growth rate is given in [41]. Taking A_n as the number of unlabelled interval orders on n elements, it is proven that $(n!)^{1-\varepsilon} < A_n < (n!)^{1+\varepsilon}$ for arbitrary $\varepsilon > 0$ and adequately large n.

††Proved in Theorem 3.2.

Intuitively one might expect that classes with slower asymptotic growth would be more amenable to polynomial solution and this indeed appears to be the case for the C_{max} objective function, but surprisingly this is not the case for L_{max}. Strangely, the class of out-forests, one of the slower growing classes, is NP-hard while the class of partial orders with width at most three is polynomially solvable. At this point, lacking any unifying theory to explain these results, the reason for this may only be speculated upon.

The author speculates that the problematic structure is D from Figure 3.1, i.e. the possibility that one task might be completed and "reveal" multiple successors that must be processed. Out-forests are composed of this structure, and the class of out-forests is a subclass of series-parallel posets – these are both NP-hard. Bipartite orders are numerous enough that they have many instances of D as a subgraph. A further examination of the reasons for these results is attempted in Section 7.2. Unfortunately, due to the limited number of results of this type, broader inferences cannot be made.

3.3 Proofs

Although some of the results in Table 3.2 that originate in this thesis follow from proofs in later chapters, others were derived solely for the purpose of this Chapter, and so are presented here. A precise formula for the number of isomorphically distinct weak orders follows.

Theorem 3.1 *The number A_n of isomorphically distinct weak orders on n elements is 2^{n-1} for all $n \geq 1$.*

Proof. Consider a poset P with at least two maximal elements j and g, i.e. there exists no q for which either $j \, P \, q$ or $g \, P \, q$. Further suppose that there is some q for which $q \, P \, j$ but $q \not P \, g$. This implies that j, g and q form an induced subgraph isomorphic to E from Figure 3.1, and consequently P cannot be a weak order.

The above implies that, given any weak order Q on n elements, all maximal elements have the same predecessors, and hence all posets produced by the removal of a single maximal element from Q are isomorphic. The definition of a weak order in terms of an excluded subgraph implies that all such resultant posets are also weak orders.

Given a weak order P, consider all of the ways that an element x may be added to P to produce P' such that

- P' is a weak order and

- x is a maximal element of P'.

Suppose there exists y such that $y\,P'\,x$ and y is a maximal element of P. If z is another maximal element of P for which $z \not P' x$ then x, y and z form an induced subgraph isomorphic to E from Figure 3.1, and consequently P' cannot be a weak order. Therefore, if there is any maximal element y of P for which $y\,P'\,x$ then for each maximal element z of P it follows that $z\,P'\,x$; this implies that x is the unique maximal element of P'. In this case there can be no induced subgraph of P' isomorphic to E from Figure 3.1.

On the other hand, suppose there does not exist any y such that $y\,P'\,x$ and y is a maximal element of P. If there exists any non-maximal element z of P such that $z \not P' x$ then (because z is non-maximal) there must be some maximal element y of P for which $z\,P\,y$. In this case x, y and z form an induced subgraph isomorphic to E from Figure 3.1, so P' cannot be a weak order. Hence, each non-maximal element z of P has $z\,P'\,x$. As stated before, all maximal elements of P' have the same set of predecessors, and so all weak orders produced in this way are members of the same isomorphism class. All of these weak orders have at least two maximal elements.

The two ways of producing a new weak order as described above result is isomorphically distinct weak orders, which can be seen from the fact that they have different numbers of maximal elements. Thus, each weak order on n elements maps to two weak orders on $n+1$ elements, so $A_{n+1} = 2A_n$ for all $n \geq 1$. This theorem follows from the fact that $A_1 = 1$. ■

The reasoning above implies a very simple method for randomly generating isomorphically distinct weak orders over a given number of elements with equal probability.

In [118] it is proven that $P|prec, p_j = 1|C_{max}$ is NP-hard, even in the case where the precedence constraints are in the form of a bipartite order. Below, the result is adapted to the $P|prec, pmtn|C_{max}$ problem. The fundamental idea is to reduce the considered scheduling problem to the clique problem, a well known graph theory problem proven in [64] to be NP-hard. A clique is defined to be a complete subgraph of an undirected graph, and the goal of the clique problem (when expressed as a decision problem) is to determine if a given graph G contains a clique of given number k of elements. Each different (G, k) defines a distinct instance of the clique problem.

Theorem 3.2 *The $P|bipartite, pmtn|C_{max}$ problem is strongly NP-hard.*

Proof. The NP-hardness of the $P|bipartite, pmtn|C_{max}$ problem will be proven by reduction from the clique problem. The fact that it is strongly NP-hard follows from the fact that the specific reduction used has $m < 3n$ and all $p_j \leq 2$, meaning all numerical values are bounded above by a polynomial in the length of the input (see [47] for more details).

If $k \leq 3$ the problem may be solved in polynomial time by brute force: all combinations of k elements are tested to see if they are a clique. From this point on, it will be assumed that $k \geq 4$.

Let V be the set of vertices and E be the set of edges of G. Naturally if

$$|V| < k \qquad \text{or} \qquad |E| < \binom{k}{2}$$

no clique of k elements exists, and if $|V| = k$ the instance of the clique problem is trivial. Assume that none of these are the case.

Now an instance φ of $P|bipartite, pmtn|C_{max}$ will be considered. It will be defined so that an optimal schedule ω for φ will allow the solution to the clique problem to be determined in polynomial time. By the assumption at the beginning of this proof, an optimal schedule can be obtained in polynomial time; hence, the assumption that the $P|bipartite, pmtn|C_{max}$ problem is polynomially solvable leads to the conclusion that the clique problem is polynomially solvable, and consequently the $P|bipartite, pmtn|C_{max}$ problem is NP-hard.

Let the instance φ of $P|bipartite, pmtn|C_{max}$ be defined as follows. Let

$$m = \max\left\{\binom{k}{2} + |V| - k, |E| - \binom{k}{2}\right\} + 1.$$

and note that, as a consequence of the assumption that $|V| > k \geq 4$,

$$m \geq \binom{k}{2} + |V| - k + 1 \geq \frac{k(k-1)}{2} + 2 \geq \frac{4(k-1)}{2} + 2 = 2k \geq k + 4. \qquad (3.1)$$

Let there be a set T_1 of tasks where each task corresponds to a member of V. Let there be another set T_2 of tasks where each task corresponds to a member of E. For $j \in T_1$ and $g \in T_2$, $j \to g$ if and only if the vertex corresponding to j is an endpoint of the edge corresponding to g; otherwise they are independent (i.e. neither $j \to g$ nor $g \to j$).

Let there be a set S_1 of $m - k$ tasks, all of which are independent of the tasks in T_1 and T_2. Note that (3.1) demonstrates the validity of this definition by showing

that $|S_1| \geq 0$. Let there be a set S_2 of

$$m - \binom{k}{2} - |V| + k - 1$$

tasks, all of which are independent of the tasks in T_1 and T_2 and preceded by all tasks in S_1. Let there be a set S_3 of

$$m - |E| + \binom{k}{2} - 1$$

tasks, all of which are preceded by all the tasks in S_1 and T_1, and all of which are independent of the tasks in T_2 and S_2. The fact that $|S_2| \geq 0$ and $|S_3| \geq 0$ follows directly from the definition of m, although note that at least one of these two sets will be empty. Finally, let there be a set S_4 containing a single task preceded by all tasks in S_1 and independent of all other tasks. All tasks $j \notin S_4$ have $p_j = 1$, while $p_g = 2$ for the task $g \in S_4$.

The tasks in $T_1 \cup T_2$ are the "graph tasks" that are the subjects of the optimisation: their position in the optimal schedule ω for φ will determine whether or not there is a clique of the desired size. The tasks in $S_1 \cup S_2 \cup S_3 \cup S_4$ are the "structure tasks" that exist to force the graph tasks to be scheduled in a certain way.

The problem instance φ described above can be derived from (G, k) in polynomial time. Observe that the precedence constraints of φ take the form of a bipartite order, where all tasks that have successors are in $T_1 \cup S_1$ and all tasks that have predecessors are in $T_2 \cup S_2 \cup S_3 \cup S_4$.

As a consequence of the fact that

$$\sum_{j \in N} p_j = |T_1| + |T_2| + |S_1| + |S_2| + |S_3| + 2|S_4|$$

$$= |V| + |E| + m - k + m - \binom{k}{2} - |V| + k - 1 + m - |E| + \binom{k}{2} - 1 + 2 = 3m$$

it can be seen that $C_{max}(\sigma) \geq 3$ for all schedules σ for the problem instance φ. In order for this to be an equality there can be no idle machines during the interval $[0, 3]$.

If $C_{max}(\sigma) = 3$, $C_j(\sigma) = 1$ for all $j \in S_1$ so that $C_g(\sigma) = 3$ for $g \in S_4$. Now consider what tasks are available for processing at time 1. By the assumption that $|V| > k$,

$$|T_1| \cup |S_1| = |V| + m - k > m,$$

and hence $C_j(\sigma) > 1$ for at least one task $j \in T_1$. Consequently, no task in S_3 is available for processing at time 1. If x tasks $j \in T_1$ have $C_j(\sigma) = 1$ then at most $\binom{x}{2}$ tasks in T_2 are made available, and strictly less than this if the vertices corresponding to the x tasks completed do not form a clique. Since at most k tasks $j \in T_1$ have $C_j(\sigma) = 1$ there are at most $\binom{k}{2}$ tasks in T_2 are available at time 1.

Supposing there are x tasks in T_1 completed at time 1, the number of tasks available at time 1 is at most

$$|V| - x + \binom{x}{2} + m - \binom{k}{2} - |V| + k = m + \frac{x^2 - 3x}{2} - \binom{k^2 - 3k}{2}. \qquad (3.2)$$

As a consequence of the assumption that $k \geq 4$, (3.2) means that there are m available tasks only when $x = k$, and fewer otherwise. Thus the only way for there not to be any idle machines during the interval $[0, 3]$, and consequently the only way in which $C_{max}(\sigma) = 3$, is if there exists a clique of k nodes in G and the corresponding tasks in T_1 are completed at time 1.

If there exists a clique of k nodes in G, $C_{max}(\sigma) = 3$ may be ensured in the following way:

- all tasks in S_1 and k tasks in T_1 corresponding to a clique of k nodes in G are processed to completion at time 1, taking up $(m - k) + k = m$ machines;
- all tasks in S_2, the remaining $|V| - k$ tasks in T_1 and the $\binom{k}{2}$ tasks in T_2 corresponding to the edges of the clique of k nodes in G are processed to completion at time 2, while the task in S_4 is processed for one unit, taking up

  $$\left(m - \binom{k}{2} - |V| + k - 1\right) + (|V| - k) + \binom{k}{2} + 1 = m$$

  machines; and
- all tasks in S_3 and S_4 and the remaining $|E| - \binom{k}{2}$ in T_2 are processed to completion at time 3, taking up

  $$\left(m - |E| + \binom{k}{2} - 1\right) + 1 + \left(|E| - \binom{k}{2}\right) = m$$

  machines.

This means that construction of an optimal schedule ω for the instance φ of the scheduling problem $P|bipartite, pmtn|C_{max}$ will provide a solution to the clique problem: if $C_{max}(\omega) = 3$ then there exists a clique of size k corresponding to the tasks $j \in T_1$ for which $C_j(\omega) = 1$, if $C_{max}(\omega) > 3$ there is no such clique. The

fact that this is a polynomial reduction demonstrates that the $P|bipartite, pmtn|C_{max}$ problem is NP-hard. The fact that this reduction has $m < 3n$ and all $p_j \leq 2$ demonstrates that the problem is strongly NP-hard. ■

Notice that in the preceding proof only one task j has $p_j \neq 1$. It is a curious fact that the problem $P|bipartite, p_j = 1 pmtn|C_{max}$ can be trivially solved in polynomial time using McNaughton's Algorithm. Thus, even a single task with processing time not equal to one means the difference between P and NP-hard.

Chapter 4

Approximation Algorithms

Since the general problem $P|prec, pmtn, u_j|L_{max}$ is NP-hard, a great deal of effort has been expended to create and analyse approximation algorithms. This chapter presents the approximation algorithms considered in this thesis, and provides some analysis of the schedules they construct. The text of this section is derived, with substantial modification, from material presented at the 9th Workshop on Models and Algorithms for Planning and Scheduling Problems [124], in collaboration with Yakov Zinder.

There is a relationship between the well known problem $P|prec, pmtn|L_{max}$ (that is, without u_j) and the other classical scheduling problem denoted by $P|prec, p_j = 1|L_{max}$. The latter differs from the $P|prec, pmtn|L_{max}$ problem by two assumptions:

1. no preemptions (interruptions of task processing) are allowed, and

2. the machine time required for the task completion is the same for all tasks and is equal to one time unit.

The problem $P|prec, p_j = 1|L_{max}$ frequently serves as a testing ground for new scheduling ideas, and many known polynomial-time approximation algorithms for $P|prec, pmtn|L_{max}$ are modifications of the algorithms for $P|prec, p_j = 1|L_{max}$. Thus, the polynomial-time algorithms for $P|prec, pmtn|L_{max}$, presented in [105], [55] and [120], are modifications of the algorithms in, respectively, [21], [45] and [119] for the $P|prec, p_j = 1|L_{max}$ problem.

More specifically, algorithms developed for $P|prec, p_j = 1|L_{max}$, including [21], [45] and [119], can be viewed as two-phase procedures, where the first phase assigns to each task j a number, say μ_j, – the task's priority (urgency), and then the second phase constructs a schedule using the rule that if two tasks compete for a machine, preference is given to the task with a higher priority. Without loss of generality, in what follows it will be assumed that the larger μ_j indicates a higher priority. Hence,

the second phase of an algorithm for $P|prec, p_j = 1|L_{max}$ can be viewed as a greedy algorithm choosing a task with the largest μ among all available for scheduling.

A modification of such two-phase heuristic procedure is achieved by

- borrowing the method for computing μ's (with all necessary adjustments accounting for preemptions and arbitrary processing times),
- introducing the notion of priorities (this new definition of priority must take into account preemptions and differences in processing times), and
- constructing a schedule using an algorithm scheduling tasks according to their priorities.

The method of transforming a two-phase heuristic procedure for $P|prec, p_j = 1|L_{max}$ into an algorithm for the $P|prec, pmtn|L_{max}$ problem, outlined above, remains valid for the $P|prec, pmtn, u_j|L_{max}$ problem as well. Now, however, this modification should address not only the fact that different tasks may have different processing times and the possibility of preemptions, but also the possibility of processing a task concurrently on several machines.

4.1 Families of Algorithms

It is reasonable to design algorithms for the $P|prec, pmtn, u_j|L_{max}$ problem as two-phase heuristic procedures, where the first phase specifies μ_j for each task j and the second phase constructs a schedule by scheduling tasks according to their priorities – as stated earlier, unlike the $P|prec, p_j = 1|L_{max}$ problem, a task's μ and its priority are not synonymous. The algorithm used in the second phase will be denoted Algorithm P, to be described later, and it is a modification for the problem $P|prec, pmtn, u_j|L_{max}$ of the algorithm presented in [105]. Thus, the approximation algorithms considered in this thesis for the problem $P|prec, pmtn, u_j|L_{max}$ differ only in their method of calculation of μ's; more precisely let each algorithm $\mathscr{A}$ define some function $\mu^{\mathscr{A}}(j, \varphi)$ such that when given a problem instance φ as defined in (1.1), $\mathscr{A}$ sets $\mu_j = \mu^{\mathscr{A}}(j, \varphi)$ for all $j \in N$. Let $\mathfrak{P}$ denote the family of algorithms that function as described above, i.e. $\mathfrak{P}$ contains all algorithms $\mathscr{A}$ that define $\mu_j = \mu^{\mathscr{A}}(j, \varphi)$ and then call Algorithm P to construct the schedule using these μ's. It is convenient to subdivide family $\mathfrak{P}$ on the basis of how the algorithms calculate the μ's.

Let two problem instances be termed isomorphic if one may be converted to the other by renumbering the tasks. Note that such a renumbering of the tasks in N produces a different tuple as specified in (1.1). In general, an algorithm might generate completely different μ's for two isomorphic problem instances. No algorithms

in family $\mathfrak{P}$ that are known to this author use this functionality – on the surface it seems to be of marginal utility for a task's μ to be determined randomly or to be dependent on an arbitrary index.* Consequently, an algorithm $\mathscr{A}$ will be called "stable" if renumbering of tasks will not change the μ's generated, and a function $\mu^{\mathscr{A}}(j, \varphi)$ will be defined as stable in an analogous fashion. Let $\mathfrak{S}$ be the family of all scheduling algorithms $\mathscr{A}$ such that

1. there is a stable function $\mu^{\mathscr{A}}$ that accepts as its first argument a task index and as its second argument a problem instance,

2. algorithm $\mathscr{A}$ sets μ_j equal to $\mu^{\mathscr{A}}(j, \varphi)$ for all $j \in N$, then

3. algorithm $\mathscr{A}$ calls Algorithm P to construct a schedule based on these μ's.

Observe that this definition implies that $\mathfrak{S} \subseteq \mathfrak{P}$.

The algorithms in [21], [45] and [119] for the much studied $P|prec, p_j = 1|L_{max}$ problem (adapted in this thesis to the problem $P|prec, pmtn, u_j|L_{max}$) are based on the following two observations. First, if several tasks are available for processing at some point in time, then in order to minimise L_{max}, it is reasonable to give preference, i.e. to schedule first, a task j for which due date d_j is the smallest among all due dates of the tasks available for processing at the considered point in time. Hence, for each task j it is reasonable to set $\mu_j = -d_j$.

Second, as defined in Chapter 1, for any task j denote the set of its successors by K_j, i.e. K_j is the set of all g such that $j \to g$. If K_j contains an urgent task, that is there exists $g \in K_j$ with a very large μ_g (or equivalently, a very small d_g), this may require j to be completed earlier than might be suggested by d_j and therefore μ_j should be larger than $-d_j$.

Based on the above two observations, the algorithms in [21], [45] and [119] first recalculate the original priority $\mu_j = -d_j$ of each task j. This recalculation may affect only the priority of tasks j with $K_j \neq \emptyset$. The original priority of each j with $K_j = \emptyset$ remain unchanged. The recalculation for each j is based on d_j and the following data:

$$\varphi_j = (m, \to_j, \{(g, d_g) : \ g \in K_j\}), \tag{4.1}$$

where $\to_j$ is the partially ordered set of the successors of j, i.e. the set K_j with the precedence constraints that exist between the tasks in K_j. So, the algorithms in [21], [45] and [119] recalculate μ_j for each task j (or equivalently, recalculate the due date d_j of each task j) using a certain function $\mu(d_j, \varphi_j)$ and then schedule the tasks, giving preference to tasks with higher priorities among all tasks available for

*It is proven in Theorem 5.32 that not being able to do this is actually a significant limitation.

processing. The algorithms in [21], [45] and [119] differ from each other only in the function $\mu(d_j, \varphi_j)$.

For the $P|prec, pmtn, u_j|L_{max}$ problem, the remaining processing time of any task j at time point t in schedule σ will be denoted by $p_j(t, \sigma)$. If, starting at t, this task is processed until its completion by the largest possible number of machines, i.e. u_j, then the task's lateness will be

$$t + \frac{p_j(t, \sigma)}{u_j} - d_j.$$

Therefore, if several tasks are available for processing in schedule σ at point in time t, in order to minimise L_{max}, it is reasonable to give preference, i.e. to allocate machine time first, to a task j which has the largest value of

$$\frac{p_j(t, \sigma)}{u_j} + \mu_j, \tag{4.2}$$

where $\mu_j = -d_j$, among all the tasks competing for machine time at t. This will hereafter be known as the priority of a task, and will be denoted $\zeta_j(t, \sigma)$. Let the priority of a task j at time zero be denoted ζ_j.

Moreover, as discussed above, in (4.2) it is reasonable to replace each $\mu_j = -d_j$ with its recalculated value that takes into account the urgency of the tasks' successors. In other words, the resultant schedule η should be constructed using (4.2) as the priority of task j in schedule η at any point in time t, where

$$\mu_j = \mu(d_j, \varphi_j); \tag{4.3}$$

here, μ is a function which specifies how the successors of a task are taken into account in computing the task's priority and

$$\varphi_j = (m, \rightarrow_j, \{(g, p_g) : \ g \in K_j\}, \{(g, d_g) : \ g \in K_j\}, \{(g, u_g) : \ g \in K_j\}) . \tag{4.4}$$

Observe that, in contrast to (4.1), φ_j incorporates for each successor of j not only its due date but also its processing time and the maximum permissible number of machines which can process this successor simultaneously. Further observe that if $K_j \neq \emptyset$, φ_j is another instance of the general problem $P|prec, pmtn, u_j|L_{max}$, as defined in (1.1). This problem instance is produced from the original problem instance simply by discarding all tasks which are not in K_j.

The above discussion leads to the following definition of a family $\mathfrak{F}$ of successor-based scheduling algorithms: $\mathfrak{F}$ consists of all algorithms $\mathscr{A}$ such that

1. there is a function $\mu^{\mathscr{A}}$ that accepts as its first argument a real number and as its second argument a problem instance,

2. algorithm $\mathscr{A}$ sets μ_j equal to $\mu^{\mathscr{A}}(d_j, \varphi_j)$ for all $j \in N$, then

3. algorithm $\mathscr{A}$ calls Algorithm P to construct a schedule based on these μ's.

In contrast to the definition of the family $\mathfrak{P}$, algorithms in $\mathfrak{F}$ only have access to a task's due date and successors when defining its μ; hence $\mathfrak{F} \subset \mathfrak{P}$. Note that not all successor-based algorithms are stable, and not all stable algorithms are successor-based.

Each algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ differs only by which function $\mu^{\mathscr{A}}$ it employs to determine the μ's that will be used in Algorithm P. This is a very broad class of algorithms, encompassing many existing approximation algorithms and infinitely many as-yet undiscovered algorithms. All approximation algorithms considered in this section are members of the family $\mathfrak{F} \cap \mathfrak{S}$

4.2 Scheduling Using Priorities

This section describes Algorithm P and investigates the properties of schedules constructed by algorithms from family $\mathfrak{P}$. To aid the analysis of these algorithms, an alternate criterion

$$G(\sigma) = \max_{j \in N}\{C_j(\sigma) + \mu_j\},$$

is defined. This can be considered to be an alternative to L_{max} that simplifies the analysis of schedules produced by Algorithm P.

Any numbers μ_j such that $\mu_j \geq -d_j$ for all $j \in N$, and $\mu_j = -d_j$ for each j which satisfies $K_j = \emptyset$, will be referred to as modified due dates. Modified due dates will be called preserving if, for any schedule σ, $G(\sigma) = L_{max}(\sigma)$.* Hence, in the case of preserving μ's, the problems $P|prec, pmtn|L_{max}$ and $P|prec, pmtn|G$ are equivalent. It is important to emphasise that family $\mathfrak{P}$ includes algorithms for which μ's are not modified due dates.

For any j such that $K_j \neq \emptyset$, let ω^j be an optimal schedule for φ_j. The following lemma, original to this thesis, provides a general condition which guarantees that μ's are preserving. In previous works where results regarding preserving μ's are introduced, it is only proven that the specific μ's produced by the algorithm under consideration are preserving: no general conditions are presented.

*It is also important to note that, in [119] and most other papers where they are used, the μ's are defined so that $G(\sigma) = L_{max}(\sigma) + d_{max}$, where d_{max} is defined as in (2.2). This thesis does not define tasks' μ's in this way because doing so would conflict with the definition of family $\mathfrak{F}$.

Lemma 4.1 *If modified due dates satisfy the condition that, for each $j \in N$ such that $K_j \neq \emptyset$,*

$$\mu_j \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\},$$

then these modified due dates are preserving.

Proof. Let σ be an arbitrary schedule and let j be a task satisfying

$$C_j(\sigma) + \mu_j = \max_{g \in N}\{C_g(\sigma) + \mu_g\}.$$

If $\mu_j = -d_j$, then taking into account that $-d_g \leq \mu_g$ for all tasks,

$$\max_{g \in N}\{C_g(\sigma) - d_g\} \leq \max_{g \in N}\{C_g(\sigma) + \mu_g\} = C_j(\sigma) + \mu_j$$

$$= C_j(\sigma) - d_j \leq \max_{g \in N}\{C_g(\sigma) - d_g\}.$$

Hence, the equality $\mu_j = -d_j$ implies $L_{max}(\sigma) = G(\sigma)$.

Suppose that $\mu_j > -d_j$, then

$$\max_{g \in N}\{C_g(\sigma) - d_g\} \leq \max_{g \in N}\{C_g(\sigma) + \mu_g\} = C_j(\sigma) + \mu_j$$

$$\leq C_j(\sigma) + L_{max}(\omega^j) = C_j(\sigma) + \max_{g \in K_j}\{C_g(\omega^j) - d_g\}$$

$$\leq C_j(\sigma) + \max_{g \in K_j}\{[C_g(\sigma) - C_j(\sigma)] - d_g\}$$

$$= \max_{g \in K_j}\{C_g(\sigma) - d_g\} \leq \max_{g \in N}\{C_g(\sigma) - d_g\}$$

which again gives $L_{max}(\sigma) = G(\sigma)$. ■

Since μ's together with the remaining processing times specify tasks' priorities and because the idea is to schedule tasks according to their priorities which change with the processing of tasks, the scheduling algorithm (referred to as Algorithm P) schedules tasks in several stages. Each stage corresponds to a point in time that will be referred to as a point of allocation (or allocation point). The terminology indicates that at each point of allocation except the last, each task which is available for processing at this point in time is assigned the machine time which this task will receive between the current and the next point of allocation. Of course, some low-priority tasks may be allocated zero processing time between these two successive points of allocation. Algorithm P is based on three simple principles:

(a) for any two tasks available at a point of allocation, if the priority of one task

is greater than or equal to the priority of the second task, then at the next point of allocation, the priority of the first cannot be less than the priority of the second;

(b) a machine can be idle between two consecutive allocation points only if the sum of u_j of all tasks available for processing is less that m and each such task j is continuously processed on u_j machines in the time interval between these points; and

(c) a task can receive some machine time between two consecutive points of allocation only if each task j which is available for processing at the first of these points and which has at this point a higher priority, is continuously processed on u_j machines between these two points.

At each point of allocation Algorithm P specifies

- the tasks that should be processed continuously between this and the next point of allocation,
- the tasks which will not be processed at all between these consecutive points of allocation, and
- the remaining available tasks which are allocated an intermediate quantity of machine time.

Algorithm P computes the next point of allocation as the latest point in time until which this pattern of machine time allocation does not violate one of the above principles (a), (b) and (c). In other words, points of allocation are points in time where the pattern of tasks' processing is changed in order to satisfy these principles. Once the amount of processing time for each task j has been determined, Algorithm P will call the Modified McNaughton Algorithm to schedule this processing.

4.2.1 The Modified McNaughton Algorithm

The goal of the Modified McNaughton Algorithm is to construct an optimal schedule for the problem $P|pmtn, u_j|C_{max}$, that is the problem of scheduling tasks with preemptions, with the goal of minimising the makespan. The original version of this algorithm, described in [81], dealt only with the problem $P|pmtn|C_{max}$ but the modification considered here is well known, e.g. Chapter 25 of [78]. Although it will be used throughout this thesis to schedule fractions of tasks on arbitrary intervals, for the sake of simplicity it will be described in its original form, where whole tasks are scheduled on an interval beginning at time 0.

First, Δ is selected as the smallest value such that

$$\sum_{j \in N} p_j \leq m\Delta \qquad \text{and} \qquad \max_{j \in N} \frac{p_j}{u_j} \leq \Delta. \tag{4.5}$$

It is easy to see that (4.5) are necessary conditions for the feasibility of this allocation. The Modified McNaughton Algorithm shows that these two conditions are also sufficient.

The Modified McNaughton Algorithm schedules tasks in the time interval $[0, \Delta]$ machine by machine and task by task. The order in which the machines and tasks are considered is not important. Each task is scheduled on a machine, either until it is completed or until time Δ. In the latter case the processing of the task "wraps around" and continues on subsequent machines.

The Modified McNaughton Algorithm

1. Set $\tau := 0$.
2. If all tasks in N have been considered, terminate the algorithm. Otherwise, select any $j \in N$ that has not already been considered.
3. If $\tau + p_j < \Delta$, process j continuously during the time interval $[\tau, \tau + p_j]$, set $\tau := \tau + p_j$ and go to Step 2. Otherwise, process j continuously during the time interval $[\tau, \Delta]$ and continue to Step 4.
4. Compute
$$q_j = \left\lfloor \frac{\tau + p_j - \Delta}{\Delta} \right\rfloor \tag{4.6}$$
and process j continuously during the time interval $[0, \Delta]$ on q_j machines.
5. Process j continuously during the time interval $[0, p_j - (\Delta - \tau) - \Delta q_j]$, set
$$\tau := p_j - (\Delta - \tau) - \Delta q_j$$
and go to Step 2.

This algorithm is useful as a subroutine in algorithms for more complex problems: if a schedule can be broken up into intervals of time, and each interval has assigned to it a given quantity of processing for each task that is available during the interval, the Modified McNaughton Algorithm can be used to construct a schedule tasks during each interval.

4.2.2 Algorithm P

The schedule constructed by Algorithm P will be denoted η. In the description of Algorithm P below, $0 = t_0 < t_1 < \cdots < t_s$ are the points of allocation and A_i denotes the set of all tasks available for processing at t_i, that is

$$A_i = \{j : j \in N \wedge p_j(t_i, \eta) > 0 \wedge \forall g \in J_j : p_g(t_i, \eta) = 0\}.$$

For the purpose of scheduling using the Modified McNaughton Algorithm, the length of the time interval between two consecutive points of allocation t_i and t_{i+1} is Δ, i.e. $\Delta = t_{i+1} - t_i$.

According to the three principles mentioned earlier, when determining how much of each task will be processed in the interval between two allocation points, Algorithm P partitions each A_i into three sets N_1, N_2, and N_3. Define $U_a = \sum_{j \in N_a} u_j$, i.e. the maximum number of machines that may process all tasks in the corresponding set simultaneously. The first set, N_1, is the largest possible subset of A_i such that $U_1 \leq m$ and the priority of each task in N_1 at t_i is strictly greater than the priorities at t_i of all remaining tasks in A_i. If $U_1 = m$ or $A_i = N_1$, set $N_2 = \emptyset$. Otherwise, N_2 is comprised of tasks which have the same priority at t_i. This common priority is strictly less than the priority at t_i of any task in N_1 but is strictly greater than that of any task in A_i which is not in N_1 and N_2. The tasks in A_i which are not in N_1 or N_2 form the set N_3. Any of the sets N_1, N_2 or N_3 can be empty.

Let the quantity of processing a task j receives during the time interval currently under consideration be denoted v_j.

If N_1 is not empty, in order to satisfy principles (b) and (c), each task $j \in N_1$ should be continuously processed on u_j machines between t_i and t_{i+1}, that is,

$$v_j = u_j \Delta \qquad \text{for each } j \in N_1. \tag{4.7}$$

If N_2 is not empty, according to the principle (a), all tasks in N_2 should receive the same amount of processing time. The total processing time allocated to N_2 can not exceed $\Delta(m - U_1)$, because only $m\Delta$ units of processing time are available between t_i and t_{i+1} and $U_1\Delta$ units have been allocated to N_1. On the other hand, by the definition of N_1 and N_2, if $N_2 \neq \emptyset$ then

$$U_2 > m - U_1.$$

Therefore, the processing time allocated to each task in N_2 must be less than Δ. By virtue of the principle (b), this implies that all machines should be busy between t_i

and t_{i+1}. In other words, in the case of $N_2 \neq \emptyset$, the total processing time allocated to N_2 must be $\Delta(m - U_1) > 0$ and

$$v_j = u_j \frac{m - U_1}{U_2} \Delta \qquad \text{for each } j \in N_2. \tag{4.8}$$

The tasks in N_3 do not receive any processing time between t_i and t_{i+1}, i.e.

$$v_j = 0 \qquad \text{for each } j \in N_3. \tag{4.9}$$

Observe that the above allocation satisfies (4.5), and therefore, using the allocated processing times, the Modified McNaughton Algorithm is able to schedule the tasks which constitute $N_1 \cup N_2$ between t_i and t_{i+1}.

The next point of allocation t_{i+1} is defined by Δ. Algorithm P chooses for Δ the largest value which guarantees the principle (a) and the condition $v_j \leq p_j(t_i, \eta)$ for all $j \in A_i$. The latter imposes the following constraints on Δ:

$$\min_{j \in N_1} \frac{p_j(t_i, \eta)}{u_j} \geq \Delta, \tag{4.10}$$

$$\min_{j \in N_2} \frac{p_j(t_i, \eta)}{u_j} \geq \frac{m - U_1}{U_2} \Delta. \tag{4.11}$$

The next three constraints are imposed by the principle (a). Each of these constraints is used by Algorithm P in calculating Δ only if the sets involved in the constraint are nonempty. According to the principle (a), the minimal priority at t_{i+1} among all tasks in N_1 should be not less than the priority at t_{i+1} of tasks in N_2 (all tasks in N_2 have at t_{i+1} the same priority) and should not be less than the maximum priority at t_{i+1} among all tasks in N_3 (since tasks in N_3 are not processed between t_i and t_{i+1}, their priorities do not change between t_i and t_{i+1}). Denoting the priority of task j at time t in schedule σ by $\zeta_j(t_i, \eta)$, the preceding requirements give

$$\min_{j \in N_1} \zeta_j(t_i, \eta) - \Delta \geq \zeta_g(t_i, \eta) - \frac{m - U_1}{U_2} \Delta \tag{4.12}$$

for $g \in N_2$, and

$$\min_{j \in N_1} \zeta_j(t_i, \eta) - \Delta \geq \max_{j \in N_3} \zeta_j(t_i, \eta). \tag{4.13}$$

Similarly, the principle (a) requires that

$$\zeta_g(t_i, \eta) - \frac{m - U_1}{U_2} \Delta \geq \max_{j \in N_3} \zeta_j(t_i, \eta) \tag{4.14}$$

for $g \in N_2$, which guarantees that the priority at t_{i+1} of tasks in N_2 (all tasks in

N_2 have the same priority at t_{i+1}) is not less than the maximum priority among all tasks in N_3 at t_{i+1} (since tasks in N_3 are not processed between t_i and t_{i+1}, there priorities do not change between t_i and t_{i+1}). These definitions allow Algorithm P to be defined.

Algorithm P

1. Set $i = 0$ and $t_i = 0$.
2. Partition A_i into the sets N_1, N_2, and N_3.
3. Calculate Δ as the largest value satisfying (4.10), (4.11), (4.12), (4.13), and (4.14).
4. Using (4.7) and (4.8), assign to each $j \in N_1 \cup N_2$ the processing time v_j.
5. Set $t_{i+1} = t_i + \Delta$ and, using the Modified McNaughton Algorithm, schedule the tasks which constitute $N_1 \cup N_2$ during the time interval $[t_i, t_{i+1}]$.
6. If all tasks have been completed, then stop. Otherwise, set $i = i + 1$ and go to Step 2.

As has been described above, in constructing η, Algorithm P generates an increasing sequence of points of allocation $t_0, t_1, \ldots, t_s$. At each t_i but the last, Algorithm P partitions the set A_i – the set of all tasks available for processing at t_i, into three subsets N_1, N_2 and N_3. In order to be able to refer to the subsets of the same type but specified for different points of allocation, the sets N_1, N_2 and N_3 associated with t_i will be denoted N_1^i, N_2^i and N_3^i. Hence, $A_i = N_1^i \cup N_2^i \cup N_3^i$.

Several properties of Algorithm P will now be established, in order to facilitate its analysis. The lemma below establishes the fact that each interval between consecutive points of allocation has non-zero length. Recall that s is the index of the last point of allocation in the considered schedule.

Lemma 4.2 *For each i such that $0 \leq i < s$, $t_{i+1} > t_i$.*

Proof. Recall that $t_{i+1} - t_i = \Delta$ is determined as the smallest non-negative value such that at least one of (4.10), (4.11), (4.12), (4.13) and (4.14) is an equality, and suppose that $\Delta = 0$. If (4.10) or (4.11) is an equality, this implies that some $j \in N_1^i \cup N_2^i$ has $p_j(t_i, \eta) = 0$, contradicting the fact that $(N_1^i \cup N_2^i) \subseteq A_i$. On the other hand, if (4.12), (4.13) or (4.14) is an equality, there exists $j \in N_e^i$ and $g \in N_f^i$, for some $e \in \{1, 2, 3\}$ and $f \in \{1, 2, 3\} \setminus \{e\}$, such that $\zeta_j(t_i, \eta) = \zeta_g(t_i, \eta)$, contradicting

the construction of N_1, N_2 and N_3. Since both cases result in contradictions, $\Delta > 0$ and the lemma is proven. ■

If at least two of N_1^i, N_2^i and N_3^i are non-empty, define

$$\xi_i = \min_{e,f:e<f \wedge N_e^i \neq \emptyset \wedge N_f^i \neq \emptyset} \left(\min_{j \in N_e^i, g \in N_f^i} (\zeta_j(t_i, \eta) - \zeta_g(t_i, \eta)) \right),$$

i.e. ξ_i is the smallest difference in priorities between any two tasks in different sets N_e^i and N_f^i. The lemma below establishes, under certain conditions, an upper bound on the length of an interval between allocation points.

Lemma 4.3 *For each i such that $0 \leq i < s$, if at least two of N_1^i, N_2^i and N_3^i are non-empty, $t_{i+1} - t_i < mn\xi_i$.*

Proof. Select $e \in \{1, 2\}$ and $f \in \{2, 3\}$ for which $e < f$ and such that there exist tasks $j \in N_e^i$ and $g \in N_f^i$ for which

$$\zeta_j(t_i, \eta) - \zeta_g(t_i, \eta) = \xi_i.$$

Three cases are now considered.

- If $j \in N_1^i$ and $g \in N_2^i$ then, from (4.12) and the fact that $U_1 + U_2 > m$ when $N_2^i \neq \emptyset$,
$$t_{i+1} - t_i \leq \frac{U_2}{U_1 + U_2 - m} \xi_i \leq U_2 \xi_i \leq m|N_2^i|\xi_i < mn\xi_i.$$

- If $j \in N_1^i$ and $g \in N_3^i$ then, from (4.13) and the fact that $j \neq g$ implies $n \geq 2$,
$$t_{i+1} - t_i \leq \xi_i < mn\xi_i.$$

- If $j \in N_2^i$ and $g \in N_3^i$ then, from (4.14) and the fact that $U_1 < m$ when $N_2^i \neq \emptyset$,
$$t_{i+1} - t_i \leq \frac{U_2}{m - U_1} \xi_i \leq U_2 \xi_i \leq m|N_2^i|\xi_i < mn\xi_i.$$

Thus the lemma holds in all three cases. ■

The lemma below justifies (4.12), (4.13) and (4.14).

Lemma 4.4 *For each i such that $0 \leq i < s$, any $e \in \{1, 2\}$ for which $N_e^i \neq \emptyset$ and any $f \in \{2, 3\}$ for which $e < f$ and $N_f^i \neq \emptyset$, $\zeta_j(t_{i+1}, \eta) \geq \zeta_g(t_{i+1}, \eta)$ for all $j \in N_e^i$ and $g \in N_f^i$.*

Proof. For all $q \in N_1^i$, (4.7) implies

$$\zeta_q(t_{i+1}, \eta) = \zeta_q(t_i, \eta) - (t_{i+1} - t_i).$$

Likewise, for all $q \in N_2^i$, (4.8) implies

$$\zeta_q(t_{i+1}, \eta) = \zeta_q(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i).$$

Finally, for all $q \in N_2^i$, (4.9) implies

$$\zeta_q(t_{i+1}, \eta) = \zeta_q(t_i, \eta).$$

As in the previous lemma, three cases are considered.

- If $j \in N_1^i$ and $g \in N_2^i$ then (4.12) implies

$$\zeta_j(t_{i+1}, \eta) = \zeta_j(t_i, \eta) - (t_{i+1} - t_i) \geq \min_{q \in N_1^i} \zeta_q(t_i, \eta) - (t_{i+1} - t_i)$$

$$\geq \zeta_g(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i) = \zeta_g(t_{i+1}, \eta).$$

- If $j \in N_1^i$ and $g \in N_3^i$ then (4.13) implies

$$\zeta_j(t_{i+1}, \eta) = \zeta_j(t_i, \eta) - (t_{i+1} - t_i) \geq \min_{q \in N_1^i} \zeta_q(t_{i+1}, \eta) - (t_{i+1} - t_i)$$

$$\geq \max_{q \in N_3^i} \zeta_q(t_i, \eta) \geq \zeta_g(t_i, \eta) = \zeta_g(t_{i+1}, \eta).$$

- Finally, if $j \in N_2^i$ and $g \in N_3^i$ then (4.14) implies

$$\zeta_j(t_{i+1}, \eta) = \zeta_j(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i)$$

$$\geq \max_{q \in N_3^i} \zeta_q(t_i, \eta) \geq \zeta_g(t_i, \eta) = \zeta_g(t_{i+1}, \eta).$$

Thus the lemma holds in all three cases. ■

Corollary 4.5 *For each i such that $0 \leq i < s$ and any $j \in A_i$ and $g \in A_i$, $\zeta_j(t_i, \eta) \geq \zeta_g(t_i, \eta)$ implies $\zeta_j(t_{i+1}, \eta) \geq \zeta_g(t_{i+1}, \eta)$.*

Proof. Consider any $j \in A_i$ and $g \in A_i$ for which $\zeta_j(t_i, \eta) \geq \zeta_g(t_i, \eta)$. If $j \in N_e^i$ and $g \in N_f^i$ for $e \neq f$ then the corollary follows directly from Lemma 4.4. On the

other hand, if $\{j, g\} \subseteq N_e^i$ for some $e \in \{1, 2, 3\}$ then the corollary follows from (4.7), (4.8) and (4.9). ■

The lemma below demonstrates that, if no task is completed in $(t_i, t_{i+1}]$ then two tasks become equal in priority at time t_{i+1}.

Lemma 4.6 *For each i such that $0 \leq i < s$, if no task is completed in $(t_i, t_{i+1}]$, then there exists $e \in \{1, 2\}$ for which $N_e^i \neq \emptyset$, $f \in \{2, 3\}$ for which $e < f$ and $N_f^i \neq \emptyset$, and $\zeta_j(t_{i+1}, \eta) = \zeta_g(t_{i+1}, \eta)$ for some $j \in N_e^i$ and $g \in N_f^i$.*

Proof. Recall that the point of allocation t_{i+1} is determined such that at least one of (4.10), (4.11), (4.12), (4.13), and (4.14) is an equality. These five cases are considered below.

- If (4.10) is an equality then there exists a task $j \in N_1^i$ for which

$$\frac{p_j(t_i, \eta)}{u_j} = \min_{q \in N_1^i} \frac{p_q(t_i, \eta)}{u_q} = t_{i+1} - t_i.$$

  Consequently, by (4.7), $v_j = p_j(t_i, \eta)$ and $C_j(\eta) \in (t_i, t_{i+1}]$.

- If (4.11) is an equality then, by similar reasoning, there exists a task $j \in N_2^i$ for which

$$\frac{p_j(t_i, \eta)}{u_j} = \min_{q \in N_2^i} \frac{p_q(t_i, \eta)}{u_q} = \frac{m - U_1}{U_2}(t_{i+1} - t_i).$$

  Consequently, by (4.8), $v_j = p_j(t_i, \eta)$ and $C_j(\eta) \in (t_i, t_{i+1}]$ as in the previous case.

- If (4.12) is an equality then, for some $j \in N_1^i$ and $g \in N_2^i$,

$$\zeta_j(t_{i+1}, \eta) = \zeta_j(t_i, \eta) - (t_{i+1} - t_i) = \min_{q \in N_1^i} \zeta_q(t_i, \eta) - (t_{i+1} - t_i)$$

$$= \zeta_g(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i) = \zeta_g(t_{i+1}, \eta).$$

- If (4.13) is an equality, then, for some $j \in N_1^i$ and $g \in N_3^i$,

$$\zeta_j(t_{i+1}, \eta) = \zeta_j(t_i, \eta) - (t_{i+1} - t_i) = \min_{q \in N_1^i} \zeta_q(t_i, \eta) - (t_{i+1} - t_i)$$

$$= \max_{q \in N_3^i} \zeta_q(t_i, \eta) = \zeta_g(t_i, \eta) = \zeta_g(t_{i+1}, \eta).$$

- Finally, if (4.14) is an equality then, for some $j \in N_2^i$ and $g \in N_3^i$,

$$\zeta_j(t_{i+1}, \eta) = \zeta_j(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i)$$

$$= \max_{q \in N_3^i} \zeta_q(t_i, \eta) = \zeta_g(t_i, \eta) = \zeta_g(t_{i+1}, \eta).$$

Thus the lemma holds in all five cases. ■

The lemma below demonstrates that, in the absence of task completion, the set N_2 is absorbing, i.e. once tasks enter set N_2 they will not leave.

Lemma 4.7 *For each i such that $0 \leq i < s$, if no task is completed in $(t_i, t_{i+1}]$, $N_1^{i+1} \subseteq N_1^i$, $N_3^{i+1} \subseteq N_3^i$ and $N_2^i \subset N_2^{i+1}$.*

Proof. Suppose $N_2^i \neq \emptyset$, and define $\psi_{i+1} = \zeta_q(t_{i+1}, \eta)$ and $\psi_i = \zeta_q(t_i, \eta)$ for any $q \in N_2^i$. Let $U_a = \sum_{j \in N_a^i} u_j$ for all $a \in \{1, 2, 3\}$. By Lemma 4.4,

$$\sum_{j \in A_i : \zeta_j(t_{i+1}, \eta) \geq \psi_{i+1}} u_j \geq U_1 + U_2 > m \quad \text{and} \quad \sum_{j \in A_i : \zeta_j(t_{i+1}, \eta) > \psi_{i+1}} u_j \leq U_1 < m.$$

Hence, if no task is completed in $(t_i, t_{i+1}]$,

$$N_1^{i+1} = \{j : j \in A_i \wedge \zeta_j(t_{i+1}, \eta) > \psi_{i+1}\} \subseteq \{j : j \in A_i \wedge \zeta_j(t_i, \eta) > \psi_i\} = N_1^i$$

and

$$N_3^{i+1} = \{j : j \in A_i \wedge \zeta_j(t_{i+1}, \eta) < \psi_{i+1}\} \subseteq \{j : j \in A_i \wedge \zeta_j(t_i, \eta) < \psi_i\} = N_3^i.$$

Lemma 4.4 and Lemma 4.6 together demonstrate that

$$N_2^{i+1} = \{j \in A_i : \zeta_j(t_{i+1}, \eta) = \psi_{i+1}\} \supset \{j \in A_i : \zeta_j(t_i, \eta) = \psi_i\} = N_2^i$$

so the lemma is proven for the the case where $N_2^i \neq \emptyset$.

On the other hand, suppose $N_2^i = \emptyset$ and observe that (4.13) must be an equality if no task is completed in $(t_i, t_{i+1}]$. Lemma 4.6 implies that $\zeta_j(t_{i+1}, \eta) = \zeta_g(t_{i+1}, \eta)$ for some $j \in N_1^i$ and $g \in N_3^i$. Define $\psi = \zeta_g(t_{i+1}, \eta)$ and note that, by Lemma 4.4,

$$\sum_{j \in A_i : \zeta_j(t_{i+1}, \eta) \geq \psi} u_j > U_1 = m \quad \text{and} \quad \sum_{j : \zeta_j(t_{i+1}, \eta) > \psi} u_j < U_1 = m.$$

Hence,

$$N_1^{i+1} = \{j : j \in A_i \wedge \zeta_j(t_{i+1}, \eta) > \psi\} \subset N_1^i$$

and

$$N_3^{i+1} = \{j : j \in A_i \wedge \zeta_j(t_{i+1}, \eta) < \psi\} \subset \{j : j \in A_i \wedge \zeta_j(t_i, \eta) < \psi\} = N_3^i.$$

Finally,

$$N_2^{i+1} = \{j : j \in A_i \wedge \zeta_j(t_{i+1}, \eta) = \psi\} \supset \emptyset = N_2^i.$$

■

Corollary 4.8 *The index s of the final point of allocation obeys $s \leq n^2$.*

Proof. Define $i_0 = 0$, and for all $c \geq 1$, if $i_{c-1} < s$, further define i_c to be the smallest $i > i_{c-1}$ such that $A_i \neq A_{i_{c-1}}$, i.e. a task $q \in A_{i_{c-1}}$ has $C_q(\eta) \in (t_{i_c-1}, t_{i_c}]$. Let b be such that $i_b = s$. Observe that $b \leq n$. Further observe that, by Lemma 4.7, $|N_3^i| < |N_3^{i+1}| \leq n$ for all $i_{c-1} \leq i < i_c$ and all $1 \leq c \leq b$. ■

The lemma below demonstrates that the priorities of available tasks do not grow further apart from one point of allocation to the next.

Lemma 4.9 *For each i such that $0 \leq i < s$ and $|A_i| \geq 2$, and for any two tasks $j \in A_i$ and $g \in A_i \setminus \{j\}$,*

$$|\zeta_j(t_i, \eta) - \zeta_g(t_i, \eta)| \geq |\zeta_j(t_{i+1}, \eta) - \zeta_g(t_{i+1}, \eta)|.$$

Proof. Suppose, without loss of generality, that $\zeta_j(t_i, \eta) \geq \zeta_g(t_i, \eta)$. Three cases are considered.

- If $j \in N_1^i$,

$$|\zeta_j(t_{i+1}, \eta) - \zeta_g(t_{i+1}, \eta)| = \zeta_j(t_i, \eta) - (t_{i+1} - t_i) - \zeta_g(t_{i+1}, \eta)$$

$$\leq \zeta_j(t_i, \eta) - (t_{i+1} - t_i) - (\zeta_g(t_i, \eta) - (t_{i+1} - t_i))$$

$$= \zeta_j(t_i, \eta) - \zeta_g(t_i, \eta) = |\zeta_j(t_i, \eta) - \zeta_g(t_i, \eta)|.$$

- If $j \in N_2^i$, then $g \in N_2^i \cup N_3^i$ and

$$|\zeta_j(t_{i+1}, \eta) - \zeta_g(t_{i+1}, \eta)| = \zeta_j(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i) - \zeta_g(t_{i+1}, \eta)$$

$$\leq \zeta_j(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i) - \left(\zeta_g(t_i, \eta) - \frac{m - U_1}{U_2}(t_{i+1} - t_i)\right)$$

$$= \zeta_j(t_i, \eta) - \zeta_g(t_i, \eta) = |\zeta_j(t_i, \eta) - \zeta_g(t_i, \eta)|.$$

- Finally, if $j \in N_3^i$, then $g \in N_3^i$ and

$$|\zeta_j(t_{i+1}, \eta) - \zeta_g(t_{i+1}, \eta)| = |\zeta_j(t_i, \eta) - \zeta_g(t_i, \eta)|.$$

Thus the lemma holds in all three cases. ■

Defining

$$u_{max} = \max_{j \in N} u_j$$

the lemma below provides a bound on the time gap between two consecutive points of allocation.

Lemma 4.10 *For any $0 \leq i < s$ and any $j \in N_1^i \cup N_2^i$,*

$$t_{i+1} - t_i \leq \frac{p_j}{u_j} n u_{max}.$$

Proof. If $j \in N_1^i$, according to (4.7) and (4.10)

$$\frac{p_j}{u_j} \geq \frac{p_j(t_i, \eta)}{u_j} \geq \min_{g \in N_1^i} \frac{p_g(t_i, \eta)}{u_g} \geq t_{i+1} - t_i \geq \frac{1}{n u_{max}}(t_{i+1} - t_i).$$

On the other hand, if $j \in N_2^i$ then $U_1 < m$, $U_2 \leq |N_2^i| u_{max} \leq n u_{max}$ and

$$\frac{p_j}{u_j} \geq \frac{p_j(t_i, \eta)}{u_j} \geq \min_{g \in N_2^i} \frac{p_g(t_i, \eta)}{u_g} \geq \frac{m - U_1}{U_2}(t_{i+1} - t_i) \geq \frac{1}{n u_{max}}(t_{i+1} - t_i).$$

■

4.3 Mu Assignment Algorithms

As stated earlier, there are many ways to assign μ's to tasks. Four well known algorithms are described in this section.

4.3.1 The Brucker-Garey-Johnson Algorithm

The concept of the Brucker-Garey-Johnson Algorithm, *BGJ* for short, originated in [21]. It is based on the simple idea that μ_j should be at least equal to the priority of any task $g \in K_j$, i.e.

$$\mu_j \geq \mu_g + \frac{p_g}{u_g}, \qquad \text{for all pairs } j \to g. \tag{4.15}$$

The Brucker-Garey-Johnson Algorithm

1. For every task j such that $K_j = \emptyset$, set $\mu_j := -d_j$.
2. If all tasks $j \in N$ are assigned μ_j, then execute Algorithm P. Otherwise, select $j \in N$ such that
   2.1. μ_j has not been specified and
   2.2. for each $g \in K_j$, μ_g has already been specified.
3. Set
$$\mu_j := \max\left\{-d_j, \max_{g \in K_j}\left(\frac{p_g}{u_g} + \mu_g\right)\right\} \tag{4.16}$$
   and go to Step 2.

It can be seen that $\mathscr{BGJ} \in \mathfrak{F} \cap \mathfrak{S}$: the value of μ_j is based only on d_j and the properties of the tasks in K_j, and renumbering the tasks does not affect the value of μ_j.

One very useful property of this algorithm (as opposed to the one proposed in [119] for example) is that it produces μ's that are preserving.

Lemma 4.11 *The μ's produced by the Brucker-Garey-Johnson Algorithm are preserving.*

Proof. Step 1 sets $\mu_j = -d_j$ for all j such that $K_j = \emptyset$, and (4.16) implies that $\mu_j \geq -d_j$ for all other j. Thus these μ's fit the definition of modified due dates.

Furthermore, consider any task $j_0 \in N$ such that $K_{j_0} \neq \emptyset$. Suppose that, for some $a \geq 0$, task j_a has already been selected but task j_{a+1} has not. If $\mu_{j_a} > -d_{g_a}$, select $j_{a+1} \in K_{j_a}$ such that

$$\frac{p_{j_{a+1}}}{u_{j_{a+1}}} + \mu_{j_{a+1}} = \max_{g \in K_{j_a}}\left(\frac{p_g}{u_g} + \mu_g\right).$$

Otherwise set $b = a$.

If $b = 0$ then

$$\mu_{j_0} = -d_{j_0} \leq \max\left\{L_{max}\left(\omega^{j_0}\right), -d_{j_0}\right\}.$$

On the other hand, if $b > 0$,

$$\mu_{j_0} = \frac{p_{j_1}}{u_{j_1}} + \mu_{j_1} = \sum_{a=1}^{b} \frac{p_{j_a}}{u_{j_a}} - d_{j_b} \leq C_{j_b}\left(\omega^{j_0}\right) - d_{j_b}$$

$$\leq L_{max}\left(\omega^{j_0}\right) \leq \max\left\{L_{max}\left(\omega^{j_0}\right), -d_j\right\}.$$

Hence, in either case the lemma follows from Lemma 4.1. ■

To illustrate this algorithm, an example has been provided below.

Example 4.12 *When the Brucker-Garey-Johnson Algorithm is applied to the problem instance defined in Example 1.1 on page 17, the computations are as follows.*

- *Tasks 5, 6, 7, 11, 14 and 15 are the only tasks with no successors, so*

$$\mu_6 = \mu_7 = \mu_{14} = \mu_{15} = -15 \qquad \textit{and} \qquad \mu_5 = \mu_{11} = -11.$$

- *Since tasks 3, 4, 12 and 13 have only successors for which μ's have been determined, their μ's may be calculated as*

$$\mu_3 = \mu_4 = \mu_{12} = \mu_{13} = \max\left\{-15, \frac{6}{2} - 15\right\} = -12.$$

- *Now tasks 2, 8 and 10 may have a value of μ calculated. This calculation gives*

$$\mu_2 = \mu_{10} = \max\left\{-11, \frac{2}{2} - 11, \frac{2}{1} - 12\right\} = -10$$

and

$$\mu_8 = \max\left\{-7, \frac{2}{2} - 11, \frac{2}{1} - 12\right\} = -7.$$

- *Finally, only tasks 1 and 9 remain, with*

$$\mu_1 = \mu_9 = \max\left\{-6, \frac{6}{1} - 10\right\} = -4.$$

When these μ's are supplied to Algorithm P the schedule depicted in Figure 4.1 is produced.

This schedule is elaborated on below.

- *At the beginning of execution, $A_0 = \{1, 8, 9\}$ and since*

$$\zeta_1(0, \eta) = \zeta_9(0, \eta) = \frac{6}{2} - 4 = -1 \qquad \textit{and} \qquad \zeta_8(0, \eta) = \frac{3}{3} - 7 = -6$$

it follows from the fact that $u_1 + u_9 = 4 > 3 = m$ that $N_1 = \{1, 9\}$, $N_2 = \emptyset$ and $N_3 = \{8\}$. The next point of allocation is $t_1 = 4$, with both tasks processed to completion.

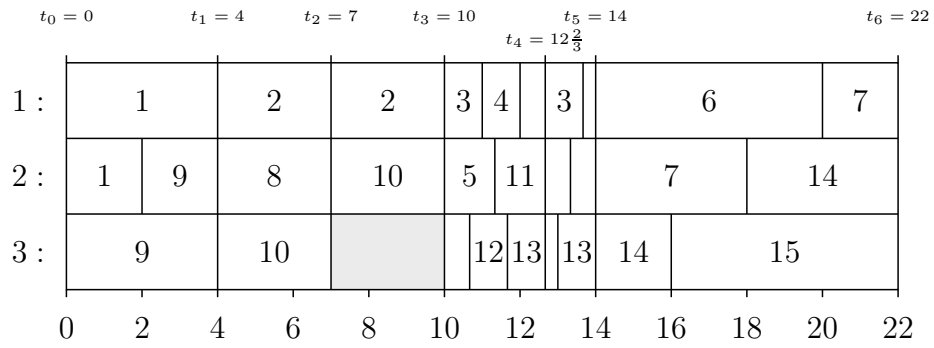


Figure 4.1: Gantt chart of the schedule produced by the Brucker-Garey-Johnson Algorithm when applied to Example 1.1.

- *Now* $A_1 = \{2, 8, 10\}$ *and since*

$$\zeta_2(t_1, \eta) = \zeta_{10}(t_1, \eta) = \frac{6}{1} - 10 = -4$$

it can be seen that $N_1 = \{2, 10\}$, $N_2 = \{8\}$ *and* $N_3 = \emptyset$. *The next point of allocation is* $t_2 = 7$, *and all three tasks have received an equal amount of processing, i.e.* 3 *units.*

- *Since task 8 has been completed,* $A_2 = \{2, 10\}$, *both of which are now in* N_2. *The next point of allocation is* $t_3 = 10$, *at which point both tasks are completed. During the interval* $[t_2, t_3]$, *one machine is left idle.*

- *Now* $A_3 = \{3, 4, 5, 11, 12, 13\}$, *and since*

$$\zeta_3(t_3, \eta) = \zeta_4(t_3, \eta) = \zeta_{12}(t_3, \eta) = \zeta_{13}(t_3, \eta) = \frac{2}{1} - 12 = -10$$

and

$$\zeta_5(t_3, \eta) = \zeta_{11}(t_3, \eta) = \frac{2}{2} - 11 = -10$$

it follows that $N_1 = N_3 = \emptyset$ *and* $N_2 = A_3$. *The next point of allocation is* $t_4 = 12\frac{2}{3}$, *and during* $[t_3, t_4]$ *each of tasks 3, 4, 12 and 13 receives* 1 *unit of processing while each of tasks 5 and 11 receives* 2 *units of processing. The latter two tasks are completed, while the former are processed during the next interval.*

- *As a consequence,* $A_4 = N_2 = \{3, 4, 12, 13\}$ *and the next point of allocation is* $t_5 = 14$. *During* $[t_4, t_5]$ *all four of the aforementioned tasks receive* 1 *unit of processing. All are completed.*

Task	$C_j(\sigma)$	$L_j(\sigma)$	Task	$C_j(\sigma)$	$L_j(\sigma)$
1	4	-2	9	4	-2
2	10	-1	10	10	-1
3	$13\frac{2}{3}$	$-1\frac{1}{3}$	11	$12\frac{2}{3}$	$1\frac{2}{3}$
4	14	-1	12	14	-1
5	$12\frac{2}{3}$	$1\frac{2}{3}$	13	14	-1
6	20	5	14	22	7
7	22	7	15	22	7
8	7	0	max	22	7

Table 4.1: Table of completion times and latenesses produced by the Brucker-Garey-Johnson Algorithm when applied to Example 1.1.

- *Finally,* $A_5 = \{6, 7, 14, 15\}$ *remain, and all have equal priority, meaning* $N_2 = A_5$*. The final point of allocation is* $t_6 = 22$*, and the remaining four tasks are processed for their entire* 6 *units of processing time and are completed.*

Table 4.1 specifies the values of $C_j(\sigma)$ *and* $L_j(\sigma)$ *for all of the tasks.*

4.3.2 The Garey-Johnson Algorithm

The concept of the Garey-Johnson Algorithm ($\mathscr{GJ}$ for short) originates from [45]. In order to describe the method of calculating μ's it is convenient to introduce the following notation:

$$a_j = \min_{g \in K_j} \mu_g \qquad \text{and} \qquad b_j = \max_{g \in K_j} \left(\frac{p_g}{u_g} + \mu_g \right). \tag{4.17}$$

In other words a_j and b_j are the minimum and maximum priorities of successors of j which can occur during the construction of schedule η.

In computing μ_j, the Garey-Johnson Algorithm examines all $a_j \leq \rho \leq b_j$. Consider an arbitrary task $g \in K_j$. If $\mu_g \leq \rho$ then the minimum amount of processing time which task g must receive in order to have priority not greater than ρ is

$$\max\left\{0, p_g - u_g(\rho - \mu_g)\right\}.$$

If every uncompleted successor of j has priority not greater than ρ and $\rho \leq \mu_g$ then g is not one of these uncompleted tasks. In other words, g has already received p_g units of processing time. By combining these two observations for each $a_j \leq \rho \leq b_j$

define

$$\beta_g(\rho) = \min\left[p_g, \max\left\{0, p_g - u_g(\rho - \mu_g)\right\}\right]. \tag{4.18}$$

Given this definition of $\beta_g(\rho)$,

$$\sum_{g \in K_j} \beta_g(\rho)$$

is the minimum machine time that must be allocated to tasks in K_j in order to ensure that the maximum priority among all incomplete tasks in K_j does not exceed ρ.

The Garey-Johnson Algorithm

1. For every task j such that $K_j = \emptyset$, set $\mu_j := -d_j$.
2. If all tasks $j \in N$ are assigned μ_j, then execute Algorithm P. Otherwise, select $j \in N$ such that
   2.1. μ_j has not been specified and
   2.2. for each $g \in K_j$, μ_g has been already specified.
3. Set

$$\mu_j := \max\left\{-d_j, \max_{a_j \le \rho \le b_j}\left(\rho + \frac{1}{m}\sum_{g \in K_j}\beta_g(\rho)\right)\right\}, \tag{4.19}$$

   and go to Step 2.

Note that the expression in the inner maximum of (4.19) is piecewise linear in ρ. The maximum must occur either at one of the end points of the interval $[a_j, b_j]$ or at one of the points where the slope changes. The number of these points is $O(n)$, so the maximum may be found in linear time. The lemma below describes a property of the μ's produced by this algorithm.

Lemma 4.13 *Equation (4.15) holds for the μ's produced by the Garey-Johnson Algorithm.*

Proof. Consider any task j for which $K_j \neq \emptyset$. From (4.17), (4.18) and (4.19),

$$\mu_j \ge \max_{a_j \le \rho \le b_j}\left(\rho + \frac{1}{m}\sum_{g \in K_j}\beta_g(\rho)\right) \ge b_j + \frac{1}{m}\sum_{g \in K_j}\beta_g(b_j) \ge \max_{g \in K_j}\left(\frac{p_g}{u_g} + \mu_g\right)$$

and the lemma is proven. ■

The Garey-Johnson Algorithm can be viewed as being able to "see" more information about a task: the Brucker-Garey-Johnson Algorithm is only able to see a

task's most urgent successor, while the Garey-Johnson Algorithm is able to see the priorities and processing times of all successors.

It can be seen that $\mathscr{GJ} \in \mathfrak{F} \cap \mathfrak{S}$: the value of μ_j is based only on d_j, m and the properties of the tasks in K_j, and renumbering the tasks does not affect the value of μ_j.

Like the Brucker-Garey-Johnson Algorithm, this algorithm produces μ's that are preserving.

Lemma 4.14 *The μ's produced by the Garey-Johnson Algorithm are preserving.*

Proof. Step 1 of the Garey-Johnson Algorithm sets $\mu_j = -d_j$ for all j such that $K_j = \emptyset$. On the other hand, (4.19) implies that $\mu_j \geq -d_j$ for all other j. Thus the μ's, produced by the Garey-Johnson Algorithm, fit the definition of modified due dates.

The rest of the lemma will be proven by induction. Note that if $n = 1$, only one μ is produced by the Garey-Johnson Algorithm and, by Lemma 4.1, it is preserving. Now, assume that the μ's produced by the Garey-Johnson Algorithm are preserving if $n \leq k$ for some $k \geq 1$ and consider the case where $n = k + 1$. It will now be proven that

$$\mu_j \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\}$$

for all $j \in N$, which, as a consequence of Lemma 4.1, will prove the lemma.

First note that either $\mu_j > -d_j$ or

$$\mu_j = -d_j \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\}.$$

Observe that when $\mu_j > -d_j$, there exists some ρ such that $a_j \leq \rho \leq b_j$ and

$$\mu_j = \rho + \frac{1}{m}\sum_{g \in K_j} \beta_g(\rho), \tag{4.20}$$

and, by the definition (4.17) of b_j, there exists a task $q \in K_j$ such that

$$\frac{p_q}{u_q} + \mu_q = b_j \geq \rho. \tag{4.21}$$

Define the set

$$Q = \left\{g : g \in K_j \wedge \frac{p_g}{u_g} + \mu_g \geq \rho\right\}$$

and for all $r \in Q$ define

$$e_r = \max\left\{t : C_r\left(\omega^j\right) \geq t \wedge \zeta_r\left(t, \omega^j\right) \geq \rho\right\},$$

i.e. the point in time during schedule ω^j at which either task r is completed or the priority of task r becomes equal to ρ, whichever comes first.

Since at least

$$\sum_{g \in K_j} \beta_g(\rho)$$

units of processing time are required to make the priority of all uncompleted successors of j at most ρ, (4.21) implies the existence of a task $r \in Q$ such that

$$e_r \geq \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho),$$

further implying

$$C_r\left(\omega^j\right) \geq \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) + \frac{p_r\left(\frac{1}{m} \sum_{g \in K_j} \beta_g(\rho), \omega^j\right)}{u_r} \tag{4.22}$$

and

$$\frac{p_r\left(\frac{1}{m} \sum_{g \in K_j} \beta_g(\rho), \omega^j\right)}{u_r} + \mu_r = \zeta_r\left(\frac{1}{m} \sum_{g \in K_j} \beta_g(\rho), \omega^j\right) \geq \rho. \tag{4.23}$$

From the inductive assumption, $|K_j| \leq n-1 = k$ implies that the Garey-Johnson Algorithm produces preserving μ's when applied to the problem instance defined by φ_j. Consequently it follows from (4.20), (4.22), (4.23) and the definition of preserving μ's that

$$\mu_j = \rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) \leq \rho + C_r\left(\omega^j\right) - \frac{p_r\left(\frac{1}{m} \sum_{g \in K_j} \beta_g(\rho), \omega^j\right)}{u_r}$$

$$\leq C_r(\omega^j) + \mu_r \leq G(\omega^j) = L_{max}(\omega^j) \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\}.$$

For $n = k+1$ the lemma follows from Lemma 4.1, and has thus been proven by induction on n. ■

To illustrate this algorithm, an example has been provided below.

Example 4.15 *When the Garey-Johnson Algorithm is applied to the problem in-*

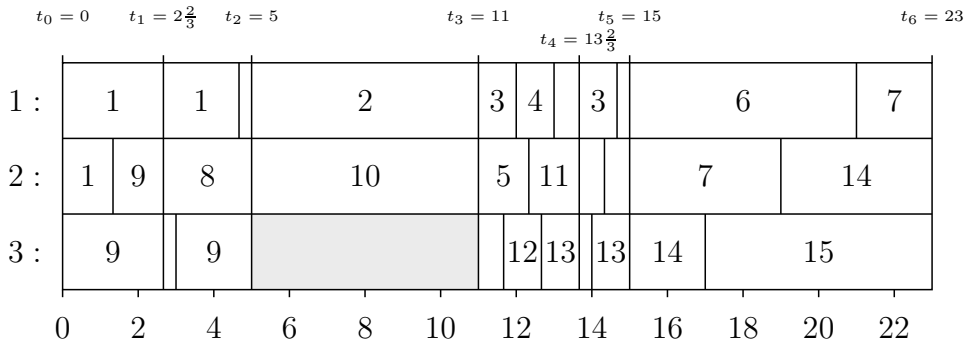


Figure 4.2: Gantt chart of the schedule produced by the Garey-Johnson Algorithm when applied to Example 1.1.

stance defined in Example 1.1 on page 17, the computations are as follows.

- *Tasks 5, 6, 7, 11, 14 and 15 are the only tasks with no successors, so*

$$\mu_6 = \mu_7 = \mu_{14} = \mu_{15} = -15 \qquad \textit{and} \qquad \mu_5 = \mu_{11} = -11.$$

- *Since tasks 3, 4, 12 and 13 have only successors for which μ's have been determined, their μ's may be calculated as*

$$\mu_3 = \mu_4 = \mu_{12} = \mu_{13} = \max\left\{-15, \frac{6}{3} - 15, \frac{0}{3} - 12\right\} = -12.$$

- *As a consequence of this,*

$$\mu_2 = \mu_{10} = \max\left\{-11, \frac{18}{3} - 15, \frac{6}{3} - 12, \frac{4}{3} - 11, \frac{0}{3} - 10\right\} = -9,$$

  and

$$\mu_8 = \max\left\{-7, \frac{36}{3} - 15, \frac{12}{3} - 12, \frac{8}{3} - 11, \frac{0}{3} - 10\right\} = -3.$$

- *Finally,*

$$\mu_1 = \mu_9 = \max\left\{-6, \frac{24}{3} - 15, \frac{12}{3} - 12, \frac{10}{3} - 11, \frac{6}{3} - 10, \frac{6}{3} - 9, \frac{0}{3} - 3\right\} = -3.$$

When these μ's are supplied to Algorithm P the schedule depicted in Figure 4.2 is produced.

This schedule is elaborated on below.

Task	$C_j(\sigma)$	$L_j(\sigma)$	Task	$C_j(\sigma)$	$L_j(\sigma)$
1	$4\frac{2}{3}$	$-1\frac{1}{3}$	9	5	-1
2	11	0	10	11	0
3	$14\frac{2}{3}$	$-\frac{1}{3}$	11	$13\frac{2}{3}$	$2\frac{2}{3}$
4	15	0	12	15	0
5	$13\frac{2}{3}$	$2\frac{2}{3}$	13	15	0
6	21	6	14	23	8
7	23	8	15	23	8
8	5	-2	max	23	8

Table 4.2: Table of completion times and latenesses produced by the Garey-Johnson Algorithm when applied to Example 1.1.

- *The first point of allocation is* $t_1 = 2\frac{2}{3}$*, and* 4 *units each of tasks 1 and 9 are processed during the interval* $[t_0, t_1]$*.*

- *At this point, the two aforementioned tasks become equal in priority to task 8, so the remaining* 2 *units of processing for tasks 1 and 9, along with all* 3 *units of task 8, are processed during* $[t_1, t_2]$*, with* $t_2 = 5$*. All three tasks are completed.*

- *Now, only tasks 2 and 10 are available for processing. Both are processed until completion at time* $t_3 = 11$*, with a single machine idle during this time.*

- *At this point, tasks 3, 4, 5, 11, 12 and 13 become available, and all have equal priority so* $t_4 = 13\frac{2}{3}$ *and during* $[t_3, t_4]$ *tasks 3, 4, 12 and 13 receive* 1 *unit of processing each while tasks 5 and 11 receive* 2 *units of proceeding each, completing the latter pair of tasks.*

- *The next point of allocation is* $t_5 = 15$*, and during* $[t_4, t_5]$ *each of 3, 4, 12 and 13 receive* 1 *unit of processing and are completed. Tasks 6, 7, 14 and 15 are now available.*

- *The final point of allocation is* $t_6 = 23$*, with all tasks being processed to completion during* $[t_5, t_6]$*.*

The table below specifies the values of $C_j(\sigma)$ *and* $L_j(\sigma)$ *for all of the tasks.*

It is worth noting that, even though the Garey-Johnson Algorithm takes more information into account than the Brucker-Garey-Johnson Algorithm, it produces a worse schedule for the problem instance defined in Example 1.1. It is important

to keep this in mind, as it will likely be beneficial in practice to employ several approximation algorithms, selecting the best output in each case.

4.3.3 The Zinder-Roper Algorithm

The concept of the Zinder-Roper Algorithm ($\mathscr{ZR}$ for short) originates from [119]. Each μ_j as computed by the Brucker-Garey-Johnson Algorithm and the Garey-Johnson Algorithm, can be viewed as a lower estimate of the urgency of successors of j. In contrast, the algorithm below can be viewed as one that "communicates" to j an upper estimate of this urgency. It achieves this by considering the problem instance φ_j (see (4.4) on page 66 for the full definition) defined on the set K_j of successors of j, and using the Zinder-Roper Algorithm itself to determine the value of μ_j.

The Zinder-Roper Algorithm

1. For every task j such that $K_j = \emptyset$, set $\mu_j := -d_j$.
2. If all tasks $j \in N$ are assigned μ_j, then execute Algorithm P. Otherwise, select $j \in N$ such that
   2.1. μ_j has not been specified and
   2.2. for each $g \in K_j$, μ_g has been already specified.
3. Using Algorithm P and the μ's already generated, construct a schedule η^j for the problem instance φ_j, set
$$\mu_j := \max\left\{-d_j, G\left(\eta^j\right)\right\}, \tag{4.24}$$
   and go to Step 2.

The algorithm above is conceptually simple: when determining how urgent a task j is, it constructs a schedule using only tasks in K_j. This has the benefit of being able to use the entire structure of precedence constraints on K_j in determining μ_j. It is not necessarily the case that $G(\sigma) = L_{max}(\sigma)$ when μ's are determined by the Zinder-Roper Algorithm, i.e. the μ's are not necessarily preserving. However, $\mathscr{ZR} \in \mathfrak{F} \cap \mathfrak{S}$ and property (4.15) does apply to the μ's generated by the Zinder-Roper Algorithm, as proven in the lemma below.

Lemma 4.16 *Equation (4.15) holds for the μ's produced by the Zinder-Roper Algorithm.*

Proof. Consider any task j for which $K_j = \emptyset$. From (4.24),

$$\mu_j \geq G\left(\eta^j\right) = \max_{g \in K_j} \left(C_g\left(\eta^j\right) + \mu_g\right) \geq \max_{g \in K_j} \left(\frac{p_g}{u_g} + \mu_g\right)$$

and the lemma is proven. ■

To illustrate this algorithm, an example has been provided below.

Example 4.17 *When the Zinder-Roper Algorithm is applied to the problem instance defined in Example 1.1 on page 17, the computations are as follows.*

- *Tasks 5, 6, 7, 11, 14 and 15 are the only tasks with no successors, so* $\mu_6 = \mu_7 = \mu_{14} = \mu_{15} = -15$ *and* $\mu_5 = \mu_{11} = -11$.

- *Problem subinstances* φ_3, φ_4, φ_{12} *and* φ_{13} *are isomorphic and* $d_3 = d_4 = d_{12} = d_{13}$ *so all four tasks must have the same value of* μ. *Scheduling the single successor* g *of task* j *results in* $C_g\left(\eta^j\right) = 3$, *giving* $\mu_3 = \mu_4 = \mu_{12} = \mu_{13} = \max\{-15, -12\} = -12$.

- *Problem subinstances* φ_2 *and* φ_{10} *are isomorphic and* $d_2 = d_{10}$ *so these two tasks must have the same value of* μ. *Note that*

$$\frac{p_5}{u_5} + \mu_5 = \frac{1}{1} - 11 = \frac{2}{1} - 12 = \frac{p_3}{u_3} + \mu_3 = \frac{p_4}{u_4} + \mu_4. \tag{4.25}$$

  Consequently, schedule η^2 *sets* $t_1 = 1\frac{1}{3}$ *and schedules* 1 *unit each of tasks 3 and 4, and* 2 *units of task 5 during the interval* $[t_0, t_1]$. *The next point of allocation is* $t_2 = 2\frac{1}{3}$, *and during the interval* $[t_1, t_2]$, 1 *unit each of tasks 3 and 4 is processed, with a single machine left idle for the duration. The final point of allocation is* $t_3 = 6\frac{1}{3}$. *There are* 6 *units each of tasks 6 and 7 processed during the interval* $[t_2, t_3]$, *resulting in their completion. Hence,*

$$\mu_2 = \mu_{10} = \max\left\{-11, -8\frac{2}{3}\right\} = -8\frac{2}{3}.$$

- *As a consequence of (4.25), schedule* η^8 *sets* $t_1 = 2\frac{2}{3}$, *with* 1 *unit each of tasks 3, 4, 12 and 13 and* 2 *units each of tasks 5 and 11 being processed during the interval* $[t_0, t_1]$. *The latter two tasks are completed in this interval. The next point of allocation is* $t_2 = 4$, *and during the interval* $[t_1, t_2]$ *tasks 3, 4, 12 and 13 each receive* 1 *unit of processing, and are completed. The final point of*

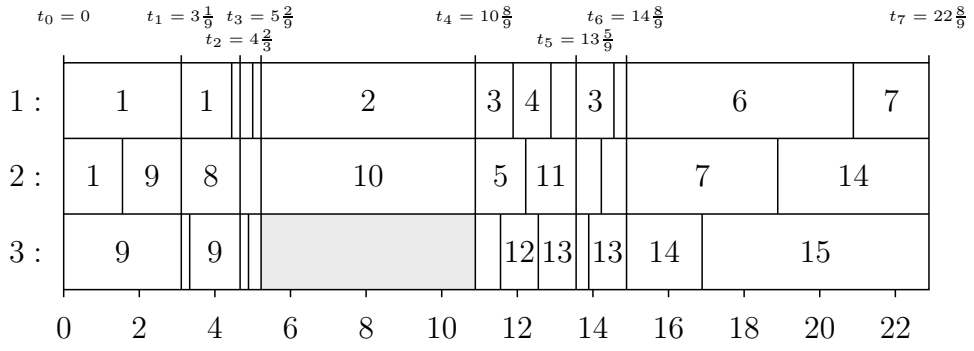


Figure 4.3: Gantt chart of the schedule produced by the Zinder-Roper Algorithm when applied to Example 1.1.

allocation is $t_3 = 12$ *with* 6 *units each of tasks 6, 7, 14 and 15 being processed during the interval* $[t_2, t_3]$. *This results in* $\mu_8 = \max\{-7, -3\} = -3$.

- *Finally, the schedules* η^1 *and* η^9 *are simply the schedules* η^2 *and* η^{10} *with an additional task beforehand. Thus,*

$$\mu_1 = \mu_9 = \max\left\{-6, -2\frac{2}{3}\right\} = -2\frac{2}{3}.$$

Once all of the μ*'s have been calculated they are supplied as arguments to Algorithm P, which produces the schedule depicted below.*

This schedule is elaborated on below.

- *Firstly,* $t_1 = 3\frac{1}{9}$ *and tasks 1 and 9 each receive* $4\frac{2}{3}$ *units of processing during the interval* $[t_0, t_1]$. *At time* t_1 *the tasks 1, 8 and 9 have equal priority, but because they have unequal* u_j *they are processed for different amounts until* $t_2 = 4\frac{2}{3}$.

- *Tasks 1 and 9 are processed for* $1\frac{1}{3}$ *units of time during* $[t_1, t_2]$ *while task 8 is processed for* 2 *units of time. Tasks 1 and 9 are completed. Task 8 now has the same priority as the newly available tasks 2 and 10.*

- *The next point of allocation is* $t_3 = 5\frac{2}{9}$, *with tasks 2 and 9 receiving* $\frac{1}{3}$ *units of processing each, and task 8 receiving* 1 *unit of processing during* $[t_2, t_3]$, *at which point it is complete.*

- *This is followed by the point of allocation* $t_4 = 10\frac{8}{9}$. *During* $[t_3, t_4]$ *one machine is idle while tasks 2 and 10 receive* $5\frac{2}{3}$ *units of processing, and both are completed.*

Task	$C_j(\sigma)$	$L_j(\sigma)$	Task	$C_j(\sigma)$	$L_j(\sigma)$
1	$4\frac{4}{9}$	$-1\frac{5}{9}$	9	$4\frac{2}{3}$	$-1\frac{1}{3}$
2	$10\frac{8}{9}$	$-\frac{1}{9}$	10	$10\frac{8}{9}$	$-\frac{1}{9}$
3	$14\frac{5}{9}$	$-\frac{4}{9}$	11	$13\frac{5}{9}$	$2\frac{5}{9}$
4	$14\frac{8}{9}$	$-\frac{1}{9}$	12	$14\frac{8}{9}$	$-\frac{1}{9}$
5	$13\frac{5}{9}$	$2\frac{5}{9}$	13	$14\frac{8}{9}$	$-\frac{1}{9}$
6	$20\frac{8}{9}$	$5\frac{8}{9}$	14	$22\frac{8}{9}$	$7\frac{8}{9}$
7	$22\frac{8}{9}$	$7\frac{8}{9}$	15	$22\frac{8}{9}$	$7\frac{8}{9}$
8	$5\frac{2}{9}$	$-1\frac{7}{9}$	max	$22\frac{8}{9}$	$7\frac{8}{9}$

Table 4.3: Table of completion times and latenesses produced by the Zinder-Roper Algorithm when applied to Example 1.1.

- *Subsequently* $t_5 = 13\frac{5}{9}$ *and during* $[t_4, t_5]$ *tasks 3, 4, 12 and 13 receive* 1 *unit of processing each while tasks 5 and 11 receive* 2 *units of processing each. The latter two tasks are completed, while the former are processed during the next interval.*

- *The next point of allocation is* $t_6 = 14\frac{8}{9}$ *and during* $[t_5, t_6]$ *each of the four aforementioned tasks receives* 1 *unit of processing. All are completed. Finally, only the tasks 6, 7, 14 and 15 remain, and all have equal priority.*

- *The final point of allocation is* $t_7 = 22\frac{8}{9}$*, and the remaining four tasks are processed for their entire* 6 *units of processing time and are completed.*

The table below specifies the values of $C_j(\sigma)$ *and* $L_j(\sigma)$ *for all of the tasks.*

4.3.4 The Optimal Recursive Algorithm

The idea of the Optimal Recursive Algorithm ($\mathscr{OR}$ for short) originates in [119]. While all of the previous algorithms could be implemented in polynomial time, this algorithm requires a recursive solution of the general problem $P|prec, pmtn|L_{max}$, which is NP-hard. This algorithm is not itself the subject of direct study, but rather exists for the purpose of comparison with other algorithms, particularly the Zinder-Roper Algorithm which it resembles.

The only difference between the Optimal Recursive Algorithm and the Zinder-Roper Algorithm is that the former uses an optimal schedule on the set of successors of a task, while the latter uses a (potentially non-optimal) schedule generated by Algorithm P and the existing values of the μ's. Similar to the previous scheduling algorithms described in this section, $\mathscr{OR} \in \mathfrak{F} \cap \mathfrak{S}$ and (4.15) holds for the μ's

The Optimal-Recursive Algorithm

1. For every task j such that $K_j = \emptyset$, set $\mu_j := -d_j$.
2. If all tasks $j \in N$ are assigned μ_j, then execute Algorithm P. Otherwise, select $j \in N$ such that
   2.1. μ_j has not been specified and
   2.2. for each $g \in K_j$, μ_g has been already specified.
3. Construct an optimal schedule schedule ω^j for the problem instance φ_j, set
$$\mu_j := \max\left\{-d_j, G\left(\omega^j\right)\right\}, \tag{4.26}$$
   and go to Step 2.

generated by the Optimal Recursive Algorithm, the latter of which is proven in the lemma below.

Lemma 4.18 *Equation (4.15) holds for the μ's produced by the Optimal Recursive Algorithm.*

Proof. Consider any task j for which $K_j = \emptyset$. From (4.26),

$$\mu_j \geq G\left(\omega^j\right) = \max_{g \in K_j}\left(C_g\left(\omega^j\right) + \mu_g\right) \geq \max_{g \in K_j}\left(\frac{p_g}{u_g} + \mu_g\right)$$

and the lemma is proven. ■

Unlike the Zinder-Roper Algorithm, the Optimal Recursive Algorithm produces μ's that are preserving.

Lemma 4.19 *The μ's produced by the Optimal-Recursive Algorithm are preserving.*

Proof. Step 1 sets $\mu_j = -d_j$ for all j such that $K_j = \emptyset$, and (4.26) implies that $\mu_j \geq -d_j$ for all other j. Thus these μ's fit the definition of modified due dates.

The remainder of the lemma will be proven by induction on n, in a similar manner to Lemma 4.14. First note that if $n = 1$, the μ produced by the Optimal Recursive Algorithm is preserving. Now assume that the μ's produced by the Optimal Recursive Algorithm are preserving if $n \leq k$ for some $k \geq 1$ and consider the case where $n = k + 1$.

For any task j for which $\mu_j = -d_j$, it follows directly that

$$\mu_j \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\}.$$

On the other hand, suppose $\mu_j = G(\omega^j)$. Since $|K_j| \leq n-1 = k$, the definition of preserving μ's implies

$$\mu_j = G\left(\omega^j\right) = L_{max}\left(\omega^j\right) \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\}.$$

Consequently, Lemma 4.1 implies that the lemma holds for $n = k+1$, and thus it has thus been proven by induction on n. ■

To illustrate this algorithm, an example has been provided below.

Example 4.20 *When the Optimal Recursive Algorithm is applied to the problem instance defined in Example 1.1 on page 17, the computations are as follows.*

- *Tasks 5, 6, 7, 11, 14 and 15 are the only tasks with no successors, so* $\mu_6 = \mu_7 = \mu_{14} = \mu_{15} = -15$ *and* $\mu_5 = \mu_{11} = -11$.
- *Problem subinstances* φ_3, φ_4, φ_{12} *and* φ_{13} *are isomorphic and* $d_3 = d_4 = d_{12} = d_{13}$ *so all four tasks must have the same value of* μ. *Scheduling the single successor* g *of task* j *results in* $C_g(\eta^j) = 3$, *giving* $\mu_3 = \mu_4 = \mu_{12} = \mu_{13} = \max\{-15, -12\} = -12$.
- *Problem subinstances* φ_2 *and* φ_{10} *are isomorphic and* $d_2 = d_{10}$ *so these two tasks must have the same value of* μ. *One example of an optimal schedule* ω^2 *processes tasks 3, 4 and 5 to completion during the time interval* $[0, 2]$, *and schedules tasks 6 and 7 to completion during* $[2, 6]$. *Hence,*

$$\mu_2 = \mu_{10} = \max\left\{-d_2, C_3\left(\omega^2\right) + \mu_3, C_5\left(\omega^2\right) + \mu_5, C_6\left(\omega^2\right) + \mu_6\right\}$$

$$= \max\{-11, -10, -9, -9\} = -9.$$

- *As a consequence of (4.25), one example of an optimal schedule* ω^8 *processes tasks 3, 4, 5, 11, 12 and 13 to completion during the time interval* $[0, 4]$, *and schedules tasks 6, 7, 14 and 15 to completion during* $[4, 12]$. *Hence,*

$$\mu_8 = \max\left\{-d_8, \max_{j \in K_j}\left(C_j\left(\omega^8\right) + \mu_j\right)\right\} = \max\{-7, -3\} = -3.$$

- *Finally, the schedules* η^1 *and* η^9 *are simply the schedules* η^2 *and* η^{10} *respectively, with an additional task beforehand. Thus,*

$$\mu_1 = \mu_9 = \max\{-6, -3\} = -3.$$

These values are identical to those produced by the Garey-Johnson Algorithm, and consequently the schedule depicted in Figure 4.2 is produced by Algorithm P.

Note that the μ's produced by the Optimal Recursive Algorithm for tasks 2 and 10 is -9, slightly smaller than the $-8\frac{2}{3}$ produced by the Zinder-Roper Algorithm; this can be viewed as the Zinder-Roper Algorithm in some sense "overestimating" the urgency of the completion of tasks 2 and 10. Also note that, in spite of the fact that the Optimal Recursive Algorithm uses more information, and has a far greater computational complexity, than the Brucker-Garey-Johnson Algorithm, the schedule it produces for the problem instance defined in Example 1.1 is worse.

4.4 Schedule Structure

Before any theorems can be proven, it is necessary to describe the structure of a schedule produced by Algorithm P, i.e. a schedule produced by any algorithms in family $\mathfrak{P}$. This is common to many examinations of the theoretical properties of scheduling algorithms in the literature – analogous statements to many of the following lemmas can be found, for example, in [105].

The following lemma allows a simplification of the reasoning about schedules constructed by Algorithm P. It permits the analysis of schedules solely by the priorities of tasks at points of allocation, rather than having to examine the priorities at arbitrary points in time.

Lemma 4.21 *There exist a point of allocation t_i and a task j such that*

$$C_j(\eta) \geq t_i \qquad \textit{and} \qquad t_i + \zeta_j(t_i, \eta) = G(\eta).$$

Proof. Observe that

$$C_g(\eta) \geq t \qquad \text{and} \qquad t + \zeta_g(t, \eta) = G(\eta)$$

holds for $t = C_g(\eta)$ and any g such that

$$C_g(\eta) + \mu_g = G(\eta).$$

Hence, if $C_g(\eta)$ coincides with a point of allocation, the lemma holds.

Suppose for a task g selected as above that $t_{i-1} < C_g(\eta) < t_i$, where t_{i-1} and t_i are consecutive points of allocation. Then, $g \in N_2^{i-1}$ and the total machine time that Algorithm P allocates at t_{i-1} to the tasks in $N_1^{i-1} \cup N_2^{i-1}$ is $m(t_i - t_{i-1})$. Hence,

by the Modified McNaughton Algorithm, there exists $j \in N_2^{i-1}$ such that $C_j(\eta) \geq t_i$. Therefore,

$$G(\eta) \geq C_j(\eta) + \mu_j \geq t_i + \zeta_j(t_i, \eta) = t_i + \zeta_g(t_i, \eta) > C_g(\eta) + \mu_g = G(\eta)$$

which contradicts the selection of g, and thus the lemma is proven. ■

Let t_{i^*} be the smallest value of t that is a point of allocation satisfying Lemma 4.21, and denote by x a corresponding task j, i.e.

$$C_x(\eta) \geq t_{i^*} \qquad \text{and} \qquad t_{i^*} + \zeta_x(t_{i^*}, \eta) = G(\eta).$$

Define

$$\rho^* = \zeta_x(t_{i^*}, \eta). \tag{4.27}$$

In much of the analysis contained in this thesis, these values are used as points of reference. Processing that occurs after time t_{i^*} can be ignored, as can processing that is less urgent that ρ^*, greatly simplifying this analysis.

The value of i^* can have significant implications for the nature of the schedule, as demonstrated by the following lemma.

Lemma 4.22 *If $i^* = 0$ the schedule η minimises the criterion G.*

Proof. If $i^* = 0$, there is some task x for which

$$\frac{p_x}{u_x} + \mu_x = t_0 + \zeta_x(t_0, \eta) = G(\eta).$$

By the definition of u_x, for any schedule σ

$$C_x(\sigma) \geq \frac{p_x}{u_x},$$

and hence

$$G(\sigma) \geq C_x(\sigma) + \mu_x \geq \frac{p_x}{u_x} + \mu_x = G(\eta).$$

Since all schedules σ have $G(\sigma) \geq G(\eta)$, η minimises the criterion G. ■

Since the time point t_{i^*} is important to the analysis of schedules produced by Algorithm P, the next lemma is introduced in order to understand the processing of tasks immediately prior to this time.

Lemma 4.23 *If $i^* > 0$, then $N_1^{i^*-1} = \emptyset$.*

Proof. Suppose that $N_1^{i^*-1} \neq \emptyset$. Then, by Corollary 4.5, there exists $g \in N_1^{i^*-1}$, which may coincide with x, such that

$$t_{i^*} + \zeta_g(t_{i^*}, \eta) \geq t_{i^*} + \zeta_x(t_{i^*}, \eta) = t_{i^*} + \rho^* = G(\eta).$$

Then, by (4.7),

$$t_{i^*-1} + \zeta_g(t_{i^*-1}, \eta) = t_{i^*-1} + \frac{p_g(t_{i^*-1}, \eta)}{u_g} + \mu_g$$

$$= t_{i^*-1} + \frac{p_g(t_{i^*}, \eta)}{u_g} + (t_{i^*} - t_{i^*-1}) + \mu_g = t_{i^*} + \zeta_g(t_{i^*}, \eta) \geq G(\eta)$$

which contradicts the selection of t_{i^*}. Hence, the lemma is proven. ■

For all $0 \leq i < s$, define the set

$$H_i(\rho) = \{j : j \in A_i \wedge \zeta_j(t_{i+1}, \eta) \geq \rho\}.$$

In essence, this represents the set of all important tasks (with priority not less than ρ at time t_{i+1}) that are available at t_i. This is used to avoid the consideration of the processing of all tasks, focusing only on those which have a sufficiently high priority.

For a given value of ρ, an interval $[t_i, t_{i+1}]$ will be called ρ-complete if

$$\sum_{j \in H_i(\rho)} u_j \geq m,$$

and ρ-incomplete otherwise. This idea of ρ-complete and ρ-incomplete intervals is introduced for the purpose of further simplifying the analysis of schedules: often, the precise scheduling of tasks during an interval can be ignored, with attention given only to whether the interval is ρ-complete and ρ-incomplete.

The following lemma demonstrates that, during ρ-complete intervals, only tasks with priority not less than ρ are processed.

Lemma 4.24 *For any ρ-complete interval $[t_i, t_{i+1}]$, $(N_1^i \cup N_2^i) \subseteq H_i(\rho)$.*

Proof. Suppose that there exists $j \in (N_1^i \cup N_2^i) \setminus H_i(\rho)$. By the definition of $H_i(\rho)$, it follows that $\zeta_j(t_{i+1}, \eta) < \zeta_g(t_{i+1}, \eta)$ for all $g \in H_i(\rho)$. By Lemma 4.5, it follows that $\zeta_j(t_i, \eta) < \zeta_g(t_i, \eta)$ for all $g \in H_i(\rho)$.

By the definition of a ρ-complete interval,

$$\sum_{g \in L} u_g \geq m, \qquad \text{where} \qquad L = \{g : g \in A_i \wedge \zeta_j(t_i, \eta) < \zeta_g(t_i, \eta)\}.$$

This implies that $j \notin N_1^i \cup N_2^i$, and the lemma is proven. ■

Corollary 4.25 *For any ρ-complete interval $[t_i, t_{i+1}]$,*

$$\sum_{j \in H_i(\rho)} [p_j(t_i, \eta) - p_j(t_{i+1}, \eta)] = m(t_{i+1} - t_i).$$

One of the conditions in the statement of the following lemma is that the μ's obey (4.15), and therefore this lemma holds for all four scheduling algorithms described in this chapter. This lemma will be used, along with Lemma 4.29 and Corollary 4.28, to demonstrate the existence of a chain of "high priority" tasks of a given length. This in turn will be used to establish several performance guarantees in terms of the length of the longest chain of tasks in the considered problem instance, in a similar manner to [105].

Lemma 4.26 *If all μ's obey (4.15), for any $0 < i' < s$, any ρ, any $j \in A_{i'}$ for which $\zeta_j(t_{i'}, \eta) \geq \rho$, and any i such that $0 \leq i < i'$, either $j \in H_i(\rho)$ or there exists a task $g \in H_i(\rho)$ such that $g \to j$.*

Proof. Consider any ρ, i, i' and j as specified in the statement of the lemma. If $j \in A_i$ then let $g = j$ and observe that

$$\zeta_g(t_{i+1}, \eta) = \zeta_j(t_{i+1}, \eta) = \frac{p_j(t_{i+1}, \eta)}{u_j} + \mu_j \geq \frac{p_j(t_{i'}, \eta)}{u_j} + \mu_j = \zeta_j(t_{i'}, \eta) \geq \rho.$$

On the other hand, if $j \notin A_i$, the fact that $C_j(\eta) > t_{i'} > t_i$ implies the existence of some task $g \in J_j \cap A_i$. In this case (4.15) implies

$$\zeta_g(t_{i+1}, \eta) = \frac{p_g(t_{i+1}, \eta)}{u_g} + \mu_g \geq \mu_g \geq \frac{p_j}{u_j} + \mu_j$$

$$\geq \frac{p_j(t_{i'}, \eta)}{u_j} + \mu_j = \zeta_j(t_{i'}, \eta) \geq \rho.$$

Thus in either case $\zeta_g(t_{i+1}, \eta) \geq \rho$ and $g \in A_i$, implying $g \in H_i(\rho)$. ■

The lemma below is presented in order to obtain Corollary 4.28. It establishes a link between ρ-incompleteness and the processing of high priority tasks

during an interval. Note that the lemma also applies to ρ-complete intervals where $\sum_{j \in H_i(\rho)} u_j = m$.

Lemma 4.27 *For any ρ, any $0 \leq i < s$ such that $\sum_{j \in H_i(\rho)} u_j \leq m$, and for any $j \in H_i(\rho)$, it follows that $j \in N_1^i$.*

Proof. Consider any ρ, i and j as in the statement of the lemma, and note that, for any $g \in A_i \setminus H_i(\rho)$, it follows that

$$\zeta_g(t_{i+1}, \eta) < \zeta_j(t_{i+1}, \eta).$$

By Lemma 4.9 it follows that

$$\zeta_g(t_i, \eta) < \zeta_j(t_i, \eta)$$

and hence

$$\{q : \zeta_q(t_i, \eta) \geq \zeta_j(t_i, \eta)\} \subseteq H_i(\rho).$$

Finally, since $\sum_{j \in H_i(\rho)} u_j \leq m$ it follows that

$$\sum_{q : \zeta_q(t_i, \eta) \geq \zeta_j(t_i, \eta)} u_q \leq m$$

and thus

$$\{q : \zeta_q(t_i, \eta) \geq \zeta_j(t_i, \eta)\} \subseteq N_1^i,$$

implying $j \in N_1^i$. ■

The corollary below follows from Lemma 4.26 and Lemma 4.27. As stated earlier, it will be used to prove the existence of a chain of high priority tasks. It is quite similar to reasoning found in [105].

Corollary 4.28 *If all μ's obey (4.15), for any $0 < i' < s$, any ρ, any $j \in H_{i'}(\rho)$ and any i such that $0 \leq i < i'$ and $[t_i, t_{i+1}]$ is ρ-incomplete, either $j \in H_i(\rho) \cap N_1^i$ or there exists a task $g \in H_i(\rho) \cap N_1^i$ such that $g \to j$.*

Repeated application of the preceding corollary demonstrates the existence of a chain of tasks being processed in N_1 throughout all ρ-incomplete intervals. Additionally, if $\rho \leq \rho^*$, Lemma 4.29 implies that task x must be preceded by such a chain.

To simplify the notation where possible, let $H_i = H_i(\rho^*)$, i.e. if the argument ρ of $H_i(\rho)$ it not included it is assumed that $\rho = \rho^*$. Define

$$H = \bigcup_{i=0}^{i^*-1} H_i.$$

The following lemma demonstrates the existence of at least one task in H_{i^*-1} that is not in any earlier ρ^*-incomplete interval.

Lemma 4.29 *If $i^* > 0$ there exists a task $q \in H_{i^*-1}$ such that, for all ρ^*-incomplete intervals $[t_i, t_{i+1}]$ for which $i < i^*$, $q \notin H_i$.*

Proof. Consider the largest $i < i^*$ for which $[t_i, t_{i+1}]$ is rho^*-incomplete, and observe that, by Lemma 4.23,

$$\sum_{j \in H_i} u_j < m \qquad \text{while} \qquad \sum_{j \in H_{i^*-1}} u_j > m. \tag{4.28}$$

Since $p_j(t_i, \eta) \geq p_j(t_{i+1}, \eta)$ for all j and i, it follows that $(A_i \cap H_{i^*-1}) \subseteq H_i$. Hence, (4.28) implies the existence of a task $q \in H_{i^*-1} \setminus H_i$, and hence there is some $g \in J_q$ for which $C_g(\eta) > t_i$. ■

The corollary below follows from Lemma 4.26 and Lemma 4.29.

Corollary 4.30 *If all μ's obey (4.15), $|H_i| \geq 1$ for all $0 \leq i < i^*$.*

The corollary below follows from Lemma 4.23 and Lemma 4.29.

Corollary 4.31 *If $i^* > 0$,*

$$\sum_{j \in H_{i^*-1}} u_j > m.$$

With the definitions and results given in this chapter, it is now possible to analyse the approximation algorithms $\mathscr{BGJ}$, $\mathscr{GJ}$ and $\mathscr{ZR}$. This will be done in the following two chapters, Chapter 5 presenting several performance guarantees and Chapter 6 examining the solvable cases.

Chapter 5

Performance Guarantees

Since the $P|prec, pmtn, u_j|L_{max}$ problem considered in this thesis is NP-hard, approximation algorithms are a much studied area of research. It is the aspiration when using approximation algorithms, that the algorithm selected will produce schedules that, while not necessarily optimal, are "close" to optimal in some sense. One way of demonstrating this is to provide some bound on the relationship between $L_{max}(\eta)$, the maximum lateness of the schedule generated by the approximation algorithm, and $L_{max}(\omega)$, the maximum lateness of an optimal schedule.

Generally in the literature there have been two kinds of bounds:

- difference bounds, i.e. upper bounds on $L_{max}(\eta) - L_{max}(\omega)$, seen for example in [105]; and

- ratio bound, i.e. upper bounds on

$$\frac{L_{max}(\eta) + d_{max}}{L_{max}(\omega) + d_{max}} \qquad \text{where} \qquad d_{max} = \max_{j \in N} d_j,$$

  often expressed in the form

$$L_{max}(\eta) \leq \alpha L_{max}(\omega) + (\alpha - 1) d_{max},$$

  as seen for example in [120].

One desirable property of either kind of bound is that of being tight, i.e. no smaller bound exists. Tightness has generally been much easier to prove for difference bounds than for ratio bounds. The goal of this chapter is to provide new bounds, often based on existing bounds for other problems, for the considered approximation algorithms. A completely new result, presented in Section 5.3, demonstrates that bounds on the worst case performance of algorithms in family $\mathfrak{F} \cap \mathfrak{S}$

cannot have performance guarantees arbitrarily close to optimal. Even more surprisingly, the Garey-Johnson Algorithm is in some sense the "dominant" algorithm of family $\mathfrak{F} \cap \mathfrak{S}$, with a performance guarantee that cannot be improved upon by any algorithm in $\mathfrak{F} \cap \mathfrak{S}$.

Several definitions are required before the bounds may be described. Firstly

$$u_{min} = \min_{j \in N} u_j$$

is the maximum parallelism allowed for a given problem instance. This is a parameter that appears in all bounds in this section. Indeed, one of the motivating factors in considering the $P|prec, pmtn, u_j|L_{max}$ problem is to investigate the effect that parallelism has on the performance of approximation algorithms. The problem instance specified in Example 1.1 on page 17 has $u_{min} = u_2 = 1$.

Another useful definition is

$$r = \min_{j \in N} \frac{p_j}{u_j},$$

the minimum ratio of processing time to parallelism and the least amount of time in which a task may be completed. This may be viewed as analogous to p_{min}, the minimum processing time, which appeared in a bound in [105]. The problem instance specified in Example 1.1 has $r = \frac{p_8}{u_8} = 1$.

Finally, the length of the longest chain must be defined. A chain is a totally ordered set of tasks $j_1 \to j_2 \to \cdots \to j_h$. The length of a chain C may be defined as

$$\sum_{j \in C} \frac{p_j}{u_j}.$$

Let l be the length of the longest chain. The problem instance specified in Example 1.1 has $l = 14$, arising from a chain $1 \to 2 \to 3 \to 6$.

All of the above values differ depending on the problem instance considered. Bounds on the performance of given algorithms involve these values, indicating that some problem instances are more amenable to solution than others.

5.1 The Brucker-Garey-Johnson Algorithm

The performance guarantee established by Theorem 5.2 is a generalisation to the $P|prec, pmtn, u_j|L_{max}$ problem of the bound for $P|prec, pmtn|L_{max}$ presented in [105]. It is an improvement over [105], not just because it is for a more general

problem, but also because the bound below is tight for all $m \geq u_{min}$ and $l \geq r$: the bound presented in [105] was only proven tight for integer values of $\frac{l}{r}$. Additionally, the bound applies in general to any μ assignment algorithm for which the μ's are preserving and obey (4.15). However, it will be proven tight in the specific case of the Brucker-Garey-Johnson Algorithm only. These results were originally presented in [124].

The two criteria, that the μ's be preserving, and that they obey (4.15), do not seem very stringent. It is noteworthy that these are the only criteria necessary to guarantee a bound of reasonable quality.

5.1.1 Proving the Bound

Let c be the total length of all ρ^*-complete intervals in $[0, t_{i^*}]$ and let w be the total length of all ρ^*-incomplete intervals in $[0, t_{i^*}]$. These definitions will be used throughout this section.

Lemma 5.1 *For any schedule η produced by Algorithm P using preserving μ's which obey (4.15), and any optimal schedule ω,*

$$L_{max}(\eta) - L_{max}(\omega) \leq \frac{m - u_{min}}{m} w.$$

Proof. First suppose that $i^* = 0$, and note that in this case $w = 0$. Additionally, Lemma 4.22 implies that η minimises G, and so the lemma follows from the definition of preserving μ's.

On the other hand, suppose that $i^* > 0$, and hence Lemma 4.29 applies. By the definition of t_{i^*},

$$L_{max}(\eta) = t_{i^*} + \rho^* = c + w + \rho^*. \tag{5.1}$$

By (4.15), Corollary 4.28 applies – Lemma 4.29 implies $H_{i^*-1} \neq \emptyset$ and hence Corollary 4.28 implies that $|H_i| \geq 1$ for all $0 \leq i < i^*$. Define

$$\delta_i = \begin{cases} m & \text{if } [t_i, t_{i+1}] \text{ is } \rho^*\text{-complete} \\ u_{min} & \text{if } [t_i, t_{i+1}] \text{ is } \rho^*\text{-incomplete} \end{cases}$$

for all $0 \leq i < i^*$. The definition of preserving μ's, along with Corollary 4.25 and Lemma 4.27, implies

$$L_{max}(\omega) = G(\omega) \geq \rho^* + \frac{1}{m} \sum_{i=0}^{i^*-1} \sum_{j \in H_i(\rho_k)} \left(p_j(t_i, \eta) - p_j(t_{i+1}, \eta) \right)$$

$$\geq \rho^* + \frac{1}{m}\sum_{i=0}^{i^*-1}\delta_i(t_{i+1} - t_i) = \rho^* + c + \frac{u_{min}}{m}w.$$

The bound in the statement of the lemma is obtained by substituting (5.1) into the inequality above. ■

The bound that is the central result of this section is presented in the theorem below. Note that it holds for every possible combination of m, u_{min}, l and r.

Theorem 5.2 *For any schedule η produced by Algorithm P using preserving μ's which obey (4.15), and any optimal schedule ω,*

$$L_{max}(\eta) - L_{max}(\omega) \leq \begin{cases} 0 & \text{if } l < 2r \\ \dfrac{m - u_{min}}{m}(l - r) & \text{otherwise} \end{cases} \tag{5.2}$$

for all combinations of $l \geq r > 0$ and $m \geq u_{min} \geq 1$.

Proof. Note that, by (4.15), Corollary 4.28 applies and observe that, if $i^* = 0$, Lemma 4.22 and the definition of preserving μ's together imply that $L_{max}(\eta) - L_{max}(\omega) \leq 0$, and therefore the theorem holds in this case. For the remainder of the proof, assume that $i^* > 0$.

Consider the case where $l < 2r$. According to Corollary 4.28, for all ρ^*-incomplete intervals $[t_i, t_{i+1}]$ for which $i < i^*$, each task in H_{i^*-1} is either in H_i or has a predecessor in H_i. The condition that $l < 2r$ implies that there are no precedence constraints – if there was any pair of tasks j and g for which $j \to g$,

$$l \geq \frac{p_j}{u_j} + \frac{p_g}{u_g} \geq 2r.$$

By Corollary 4.31,

$$\sum_{j \in H_{i*-1}} u_j > m.$$

Since there are no precedence constraints, $A_{i+1} \subseteq A_i$, and since $p_j(t_{i+1}, \eta) \leq p_j(t_i, \eta)$ for all j and i, it follows that $[t_i, t_{i+1}]$ is ρ-complete for all $0 \leq i < i^*$, and therefore $w = 0$. Hence, Lemma 5.1 implies that $L_{max}(\eta) - L_{max}(\omega) \leq 0$ and the theorem holds.

On the other hand, suppose $l \geq 2r$. Notice that Corollary 4.28 and Lemma 4.29 imply the existence of a chain of tasks. Specifically, construct a chain $C = \{j_0, j_1, \cdots\}$ using the Brucker-Garey-Johnson Chain Algorithm.

Since Step 2 of the algorithm above always selects a smaller value of i, this algorithm always terminates. Lemma 4.29 guarantees the existence of task j_0 as

The Brucker-Garey-Johnson Chain Algorithm

1. Initialise $e := 1$ and $i := i^* - 1$. Select j_0 such that $j_0 \in H_{i^*-1}$ and, for all ρ^*-incomplete intervals $[t_i, t_{i+1}]$ for which $i < i^*$, $j_0 \notin H_i$.
2. If there exists an i' such that $0 \leq i' < i$ and $[t_{i'}, t_{i'+1}]$ is ρ^*-incomplete, update i to be the largest such i'. If no such i' exists, terminate the algorithm.
3. If $j_{e-1} \notin A_i$, select as j_e any task in $J_{j_{e-1}} \cap H_i$ and increment e.
4. Go to Step 2.

specified in Step 1. Corollary 4.28, and the fact that $j_{e-1} \in H$ every time Step 3 is executed, imply that either $j_{e-1} \in H_i$ or $J_{j_{e-1}} \cap H_i \neq \emptyset$. Consequently, the selection of j_e in Step 3 of the above algorithm is valid.

The length of chain C provides a lower bound on l, and since each task $j \in C \backslash \{j_0\}$ was selected from H_i for some ρ^*-incomplete interval $[t_i, t_{i+1}]$, Lemma 4.27 implies that $j \in N_1^i$. Furthermore, since r is a lower bound on the length of j_0,

$$l \geq \sum_{j \in C} \frac{p_j}{u_j} = \frac{p_{j_0}}{u_{j_0}} + \sum_{j \in C \backslash \{j_0\}} \frac{p_j}{u_j} \geq r + w.$$

The theorem is proven by combining the above bound on $l - r$ with Lemma 5.1. ■

Corollary 5.3 *For any schedule η produced by Algorithm P using preserving μ's which obey (4.15), and any optimal schedule ω,*

$$L_{max}(\eta) \leq \left(2 - \frac{u_{min}}{m}\right) L_{max}(\omega) + \left(1 - \frac{u_{min}}{m}\right)(d_{max} - r).$$

Proof. This corollary follows from the substitution of $L_{max}(\omega) \geq l - d_{max}$ into (5.2). ■

5.1.2 Proving Tightness

The tightness of (5.2) for the Brucker-Garey-Johnson Algorithm is proven below, by specifying a problem instance for which the Brucker-Garey-Johnson Algorithm precisely meets (5.2). Lemma 5.4 establishes the maximum lateness of the schedule $\eta^{\mathscr{BGJ}}$ produced by the application of the Brucker-Garey-Johnson Algorithm to this problem instance, Lemma 5.5 establishes a bound on the optimal maximum lateness of the problem instance, and they are combined in Theorem 5.6.

For any combination of $l \geq 2r > 0$ and $m \geq u_{min} \geq 1$, consider an instance φ of the $P|prec, pmtn, u_j|L_{max}$ problem, where the tasks form three sets: X, Y and

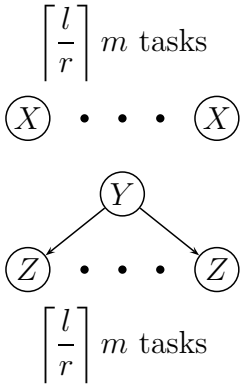


Figure 5.1: Precedence graph of φ.

Z. The set X is comprised of $\left\lceil \frac{l}{r} \right\rceil m$ identical tasks. The set Y is comprised of only one task, and the set Z is comprised of $\left\lceil \frac{l}{r} \right\rceil m$ identical tasks. The sole task in Y precedes all tasks in Z. No other precedence constraints exist between the tasks. The processing times, the permissible parallelisms of tasks' processing, and the due dates are as follows:

- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = r$ for all $j \in X$;
- $p_j = (l - r)u_{min}$, $u_j = u_{min}$ and $d_j = l$ for $j \in Y$; and
- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = l + r$ for all $j \in Z$.

Figure 5.1 depicts the partially ordered set of tasks in φ. In this figure, each task is labelled with the name of the set to which it belongs.

Observe that

- the longest chain is composed of the task constituting Y and any task in Z, and has a length of l;
- all tasks j have $u_j = u_{min}$; and
- all tasks j have $\frac{p_j}{u_j} \geq r$, with $\frac{p_j}{u_j} = r$ for all $j \in X \cup Z$.

Lemma 5.4 *When the Brucker-Garey-Johnson Algorithm is applied to φ as defined above,*

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) = 2\left\lceil \frac{l}{r} \right\rceil ru_{min} - 2r$$

for any combination of $l \geq 2r > 0$ and $m \geq u_{min} \geq 1$.

Proof. From (4.16), the values of μ_j assigned by the Brucker-Garey-Johnson Algorithm are

- for all $j \in Z$, $\mu_j = -d_j = -l - r$;
- for the sole task $j \in Y$,

$$\mu_j = \max\left\{-d_j, \max_{g \in K_j}\left(\frac{p_g}{u_g} + \mu_g\right)\right\} = \max\{-l, -l\} = -l;$$

and

- for all $j \in X$, $\mu_j = -d_j = -r$.

These values produce a schedule $\eta^{\mathscr{BGJ}}$ as follows.

At time $t_0 = 0$, the tasks in $X \cup Y$ are available while the tasks in Z are not, i.e. $A_0 = X \cup Y$. For all $j \in X$, $\zeta_j\left(t_0, \eta^{\mathscr{BGJ}}\right) = r - r = 0$ and for the sole task $j \in Y$, $\zeta_j\left(t_0, \eta^{\mathscr{BGJ}}\right) = l - r - l = -r$. Since the tasks in X have the highest priority, and since $l \geq 2r$ implies

$$\sum_{j \in X} u_j = \left\lceil \frac{l}{r} \right\rceil m u_{min} > m,$$

$N_1^0 = \emptyset$, $N_2^0 = X$ and $N_3^0 = Y$. Recall that, when Algorithm P calculates the length Δ of the interval between the current and next points of allocation, Δ is constrained by some (possibly strict) subset of the inequalities (4.10), (4.11), (4.12), (4.13) and (4.14). Since $N_1^0 = \emptyset$, only (4.11) and (4.14) are used in the determination of t_1. Since (4.11) gives

$$t_1 = \Delta \leq \frac{\left\lceil \frac{l}{r} \right\rceil m u_{min}}{m - 0} \min_{j \in N_2^0} \frac{p_j\left(t_0, \eta^{\mathscr{BGJ}}\right)}{u_j} = \left\lceil \frac{l}{r} \right\rceil r u_{min},$$

and (4.14) gives, for any $g \in N_2^0$,

$$t_1 = \Delta \leq \frac{\left\lceil \frac{l}{r} \right\rceil m u_{min}}{m - 0} \left(\zeta_g\left(t_0, \eta^{\mathscr{BGJ}}\right) - \max_{j \in N_3^0} \zeta_j\left(t_0, \eta^{\mathscr{BGJ}}\right)\right) = \left\lceil \frac{l}{r} \right\rceil r u_{min},$$

it follows that

$$t_1 = \left\lceil \frac{l}{r} \right\rceil r u_{min}.$$

For all $j \in X$, (4.8) implies that $v_j = r$, and thus all tasks in X are completed by time t_1. This gives

$$C_j\left(\eta^{\mathscr{BGJ}}\right) - d_j \leq \left\lceil \frac{l}{r} \right\rceil r u_{min} - r \qquad \text{for all } j \in X. \tag{5.3}$$

Similarly, for the sole task $j \in Y$, (4.9) implies $v_j = 0$, i.e. j is not processed at all before time t_1.

All of this implies that, at time t_1, only the sole task in Y is available, i.e. $A_1 = Y$. Since

$$\sum_{j \in Y} u_j = u_{min} \leq m,$$

$N_1^1 = Y$, $N_2^1 = \emptyset$ and $N_3^1 = \emptyset$, and only (4.10) is used in the determination of t_1. Hence, (4.10) determines the length of the interval as

$$t_2 - t_1 = \Delta = \min_{j \in N_1^1} \frac{p_j\left(t_1, \eta^{\mathscr{BGJ}}\right)}{u_j} = l - r.$$

From (4.7), it follows that the sole task $j \in Y$ is completed, giving

$$C_j\left(\eta^{\mathscr{BGJ}}\right) - d_j = t_2 - l = \left\lceil \frac{l}{r} \right\rceil r u_{min} - r \qquad \text{for the sole task } j \in Y. \tag{5.4}$$

Now the only tasks available are the tasks from Z, i.e. $A_2 = Z$. Since $l \geq 2r$ implies

$$\sum_{j \in Z} u_j = \left\lceil \frac{l}{r} \right\rceil m u_{min} > m,$$

$N_1^2 = \emptyset$, $N_2^2 = Z$ and $N_3^2 = \emptyset$, and only (4.11) is used in the determination of t_1, giving

$$t_3 - t_2 = \Delta = \frac{\left\lceil \frac{l}{r} \right\rceil m u_{min}}{m - 0} \min_{j \in N_2^2} \frac{p_j\left(t_2, \eta^{\mathscr{BGJ}}\right)}{u_j} = \left\lceil \frac{l}{r} \right\rceil r u_{min}.$$

By (4.8), all tasks in Z are completed by t_3, giving

$$C_j\left(\eta^{\mathscr{BGJ}}\right) - d_j = t_3 - l - r = 2\left\lceil \frac{l}{r} \right\rceil r u_{min} - 2r \qquad \text{for all } j \in Z. \tag{5.5}$$

The lemma follows from (5.3), (5.4) and (5.5), along with the fact that (5.5) is an equality for some $j \in Z$, and the fact that

$$2\left\lceil \frac{l}{r} \right\rceil r u_{min} - 2r > \left\lceil \frac{l}{r} \right\rceil r u_{min} - r + \left\lceil \frac{r}{r} \right\rceil r - r = \left\lceil \frac{l}{r} \right\rceil r u_{min} - r.$$

■

Lemma 5.5 *For any optimal schedule ω for φ,*

$$L_{max}(\omega) \leq 2\left\lceil \frac{l}{r} \right\rceil r u_{min} - 2r - \frac{m - u_{min}}{m}(l - r)$$

for any combination of $l \geq 2r > 0$ *and* $m \geq u_{min} \geq 1$.

Proof. Consider the schedule σ defined as follows. Since

$$\max_{j \in X \cup Y} \frac{p_j}{u_j} = l - r < l \leq \left\lceil \frac{l}{r} \right\rceil r < \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{u_{min}}{m}(l-r) = \frac{1}{m} \sum_{j \in X \cup Y} p_j,$$

use the Modified McNaughton Algorithm to schedule all $j \in X \cup Y$ during

$$\left[0, \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{u_{min}}{m}(l-r)\right],$$

giving

$$C_j(\sigma) - d_j \leq \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{u_{min}}{m}(l-r) - r \qquad \text{for all } j \in X \tag{5.6}$$

and

$$C_j(\sigma) - d_j \leq \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{u_{min}}{m}(l-r) - l \qquad \text{for the sole task } j \in Y. \tag{5.7}$$

Finally, use the Modified McNaughton Algorithm to schedule all tasks in Z during

$$\left[\left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{u_{min}}{m}(l-r), 2\left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{u_{min}}{m}(l-r)\right],$$

giving

$$C_j(\sigma) - d_j \leq 2\left\lceil \frac{l}{r} \right\rceil r u_{min} - 2r - \frac{m - u_{min}}{m}(l-r) \qquad \text{for all } j \in Z. \tag{5.8}$$

Combining (5.6), (5.7) and (5.8), along with the fact that the maximum lateness of any feasible schedule is an upper bound of the maximum lateness of an optimal schedule, gives

$$L_{max}(\omega) \leq L_{max}(\sigma) \leq 2\left\lceil \frac{l}{r} \right\rceil r u_{min} - 2r - \frac{m - u_{min}}{m}(l-r).$$

■

Theorem 5.6 *For the Brucker-Garey-Johnson Algorithm the bound (5.2) is tight for all combinations of* $l \geq r > 0$ *and* $m \geq u_{min} \geq 1$.

Proof. Let $\eta^{\mathscr{BGJ}}$ be a schedule produced by the Brucker-Garey-Johnson Algorithm. When $l < 2r$, (5.2) is zero and thus the bound must be tight. On the other

hand, if $l \geq 2r$, combining Theorem 5.2, Lemma 5.4 and Lemma 5.5 gives

$$\frac{m - u_{min}}{m}(l - r) \geq L_{max}\left(\eta^{\mathscr{PGJ}}\right) - L_{max}(\omega) \geq \frac{m - u_{min}}{m}(l - r),$$

and the theorem is proven. ■

This section has established a performance guarantee under very broad conditions, and has proven that it is met for the Brucker-Garey-Johnson Algorithm for all possible combinations of m, u_{min}, l and r. This expands on what is already known by adding variable task parallelisms, providing a comprehensive proof of tightness, and establishing broad conditions under which the bound holds.

5.2 The Garey-Johnson Algorithm

The bound on $L_{max}\left(\eta^{\mathscr{GJ}}\right)$ proven in this section is a non-trivial adaptation to the $P|prec, pmtn, u_j|L_{max}$ problem of the bound for $P|prec, p_j = 1|L_{max}$ presented in [36]. It is a tighter bound than any known to the author. The results of this section and the next section were originally presented in [116] in collaboration with Yakov Zinder and Claire Hanen.

For the purposes of this section, let $i_0 = i^*$. If $i_0 = 0$, Lemma 4.22 demonstrates that the schedule $\eta^{\mathscr{GJ}}$ is optimal. Therefore, for the purposes of the following construction, assume that $i_0 > 0$. Define x_0 to be any task such that $x_0 \in H_{i^*-1}$ and, for all ρ^*-incomplete intervals $[t_i, t_{i+1}]$ for which $i < i^*$, $x_0 \notin H_i$. The existence of x_0 follows from Lemma 4.29. Starting with the pair (i_0, x_0), consider a sequence $(i_0, x_0), (i_1, x_1), \ldots$ generated as follows. Suppose task x_q has been defined and x_{q+1} has not. Let

$$\rho_q = \zeta_{x_q}\left(t_{i_q}, \eta^{\mathscr{GJ}}\right). \tag{5.9}$$

As a consequence of this, $\rho^* = \rho_0$. Tasks x_q will be defined so that

$$x_q \in A_{i_q-1} \cup A_{i_q}. \tag{5.10}$$

Note that (5.10) holds for $q = 0$.

By (5.9), (5.10) and Lemma 4.26, if $i_q > 0$ there exists some $g \in H_{i_q-1}(\rho_q)$ for which either $g = x_q$ or $g \to x_q$. Hence, Lemma 4.13 and Lemma 4.26 imply $H_i(\rho_q) \neq \emptyset$ for all $i < i_q$.

If, for all $i < i_q$, $|H_i(\rho_q)| > 1$, define $k = q$ and let (i_k, x_k) be the last pair in the considered sequence. Otherwise, let i_{q+1} be the largest among all i such that $i < i_q$ and $|H_i(\rho_q)| = 1$. Let x_{q+1} be the sole member of $H_{i_{q+1}}(\rho_q)$. This

construction process guarantees that (5.10) remains valid for all $0 \leq q \leq k$. Observe that, although the points of allocation t_i are numbered in increasing order of t_i, i.e. $t_0 < t_1 < \cdots$, the points t_{i_q} are numbered in decreasing order of t_{i_q}, that is $t_{i_0} > t_{i_1} > \cdots$.

In order to simplify notation, define $i_{k+1} = -1$. For the specified sequence of pairs $(i_0, x_0), (i_1, x_1), \ldots, (i_k, x_k)$, c_q is defined to be the total length of all ρ_q-complete intervals in schedule $\eta^{\mathscr{GJ}}$ during the time interval $[t_{i_{q+1}+1}, t_{i_q}]$. Likewise, w_q is defined to be the total length of all ρ_q-incomplete intervals in schedule $\eta^{\mathscr{GJ}}$ during the time interval $[t_{i_{q+1}+1}, t_{i_q}]$. For $1 \leq q \leq k$, let

$$z_q = t_{i_q+1} - t_{i_q}. \tag{5.11}$$

The complicated set of definitions above is required to examine the structure of $\eta^{\mathscr{GJ}}$ in much greater detail than the analysis of η in Subsection 5.1.1. In particular, intervals $[t_{i_q}, t_{i_q+1}]$ will undergo significant analysis in order to demonstrate that they make no net contribution to the bound (5.16) established in Theorem 5.14. In order to accomplish this, the following lemma establishes the completion times of some x_q – although sometimes $x_q = x_{q-1}$, if $i_{q-1} \neq i_q + 1$ it can be demonstrated that $x_q \neq x_{q-1}$, allowing the completion time to be established.

Lemma 5.7 *For all $1 \leq q \leq k$, if $i_{q-1} \neq i_q + 1$ then $C_{x_q}\left(\eta^{\mathscr{GJ}}\right) = t_{i_q+1}$.*

Proof. If $i_q \neq i_{q-1} - 1$ then, by the definition of i_q, $i_q + 2 \leq i_{q-1}$ and $|H_{i_q+1}(\rho_{q-1})| \geq 2$. By Corollary 4.28, all tasks $j \in H_{i_q+1}(\rho_{q-1})$ have either $x_q = j$ or $x_q \to j$. Since $H_{i_q+1}(\rho_{q-1}) \subseteq A_{i_q+1}$ and since A_{i_q+1} is a set of independent tasks, the former case is ruled out and thus $H_{i_q+1}(\rho_{q-1}) \subseteq K_{x_q}$. This implies that $C_{x_q}\left(\eta^{\mathscr{GJ}}\right) \leq t_{i_q+1}$. By Lemma 4.27, $x_q \in N_1^{i_q}$, so (4.7) implies $C_{x_q}\left(\eta^{\mathscr{GJ}}\right) \geq t_{i_q+1}$ and the lemma is proven. ■

Corollary 5.8 *For all $1 \leq q \leq k$, if $x_q \neq x_{q-1}$ then $C_{x_q}\left(\eta^{\mathscr{GJ}}\right) = t_{i_q+1}$.*

Proof. If $i_{q-1} \neq i_q + 1$, this follows directly from Lemma 5.7. If $i_{q-1} = i_q + 1$, the fact that $x_{q-1} \in H_{i_q+1}(\rho_{q-1})$ and Corollary 4.28 imply that $x_q \in N_1^{i_q}$. As a consequence of $x_q \neq x_{q-1}$ it follows that $C_{x_q}\left(\eta^{\mathscr{GJ}}\right) \leq t_{i_q+1}$, so (4.7) implies $C_{x_q}\left(\eta^{\mathscr{GJ}}\right) \geq t_{i_q+1}$ and the corollary follows. ■

Recall that, in the definition of the Garey-Johnson Algorithm, for each task j with $K_j \neq \emptyset$, two values a_j and b_j, as defined in (4.18), are used to bound the

priority ρ in (4.19) when calculating the value of μ_j. The lemma below is the first step towards the establishment of a relationship between successive values of ρ_q.

Lemma 5.9 *For all* $1 \leq q \leq k$, *if* $i_{q-1} \neq i_q + 1$,

$$a_{x_q} \leq \rho_{q-1} \leq b_{x_q}.$$

Proof. If $i_{q-1} \neq i_q + 1$, Lemma 5.7 implies $x_q \neq x_{x-1}$ so Corollary 4.28 implies $x_q \to x_{q-1}$. As a consequence of (4.15) and (5.9),

$$a_{x_q} = \min_{g \in K_{x_q}} \mu_g \leq \mu_{x_{q-1}} \leq \zeta_{x_{q-1}} \left(t_{i_{q-1}}, \eta^{\mathscr{GJ}} \right) = \rho_{q-1}$$

$$\leq \frac{p_{x_{q-1}}}{u_{x_{q-1}}} + \mu_{x_{q-1}} \leq \max_{g \in K_{x_q}} \left(\frac{p_g}{u_g} + \mu_g \right) = b_{x_q},$$

and the lemma is proven. ■

The lemma below examines the relationship between u_{min}, a parameter of the bound (5.16), and w_q, an important parameter used in the establishment of that bound.

Lemma 5.10 *For all* $0 \leq q \leq k$, *if* $m \leq 2u_{min}$, $w_q = 0$.

Proof. If $m \leq 2u_{min}$, $|H_i(\rho_q)| \geq 2$ implies

$$\sum_{j \in H_i(\rho_q)} u_j \geq m,$$

meaning $[t_i, t_{i+1}]$ is ρ_q-complete. By the definition of i_q, $|H_i(\rho_q)| \geq 2$ for all $i_{q+1} < i < i_q$ and consequently $w_q = 0$. ■

The lemma below further advances the understanding of the relationship between ρ_q's.

Lemma 5.11 *For all* $1 \leq q \leq k$, *if* $i_{q-1} \neq i_q + 1$,

$$\mu_{x_q} \geq \rho_{q-1} + c_{q-1} + \frac{2u_{min}}{m} w_{q-1}.$$

Proof. As a consequence of Corollary 4.28,

$$H_i(\rho_{q-1}) \subseteq (\{x_q\} \cup K_{x_q}) \qquad \text{for all} \qquad i_q < i < i_{q-1}. \tag{5.12}$$

If $i_{q-1} \neq i_q + 1$, Lemma 5.7 implies $x_q \notin H_i(\rho_{q-1})$ for any $i > i_q$ and thus $V \subseteq K_{x_q}$, where

$$V = \bigcup_{i=i_q+1}^{i_{q-1}-1} H_i(\rho_{q-1}).$$

For all $j \in V$, define

$$\iota(j) = 1 + \max\{i : i_q + 1 \leq i < i_{q-1} \wedge j \in H_i(\rho_{q-1})\},$$

i.e. $\iota(j)$ is the latest point of allocation in the currently considered interval $[i_q{+}1, i_{q-1}]$ that task j has a high priority. This definition implies, for all $j \in V$,

$$\zeta_j\left(t_{\iota(j)}, \eta^{\mathscr{GJ}}\right) = \frac{p_j\left(t_{\iota(j)}, \eta^{\mathscr{GJ}}\right)}{u_j} + \mu_j \geq \rho_{q-1};$$

rearranging gives

$$p_j\left(t_{\iota(j)}, \eta^{\mathscr{GJ}}\right) \geq u_j\left(\rho_{q-1} - \mu_j\right). \tag{5.13}$$

Define

$$\delta_i = \begin{cases} m & \text{if } [t_i, t_{i+1}] \text{ is } \rho_{q-1}\text{-complete} \\ 2u_{min} & \text{if } [t_i, t_{i+1}] \text{ is } \rho_{q-1}\text{-incomplete} \end{cases}$$

for all $i_q < i < i_{q-1}$. It follows from (4.18), (4.19), (5.12), (5.13) and Lemma 5.9, that

$$\begin{aligned} \mu_{x_q} &\geq \rho_{q-1} + \frac{1}{m}\sum_{j \in K_{x_q}} \beta_j(\rho_{q-1}) \geq \rho_{q-1} + \frac{1}{m}\sum_{j \in K_{x_q}} \left(p_j - u_j\left(\rho_{q-1} - \mu_j\right)\right) \\ &\geq \rho_{q-1} + \frac{1}{m}\sum_{j \in V} \left(p_j - p_j\left(t_{\iota(j)}, \eta^{\mathscr{GJ}}\right)\right) \\ &\geq \rho_{q-1} + \frac{1}{m}\sum_{i=i_q+1}^{i_{q-1}-1} \sum_{j \in H_i(\rho_{q-1})} \left(p_j\left(t_i, \eta^{\mathscr{GJ}}\right) - p_j\left(t_{i+1}, \eta^{\mathscr{GJ}}\right)\right) \\ &\geq \rho_{q-1} + \frac{1}{m}\sum_{i=i_q+1}^{i_{q-1}-1} \delta_i(t_{i+1} - t_i). \end{aligned}$$

If $m \geq 2u_{min} + 1$, this gives

$$\mu_{x_q} \geq \rho_{q-1} + \frac{1}{m}\sum_{i=i_q+1}^{i_{q-1}-1} \delta_i(t_{i+1} - t_i) = \rho_{q-1} + c_{q-1} + \frac{2u_{min}}{m} w_{q-1}.$$

If, on the other hand, $m \leq 2u_{min}$, Lemma 5.10 states that $w_{q-1} = 0$, and thus

$c_{q-1} = t_{i_{q-1}} - t_{i_q+1}$. Consequently, the preceding reasoning leads to

$$\mu_{x_q} \geq \rho_{q-1} + \frac{1}{m} \sum_{i=i_q+1}^{i_{q-1}-1} \delta_i (t_{i+1} - t_i) = \rho_{q-1} + t_{i_{q-1}} - t_{i_q+1} = \rho_{q-1} + c_{q-1} + \frac{2u_{min}}{m} w_{q-1}$$

and the lemma is proven. ■

The lemma below is the culmination of the preceding reasoning. It establishes the desired relationship between successive values of ρ_q. Specifically they are related in terms of m and u_{min}, parameters of the problem that appear in the performance guarantee (5.16), and z_q, c_{q-1} and w_{q-1}, which are used in Theorem 5.14 to derive this bound.

Lemma 5.12 *For all* $1 \leq q \leq k$,

$$\rho_q \geq z_q + c_{q-1} + \frac{2u_{min}}{m} w_{q-1} + \rho_{q-1}.$$

Proof. By Lemma 4.27, $x_q \in N_1^{i_q}$. Consequently, by (4.7), (5.9) and (5.11),

$$\rho_q = \frac{p_{x_q}\left(t_{i_q}, \eta^{\mathscr{GJ}}\right)}{u_{x_q}} + \mu_{x_q} = z_q + \frac{p_{x_q}\left(t_{i_q+1}, \eta^{\mathscr{GJ}}\right)}{u_{x_q}} + \mu_{x_q}. \tag{5.14}$$

If $x_q = x_{q-1}$, Lemma 5.7 implies $i_{q-1} = i_q + 1$ and thus $c_{q-1} = w_{q-1} = 0$. Hence, (5.14) implies

$$\rho_q = z_q + \frac{p_{x_q}\left(t_{i_q+1}, \eta^{\mathscr{GJ}}\right)}{u_{x_q}} + \mu_{x_q} = z_q + \rho_{q-1} = z_q + \rho_{q-1} + c_{q-1} + \frac{2u_{min}}{m} w_{q-1}.$$

Suppose, on the other hand, that $x_q \neq x_{q-1}$. In this case, Corollary 5.8 and Lemma 5.11 imply

$$\rho_q \geq z_q + \frac{p_{x_q}\left(t_{i_q+1}, \eta^{\mathscr{GJ}}\right)}{u_{x_q}} + \mu_{x_q} = z_q + \mu_{x_q} \geq z_q + \rho_{q-1} + c_{q-1} + \frac{2u_{min}}{m} w_{q-1}.$$

Consequently, the lemma is proven in both cases. ■

Having gained an understanding of the structure of schedule $\eta^{\mathscr{GJ}}$, it now remains to use this knowledge to derive the desired bound. The lemma below is analogous to Lemma 5.1 in Subsection 5.1.1 – it establishes a bound on the performance of the Garey-Johnson Algorithm, but not in the desired terms. It will be used to derive (5.16) in Theorem 5.14.

Lemma 5.13 *For any schedule $\eta^{\mathscr{GJ}}$ produced by the Garey-Johnson Algorithm and any optimal schedule ω,*

$$L_{max}\left(\eta^{\mathscr{GJ}}\right) - L_{max}\left(\omega\right) \leq \frac{m - 2u_{min}}{m} \sum_{k \geq q \geq 0} w_q.$$

Proof. By the definition of t_{i_0},

$$L_{max}\left(\eta^{\mathscr{GJ}}\right) = t_{i_0} + \rho_0 = c_k + w_k + \sum_{k > q \geq 0} \left(z_{q+1} + c_q + w_q\right) + \rho_0. \tag{5.15}$$

Lemma 4.14 and Lemma 5.12 together imply, similar to the proof of Lemma 5.11,

$$L_{max}(\omega) = G(\omega) \geq \rho_k + \frac{1}{m} \sum_{i=0}^{i_k - 1} \sum_{j \in H_i(\rho_k)} \left(p_j\left(t_i, \eta^{\mathscr{GJ}}\right) - p_j\left(t_{i+1}, \eta^{\mathscr{GJ}}\right)\right)$$

$$\geq c_k + \frac{2u_{min}}{m} w_k + \rho_k \geq c_k + \frac{2u_{min}}{m} w_k + \sum_{k > q \geq 0} \left(z_{q+1} + c_q + \frac{2u_{min}}{m} w_q\right) + \rho_0.$$

The bound in the statement of the lemma is obtained by substituting (5.15) into the inequality above. ■

The theorem below is very similar to Theorem 5.2 – the primary difference is in the consideration of the additional case where $m \leq 2u_{min}$.

Theorem 5.14 *For any schedule $\eta^{\mathscr{GJ}}$ produced by the Garey-Johnson Algorithm and any optimal schedule ω,*

$$L_{max}\left(\eta^{\mathscr{GJ}}\right) - L_{max}\left(\omega\right) \leq \begin{cases} 0 & \text{if } m \leq 2u_{min} \text{ or } l < 2r \\ \dfrac{m - 2u_{min}}{m}(l - r) & \text{otherwise} \end{cases} \tag{5.16}$$

for all combinations of $l \geq r > 0$ and $m \geq u_{min} \geq 1$.

Proof. If $m \leq 2u_{min}$, Lemma 5.10 implies

$$\sum_{k \geq q \geq 0} w_q = 0. \tag{5.17}$$

Hence, Lemma 5.13 gives $L_{max}\left(\eta^{\mathscr{GJ}}\right) - L_{max}\left(\omega\right) \leq 0$, and the theorem holds.

The consideration of the case where $l < 2r$ is analogous to the proof of Theorem 5.2 – as before $l < 2r$ implies there are no precedence constraints, which implies there are no ρ_0-incomplete intervals, and hence Lemma 5.13 implies that $L_{max}\left(\eta^{\mathscr{GJ}}\right) - L_{max}\left(\omega\right) \leq 0$ and the theorem holds.

Finally, assume that $m \geq 2u_{min} + 1$ and $l \geq 2r$. Notice that Corollary 4.28 and Lemma 4.29 imply the existence of a chain of tasks. Specifically, construct a chain $C = \{j_0, j_1, \cdots\}$ using the Garey-Johnson Chain Algorithm.

The Garey-Johnson Chain Algorithm

1. Initialise $e := 1$, $i := i_0 - 1$ and $q := 0$. Set $j_0 = x_0$.
2. If there exists an i' such that $1 \leq i' < i$ and $[t_{i'}, t_{i'+1}]$ is ρ_q-incomplete, update i to be the largest such i'. If no such i' exists, terminate the algorithm.
3. If $j_{e-1} \notin A_i$, select as j_e any task in $J_{j_{e-1}} \cap H_i(\rho_q)$ and increment e.
4. If $j_{e-1} = x_{q+1}$, increment q.
5. Go to Step 2.

Since Step 2 of the algorithm above always selects a smaller value of i, this algorithm always terminates. At Step 3, either q was incremented in Step 4 after the selection of task j_{e-1} or it was not. In the former case, $j_{e-1} = x_q$, and Lemma 4.26 demonstrates that there exists $g \in H_i(\rho_q)$ such that either $g = j_{e-1}$ or $g \to j_{e-1}$. On the other hand, if q was not incremented in Step 4 after the selection of task j_{e-1}, let $g = j_{e-1}$. In all cases, Corollary 4.28 implies that g, and thus j_{e-1}, has a predecessor in $H_i(\rho_q)$. Consequently, the selection of j_e in Step 3 of the above algorithm is valid.

The length of chain C provides a lower bound on l, and since each task $j \in C \setminus \{x_0\}$ was selected from $H_i(\rho_q)$ for some ρ_q-incomplete interval $[t_i, t_{i+1}]$, Lemma 4.27 implies that $j \in N_1^i$. Furthermore, since r is a lower bound on the length of x_0,

$$l \geq \sum_{j \in C} \frac{p_j}{u_j} = \frac{p_{x_0}}{u_{x_0}} + \sum_{j \in C \setminus \{x_0\}} \frac{p_j}{u_j} \geq r + \sum_{k \geq q \geq 0} w_q.$$

The theorem is proven by combining the above bound on $l - r$ with Lemma 5.13. ■

In the next section, the bound (5.16) established above will be demonstrated to be tight for all $m \geq u_{min} \geq 1$ and $l \geq r > 0$, a comprehensive demonstration of tightness not found in [105], which provided a similar, but weaker, bound for the Brucker-Garey-Johnson Algorithm.

The corollary below is a generalisation of the well known result that the Garey-Johnson Algorithm solves the $P2|prec, pmtn|L_{max}$ problem.

Corollary 5.15 *The Garey-Johnson Algorithm solves the $P2|prec, pmtn, u_j|L_{max}$ problem.*

The corollary below provides a ratio bound for the Garey-Johnson Algorithm that has a similar form to the bound provided by Corollary 5.3 for the Brucker-Garey-Johnson Algorithm.

Corollary 5.16 *For any schedule $\eta^{\mathscr{GJ}}$ produced by the Garey-Johnson Algorithm and any optimal schedule ω,*

$$L_{max}\left(\eta^{\mathscr{GJ}}\right) \leq \begin{cases} L_{max}\left(\omega\right) & \text{if } m \leq 2u_{min} \\ \left(2 - \dfrac{2u_{min}}{m}\right) L_{max}(\omega) + \left(1 - \dfrac{2u_{min}}{m}\right)(d_{max} - r) & \text{otherwise} \end{cases}.$$

Proof. As before, this corollary follows from the substitution of $L_{max}(\omega) \geq l - d_{max}$ into (5.16). ■

5.3 Dominance of Performance Guarantee

This section shows that no algorithm in family $\mathfrak{F} \cap \mathfrak{S}$ can have a better performance guarantee than (5.16). To make this statement more rigorous, the following definitions are required.

Let $\Phi(m, u_{min}, l, r)$ be the set of all instances of the problem $P|prec, pmtn, u_j|L_{max}$ with the same values of m, u_{min}, l and r. Observe that, for any combination of $l \geq r > 0$ and $m \geq u_{min} \geq 1$, $\Phi(m, u_{min}, l, r)$ is comprised of infinitely many such instances which differ from each other by the number of tasks, their processing times, permissible parallelisms, due dates and precedence constraints. For any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$, let $\Gamma^{\mathscr{A}}(m, u_{min}, l, r)$ be the least upper bound on the difference $L_{max}\left(\eta^{\mathscr{A}}\right) - L_{max}\left(\omega\right)$ over all instances in $\Phi(m, u_{min}, l, r)$. In other words, for all schedules $\eta^{\mathscr{A}}$, created by algorithm $\mathscr{A}$ for instances in $\Phi(m, u_{min}, l, r)$,

$$L_{max}\left(\eta^{\mathscr{A}}\right) - L_{max}\left(\omega\right) \leq \Gamma^{\mathscr{A}}(m, u_{min}, l, r),$$

and for any $\varepsilon > 0$, there exists an instance in $\Phi(m, u_{min}, l, r)$ for which

$$L_{max}\left(\eta^{\mathscr{A}}\right) - L_{max}\left(\omega\right) > \Gamma^{\mathscr{A}}(m, u_{min}, l, r) - \varepsilon.$$

It will be shown that, for any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ and any combination of $l \geq r > 0$ and $m \geq u_{min} \geq 1$, $\Gamma^{\mathscr{A}}(m, u_{min}, l, r)$ is not less than the right hand side of (5.16). Since the Garey-Johnson Algorithm is a member of family $\mathfrak{F} \cap \mathfrak{S}$, this result implies that (5.16) is tight for this algorithm.

This can be viewed as a vast improvement on the result given in [117], where it is proven that one already known algorithm within an infinite family of algorithms, specifically the nonpreemptive version of the Zinder-Roper Algorithm, has a better performance guarantee than all other algorithms in that family. What differs in [117] is that

- the problem considered in [117] is the $P|prec, p_j = 1|L_{max}$ problem;
- the bound examined in [117] is a ratio bound, rather than a difference bound;
- in [117] the value that the Zinder-Roper Algorithm minimises is a function only of the number of machines m, in contrast to the four parameters of $\Gamma^{\mathscr{A}}$; and
- the bound considered in [117] is only asymptotically tight.

The last point is the most important. A tight bound of the form considered in [117] is derived in [119], but includes a constant term that is not accounted for in [117]. Consequently, it may be the case that another algorithm has a better performance guarantee than the Zinder-Roper Algorithm by improving on this constant term. It might also be the case that the Zinder-Roper Algorithm has the best performance guarantee for some values of m, but not for others. In contrast to this, it is demonstrated in this section that the Garey-Johnson Algorithm has the best performance guarantee, of the form appearing in (5.16), for every combination of the four parameters $l \geq r > 0$ and $m \geq u_{min} \geq 1$. Hence, this section significantly advances the body of knowledge in this area of research.

Consider all instances of the $P|prec, pmtn, u_j|L_{max}$ problem with the same number of machines m. Recall that, by the definition of family $\mathfrak{F} \cap \mathfrak{S}$, all algorithms $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ determine the values of μ's by a stable function $\mu^{\mathscr{A}}$. Observe that for any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ and any two tasks g and q such that $K_g = K_q = \emptyset$,

$$\mu_g = \mu^{\mathscr{A}}(d_g, (m, \emptyset, \emptyset, \emptyset, \emptyset)) \qquad \text{and} \qquad \mu_q = \mu^{\mathscr{A}}(d_q, (m, \emptyset, \emptyset, \emptyset, \emptyset)). \tag{5.18}$$

This method of computing μ_g and μ_q remains the same regardless of whether this computation occurs in the construction of a schedule for one problem instance or in the construction of two schedules for two different problem instances. In other words, (5.18) does not depend on whether tasks g and q are in the same instance or in different instances. Furthermore, if for these two tasks, $d_g = d_q$, then $\mu_g = \mu_q$ regardless of whether this computation occurs in the construction of a schedule

for one problem instance or in the construction of two schedules for two different problem instances. This observation is particularly important to understand when considering the proofs in this section. Specifically, three different problem instances are considered, and schedules produced by an arbitrary algorithm from family $\mathfrak{F} \cap \mathfrak{S}$ are compared. It is demonstrated that it is impossible for any single algorithm from this family to improve on the performance guarantee (5.14) in all three cases.

Consider any specific values of m, u_{min}, l and r, such that $l \geq 2r$ and $m \geq 2u_{min} + 1$. For these values, consider a problem instance $\varphi^1 \in \Phi(m, u_{min}, l, r)$ which has no precedence constraints and has the parameters

$$N = \left\{0, \ldots, \left\lceil \frac{l}{r} \right\rceil m\right\},$$

$p_0 = l(m-1)$, $u_0 = m-1$ and for all $j \in \{1, \ldots, n-1\}$, $p_j = ru_{min}$, $u_j = u_{min}$, and $d_j = d_0 - l$. The due date d_0 can take any value.

Observe that, for φ^1,

- the longest chain is composed solely of task 0, and has a length of l;
- all tasks j have $u_j \geq u_{min}$ and u_{min} is achievable, for example $u_1 = u_{min}$; and
- all tasks j have $\frac{p_j}{u_j} \geq r$ and r is achievable, for example $\frac{p_1}{u_1} = r$.

As was the case with the proof of tightness for the Brucker-Garey-Johnson Algorithm, an upper bound on the optimal maximum lateness is compared with the maximum lateness produced by the algorithm under consideration. The lemma below considers the optimal maximum lateness.

Lemma 5.17 *For any optimal schedule ω for φ^1,*

$$L_{max}(\omega) \leq \left\lceil \frac{l}{r} \right\rceil ru_{min} + l - d_0$$

for any combination of $l \geq 2r > 0$ and $m > 2u_{min} \geq 2$.

Proof. Consider a schedule σ^1 that has all of tasks $j \in N \setminus \{0\}$ processed during the interval

$$\left[0, \left\lceil \frac{l}{r} \right\rceil ru_{min}\right].$$

That means that all m machines are busy during this interval, processing the tasks in $N \setminus \{0\}$. It is easy to see that this fragment of the considered schedule can be

obtained using the Modified McNaughton Algorithm. Suppose also that task 0 is processed in σ during the interval

$$\left[\left\lceil\frac{l}{r}\right\rceil ru_{min}, \left\lceil\frac{l}{r}\right\rceil ru_{min}+l\right].$$

The maximum lateness of this schedule provides an upper bound on the optimal schedule and so

$$L_{max}(\omega) \leq L_{max}\left(\sigma^1\right) = \left\lceil\frac{l}{r}\right\rceil ru_{min}+l-d_0,$$

and the lemma is proven. ■

In order to understand the behaviour of an arbitrary algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$, the lemma below addresses the manner in which the tasks are partitioned into N_1^0, N_2^0 and N_3^0.

Lemma 5.18 *When any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ is applied to φ^1, for any combination of $l \geq 2r > 0$ and $m > 2u_{min} \geq 2$,*

$$(N \setminus \{0\}) \subseteq N_2^0.$$

Proof. According to (4.3) and (4.4), for any $j \in N$, $\mu_j = \mu^{\mathscr{A}}(d_j, \varphi_j)$, where $\mu^{\mathscr{A}}$ is a function, specifying the algorithm $\mathscr{A}$. For the problem instance φ^1 and for all tasks $j \in N$, $\varphi_j = (m, \emptyset, \emptyset, \emptyset, \emptyset)$. Since all $j \in N \setminus \{0\}$ have equal d_j, they also have equal μ_j, i.e.

$$\mu_1 = \mu_2 = \cdots = \mu_{n-1}.$$

Consequently,

$$\sum_{j \in N \setminus \{0\}} u_j = \left\lceil\frac{l}{r}\right\rceil mu_{min} \geq 2m$$

implies $(N \setminus \{0\}) \subseteq N_2^0$. ■

Let T be the point at which the last of the tasks in $N \setminus \{0\}$ is completed in $\eta^{\mathscr{A}}$, i.e.

$$T = \max_{j \in N \setminus \{0\}} C_j\left(\eta^{\mathscr{A}}\right).$$

The lemma below examines the consequences of a large value of μ_0 relative to the values of the other μ's, as stated in (5.19). This will be expanded upon in Lemma 5.20, and it will then be demonstrated that these consequences are incompatible with a value of $\Gamma^{\mathscr{A}}(m, u_{min}, l, r)$ that improves upon the bound specified in (5.16).

Lemma 5.19 *For any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$, when $\mathscr{A}$ is applied to φ^1, if*

$$\mu_0 + \frac{2u_{min} - 1}{m - 1} l > \mu_1 = \mu_2 = \cdots = \mu_{n-1}, \tag{5.19}$$

then either $C_0\left(\eta^{\mathscr{A}}\right) < T$ or $N_2^i = N$ for some point of allocation t_i.

Proof. Recall that, when Algorithm P calculates the length Δ of the interval between the current point of allocation and the next, Δ is constrained by some (possibly strict) subset of the inequalities (4.10), (4.11), (4.12), (4.13) and (4.14). Three cases will be considered below, based on which set – N_1^0, N_2^0 or N_3^0 – contains task 0.

- Suppose $0 \in N_1^0$. Let U_1 and U_2 be as specified at the introduction of (4.10) – (4.14). In the considered case, Lemma 5.18 implies

  $$U_1 = u_0 = m - 1 \qquad \text{and} \qquad U_2 = \left|N_2^0\right| u_{min} = \left\lceil \frac{l}{r} \right\rceil m u_{min}.$$

  Consequently, taking into account (4.10),

  $$\frac{U_2}{m - U_1} \min_{j \in N_2^0} \frac{p_j\left(t_0, \eta^{\mathscr{A}}\right)}{u_j} = U_2 r = \left\lceil \frac{l}{r} \right\rceil m r u_{min} > \left\lceil \frac{l}{r} \right\rceil r u_{min}$$

  $$\geq l u_{min} \geq l = \min_{j \in N_1^0} \frac{p_j\left(t_0, \eta^{\mathscr{A}}\right)}{u_j} \geq \Delta,$$

  where $\Delta = t_1 - t_0$. So, when Algorithm P computes Δ, (4.11) is not an equality, i.e.

  $$\min_{j \in N_2} \frac{p_j\left(t_i, \eta^{\mathscr{A}}\right)}{u_j} > \frac{m - U_1}{U_2} \Delta. \tag{5.20}$$

  In words, this means that no task in N_2^0, i.e. no task in $N \setminus \{0\}$, can be completed at or before time t_1 because the completion of a task in N_1^0, in this case task 0, constrains t_1 to be too small.

  According to the description of Algorithm P, at least one of (4.10) – (4.14) must be an equality. Since $N_3^0 = \emptyset$, (4.13) and (4.14), the conditions that a task in N_1^0 or N_2^0 respectively become equal in priority to a task in N_3^0, cannot be considered. Consequently, the fact that (4.11) is not an equality implies that either (4.10) or (4.12) is an equality – respectively, that either a task in N_1^0 (in this case task 0) is completed, or a task in N_1^0 (again, this must be task 0) becomes equal in priority to the common priority of the tasks in N_2^0. These two sub-cases are considered below.

– Recall that the value of T must be at least equal to the total processing of all tasks that came before it divided by m, i.e.

$$T \geq \frac{1}{m} \sum_{j \in N} \left(p_j - p_j \left(T, \eta^{\mathscr{A}} \right) \right) = \frac{1}{m} \left(p_0 - p_0 \left(T, \eta^{\mathscr{A}} \right) \right) + \left\lceil \frac{l}{r} \right\rceil r u_{min}. \quad (5.21)$$

Consequently, if (4.10) is an equality,

$$C_0 \left(\eta^{\mathscr{A}} \right) = t_1 = l \leq \left\lceil \frac{l}{r} \right\rceil r u_{min} < T.$$

– If (4.10) is not an equality, then

$$p_0 \left(t_1, \eta^{\mathscr{A}} \right) \geq \frac{p_0 \left(t_1, \eta^{\mathscr{A}} \right)}{u_0} > 0$$

and hence task 0 is available for processing at time t_1. Furthermore, (4.12) is an equality, i.e. the common priority of the tasks in N_2^0 (in this case the tasks constituting $N \setminus \{0\}$) is equal to the priority of a task in N_1^0 (which in this case is task 0). Consequently $p_0 \left(T, \eta^{\mathscr{A}} \right) > 0$ and, for all $j \in N \setminus \{0\}$,

$$\frac{p_j \left(t_1, \eta^{\mathscr{A}} \right)}{u_j} + \mu_j = \frac{p_0 \left(t_1, \eta^{\mathscr{A}} \right)}{u_0} + \mu_0. \quad (5.22)$$

This implies $N_2^1 = N$.

- If $0 \in N_2^0$, then $N_2^0 = N$.

- Suppose that $0 \in N_3^0$. The inequality $m \geq 2u_{min} + 1$ and (5.19) together imply

$$\mu_1 < \mu_0 + \frac{2u_{min} - 1}{m - 1} l \leq \mu_0 + \frac{m - 2}{m - 1} l < \mu_0 + l.$$

Hence, $\mu_1 - l - \mu_0$ is negative. Then, applying (4.14),

$$\min_{j \in N_2^0} \frac{p_j \left(t_0, \eta^{\mathscr{A}} \right)}{u_j} = \frac{p_1 \left(t_0, \eta^{\mathscr{A}} \right)}{u_1} > \frac{p_1 \left(t_0, \eta^{\mathscr{A}} \right)}{u_1} + \mu_1 - l - \mu_0$$

$$= \frac{p_1 \left(t_0, \eta^{\mathscr{A}} \right)}{u_1} + \mu_1 - \max_{j \in N_3^0} \left\{ \frac{p_j \left(t_0, \eta^{\mathscr{A}} \right)}{u_j} + \mu_j \right\} \geq \frac{m - U_1}{U_2} \Delta.$$

Consequently, when Algorithm P computes Δ, (4.11) is not an equality, i.e. (5.20) holds in this case as well.

According to the description of Algorithm P, at least one of (4.10) – (4.14) must be an equality. Since $N_1^0 = \emptyset$, (4.10), (4.12) and (4.13), respectively the conditions that a task in N_1^0 is complete, that a task in N_1^0 becomes equal in priority to a task in N_2^0 and that a task in N_1^0 becomes equal in priority to a task in N_3^0, cannot be considered. Consequently, the fact that (4.11) is not an equality implies that (4.14) is an equality, i.e. that the common priority of the tasks in N_2^0 (in this case the tasks constituting $N \setminus \{0\}$) is equal to the priority of a task in N_3^0 (which in in this case is task 0). In other words, (5.22) holds for all tasks $j \in N \setminus \{0\}$. As before, this and the inequality $p_0\left(T, \eta^{\mathscr{A}}\right) > 0$ imply $N_2^1 = N$.

Hence, the lemma holds in all cases. ■

Lemma 5.20 *For any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$, when $\mathscr{A}$ is applied to φ^1, if (5.19) holds, then either*

$$\frac{p_0\left(T, \eta^{\mathscr{A}}\right)}{m-1} + \mu_0 = \mu_1 = \mu_2 = \cdots = \mu_{n-1} \tag{5.23}$$

or $C_0\left(\eta^{\mathscr{A}}\right) < T$.

Proof. As has been shown by Lemma 5.19, at least one of two conditions must hold: either $C_0\left(\eta^{\mathscr{A}}\right) < T$ or $N_2^i = N$ for some point of allocation t_i. If the latter condition holds, when determining the point of allocation t_{i+1}, the fact that $N_1^i = N_3^i = \emptyset$ implies that (4.11) is an equality, i.e. a task in N_2^0 is completed.

Since all tasks $j \in N \setminus \{0\}$ have the same p_j, u_j and μ_j, (4.7), (4.8) and (4.9) imply they receive the same processing time between any two points of allocation. Since (4.11) is an equality, this indicates that one of the tasks in N_2^i is completed in the time interval $[t_i, t_{i+1}]$. Given the above, the completed tasks form one of three possible sets:

- all tasks in $N \setminus \{0\}$, in which case $T = t_{i+1} < C_0\left(\eta^{\mathscr{A}}\right)$;
- the task 0, in which case $C_0\left(\eta^{\mathscr{A}}\right) \leq t_{i+1} < T$; or
- all tasks in N, in which case $C_0\left(\eta^{\mathscr{A}}\right) \leq t_{i+1} = T$.

If $T = t_{i+1}$, then $p_j\left(T, \eta^{\mathscr{A}}\right) = 0$ for all $j \in N \setminus \{0\}$. Therefore, by the definition of N_2,

$$\mu_1 = \zeta_1\left(T, \eta^{\mathscr{A}}\right) = \zeta_0\left(T, \eta^{\mathscr{A}}\right) = \frac{p_1\left(T, \eta^{\mathscr{A}}\right)}{m-1} + \mu_0,$$

so (5.23) holds. Since $t_{i+1} < T$ implies $C_0\left(\eta^{\mathscr{A}}\right) < T$, either (5.23) holds or $C_0\left(\eta^{\mathscr{A}}\right) < T$, or both. ■

The following lemma uses all of the preceding results to establish limits on the μ's an algorithm $\mathscr{A}$ can create if $\Gamma^{\mathscr{A}}(m, u_{min}, l, r)$ improves upon the bound specified in (5.16). As stated earlier, the definition of family $\mathfrak{F} \cap \mathfrak{S}$ means that the μ's produced depend only on the each task's due date, it's successors, and the number of machines – the rest of the problem instance does not affect these values. This property is used in the subsequent Corollary 5.22 and Corollary 5.23.

Lemma 5.21 *For any $l \geq 2r > 0$, $m > 2u_{min} \geq 2$, and any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ for which*

$$\Gamma^{\mathscr{A}}(m, u_{min}, l, r) < \frac{m - 2u_{min}}{m}(l - r), \tag{5.24}$$

when $\mathscr{A}$ is applied to φ^1, the μ's generated by this algorithm are subject to

$$\mu_0 + \frac{2u_{min} - 1}{m - 1}l \leq \mu_1 = \mu_2 = \cdots = \mu_{n-1}.$$

Proof. Assume that this is not the case, i.e. assume that (5.19) holds. In this case, Lemma 5.20 demonstrates that either (5.23) holds, or $C_0\left(\eta^{\mathscr{A}}\right) < T$.

Suppose (5.23) holds. Therefore, taking into account (5.19) and (5.21),

$$\begin{aligned}
L_{max}\left(\eta^{\mathscr{A}}\right) &\geq T - (d_0 - l) \\
&\geq \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{1}{m}\left(l(m-1) - p_0\left(T, \eta^{\mathscr{A}}\right)\right) - d_0 + l \\
&= \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{m-1}{m}\left(l - \mu_1 + \mu_0\right) - d_0 + l \\
&> \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{m-1}{m}\left(l - \frac{2u_{min} - 1}{m-1}l\right) - d_0 + l \\
&= \left\lceil \frac{l}{r} \right\rceil r u_{min} + l - d_0 + \frac{m - 2u_{min}}{m}l.
\end{aligned}$$

Consequently, Lemma 5.17 implies

$$L_{max}\left(\eta^{\mathscr{A}}\right) - L_{max}(\omega) > \frac{m - 2u_{min}}{m}l > \frac{m - 2u_{min}}{m}(l - r), \tag{5.25}$$

which contradicts (5.24).

On the other hand, if $C_0\left(\eta^{\mathscr{A}}\right) < T$, then

$$L_{max}\left(\eta^{\mathscr{A}}\right) \geq T - (d_0 - l) \geq \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{m-1}{m} l - d_0 + l$$

$$> \left\lceil \frac{l}{r} \right\rceil r u_{min} + l - d_0 + \frac{m - 2u_{min}}{m} l,$$

which, as had been shown above, implies (5.25), which in turn contradicts (5.24). ■

Corollary 5.22 *Consider any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ and any specific values of m, u_{min}, l and r, such that $l \geq 2r > 0$, $m > 2u_{min} \geq 2$ and (5.24) holds. Then, for any two tasks g and q such that $K_g = K_q = \emptyset$ and $d_g - d_q = l$, the corresponding μ's computed by the algorithm $\mathscr{A}$, satisfy*

$$\mu_q \geq \mu_g + \frac{2u_{min} - 1}{m - 1} l. \tag{5.26}$$

Proof. This follows due to the definition of family $\mathfrak{F} \cap \mathfrak{S}$. Consider two tasks, not necessarily in the same problem instance, g and q such that $K_g = K_q = \emptyset$ and $d_g - d_q = l$. Lemma 5.21 demonstrates that tasks 0 and 1 of φ^1 obey $d_0 - d_1 = l$ and

$$\mu_1 \geq \mu_0 + \frac{2u_{min} - 1}{m - 1} l.$$

Since all algorithms $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ determine the μ's used by Algorithm P by a function $\mu^{\mathscr{A}}$, for any two tasks where the arguments of this function are identical for both tasks, these tasks will receive the same value of μ. Again, this does not even require the tasks to be in the same problem instance. Consequently, $\mu_g = \mu_0$ and $\mu_q = \mu_1$ and the corollary is proven. ■

Corollary 5.23 *Consider any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ and some values of m, u_{min}, l and r, such that $l \geq 2r > 0$, $m > 2u_{min} \geq 2$ and (5.24) holds. Then, for any positive integer κ and any two tasks g and q such that $K_g = K_q = \emptyset$ and $d_g - d_q = \kappa l$, the corresponding μ's computed by the algorithm $\mathscr{A}$, satisfy*

$$\mu_q \geq \mu_g + \kappa \frac{2u_{min} - 1}{m - 1} l. \tag{5.27}$$

Note that Corollary 5.22 and Corollary 5.23 establish a relation between due dates and μ's for any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ for which (5.24) holds, and for a particular quartet of values m, u_{min}, l and r, regardless of the behaviour of the algorithm $\mathscr{A}$

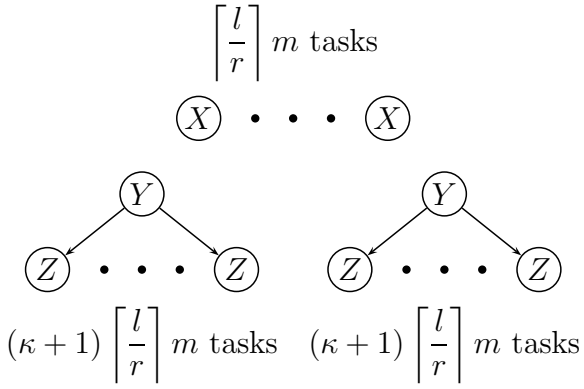


Figure 5.2: Precedence graph of φ^2.

for any other values of these parameters. This concludes the examination of the problem instance φ^1.

For any combination of $l \geq 2r$ and $m \geq 2u_{min} + 1$, consider a problem instance $\varphi^2 \in \Phi(m, u_{min}, l, r)$ where the tasks form three sets: X, Y and Z. The set X is comprised of $\left\lceil \frac{l}{r} \right\rceil m$ identical tasks. The set Y is comprised of two identical tasks. The set Z is comprised of $2(\kappa+1)\left\lceil \frac{l}{r} \right\rceil m$ identical tasks, where

$$\kappa = \left\lceil \frac{m-1}{2u_{min}-1} \right\rceil . \tag{5.28}$$

Each task in Y precedes $(\kappa+1)\left\lceil \frac{l}{r} \right\rceil m$ tasks in Z and none of the tasks in Z is preceded by both tasks in Y. No other precedence constraints exist between the tasks in addition to those already mentioned. Figure 5.2 depicts the partially ordered set of tasks in φ^2. In this figure, each task is labelled with the name of the set to which it belongs.

The parameters of the tasks are

- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = r$ for all $j \in X$;
- $p_j = (l-r)u_{min}$, $u_j = u_{min}$ and $d_j = (\kappa+1)l$ for both $j \in Y$; and
- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = (\kappa+1)\left(\left\lceil \frac{l}{r} \right\rceil ru_{min} + l\right)$ for all $j \in Z$.

Observe that

- either $j \in Y$ and any of its successors $g \in Z$ form a chain of maximal length,

with a length of

$$\frac{p_j}{u_j} + \frac{p_g}{u_g} = l;$$

- $u_j = u_{min}$ for all $j \in X \cup Y \cup Z$; and
- $\frac{p_j}{u_j} = r$ for all $j \in X \cup Z$ and $\frac{p_j}{u_j} = l - r \geq r$ for both $j \in Y$.

As with the examination of φ^1, a bound on the optimal maximum lateness is established in the lemma below.

Lemma 5.24 *For any optimal schedule ω for φ^2,*

$$L_{max}(\omega) \leq (\kappa + 2) \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l - r) - (\kappa + 1)l$$

for any combination of $l \geq 2r > 0$ and $m > 2u_{min} \geq 2$.

Proof. Consider a schedule σ^2 defined as follows. Since, for all $j \in X \cup Y$,

$$\frac{p_j}{u_j} \leq l \leq \left\lceil \frac{l}{r} \right\rceil r < \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l - r),$$

use the Modified McNaughton Algorithm to schedule all $j \in X \cup Y$ to be processed to completion during

$$\left[0, \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l - r)\right],$$

giving

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j \leq \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l - r) - r \qquad \text{for all } j \in X \tag{5.29}$$

and

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j \leq \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l - r) - (\kappa + 1)l \qquad \text{for both } j \in Y. \tag{5.30}$$

Similarly, for any tasks $j \in Z$,

$$\frac{p_j}{u_j} \leq l \leq \left\lceil \frac{l}{r} \right\rceil r < (2\kappa + 2) \left\lceil \frac{l}{r} \right\rceil r u_{min}.$$

Therefore, use the Modified McNaughton Algorithm to schedule all tasks in Z to be processed to completion during

$$\left[\left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l - r), (2\kappa + 3) \left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l - r)\right]$$

giving

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j \leq (\kappa+2)\left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l-r) - (\kappa+1)l \qquad \text{for all } j \in Z. \quad (5.31)$$

Combining (5.29), (5.30) and (5.31), along with the fact that

$$\left\lceil \frac{l}{r} \right\rceil r u_{min} \geq l > r,$$

gives

$$L_{max}(\omega) \leq L_{max}\left(\sigma^2\right) \leq (\kappa+2)\left\lceil \frac{l}{r} \right\rceil r u_{min} + \frac{2u_{min}}{m}(l-r) - (\kappa+1)l,$$

and the lemma is proven. ■

Let j and g be the tasks constituting the set Y. Observe that all tasks in Z are identical in terms of their required processing times, the permissible parallelism in their processing, and their due dates. Therefore, the tuple φ_j describing the partially ordered set of the successors of j and the tuple φ_g describing the partially ordered set of the successors of g differ only in the names of the successors of j and g. Hence, taking into account the fact that $d_j = d_g$, and the stability property of the function $\mu^{\mathscr{A}}$ which specifies any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$, $\mu_j = \mu_g$. In what follows, the common value of μ_j and μ_g for algorithm $\mathscr{A}$ will be denoted α. Similarly, since all tasks in X are identical in terms of their required processing times, the permissible parallelism in their processing, and their due dates, all tasks in X have the same μ's. In what follows, the common value of all these μ's for algorithm $\mathscr{A}$ will be denoted β. The following lemma establishes a relation (5.32) between the priorities α and β that any algorithm must have if $\Gamma^{\mathscr{A}}(m, u_{min}, l, r)$ improves upon the bound specified in (5.16). It will later be demonstrated that (5.32) constrains the algorithm to produce a schedule for a third problem instance, defined later, with a maximum lateness that deviates greatly from the optimal.

Lemma 5.25 *Consider any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ and any specific values of m, u_{min}, l and r, such that $l \geq 2r > 0$, $m > 2u_{min} \geq 2$ and (5.24) holds. When $\mathscr{A}$ is applied to φ^2, the μ's generated by this algorithm are subject to*

$$l - r + \alpha > \beta. \quad (5.32)$$

Proof. Suppose that the lemma is not true, i.e. suppose $l - r + \alpha \leq \beta$. This

implies that, for all $g \in X$ and both $q \in Y$,

$$\zeta_q\left(0, \eta^{\mathscr{A}}\right) = l - r + \mu_q \leq \mu_g = \zeta_g\left(0, \eta^{\mathscr{A}}\right) - r. \tag{5.33}$$

Consequently, since the tasks in Z are not available at time $t_0 = 0$ and since

$$\sum_{j \in X} u_j = \left\lceil \frac{l}{r} \right\rceil m u_{min} > m$$

$N_1^0 = \emptyset$, $N_2^0 = X$ and $N_3^0 = Y$. Furthermore $N_1^0 = \emptyset$ means that (4.10), (4.12), and (4.13) are not considered when determining the second point of allocation t_1. An additional consequence of (5.33) is that, from (4.11),

$$\Delta \leq \frac{U_2}{m - U_1} \min_{j \in N_2^0} \frac{p_j\left(t_i, \eta^{\mathscr{A}}\right)}{u_j} = \frac{U_2}{m - U_1} r \leq \frac{U_2}{m - U_1}\left(\zeta_g\left(0, \eta^{\mathscr{A}}\right) - \zeta_q\left(0, \eta^{\mathscr{A}}\right)\right).$$

This implies that (4.14) can only be an equality if (4.11) is also an equality, i.e. all tasks in X are processed to completion during

$$\left[0, \left\lceil \frac{l}{r} \right\rceil r u_{min}\right].$$

This gives

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j \leq \left\lceil \frac{l}{r} \right\rceil r u_{min} - r \qquad \text{for all } j \in X, \tag{5.34}$$

and this is an equality for at least one task.

At time t_1, only the tasks in Y are available, i.e. $A_1 = Y$. Consequently, the next point of allocation t_2 is determined by (4.10), meaning both tasks in Y are processed concurrently and completed during the interval

$$\left[\left\lceil \frac{l}{r} \right\rceil r u_{min}, \left\lceil \frac{l}{r} \right\rceil r u_{min} + l - r\right],$$

giving

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j = \left\lceil \frac{l}{r} \right\rceil r u_{min} - \kappa l - r \qquad \text{for both } j \in Y. \tag{5.35}$$

Finally, all tasks in Z are processed to completion during the interval

$$\left[\left\lceil \frac{l}{r} \right\rceil r u_{min} + l - r, (2\kappa + 3)\left\lceil \frac{l}{r} \right\rceil r u_{min} + l - r\right],$$

giving

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j = (\kappa + 2)\left\lceil \frac{l}{r} \right\rceil r u_{min} - \kappa l - r \qquad \text{for all } j \in Z. \tag{5.36}$$

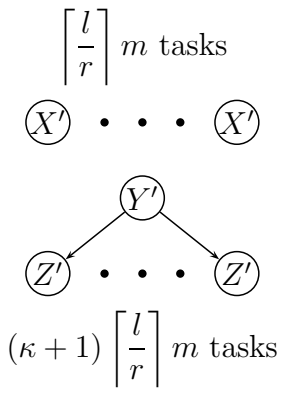


Figure 5.3: Precedence graph of φ^3.

As in the case of (5.34), this is an equality for at least one task.

Combining (5.34), (5.35) and (5.36), by (5.28) the maximum lateness of $\eta^{\mathscr{A}}$ is

$$L_{max}\left(\eta^{\mathscr{A}}\right) = (\kappa+2)\left\lceil \frac{l}{r} \right\rceil ru_{min} - \kappa l - r.$$

As a consequence of Lemma 5.24,

$$L_{max}\left(\eta^{\mathscr{A}}\right) - L_{max}(\omega) \geq \frac{m-2u_{min}}{m}(l-r),$$

contradicting (5.24). ■

This concludes the consideration of the problem instance φ^2.

For any combination of $l \geq 2r$ and $m \geq 2u_{min}+1$, consider another problem instance $\varphi^3 \in \Phi(m, u_{min}, l, r)$ where the tasks form three sets: X', Y' and Z'. The set X' is comprised of $\left\lceil \frac{l}{r} \right\rceil m$ identical tasks. The set Y' is comprised of only one task, and the set Z' is comprised of $(\kappa+1)\left\lceil \frac{l}{r} \right\rceil m$ identical tasks, where κ is specified by (5.28). The sole task in Y' precedes all tasks in Z'. No other precedence constraints exist between the tasks. Observe that the precedence constraints of φ^3 can be obtained from those of φ^2 by eliminating one of the tasks in Y together with all its successors. Figure 5.3 depicts the partially ordered set of tasks in the considered instance of $P|prec, pmtn, u_j|L_{max}$. In this figure, each task is labelled with the name of the set to which it belongs.

The parameters of the tasks are

- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = \kappa l + r$ for all $j \in X'$;
- $p_j = (l - r)m$, $u_j = m$ and $d_j = (\kappa + 1)l$ for $j \in Y'$; and
- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = (\kappa + 1)\left(\left\lceil \frac{l}{r} \right\rceil ru_{min} + l\right)$ for all $j \in Z'$.

Note that the only changes to the parameters above from those of φ^2 is that the tasks in X' have a different due date than those in X and the task in Y' has a different processing time and permissible parallelism than the two tasks in Y.

Observe that

- the longest chain is composed of the task constituting Y' and any task in Z', and has a length of l;
- all tasks j have $u_j \geq u_{min}$ with $u_j = u_{min}$ for any $j \in X' \cup Z'$; and
- all tasks j have $\frac{p_j}{u_j} \geq r$, with $\frac{p_j}{u_j} = r$ for all $j \in X' \cup Z'$.

Since all tasks in X' have the same due dates and no successors, they have the same μ's. For algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$, let γ be the common value of all these μ's, i.e. $\gamma = \mu_j$ for any $j \in X'$. Since the due date and the successors of the tasks in Y are the same as those of the task in Y', the μ of the task in Y' is equal to α. This allows a constraint to be placed on the relationship between α and γ in the lemma below. It will be combined with Corollary 5.23 and Lemma 5.25 to demonstrate that, if an algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ produces good schedules for instances φ^1 and φ^2, it will produce a bad schedule for φ^3.

Lemma 5.26 *Consider any algorithm $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ and any specific values of m, u_{min}, l and r, such that $l \geq 2r > 0$, $m > 2u_{min} \geq 2$ and (5.24) holds. When $\mathscr{A}$ is applied to φ^3, the μ's generated by this algorithm are subject to*

$$\alpha > r + \gamma.$$

Proof. Let q be any task in X and g be any task in X'. Since $d_g - d_q = \kappa l$, according to Corollary 5.23,

$$\mu_q \geq \mu_g + \kappa \frac{2u_{min} - 1}{m - 1} l.$$

Therefore, applying (5.28) and Lemma 5.25,

$$l - r + \alpha > \beta \geq \gamma + \kappa \frac{2u_{min} - 1}{m - 1} l \geq l + \gamma$$

and the lemma holds. ■

As with Lemma 5.17 and Lemma 5.24, the lemma below establishes an upper bound on the optimal maximum lateness for φ^3. Along with Lemma 5.26, this will be used in Theorem 5.28 to establish the main result of this section.

Lemma 5.27 *For any optimal schedule ω for φ^3,*

$$L_{max}(\omega) \leq \left\lceil \frac{l}{r} \right\rceil ru_{min} - \kappa l - r$$

for any combination of $l \geq 2r > 0$ and $m > 2u_{min} \geq 2$.

Proof. Consider a schedule σ^3 defined as follows. Use the Modified McNaughton Algorithm to process all tasks in X' to completion during the interval

$$\left[0, \left\lceil \frac{l}{r} \right\rceil ru_{min}\right],$$

giving

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j \leq \left\lceil \frac{l}{r} \right\rceil ru_{min} - \kappa l - r \qquad \text{for all } j \in X'. \tag{5.37}$$

This is tight for at least one task.

Process the task comprising Y' to completion during the interval

$$\left[\left\lceil \frac{l}{r} \right\rceil ru_{min}, \left\lceil \frac{l}{r} \right\rceil ru_{min} + l - r\right],$$

giving

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j = \left\lceil \frac{l}{r} \right\rceil ru_{min} - \kappa l - r \qquad \text{for the sole task } j \in Y'. \tag{5.38}$$

Note that the right hand side of the equation above is the same as that of (5.37).

Use the Modified McNaughton Algorithm to process all tasks in Z' to completion during the interval

$$\left[\left\lceil \frac{l}{r} \right\rceil ru_{min} + l - r, (\kappa + 2)\left\lceil \frac{l}{r} \right\rceil ru_{min} + l - r\right],$$

giving

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j \leq \left\lceil \frac{l}{r} \right\rceil ru_{min} - \kappa l - r \qquad \text{for all } j \in Z'. \tag{5.39}$$

Again, the right hand side of the equation above matches that of (5.37) and (5.38).

Together, (5.37), (5.38) and (5.39) give

$$L_{max}(\omega) \leq L_{max}\left(\sigma^3\right) = \left\lceil \frac{l}{r} \right\rceil r u_{min} - \kappa l - r,$$

and the lemma is proven. ■

Theorem 5.28 *For any algorithm* $\mathscr{A} \in \mathfrak{F} \cap \mathfrak{S}$ *and any* $l \geq r > 0$ *and* $m \geq u_{min} \geq 1$,

$$\Gamma^{\mathscr{A}}(m, u_{min}, l, r) \geq \begin{cases} 0 & \text{for } m \leq 2u_{min} \text{ or } l < 2r \\ \dfrac{m - 2u_{min}}{m}(l - r) & \text{otherwise} \end{cases}.$$

Proof. By definition, $\Gamma^{\mathscr{A}}(m, u_{min}, l, r) \geq 0$. Therefore, the theorem holds for $m \leq 2u_{min}$ and $l < 2r$. Hence, in what follows, it is assumed that $l \geq 2r$ and $m \geq 2u_{min} + 1$. Consider any specific values of m, u_{min}, l and r, and suppose that the theorem is not true for those values, i.e. suppose (5.24) holds.

Consider two tasks, $g \in Y$ and $g' \in Y'$. For any $h \in Z$ and any $h' \in Z'$, $p_h = p_{h'}$, $u_h = u_{h'}$ and $d_h = d_{h'}$. The tasks in Z and the tasks in Z' have no successors in their respective instances of $P|prec, pmtn, u_j|L_{max}$. Furthermore, g and g' have the same number of successors in the set Z and Z', respectively. Therefore, there exists a one-to-one mapping between tasks in K_j and $K_{j'}$ which preserves the precedence constraints, the tasks' processing times, the permissible parallelism in tasks' processing, and the tasks' due dates. Then, taking into account that $d_g = d_{g'}$ and that the function $\mu^{\mathscr{A}}$ is stable, $\mu_g = \mu_{g'}$.

Since $\mu_g = \alpha$, Lemma 5.26 implies that, for all $q \in X'$

$$\zeta_q\left(0, \eta^{\mathscr{A}}\right) = r + \mu_q < \mu_{g'} = \zeta_{g'}\left(0, \eta^{\mathscr{A}}\right) - (l - r). \tag{5.40}$$

Consequently, since the tasks in Z are not available at time $t_0 = 0$ and since

$$\sum_{j \in Y'} u_j = m,$$

$N_1^0 = Y'$, $N_2^0 = \emptyset$ and $N_3^0 = X'$. Furthermore, $N_2^0 = \emptyset$ means that (4.11), (4.12), and (4.14) are not considered when determining the second point of allocation t_1. An additional consequence of (5.40) is that, from (4.10),

$$\Delta \leq \min_{j \in N_1^0} \frac{p_j\left(t_i, \eta^{\mathscr{A}}\right)}{u_j} = l - r < \zeta_{g'}\left(0, \eta^{\mathscr{A}}\right) - \zeta_q\left(0, \eta^{\mathscr{A}}\right).$$

This implies that (4.12) cannot be an equality and hence (4.10) is an equality, i.e.

task g' is processed to completion during

$$[0, l - r]\,.$$

This gives

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j = -\kappa l - r \qquad \text{for the sole task } j \in Y'. \tag{5.41}$$

At time $t_1 = l - r$, all tasks from sets X' and Z' are available, i.e. $A_1 = X' \cup Z'$. This implies that there is at least one task from X' still being processed at time

$$l - r + \left\lceil \frac{l}{r} \right\rceil r u_{min}$$

and consequently

$$C_j\left(\eta^{\mathscr{A}}\right) - d_j \geq \left\lceil \frac{l}{r} \right\rceil r u_{min} - (\kappa - 1)l - 2r \qquad \text{for at least one } j \in X. \tag{5.42}$$

Combining (5.41) and (5.42) gives

$$L_{max}\left(\eta^{\mathscr{A}}\right) \geq \left\lceil \frac{l}{r} \right\rceil r u_{min} - (\kappa - 1)l - 2r.$$

With Lemma 5.27, the above inequality gives

$$L_{max}\left(\eta^{\mathscr{A}}\right) - L_{max}(\omega) \geq l - r > \frac{m - 2u_{min}}{m}(l - r),$$

contradicting (5.24). For any $l \geq 2r > 0$ and $m > 2u_{min} \geq 2$, the negation of (5.24) gives the bound in the statement of the theorem, and so the theorem holds. ■

Corollary 5.29

$$\Gamma^{\mathscr{GJ}}(m, u_{min}, l, r) = \begin{cases} 0 & \text{if } m \leq 2u_{min} \text{ or } l < 2r \\ \dfrac{m - 2u_{min}}{m}(l - r) & \text{otherwise} \end{cases}$$

for all $l \geq r > 0$ *and* $m \geq u_{min} \geq 1$.

Corollary 5.30 *The three machine problem* $P3|prec, pmtn, u_j|L_{max}$ *cannot be solved by any algorithm in* $\mathfrak{F} \cap \mathfrak{S}$.

The above results cannot be directly applied to the $P|prec, pmtn|L_{max}$ problem, that is the problem where all $u_j = 1$, because it might be supposed that the flexibility of arbitrary u_j is required to prove this kind of result. However, the proofs above can

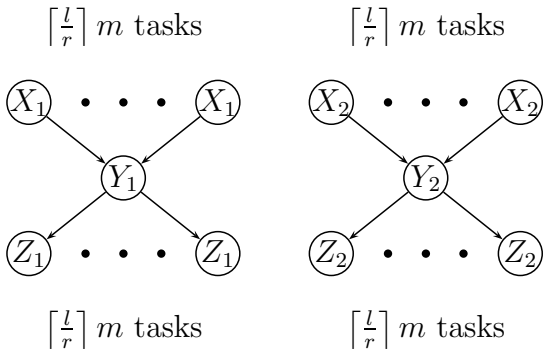


Figure 5.4: Precedence graph of φ^S.

be easily adapted by setting $u_{min} = 1$ and replacing the task task in Y' of problem instance φ^3 with m tasks, each of length $l - r$. This demonstrates that the much studied $P3|prec, pmtn|L_{max}$ problem cannot be solved by any algorithm in $\mathfrak{F} \cap \mathfrak{S}$.

It should be noted that the three problem instances used to prove this result, φ^1, φ^2 and φ^3, all have bipartite precedence constraints. This means that bipartite precedence constraints alone are sufficient to prevent any algorithm in $\mathfrak{F} \cap \mathfrak{S}$ from approaching the optimal maximum lateness more closely than $\Gamma^{\mathscr{GJ}}(m, u_{min}, l, r)$. Combined with Theorem 3.2 – proof of the NP-hardness of $P|bipartite, pmtn|L_{max}$ – this strongly suggests that little can be gained by limiting the precedence constraints to bipartite orders.

A similar result for the family $\mathfrak{S}$ of stable algorithms is supplied below. Unfortunately it is unknown if any stable algorithm achieves this bound.

For any combination of $l \geq 3r$ and $m \geq 2u_{min} + 1$, consider another problem instance $\varphi^S \in \Phi(m, u_{min}, l, r)$ where the tasks form six sets: X_1, X_2, Y_1, Y_2, Z_1 and Z_2. The sets X_1 and X_2 are each comprised of $\lceil \frac{l}{r} \rceil m$ identical tasks. The sets Y_1 and Y_2 are each comprised of only one task, each of which is identical to the other. Finally, the sets Z_1 and Z_2 are each comprised of $\lceil \frac{l}{r} \rceil m$ identical tasks. For both $a \in \{1, 2\}$, the sole task in Y_a is preceded by all tasks in X_a and precedes all tasks in Z_a. No other precedence constraints exist between the tasks. Figure 5.4 depicts the partially ordered set of tasks in the considered instance of $P|prec, pmtn, u_j|L_{max}$. In this figure, each task is labelled with the name of the set to which it belongs.

The parameters of the tasks are

- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = 0$ for all $j \in X_1 \cup X_2$;
- $p_j = (l - 2r)u_{min}$, $u_j = u_{min}$ and $d_j = 0$ for all $j \in Y_1 \cup Y_2$; and
- $p_j = ru_{min}$, $u_j = u_{min}$ and $d_j = 0$ for all $j \in Z_1 \cup Z_2$.

Note that all due dates are zero, so this is an instance of the makespan problem $P|prec, pmtn, u_j|C_{max}$.

Observe that

- the longest chain is composed of any task from $Y_1 \cup Y_2$ along with any successor of this task and any predecessor of this task, and has a length of l;
- all tasks have $u_j = u_{min}$ and
- all tasks have $\frac{p_j}{u_j} \geq r$, with $\frac{p_j}{u_j} = r$ for all $j \in X_1 \cup X_2 \cup Z_1 \cup Z_2$.

Lemma 5.31 *For any optimal schedule ω for φ^S,*

$$C_{max}(\omega) \leq 4 \left\lceil \frac{l}{r} \right\rceil ru_{min} + \frac{2u_{min}}{m}(l - 2r)$$

for any combination of $l \geq 3r$ and $m > 2u_{min} \geq 2$.

Proof. Consider a schedule σ constructed as follows. Process all tasks in X_1 to completion during

$$\left[0, \left\lceil \frac{l}{r} \right\rceil ru_{min}\right]$$

and then process all tasks in $X_2 \cup Y_1$ to completion during

$$\left[\left\lceil \frac{l}{r} \right\rceil ru_{min}, 2 \left\lceil \frac{l}{r} \right\rceil ru_{min} + \frac{u_{min}}{m}(l - 2r)\right].$$

Subsequently, process all tasks in $Y_2 \cup Z_1$ to completion during

$$\left[2 \left\lceil \frac{l}{r} \right\rceil ru_{min} + \frac{u_{min}}{m}(l - 2r), 3 \left\lceil \frac{l}{r} \right\rceil ru_{min} + \frac{2u_{min}}{m}(l - 2r)\right]$$

and process all tasks in Z_2 to completion during

$$\left[3 \left\lceil \frac{l}{r} \right\rceil ru_{min} + \frac{2u_{min}}{m}(l - 2r), 4 \left\lceil \frac{l}{r} \right\rceil ru_{min} + \frac{2u_{min}}{m}(l - 2r)\right]$$

and the lemma is proven. ■

Theorem 5.32 *For all $\mathscr{A} \in \mathfrak{S}$, all $l \geq r > 0$ and all $m \geq u_{min} \geq 1$,*

$$\Gamma^{\mathscr{A}}(m, u_{min}, l, r) \geq \begin{cases} 0 & \text{if } m \leq 2u_{min} \text{ or } l < 3r \\ \dfrac{m - 2u_{min}}{m}(l - 2r) & \text{otherwise} \end{cases}$$

in the special case when all $d_j = 0$.

Proof. As noted before, when all $d_j = 0$, $L_{max} \equiv C_{max}$.

The theorem holds in the cases of $m \leq 2u_{min}$ and $l < 3r$ because the definition of $\Gamma^{\mathscr{A}}(m, u_{min}, l, r)$ precludes negative values. Therefore, only cases where $l \geq 3r$ and $m \geq 2u_{min} + 1$ need to be considered.

Consider a schedule $\eta^{\mathscr{A}}$ produced by any stable algorithm $\mathscr{A} \in \mathfrak{S}$. At time $t_0 = 0$, only tasks from $X_1 \cup X_2$ are available, i.e. $A_0 = X_1 \cup X_2$. By the stable property all $j \in A_0$ have the same μ_j. Consequently, all stable algorithms will process all tasks in $X_1 \cup X_2$ to completion during

$$\left[0, 2\left\lceil \frac{l}{r} \right\rceil r u_{min}\right].$$

Subsequently, both tasks in $Y_1 \cup Y_2$ will be processed to completion during

$$\left[2\left\lceil \frac{l}{r} \right\rceil r u_{min}, 2\left\lceil \frac{l}{r} \right\rceil r u_{min} + l - 2r\right]$$

and then all tasks in Z will, because they must also have identical μ's, be processed to completion during

$$\left[2\left\lceil \frac{l}{r} \right\rceil r u_{min} + l - 2r, 4\left\lceil \frac{l}{r} \right\rceil r u_{min} + l - 2r\right].$$

The makespan of this schedule is

$$C_{max}\left(\eta^{\mathscr{A}}\right) = 4\left\lceil \frac{l}{r} \right\rceil r u_{min} + l - 2r.$$

Combined with Lemma 5.31, this gives

$$C_{max}\left(\eta^{\mathscr{A}}\right) - C_{max}(\omega) \geq \frac{m - 2u_{min}}{m}(l - 2r),$$

and the theorem is proven. ■

Corollary 5.33 *The three machine problem $P3|prec, pmtn, u_j|C_{max}$ cannot be solved by any algorithm in $\mathfrak{S}$.*

This demonstrates that even the broad family of stable algorithms cannot provide a great improvement over the existing bound for the Garey-Johnson Algorithm, even in the restricted case of the makespan problem. Note that, again, the proof above may be transformed into a proof for the $P3|prec, pmtn|C_{max}$ problem by setting $u_{min} = 1$.

All of the results in this section are of a completely unprecedented nature. They provide guidance to researchers who are searching for algorithms with good performance guarantees, or with the goal of solving the three machine problem. It is now known that the family $\mathfrak{F} \cap \mathfrak{S}$ of scheduling algorithms is completely dominated, in terms of performance guarantees of the type examined in this section, by a well known and much studied polynomial time algorithm.

5.4 The Zinder-Roper Algorithm

The bound for the Zinder-Roper Algorithm proven below is analogous to that proven in [119] for the equivalent algorithm for the $P|prec, p_j = 1|L_{max}$ problem, and in [120] for the $P|prec, pmtn|L_{max}$ problem. It was originally presented by the author in [122] in collaboration with Yakov Zinder.

First, a result similar to that in [120] will be proven: that, for any $0 \leq i < i^* - 1$, no task in H_i precedes all tasks in H_{i+1}. The proof below is significantly simpler than the proof for the corresponding lemma in [120]; nonetheless it is quite involved, and so will be relegated to a separate subsection.

5.4.1 Analysis of Successors

Suppose there exists a value of $\bar{\imath}$ for which $0 < \bar{\imath} < i^*$ and there exists a task $z \in A_{\bar{\imath}-1}$ such that $H_{\bar{\imath}} \subseteq K_z$. Note that the definition of z implies the lemma stated below.

Lemma 5.34 *If z exists, $H_i \subseteq K_z$ for all $\bar{\imath} \leq i < i^*$.*

Proof. The lemma follows from the definition of z if $i = \bar{\imath}$. On the other hand, if $i > \bar{\imath}$, consider any task $j \in H_i$. Lemma 4.26 implies that either $j \in H_{\bar{\imath}}$, in which case $j \in K_z$, or there exists a task $g \in H_{\bar{\imath}}$ such that $g \rightarrow j$, in which case $g \in K_z$ and hence $j \in K_z$. ■

In order to prove that no task z can exist, it will be demonstrated that any such task contradicts the selection of i^* as the smallest i for which there exists a task j such that

$$C_j\left(\eta^{\mathscr{ZR}}\right) \geq t_i \qquad \text{and} \qquad t_i + \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right) = G\left(\eta^{\mathscr{ZR}}\right).$$

Specifically, the definition of i^* implies an upper bound on the value of μ_z, stated in the lemma below.

Lemma 5.35 *If z exists,*

$$\mu_z < t_{i^*} - t_{\bar{\imath}} + \rho^*.$$

Proof. By the definition of z, $C_z\left(\eta^{\mathscr{ZR}}\right) \leq t_{\bar{\imath}}$, and hence $z \in N_1^{\bar{\imath}-1} \cup N_2^{\bar{\imath}-1}$ and

$$\zeta_z\left(t_{\bar{\imath}}, \eta^{\mathscr{ZR}}\right) = \mu_z. \tag{5.43}$$

If $z \in N_2^{\bar{\imath}-1}$ then Lemma 4.25 implies the existence of a task $g \in N_2^{\bar{\imath}-1}$ for which

$$C_g\left(\eta^{\mathscr{ZR}}\right) \geq t_{\bar{\imath}} \qquad \text{and} \qquad \zeta_g\left(t_{\bar{\imath}}, \eta^{\mathscr{ZR}}\right) = \zeta_z\left(t_{\bar{\imath}}, \eta^{\mathscr{ZR}}\right). \tag{5.44}$$

If $z \in N_1^{\bar{\imath}-1}$, then $C_z\left(\eta^{\mathscr{ZR}}\right) = t_{\bar{\imath}}$ and hence (5.44) holds for $g = z$.

By the definition of i^*, it must be the case that

$$t_{\bar{\imath}} + \zeta_g\left(t_{\bar{\imath}}, \eta^{\mathscr{ZR}}\right) < G\left(\eta^{\mathscr{ZR}}\right) = t_{i^*} + \rho^*$$

for any g for which (5.44) holds. As a consequence, the lemma follows from (5.44) and (5.43). ■

The reasoning in this subsection will demonstrate that $\mu_z = t_{i^*} - t_{\bar{\imath}} + \rho^*$, contradicting Lemma 5.35, and hence demonstrating that no task z exists. Consequently, the value of μ_z will be examined in detail, and particular attention will be given to the schedule η^z used by the Zinder-Roper Algorithm in the determination of the value of μ_z. Note that, since $K_z \neq \emptyset$, the formula (4.24) involving this schedule will always be used in calculating μ_z.

The next lemma demonstrates that there is a correspondence between the remaining processing times of successors of z in schedule $\eta^{\mathscr{ZR}}$ and schedule η^z.

Lemma 5.36 *For all $g \in K_z$*

$$p_g\left(t_{\bar{\imath}}, \eta^{\mathscr{ZR}}\right) = p_g\left(0, \eta^z\right) = p_g.$$

Proof. As a consequence of $H_{\bar{\imath}} \subseteq K_z$ and $z \in A_{\bar{\imath}-1}$, it follows that $p_g\left(t_{\bar{\imath}}, \eta^{\mathscr{ZR}}\right) = p_g$ for all $g \in K_g$. Combining this with the fact that $p_g\left(0, \eta^z\right) = p_g$ completes the proof. ■

For schedule η^z, let the points of allocation be denoted $\hat{t}_i$, the sets of available tasks be $\hat{A}_i$, the sets N_a^i partitioning $\hat{A}_i$ in Algorithm P be $\hat{N}_a^i$ and the index of the largest point of allocation be $\hat{s}$. For all $0 \leq i \leq s$ and $0 \leq \hat{\imath} \leq \hat{s}$ define

$$Q(i,\hat{\imath}) = \left\{ g : g \in K_z \wedge \left(\zeta_g \left(t_i, \eta^{\mathscr{ZR}} \right) \geq \rho^* \vee \zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) \geq \rho^* \right) \right\},$$

i.e. $Q(i,\hat{\imath})$ is the set of all successors of z with priority not less than ρ^* at either t_i in schedule $\eta^{\mathscr{ZR}}$ or at $\hat{t}_{\hat{\imath}}$ in schedule η^z.

It will be proven that, in some sense, the "structure" of the processing of high priority tasks is identical during time interval $[t_{\bar{\imath}}, t_{i^*}]$ of schedule $\eta^{\mathscr{ZR}}$ and time interval $[0, t_{i^*} - t_{\bar{\imath}}]$ of schedule η^z. For this purpose, the two lemmas below demonstrate that, if they are identical during time intervals $[t_{\bar{\imath}}, t_i]$ and $[0, t_i - t_{\bar{\imath}}]$ respectively, for some point of allocation t_i such that $[t_i, t_{i+1}]$ is ρ^*-complete, they will also be identical during $[t_{\bar{\imath}}, t_{i+1}]$ and $[0, t_{i+1} - t_{\bar{\imath}}]$ respectively.

Lemma 5.37 *If there exists $\bar{\imath} \leq i < s$ such that the interval $[t_i, t_{i+1}]$ in schedule $\eta^{\mathscr{ZR}}$ is ρ^*-complete and there exists $0 \leq \hat{\imath} < \hat{s}$ for which*

$$\zeta_g \left(t_i, \eta^{\mathscr{ZR}} \right) = \zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) \qquad \textit{for all} \qquad g \in Q(i,\hat{\imath}), \tag{5.45}$$

it follows that

$$\hat{N}_1^{\hat{\imath}} = N_1^i \subseteq Q(i,\hat{\imath}), \qquad \hat{N}_2^{\hat{\imath}} = N_2^i \subseteq Q(i,\hat{\imath}) \qquad \textit{and} \qquad \hat{N}_3^{\hat{\imath}} \cap Q(i,\hat{\imath}) = N_3^i \cap Q(i,\hat{\imath}). \tag{5.46}$$

Proof. Define

$$\rho = \min_{g \in N_1^i \cup N_2^i} \zeta_g \left(t_i, \eta^{\mathscr{ZR}} \right),$$

and note that, since $(N_1^i \cup N_2^i) \subseteq H_i$, for some $j \in N_1^i \cup N_2^i$

$$\rho = \zeta_j \left(t_i, \eta^{\mathscr{ZR}} \right) > \zeta_j \left(t_{i+1}, \eta^{\mathscr{ZR}} \right) \geq \rho^*.$$

Lemma 5.34 and the fact that $[t_i, t_{i+1}]$ is ρ^*-complete imply that

$$N_1^i \cup N_2^i = \left\{ g : g \in A_i \wedge \zeta_g \left(t_i, \eta^{\mathscr{ZR}} \right) \geq \rho \right\} \subseteq Q(i,\hat{\imath}).$$

By (5.45) and Lemma 5.34 it follows that

$$\left\{ g : g \in A_i \wedge \zeta_g \left(t_i, \eta^{\mathscr{ZR}} \right) \geq \rho \right\} = \left\{ g : g \in \hat{A}_{\hat{\imath}} \wedge \zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) \geq \rho \right\}.$$

Hence, it follows from (5.45) that

$$\sum_{g:g\in\hat{A}_{\hat{i}}\wedge\zeta_g\left(\hat{t}_{\hat{i}},\eta^z\right)\geq\rho} u_g \geq m \qquad \text{and} \qquad \sum_{g:g\in\hat{A}_{\hat{i}}\wedge\zeta_g\left(\hat{t}_{\hat{i}},\eta^z\right)>\rho} u_g < m.$$

Consequently (5.46) holds and the lemma is proven. ■

Lemma 5.38 *If there exists $\bar{\imath} \leq i < s$ such that the interval $[t_i, t_{i+1}]$ in schedule $\eta^{\mathscr{Z}\mathscr{R}}$ is ρ^*-complete and there exists $0 \leq \hat{\imath} < \hat{s}$ for which (5.45) holds, it follows that*

- $t_{i+1} - t_i = \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$ *and*
- $\zeta_g\left(t_{i+1}, \eta^{\mathscr{Z}\mathscr{R}}\right) = \zeta_g\left(\hat{t}_{\hat{\imath}+1}, \eta^z\right)$ *for all* $g \in Q(i+1, \hat{\imath}+1)$.

Proof. Recall that Algorithm P determines the length of each interval by selecting the largest value Δ that satisfies some (possibly strict) subset of the inequalities (4.10), (4.11), (4.12), (4.13) and (4.14). Specifically, of those inequalities that apply to the particular point of allocation under consideration, at least one is an equality, while the rest are strict inequalities. It will be demonstrated below that these inequalities result in $t_{i+1} - t_i = \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$, i.e. the same interval length is determined when scheduling time interval $[t_i, t_{i+1}]$ in schedule $\eta^{\mathscr{Z}\mathscr{R}}$ and time interval $\left[\hat{t}_{\hat{\imath}}, \hat{t}_{\hat{\imath}+1}\right]$ in schedule η^z. This, combined with the equations (4.7), (4.8) and (4.9) which determine the amount of processing allocated to each available task between two consecutive points of allocation, along with Lemma 5.37, proves the lemma. The five inequalities which determine Δ are considered case by case below.

- Suppose that $N_1^i \neq \emptyset$ and that (4.10) is an equality when determining the value of $t_{i+1} - t_i$.* Since $\hat{N}_1^{\hat{\imath}} = N_1^i \subseteq Q(i, \hat{\imath})$, (5.45) implies that

$$t_{i+1} - t_i = \min_{j\in N_1^i} \frac{p_j\left(t_i, \eta^{\mathscr{Z}\mathscr{R}}\right)}{u_j} = \min_{j\in\hat{N}_1^{\hat{\imath}}} \frac{p_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right)}{u_j} \geq \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}.$$

- Suppose that $N_2^i \neq \emptyset$ and that (4.11) is an equality when determining the value of $t_{i+1} - t_i$. Since $\hat{N}_1^{\hat{\imath}} = N_1^i \subseteq Q(i, \hat{\imath})$ and $\hat{N}_2^{\hat{\imath}} = N_2^i \subseteq Q(i, \hat{\imath})$, (5.45) implies that

$$t_{i+1} - t_i = \frac{\sum_{j\in N_2^i} u_j}{m - \sum_{j\in N_1^i} u_j} \min_{j\in N_2^i} \frac{p_j\left(t_i, \eta^{\mathscr{Z}\mathscr{R}}\right)}{u_j}$$

*The constraint that $N_1^i \neq \emptyset$ is required, since without it (4.10) is undefined, and is thus not used in the calculation of the length of the interval $[t_i, t_{i+1}]$. This is the reason behind the similar constraints on the non-emptiness of sets N_a^i in the four subsequent cases.

$$= \frac{\sum_{j\in\hat{N}_2^{\hat{i}}} u_j}{m - \sum_{j\in\hat{N}_1^{\hat{i}}} u_j} \min_{j\in\hat{N}_2^{\hat{i}}} \frac{p_j\left(\hat{t}_{\hat{i}}, \eta^z\right)}{u_j} \geq \hat{t}_{\hat{i}+1} - \hat{t}_{\hat{i}}.$$

- Suppose that $N_1^i \neq \emptyset$, $N_2^i \neq \emptyset$ and that (4.12) is an equality when determining the value of $t_{i+1} - t_i$. Since $\hat{N}_1^{\hat{i}} = N_1^i \subseteq Q(i,\hat{i})$ and $\hat{N}_2^{\hat{i}} = N_2^i \subseteq Q(i,\hat{i})$, (5.45) implies that, for any $g \in N_2^i$,

$$t_{i+1} - t_i = \frac{\sum_{j\in N_2^i} u_j}{\sum_{j\in N_1^i\cup N_2^i} u_j - m} \left(\min_{j\in N_1^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right) - \zeta_g\left(t_i, \eta^{\mathscr{ZR}}\right)\right)$$

$$= \frac{\sum_{j\in\hat{N}_2^{\hat{i}}} u_j}{\sum_{j\in\hat{N}_1^{\hat{i}}\cup\hat{N}_2^{\hat{i}}} u_j - m} \left(\min_{j\in\hat{N}_1^{\hat{i}}} \zeta_j\left(\hat{t}_{\hat{i}}, \eta^z\right) - \zeta_g\left(\hat{t}_{\hat{i}}, \eta^z\right)\right) \geq \hat{t}_{\hat{i}+1} - \hat{t}_{\hat{i}}.$$

- Suppose that $N_1^i \neq \emptyset$, $N_3^i \neq \emptyset$ and that (4.13) is an equality when determining the value of $t_{i+1} - t_i$. If

$$\max_{j\in N_3^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right) \geq \rho^*, \tag{5.47}$$

it follows from $\hat{N}_1^{\hat{i}} = N_1^i \subseteq Q(i,\hat{i})$, $\hat{N}_3^{\hat{i}} \cap Q(i,\hat{i}) = N_3^i \cap Q(i,\hat{i})$ and (5.45) that

$$t_{i+1} - t_i = \min_{j\in N_1^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right) - \max_{j\in N_3^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right)$$

$$= \min_{j\in\hat{N}_1^{\hat{i}}} \zeta_j\left(\hat{t}_{\hat{i}}, \eta^z\right) - \max_{j\in\hat{N}_3^{\hat{i}}} \zeta_j\left(\hat{t}_{\hat{i}}, \eta^z\right) \geq \hat{t}_{\hat{i}+1} - \hat{t}_{\hat{i}}.$$

On the other hand, if (5.47) does not hold, the ρ^*-completeness of $[t_i, t_{i+1}]$ implies

$$t_{i+1} - t_i = \min_{j\in N_1^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right) - \min_{j\in N_1^i} \zeta_j\left(t_{i+1}, \eta^{\mathscr{ZR}}\right)$$

$$\leq \min_{j\in N_1^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right) - \rho^* < \min_{j\in N_1^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right) - \max_{j\in N_3^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right).$$

This contradicts the assumption that (4.13) is an equality when determining the value of $t_{i+1} - t_i$.

- Finally, suppose that $N_2^i \neq \emptyset$ $N_3^i \neq \emptyset$ and that (4.14) is an equality when determining the value of $t_{i+1} - t_i$. If (5.47) holds, it follows from $\hat{N}_2^{\hat{i}} = N_2^i \subseteq Q(i,\hat{i})$, $\hat{N}_3^{\hat{i}} \cap Q(i,\hat{i}) = N_3^i \cap Q(i,\hat{i})$ and (5.45) that, for any $g \in N_2^i$,

$$t_{i+1} - t_i = \frac{\sum_{j\in N_2^i} u_j}{m - \sum_{j\in N_1^i} u_j} \left(\zeta_g\left(t_i, \eta^{\mathscr{ZR}}\right) - \max_{j\in N_3^i} \zeta_j\left(t_i, \eta^{\mathscr{ZR}}\right)\right)$$

$$= \frac{\sum_{j\in\hat{N}_2^{\hat{\imath}}} u_j}{m - \sum_{j\in\hat{N}_1^{\hat{\imath}}} u_j} \left(\zeta_g\left(\hat{t}_{\hat{\imath}}, \eta^z\right) - \max_{j\in\hat{N}_3^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right) \right) \geq \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}.$$

On the other hand, if (5.47) does not hold, for any $g \in N_2^i$, the ρ^*-completeness of $[t_i, t_{i+1}]$ implies

$$t_{i+1} - t_i = \frac{\sum_{j\in N_2^i} u_j}{m - \sum_{j\in N_1^i} u_j} \left(\zeta_g\left(t_i, \eta^{\mathscr{Z}\mathscr{R}}\right) - \zeta_g\left(t_{i+1}, \eta^{\mathscr{Z}\mathscr{R}}\right) \right)$$

$$\leq \frac{\sum_{j\in N_2^i} u_j}{m - \sum_{j\in N_1^i} u_j} \left(\zeta_g\left(t_i, \eta^{\mathscr{Z}\mathscr{R}}\right) - \rho^* \right)$$

$$< \frac{\sum_{j\in N_2^i} u_j}{m - \sum_{j\in N_1^i} u_j} \left(\zeta_g\left(t_i, \eta^{\mathscr{Z}\mathscr{R}}\right) - \max_{j\in N_3^i} \zeta_j\left(t_i, \eta^{\mathscr{Z}\mathscr{R}}\right) \right).$$

This contradicts the assumption that (4.14) is an equality when determining the value of $t_{i+1} - t_i$.

As a consequence of the above five cases, whichever of (4.10), (4.11), (4.12), (4.13) and (4.14) is an equality when determining the value of $t_{i+1} - t_i$,

$$t_{i+1} - t_i \geq \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}. \tag{5.48}$$

Now a similar, although much abbreviated, analysis will be performed on the determination of $\hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$.

- Suppose that $N_1^i \neq \emptyset$, $N_3^i \neq \emptyset$, that (4.13) is an equality when determining the value of $\hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$, and that

$$\max_{j\in\hat{N}_3^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right) < \rho^*. \tag{5.49}$$

As a consequence of $\hat{N}_1^{\hat{\imath}} = N_1^i \subseteq Q(i, \hat{\imath})$, (5.45) and (5.48), it follows from the ρ^*-completeness of $[t_i, t_{i+1}]$ that

$$\hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}} = \min_{j\in\hat{N}_1^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right) - \min_{j\in\hat{N}_1^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}+1}, \eta^z\right) \leq \min_{j\in\hat{N}_1^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right) - \min_{j\in N_1^i} \zeta_j\left(t_{i+1}, \eta^{\mathscr{Z}\mathscr{R}}\right)$$

$$\leq \min_{j\in\hat{N}_1^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right) - \rho^* < \min_{j\in\hat{N}_1^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right) - \max_{j\in\hat{N}_3^{\hat{\imath}}} \zeta_j\left(\hat{t}_{\hat{\imath}}, \eta^z\right).$$

This contradicts the assumption that (4.13) is an equality when determining the value of $\hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$.

- Suppose that $N_2^i \neq \emptyset$, $N_3^i \neq \emptyset$, that (4.14) is an equality when determining the value of $\hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$, and that (5.49) holds. As a consequence of $\hat{N}_2^{\hat{\imath}} = N_2^i \subseteq Q(i, \hat{\imath})$, (5.45) and (5.48), it follows from the ρ^*-completeness of $[t_i, t_{i+1}]$ that, for any $g \in N_2^i$,

$$\begin{aligned}
\hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}} &= \frac{\sum_{j \in \hat{N}_2^{\hat{\imath}}} u_j}{m - \sum_{j \in \hat{N}_1^{\hat{\imath}}} u_j} \left(\zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) - \zeta_g \left(\hat{t}_{\hat{\imath}+1}, \eta^z \right) \right) \\
&\leq \frac{\sum_{j \in \hat{N}_2^{\hat{\imath}}} u_j}{m - \sum_{j \in \hat{N}_1^{\hat{\imath}}} u_j} \left(\zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) - \zeta_g \left(t_{i+1}, \eta^{\mathscr{ZR}} \right) \right) \\
&\leq \frac{\sum_{j \in \hat{N}_2^{\hat{\imath}}} u_j}{m - \sum_{j \in \hat{N}_1^{\hat{\imath}}} u_j} \left(\zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) - \rho^* \right) \\
&< \frac{\sum_{j \in \hat{N}_2^{\hat{\imath}}} u_j}{m - \sum_{j \in \hat{N}_1^{\hat{\imath}}} u_j} \left(\zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) - \max_{j \in \hat{N}_3^{\hat{\imath}}} \zeta_j \left(\hat{t}_{\hat{\imath}}, \eta^z \right) \right).
\end{aligned}$$

  This contradicts the assumption that (4.14) is an equality when determining the value of $\hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$.

- Finally, in all other cases, identical reasoning to the consideration of $t_{i+1} - t_i$ demonstrates that

$$t_{i+1} - t_i \leq \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}},$$

  and hence

$$t_{i+1} - t_i = \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}.$$

Since the resulting intervals are the same length, since every task in $Q(i, \hat{\imath})$ is in the same set, N_1, N_2 or N_3, for both schedules, and as a consequence of (5.45),

$$p_g \left(t_i, \eta^{\mathscr{ZR}} \right) - p_g \left(t_i, \eta^{\mathscr{ZR}} \right) = \zeta_g \left(\hat{t}_{\hat{\imath}}, \eta^z \right) \qquad \text{for all} \qquad g \in Q(i, \hat{\imath}).$$

The lemma follows from the fact that $Q(i+1, \hat{\imath}+1) \subseteq Q(i, \hat{\imath})$. ■

The following corollary is simply a restatement of Lemma 5.38 with the schedules $\eta^{\mathscr{ZR}}$ and η^z swapped, and can be derived in precisely the same way.*

Corollary 5.39 *If there exists $0 \leq \hat{\imath} < \hat{s}$ such that the interval $[t_{\hat{\imath}}, t_{\hat{\imath}+1}]$ in schedule η^z is ρ^*-complete and there exists $\bar{\imath} \leq i < s$ for which (5.45) holds, it follows that*

- $t_{i+1} - t_i = \hat{t}_{\hat{\imath}+1} - \hat{t}_{\hat{\imath}}$ *and*

*Although it is somewhat infelicitous to state Lemma 5.38 and Corollary 5.39 separately when they could have been proven in a single lemma, the cost that making Lemma 5.38 agnostic with respect to the schedules $\eta^{\mathscr{ZR}}$ and η^z would have incurred, in terms of complexity of notation and exposition, is quite significant, and this genericity is needed only here.

- $\zeta_g\left(t_{i+1}, \eta^{\mathscr{ZR}}\right) = \zeta_g\left(\hat{t}_{\hat{\imath}+1}, \eta^z\right)$ *for all* $g \in Q(i+1, \hat{\imath}+1)$.

The lemma below derives a similar result for ρ^*-incomplete intervals. Note, however, that while Lemma 5.38 considered only the next point of allocation in both schedules, Lemma 5.40 considers unspecified points of allocation potentially much later in each schedule.

Lemma 5.40 *If there exists* $\bar{\imath} \leq i < i^*$ *such that the interval* $[t_i, t_{i+1}]$ *in schedule* $\eta^{\mathscr{ZR}}$ *is* ρ^**-incomplete and there exists* $0 \leq \hat{\imath} < \hat{s}$ *for which (5.45) holds, it follows that*

- *there exist* $i < i' < i^*$ *and* $\hat{\imath} < \hat{\imath}' < \hat{s}$ *for which* $t_{i'} - t_i = \hat{t}_{\hat{\imath}'} - \hat{t}_{\hat{\imath}}$, *and*
- $\zeta_g\left(t_{i'}, \eta^{\mathscr{ZR}}\right) = \zeta_g\left(\hat{t}_{\hat{\imath}'}, \eta^z\right)$ *for all* $g \in Q(i', \hat{\imath}')$.

Proof. For any $i'' > i$ such that $[t_{i''}, t_{i''+1}]$ is ρ^*-complete, Lemma 4.26 demonstrates that, for each $j'' \in H_{i''}$, either $j'' \in H_i$ or $g \to j''$ for some $g \in H_i$. If $H_{i''} \subseteq H_i$,

$$m > \sum_{q \in H_i} u_q \geq \sum_{q \in H_{i''}} u_q \geq m,$$

a contradiction. Hence, by Corollary 4.31, there must be at least one $g \in H_i \subseteq (A_i \cap Q(i, \hat{\imath}))$ such that $C_g\left(\eta^{\mathscr{ZR}}\right) \leq t_{i''}$ and

$$\mu_g \geq \frac{p_{j''}}{u_{j''}} + \mu_j > \rho^*.$$

Among all $g \in A_i \cap Q(i, \hat{\imath})$ with $\mu_g > \rho^*$, select as g' a task with the smallest value of $C_g\left(\eta^{\mathscr{ZR}}\right)$. Note that $g' \in N_1^i$. By Lemma 4.27, $C_g\left(\eta^{\mathscr{ZR}}\right)$ coincides with a point of allocation; select i' such that $t_{i'} = C_{g'}\left(\eta^{\mathscr{ZR}}\right)$. For each task $j \in A_i \cap Q(i, \hat{\imath})$, Lemma 4.27 and the fact that all time intervals comprising $[t_i, t_{i'}]$ are ρ^*-incomplete imply that either

- $\zeta_j\left(t_{i'}, \eta^{\mathscr{ZR}}\right) < \rho^*$ or
- $p_j\left(t_i, \eta^{\mathscr{ZR}}\right) - p_j\left(t_{i'}, \eta^{\mathscr{ZR}}\right) = u_j(t_{i'} - t_i)$.

As a consequence of this, for each $j \in A_i \cap Q(i, \hat{\imath})$ with $\mu_j > \rho^*$,

$$p_j\left(t_i, \eta^{\mathscr{ZR}}\right) \geq u_j(t_{i'} - t_i)$$

and hence (5.45) implies

$$C_j\left(\eta^z\right) \geq \hat{\imath} + t_{i'} - t_i.$$

Thus, for any $\hat{\imath}''$ such that

$$\hat{t}_{\hat{\imath}} \leq \hat{t}_{\hat{\imath}''} < \hat{t}_{\hat{\imath}} + t_{i'} - t_i$$

(4.15) implies

$$\left\{ j : j \in \hat{A}_{\hat{\imath}''} \wedge \zeta_j \left(\hat{t}_{\hat{\imath}''}, \eta^z \right) \geq \rho^* \right\} \subseteq (A_i \cap Q(i, \hat{\imath})).$$

Corollary 5.39 demonstrates that the interval $\left[\hat{t}_{\hat{\imath}}, \hat{t}_{\hat{\imath}+1}\right]$ in schedule η^z is not ρ^*-complete, because if it was, the interval $[t_i, t_{i+1}]$ in schedule $\eta^{\mathscr{ZR}}$ would be ρ^*-complete. Hence,

$$\sum_{j : j \in \hat{A}_{\hat{\imath}''} \wedge \zeta_j \left(\hat{t}_{\hat{\imath}''}, \eta^z \right) \geq \rho^*} u_j < m,$$

and thus Lemma 4.27 implies that $g' \in \hat{N}_1^{\hat{\imath}''}$.

This in turn implies the existence of a point of allocation $\hat{t}_{\hat{\imath}'}$ such that

$$C_{g'} \left(\eta^z \right) = \hat{t}_{\hat{\imath}} + t_{i'} - t_i = \hat{t}_{\hat{\imath}'}.$$

Consequently, at time $\hat{t}_{\hat{\imath}'}$ in schedule η^z, every task $j \in A_i \cap Q(i, \hat{\imath})$ is either processed on u_j machines continuously during $\left[\hat{t}_{\hat{\imath}}, \hat{t}_{\hat{\imath}'}\right]$, or has $\zeta_j \left(\hat{t}_{\hat{\imath}'}, \eta^z \right) < \rho^*$. This precisely replicates the processing during $[t_i, t_{i'}]$ in schedule $\eta^{\mathscr{ZR}}$, so the lemma is proven. ■

The lemma below uses the preceding working to demonstrate that there is no task z, as defined earlier in this subsection. This result will be used to derive several properties of $\eta^{\mathscr{ZR}}$ that will be used in Subsection 5.4.2 to establish a performance guarantee for the Zinder-Roper Algorithm. It will also be used in Section 6.3 to determine the class of precedence constraints for which the Zinder-Roper Algorithm is guaranteed to produce an optimal schedule, known as the "domain" of the algorithm.

Lemma 5.41 *There exists no* $1 \leq \bar{\imath} < i^*$ *and no task* $z \in H_{\bar{\imath}-1}$ *such that* $H_{\bar{\imath}} \subseteq K_z$.

Proof. If such a task z does exist, Lemma 5.36, Lemma 5.38 and Lemma 5.40 demonstrate that (5.45) holds for $i = i^*$ and $\hat{t}_{\hat{\imath}} = t_{i^*} - t_{\bar{\imath}}$. Hence (4.24) implies

$$\mu_z \geq G \left(\eta^z \right) \geq \hat{t}_{\hat{\imath}} + \zeta_x \left(\hat{t}_{\hat{\imath}}, \eta^z \right) = t_{i^*} - t_{\bar{\imath}} + \rho^*,$$

contradicting Lemma 5.35. ■

Corollary 5.42 *In schedule $\eta^{\mathscr{ZR}}$ produced by the Zinder-Roper Algorithm, $|H_i| \geq 2$ for all $0 \leq i < i^*$.*

Proof. By Corollary 4.28 if there exists any i and j such that $H_i = \{j\}$, $H_{i'} \subseteq K_j$ for all $i' > i$, contradicting Lemma 5.41. ■

Corollary 5.43 *The schedule $\eta^{\mathscr{ZR}}$ produced by the Zinder-Roper Algorithm is optimal for $P|prec, pmtn, u_j|G$ if $2u_{min} \geq m$.*

Proof. This follows from the previous corollary, since there can be no ρ^*-incomplete intervals when $2u_{min} \geq m$. ■

Corollary 5.44 *If the μ's produced by the Zinder-Roper Algorithm are preserving, schedule $\eta^{\mathscr{ZR}}$ produced by the Zinder-Roper Algorithm is optimal for $P|prec, pmtn, u_j|L_{max}$ if $2u_{min} \geq m$.*

The lemma below is a generalisation to the case of malleable tasks of a key result from [120], specifically that the Zinder-Roper Algorithm solves the $P2|prec, pmtn|L_{max}$ problem.

Lemma 5.45 *The schedule $\eta^{\mathscr{ZR}}$ produced by the Zinder-Roper Algorithm is optimal for $P|prec, pmtn, u_j|L_{max}$ if $2u_{min} \geq m$.*

Proof. Suppose this is not the case, i.e. that $\eta^{\mathscr{ZR}}$ does not minimise the maximum lateness for a given problem instance. Corollary 5.44 then implies that the μ's produced by the Zinder-Roper Algorithm are not preserving. Since the Zinder-Roper Algorithm assigns $\mu_j = -d_j$ if $K_j = \emptyset$, by Lemma 4.1 it follows that, for at least one task j with $K_j \neq \emptyset$,

$$\mu_j > \max\left\{L_{max}\left(\omega^j\right), -d_j\right\}. \tag{5.50}$$

Without loss of generality, let j have the smallest value of $|K_j|$ among all such tasks.

Recall that, by (4.24), either $\mu_j = -d_j$ or $\mu_j = G\left(\eta^j\right)$. If $\mu_j = -d_j$,

$$\mu_j = -d_j \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\},$$

contradicting (5.50).

On the other hand suppose $\mu_j = G\left(\eta^j\right)$. Note that, for all $g \in K_j$, $|K_g| \leq |K_j \setminus \{g\}| = k$, so by the selection of j, the μ's of all tasks in K_j are preserving. Thus, Corollary 5.44 implies

$$\mu_j = G\left(\eta^j\right) = L_{max}\left(\eta^j\right) = L_{max}\left(\omega^j\right) \leq \max\left\{L_{max}\left(\omega^j\right), -d_j\right\},$$

which also contradicts (5.50). Hence, either case leads to a contradiction, and the lemma holds. ■

Corollary 5.46 *The Zinder-Roper algorithm optimally solves the problem* $P2|prec, pmtn, u_j|L_{max}$.

It now remains to establish the performance guarantee.

5.4.2 The Remainder of the Proof

The bound for the Zinder-Roper Algorithm proven below is analogous to that proven in [119] for the equivalent algorithm for the $P|prec, p_j = 1|L_{max}$ problem and in [120] for the $P|prec, pmtn|L_{max}$ problem. Lemma 5.47 below establishes a relationship between μ's produced by the Zinder-Roper Algorithm, and those produced by the Optimal Recursive Algorithm; specifically it establishes the unsurprising result that the Zinder-Roper Algorithm generates larger μ's than the Optimal Recursive Algorithm. This will be used in the proof of Lemma 5.48 which establishes a bound on the difference between these values. Indeed, it is for this purpose that the Optimal Recursive Algorithm was introduced – the Zinder-Roper Algorithm is difficult to analyse directly because the μ's it produces are not necessarily preserving, while the Optimal Recursive Algorithm, which closely resembles the Zinder-Roper Algorithm, is guaranteed to produce preserving μ's.

Throughout the rest of this thesis, schedules produced by particular algorithms are given superscripts to indicate their origin; analogously, in this subsection $\mu_j^{\mathscr{ZR}}$ and $\mu_j^{\mathscr{OR}}$ represent the value of μ_j produced by the Zinder-Roper Algorithm and the Optimal Recursive Algorithm respectively, likewise $G^{\mathscr{ZR}}$ and $G^{\mathscr{OR}}$ represent the corresponding function G and $\zeta_j^{\mathscr{ZR}}$ and $\zeta_j^{\mathscr{OR}}$ represent the corresponding function ζ_j. Because this is the only portion of this thesis where μ's produced by different algorithms for the same problem instance are compared, this notation is not used outside this subsection. Finally, let η^j be the schedule generated by the Zinder-Roper Algorithm when calculating $\mu_j^{\mathscr{ZR}}$ and ω^j be the schedule generated by the Optimal Recursive Algorithm when calculating $\mu_j^{\mathscr{OR}}$.

Lemma 5.47 *For all* $j \in N$, $\mu_j^{\mathscr{ZR}} \geq \mu_j^{\mathscr{OR}}$.

Proof. This lemma will be proven by induction on $|K_j|$. First note that, if $|K_j| = 0$,

$$\mu_j^{\mathscr{ZR}} = \mu_j^{\mathscr{OR}} = -d_j$$

and the lemma holds. Now assume that, for some $k \geq 0$, the lemma holds for all $|K_j| \leq k$, and consider the case where $|K_j| = k + 1$.

Note that, for all $g \in K_j$, $|K_g| \leq |K_j \setminus \{g\}| = k$ which, when combined with the inductive assumption, implies that $\mu_g^{\mathscr{ZR}} \geq \mu_g^{\mathscr{OR}}$. Consequently, (4.24) and (4.26) imply that

$$\mu_j^{\mathscr{ZR}} = \max\left\{-d_j, G^{\mathscr{ZR}}\left(\eta^j\right)\right\} \geq \max\left\{-d_j, G^{\mathscr{OR}}\left(\eta^j\right)\right\}$$

$$\geq \max\left\{-d_j, G^{\mathscr{OR}}\left(\omega^j\right)\right\} = \mu_j^{\mathscr{OR}}.$$

Hence, the lemma holds true when $|K_j| = k + 1$, and thus it has been proven by induction. ■

Lemma 5.48 *For any $j \in N$, $K_j = \emptyset$ implies $\mu_j^{\mathscr{ZR}} = \mu_j^{\mathscr{OR}}$ and $K_j \neq \emptyset$ implies*

$$\mu_j^{\mathscr{ZR}} - \mu_j^{\mathscr{OR}} \leq G^{\mathscr{ZR}}\left(\eta^j\right) - G^{\mathscr{OR}}\left(\omega^j\right). \tag{5.51}$$

Proof. If $K_j = \emptyset$, the lemma follows directly from the definitions of the algorithms. Suppose now that $K_j \neq \emptyset$. If $\mu_j^{\mathscr{ZR}} = -d_j$, Lemma 5.47 and (4.26) imply

$$\mu_j^{\mathscr{ZR}} = -d_j \leq \max\left\{-d_j, G^{\mathscr{OR}}\left(\omega^j\right)\right\} = \mu_j^{\mathscr{OR}} \leq \mu_j^{\mathscr{ZR}}.$$

This in turn implies

$$\mu_j^{\mathscr{ZR}} - \mu_j^{\mathscr{OR}} = 0 = G^{\mathscr{OR}}\left(\omega^j\right) - G^{\mathscr{OR}}\left(\omega^j\right)$$

$$\leq G^{\mathscr{OR}}\left(\eta^j\right) - G^{\mathscr{OR}}\left(\omega^j\right) \leq G^{\mathscr{ZR}}\left(\eta^j\right) - G^{\mathscr{OR}}\left(\omega^j\right),$$

and the lemma is proven in the case where $\mu_j^{\mathscr{ZR}} = -d_j$.

On the other hand, if $\mu_j^{\mathscr{ZR}} \neq -d_j$, (4.24) and (4.26) imply

$$\mu_j^{\mathscr{ZR}} - \mu_j^{\mathscr{OR}} = G^{\mathscr{ZR}}\left(\eta^j\right) - \max\left\{-d_j, G^{\mathscr{OR}}\left(\omega^j\right)\right\} \leq G^{\mathscr{ZR}}\left(\eta^j\right) - G^{\mathscr{OR}}\left(\omega^j\right)$$

and the lemma is proven in this case as well. ■

Having established a relationship between the μ's generated by the Zinder-Roper Algorithm and the Optimal Recursive Algorithm, a relationship between the criteria $G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right)$ and $G^{\mathscr{OR}}(\omega)$ will now be derived, where $\eta^{\mathscr{ZR}}$ is a schedule generated by the Zinder-Roper Algorithm and ω is an optimal schedule for $P|prec, pmtn, u_j$

$|L_{max}$.* Since the schedule $\eta^{\mathscr{OR}}$ generated by the Optimal Recursive Algorithm is not used in this analysis, i^*, x, ρ^* and H_i will all refer to these values produced by the Zinder-Roper Algorithm for schedule $\eta^{\mathscr{ZR}}$.

Theorem 5.49 *For any schedule $\eta^{\mathscr{ZR}}$ produced by the Zinder-Roper Algorithm, and any schedule ω which is optimal for $P|prec, pmtn, u_j|L_{max}$,*

$$L_{max}\left(\eta^{\mathscr{ZR}}\right) \leq \begin{cases} L_{max}(\omega) & \text{if } m \leq 2u_{min} \\ \left(2 - \dfrac{2u_{min}}{m}\right) L_{max}(\omega) + \left(1 - \dfrac{2u_{min}}{m}\right)(d_{max} - r) & \text{otherwise} \end{cases}.$$

Proof. In the case of $m \leq 2u_{min}$, the theorem follows directly from Lemma 5.45. For the remainder of this proof it will be assumed that $m > 2u_{min}$.

If

$$G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) \leq \left(2 - \frac{2u_{min}}{m}\right) G^{\mathscr{OR}}(\omega) + \left(1 - \frac{2u_{min}}{m}\right)(d_{max} - r), \tag{5.52}$$

then the theorem follows because (4.24) implies $G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) \geq L_{max}\left(\eta^{\mathscr{ZR}}\right)$ and Lemma 4.19 demonstrates that $G^{\mathscr{OR}}(\omega) = L_{max}(\omega)$. The remainder of the theorem will be devoted to proving (5.52) by induction on the number of tasks n.

First note that

$$G^{\mathscr{OR}}(\omega) = L_{max}(\omega) \geq r - d_{max}. \tag{5.53}$$

If $n = 1$ this implies

$$G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) = G^{\mathscr{OR}}(\omega) = \left(2 - \frac{2u_{min}}{m}\right) G^{\mathscr{OR}}(\omega) + \left(1 - \frac{2u_{min}}{m}\right)(d_{max} - r)$$

and (5.52) holds. Now, as the inductive assumption, suppose there is some $k \geq 1$ such that (5.52) holds for all $n \leq k$ and consider the case of $n = k + 1$.

Note that for any task j , either $\mu_j^{\mathscr{ZR}} = G^{\mathscr{ZR}}(\eta^j)^\dagger$ or $\mu_j^{\mathscr{ZR}} = -d_j$ (or both). In the case where $\mu_j^{\mathscr{ZR}} = G^{\mathscr{ZR}}(\eta^j)$, the inductive assumption, together with $|K_j| \leq k$, implies that

$$\mu_j^{\mathscr{ZR}} = G^{\mathscr{ZR}}\left(\eta^j\right) \leq \left(2 - \frac{2u_{min}}{m}\right) G^{\mathscr{OR}}\left(\omega^j\right) + \left(1 - \frac{2u_{min}}{m}\right)\left(\max_{g \in K_j} d_g - \min_{g \in K_j} \frac{p_g}{u_g}\right)$$

*In much of this thesis it is not specified whether ω minimises G or L_{max}. This distinction only matters when the μ's are not preserving, i.e. for the Zinder-Roper Algorithm alone.

†Note that if $K_j = \emptyset$ the schedule η^j is not defined.

and as a consequence of (4.26),

$$\mu_j^{\mathscr{ZR}} \leq \left(2 - \frac{2u_{min}}{m}\right) \mu_j^{\mathscr{OR}} + \left(1 - \frac{2u_{min}}{m}\right) (d_{max} - r). \tag{5.54}$$

On the other hand, if $\mu_j^{\mathscr{ZR}} = -d_j$, Lemma 5.47 implies $\mu_j^{\mathscr{OR}} = -d_j$ and as a consequence of

$$\mu_j^{\mathscr{OR}} = -d_j \geq -d_{max}$$

it follows that

$$\mu_j^{\mathscr{ZR}} \leq \left(2 - \frac{2u_{min}}{m}\right) \mu_j^{\mathscr{OR}} + \left(1 - \frac{2u_{min}}{m}\right) d_{max}. \tag{5.55}$$

Note that (5.54) implies (5.55).

If $i^* = 0$, there exists a task x such that

$$G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) = \frac{p_x}{u_x} + \mu_x^{\mathscr{ZR}} \qquad \text{and} \qquad G^{\mathscr{OR}}(\omega) \geq \frac{p_x}{u_x} + \mu_x^{\mathscr{OR}}. \tag{5.56}$$

If $\mu_j^{\mathscr{ZR}} = G^{\mathscr{ZR}}(\eta^j)$, combining (5.56) with (5.54) gives

$$\begin{aligned} G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) &\leq \frac{p_x}{u_x} + \left(2 - \frac{2u_{min}}{m}\right) \mu_x^{\mathscr{OR}} + \left(1 - \frac{2u_{min}}{m}\right) (d_{max} - r) \\ &\leq \left(2 - \frac{2u_{min}}{m}\right) \left(\frac{p_x}{u_x} + \mu_x^{\mathscr{OR}}\right) + \left(1 - \frac{2u_{min}}{m}\right) (d_{max} - r). \\ &\leq \left(2 - \frac{2u_{min}}{m}\right) G^{\mathscr{OR}}(\omega) + \left(1 - \frac{2u_{min}}{m}\right) (d_{max} - r). \end{aligned}$$

On the other hand, if $\mu_j^{\mathscr{ZR}} = -d_j$, Lemma 5.47 implies $\mu_j^{\mathscr{OR}} = -d_j$ and hence (5.56) and (5.53) give

$$\begin{aligned} G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) &= \frac{p_x}{u_x} + \mu_x^{\mathscr{OR}} \leq G^{\mathscr{OR}}(\omega) \\ &\leq \left(2 - \frac{2u_{min}}{m}\right) G^{\mathscr{OR}}(\omega) + \left(1 - \frac{2u_{min}}{m}\right) (d_{max} - r). \end{aligned}$$

Thus, if $i^* = 0$, (5.52) holds and hence the lemma follows. From this point on, it will be assumed that $i^* > 0$.

Define

$$f_j(\sigma) = \max\left\{t : \zeta_j^{\mathscr{ZR}}(t, \sigma) \geq \rho^* \wedge t \leq C_j(\sigma)\right\}$$

for all $j \in H = \cup_{i=0}^{i^*-1} H_i$, i.e. $f_j(\sigma)$ is the largest point in time in schedule σ where task j is not complete and has high priority, according to the μ's generated by the Zinder-Roper Algorithm. Lemma 4.29 demonstrates that H is not empty.

Denote the total length in schedule $\eta^{\mathscr{ZR}}$ of all ρ^*-complete intervals as c and all ρ^*-incomplete intervals as w. Since $G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) = c + w + \rho^*$, by Corollary 5.42

$$\max_{j\in H} f_j\left(\omega\right) \geq c + \frac{2u_{min}}{m} w$$

and by Corollary 4.28 and Lemma 4.29,

$$\max_{j\in H} f_j\left(\omega\right) \geq w + r.$$

Combining both of these relations with $G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) = c + w + \rho^*$ gives

$$\begin{aligned} G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) &\leq \left(2 - \frac{2u_{min}}{m}\right) \max_{j\in H} f_j\left(\omega\right) - \left(1 - \frac{2u_{min}}{m}\right) r + \rho^* \\ &= \max_{j\in H} \left[\left(2 - \frac{2u_{min}}{m}\right) f_j(\omega) + \rho^*\right] - \left(1 - \frac{2u_{min}}{m}\right) r \end{aligned} \tag{5.57}$$

From the definition of $f_j(\sigma)$, (5.57) and Lemma 4.29 imply that

$$\begin{aligned} G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) \leq \max_{j\in H} \left[\left(2 - \frac{2u_{min}}{m}\right) f_j(\omega) + \frac{p_j\left(f_j\left(\eta^{\mathscr{ZR}}\right), \eta^{\mathscr{ZR}}\right)}{u_j} + \mu_j^{\mathscr{ZR}}\right] \\ - \left(1 - \frac{2u_{min}}{m}\right) r \end{aligned} \tag{5.58}$$

From the fact that

$$p_j\left(f_j\left(\sigma_1\right), \sigma_1\right) = p_j\left(f_j\left(\sigma_2\right), \sigma_2\right)$$

for any $j \in H$ and any two schedules σ_1 and σ_2, and from (5.55), (5.58) implies

$$\begin{aligned} G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) \leq \left(2 - \frac{2u_{min}}{m}\right) \max_{j\in H} \left[f_j\left(\omega\right) + \frac{p_j\left(f_j\left(\omega\right), \omega\right)}{u_j} + \mu_j^{\mathscr{OR}}\right] \\ + \left(1 - \frac{2u_{min}}{m}\right) (d_{max} - r). \end{aligned} \tag{5.59}$$

Finally, since $C_j(\omega) \geq f_j(\omega)$ and thus

$$C_j(\omega) \geq f_j(\omega) + \frac{p_j\left(f_j\left(\omega\right), \omega\right)}{u_j}$$

for all $j \in H$, (5.59) implies

$$G^{\mathscr{ZR}}\left(\eta^{\mathscr{ZR}}\right) \leq \left(2 - \frac{2u_{min}}{m}\right) \max_{j\in H} \left(C_j(\omega) + \mu_j^{\mathscr{OR}}\right) + \left(1 - \frac{2u_{min}}{m}\right) (d_{max} - r)$$

which in turn implies (5.52).

Consequently (5.52) holds in the case where $n = k+1$ and the theorem is proven by induction. ■

Note that, for the $P|prec, pmtn|L_{max}$ problem, [120] proved that the Zinder-Roper Algorithm has a bound

$$L_{max}\left(\eta^{\mathscr{ZR}}\right) \leq \left(2 - \frac{2}{m}\right) L_{max}(\omega) + \left(1 - \frac{2}{m}\right) d_{max},$$

i.e. $d_{max} - r$ is replaced by d_{max} in the bound established above. Thus, Theorem 5.49 above provides a slight improvement on what was already known for the case of non-malleable tasks, as well as extending the proof to malleable tasks. This improvement arises from a more careful examination of the case where $i^* = 0$.

Unfortunately, this ratio bound is derived directly, rather that as a consequence of a difference bound as in Corollary 5.3 and Corollary 5.16. To the author's knowledge it is unknown whether

$$\Gamma^{\mathscr{ZR}}(m, u_{min}, l, r) = \Gamma^{\mathscr{GJ}}(m, u_{min}, l, r),$$

where $\Gamma^{\mathscr{A}}(m, u_{min}, l, r)$ is defined as in Section 5.3. This is an area for future research.

Chapter 6

The Domain of an Algorithm

Since approximation algorithms are generally used to solve the scheduling problems described in this thesis, it is often useful to know something about the relationship between the obtained schedule and an optimal schedule for the same problem instance. The previous chapter dealt with one aspect of this issue: it provided various upper bounds on $L_{max}(\eta)$ in terms of some parameters of the considered problem instance, including $L_{max}(\omega)$. Another aspect to study is the conditions under which the schedules generated are optimal, which is the subject of this chapter for the $P|prec, pmtn|L_{max}$ problem, i.e. the problem with non-malleable tasks.

Several well known results have been obtained in this area: from [71] the Brucker-Garey-Johnson Algorithm is known to produce an optimal schedule whenever the partial orders are in the form of an in-tree (or an in-forest); from [71] and [120] respectively it can be seen that the Garey-Johnson and Zinder-Roper Algorithms produce an optimal schedule whenever $m \leq 2$, and consequently, whenever the precedence constraint graph has width at most three.

None of the algorithms developed for the considered problems have an exhaustive description of all solvable precedence constraints, here known as the domain of an algorithm. The domain $\mathcal{D}^{\mathscr{A}}$ of an algorithm $\mathscr{A}$ is defined as the class of unlabelled partial orders such that

1. if $\prec \in \mathcal{D}^{\mathscr{A}}$, for each problem instance φ with precedence constraints isomorphic to $\prec$ the schedule $\eta^{\mathscr{A}}$, produced by applying algorithm $\mathscr{A}$ to φ, minimises the maximum lateness; and

2. if $\prec \notin \mathcal{D}^{\mathscr{A}}$, there exists at least one problem instance φ with precedence constraints isomorphic to $\prec$ such that the schedule $\eta^{\mathscr{A}}$, produced by applying algorithm $\mathscr{A}$ to φ, does not minimise the maximum lateness.

In this chapter, the domain of the Brucker-Garey-Johnson, Garey-Johnson and

Zinder-Roper Algorithms for the $P|prec, pmtn|L_{max}$ problem will be derived. To the author's knowledge, these results are the first of their kind ever to be derived.

6.1 Schedule Structure

For the purpose of this chapter, it is convenient to use the notation H_i introduced in Section 4.4. As before, it will be assumed that (4.15) holds for all algorithms considered.

The lemma below is a simple modification of Lemma 4.26, hence the proof is omitted. The differences are that

- an arbitrary priority ρ is not considered; and
- it is supposed that j has a high priority at $t_{i'+1}$, rather than at $t_{i'}$.

Lemma 6.1 *If all μ's obey (4.15), for any $0 \leq i < i' < s$ and any $j \in H_{i'}$, it follows that either $j \in H_i$ or there exists a task $g \in H_i$ such that $g \rightarrow j$.*

Suppose there is some problem instance for which the considered algorithm does not provide the optimal solution for the alternate criterion G. The two lemmas below provide an insight into the structure of the resulting schedule. Specifically, they demonstrate that any schedule η produced by Algorithm P and which is not optimal for G must have $|H_a| > m$, $|H_b| < m$ and $|H_c| > m$ for $0 \leq a < b < c < i^*$. Without loss of generality, let $|H_i| \geq m$ for all $b < i < i^*$ and let $|H_i| \leq m$ for all $a < i < c$. These constraints imply that

- b is the largest $i < i^*$ for which $|H_i| < m$,
- c is the smallest $i > b$ for which $|H_i| > m$ and
- a is the largest $i < b$ for which $|H_i| > m$.

Lemma 6.2 *For some integer $i' \in [0, i^*)$, if $|H_i| \geq m$ for all integers $i \in [i', i^*)$, and if there exists some schedule σ such that*

$$p_j(t_{i'}, \sigma) \geq p_j(t_{i'}, \eta) \qquad \text{for all } j \in \bigcup_{i=i'}^{i^*-1} H_i \tag{6.1}$$

it follows that $G(\sigma) \geq G(\eta)$.

Proof. Suppose there exist i' and σ as defined in the statement of the lemma. By Corollary 4.25, the total amount of processing time allocated to the tasks in $\cup_{i=i'}^{i^*-1} H_i$ in schedule η during the time interval $[t_{i'}, t_{i^*}]$ is $m(t_{i^*} - t_{i'})$, or equivalently

$$\sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i'}, \eta) - \sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i^*}, \eta) = m(t_{i^*} - t_{i'}). \tag{6.2}$$

Combining (6.2) with (6.1) gives

$$\sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i'}, \sigma) - \sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i^*}, \eta) \geq \sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i'}, \eta) - \sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i^*}, \eta)$$

$$= m(t_{i^*} - t_{i'}) \geq \sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i'}, \sigma) - \sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i^*}, \sigma)$$

and hence

$$\sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i^*}, \sigma) \geq \sum_{j \in \cup_{i=i'}^{i^*-1} H_i} p_j(t_{i^*}, \eta). \tag{6.3}$$

If there exists a task $g \in \cup_{i=i'}^{i^*-1} H_i$ such that $p_g(t_{i^*}, \sigma) > p_g(t_{i^*}, \eta)$, it follows that $C_g(\sigma) > t_{i^*}$. On the other hand, if no such task exists, (6.3) implies that $p_j(t_{i^*}, \sigma) = p_j(t_{i^*}, \eta)$ for all $j \in \cup_{i=i'}^{i^*-1} H_i$. Hence, in either case there exists a task $q \in \cup_{i=i'}^{i^*-1} H_i$ such that

$$C_q(\sigma) \geq t_{i^*} + p_q(t_{i^*}, \sigma) \qquad \text{and} \qquad p_q(t_{i^*}, \sigma) \geq p_q(t_{i^*}, \eta). \tag{6.4}$$

Let i'' be the largest value of $i < i^*$ such that $q \in H_i$. With this definition, (6.2) implies that q is not processed during $[t_{i''+1}, t_{i^*}]$. Thus,

$$p_q(t_{i^*}, \eta) + \mu_q = p_q(t_{i''+1}, \eta) + \mu_q = \zeta_q(t_{i''+1}, \eta) \geq \rho^*.$$

As a consequence, (6.4) implies

$$G(\sigma) \geq C_q(\sigma) + \mu_q \geq t_{i^*} + p_q(t_{i^*}, \sigma) + \mu_q$$

$$\geq t_{i^*} + p_q(t_{i^*}, \eta) + \mu_q \geq t_{i^*} + \rho^* = G(\eta),$$

and the lemma is proven. ■

Lemma 6.3 *If all μ's obey (4.15), and if there exists an integer $i' \in [0, i^*)$ such that $|H_i| \leq m$ for all integers $i \in [0, i')$ and $|H_i| \geq m$ for all integers $i \in [i', i^*)$, η minimises the criterion G.*

Proof. Lemma 4.27 implies that $H_i \subseteq N_1^i$ for all $i < i'$. From (4.7) the tasks in N_1^i are processed continuously during the entire interval $[t_i, t_{i+1}]$.

The lemma will now be proven by induction on i. First note that the relation $p_j(0, \eta) \leq p_j(0, \sigma)$ holds for all tasks $j \in N$ and all schedules σ. Now assume that, for some $i < i'$, $p_j(t_i, \eta) \leq p_j(t_i, \sigma)$ holds for all tasks $j \in \cup_{\iota=i}^{i^*-1} H_\iota$ and all schedules σ. For any $j \in H_i$, the fact that $j \in N_1^i$ implies that

$$p_j(t_{i+1}, \eta) \leq p_j(t_{i+1}, \sigma) \text{ for all schedules } \sigma. \tag{6.5}$$

By (4.15) and Lemma 6.1, the set $\cup_{\iota=i+1}^{i^*-1} H_\iota$ can be partitioned into two (possibly empty) subsets: tasks in H_i and successors of tasks in H_i. For any $j \in H_i$, (6.5) holds. For any $g \in K_j$ for some $j \in H_i$, (6.5) demonstrates that $p_g(t_{i+1}, \sigma) = p_g$ for all schedules σ.

As a consequence, $p_j(t_{i'}, \eta) \leq p_j(t_{i'}, \sigma)$ for all $j \in \cup_{\iota=i'}^{i^*-1} H_\iota$ and all schedules σ. The lemma now follows from Lemma 6.2. ■

The corollary below follows from the lemma above and Corollary 4.30.

Corollary 6.4 *If all μ's obey (4.15) and if $m = 1$, η minimises the criterion G.*

All of the reasoning above, along with Corollary 4.31, combines to demonstrate the existence of the indices a, b and c as defined earlier.

Lemma 6.5 *If all μ's obey (4.15), either η minimises the criterion G, or there exist $0 \leq a < b < c < i^*$ such that $|H_a| > m$, $|H_b| < m$ and $|H_c| > m$.*

Proof. If η does not minimise G, Lemma 4.22 implies $i^* > 0$ and therefore Corollary 4.31 implies $|H_{i^*-1}| > m$. By Lemma 6.3, if there exists an integer $i' \in [0, i^*)$ such that $|H_i| \leq m$ for $i \in [0, i')$ and $|H_i| \geq m$ for $i \in [i', i^*)$, η minimises G. Hence, the lemma is proven. ■

As stated earlier, these values may be selected, without loss of generality, such that b is the largest $i < i^*$ for which $|H_i| < m$, c is the smallest $i > b$ for which $|H_i| > m$ and a is the largest $i < b$ for which $|H_i| > m$. The purpose of this reasoning is to understand the nature of a non-optimal schedule. The composition of the sets H_i in such a schedule will be shown to imply a certain structure of precedence constraints; thus, if this structure is absent, the schedule η must be optimal.

6.2 The Brucker-Garey-Johnson Algorithm

In [71] it was demonstrated that the Brucker-Garey-Johnson Algorithm will optimally schedule all problem instances where the precedence constraints take the

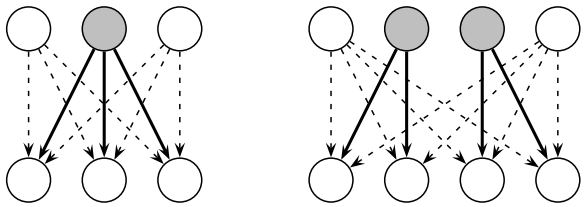


Figure 6.1: The two classes of prohibited subgraphs for $\mathcal{D}^{\mathscr{BGJ}}$.

form of an in-tree (or in-forest). This implies that all in-forests are in the domain of the Brucker-Garey-Johnson Algorithm. However, until now it was not known what other partial orders are in this domain. The domain $\mathcal{D}^{\mathscr{BGJ}}$ of the Brucker-Garey-Johnson Algorithm for the $P|prec, pmtn|L_{max}$ problem is presented in this section. To the knowledge of the author this is the first result of this kind ever discovered or published. The material of this section was originally published in [114], and has been expanded upon in this thesis.

6.2.1 The Class of Partial Orders

It will now be proved that the domain of the Brucker-Garey-Johnson Algorithm is the set of all partial orders not containing a subgraph isomorphic to any member of either one of two classes of prohibited subgraph, described below. Using the terminology introduced in Section 3.1, $\mathcal{D}^{\mathscr{BGJ}}$ can be viewed as the class of all partial orders that are P-free for all partial orders P in the two classes described below.

The first class is that of graphs with six nodes, divided into two sets A_1 and B_1, each containing three independent nodes. One node in A_1, which will be called the key node, precedes all three nodes in B_1. Any precedence constraints between the other two nodes in A_1 and the three nodes in B_1 are allowed in this class of graphs.

The second class is that of graphs with eight nodes, divided into two sets A_2 and B_2, each containing four independent nodes. Two nodes in A_2, which will be called the key nodes, precede disjoint sets, each of two nodes, in B_2 in such a way that each precedes only one of these two sets. Any precedence constraints between the other two nodes in A_2 and the four nodes in B_2 are allowed in this class of graphs.

These classes of prohibited subgraphs are represented in Figure 6.1, with the mandatory precedence constraints as solid lines and the optional constraints as dashed lines. The key nodes are shaded.

There are thirteen isomorphically distinct (i.e. unlabelled) prohibited subgraphs

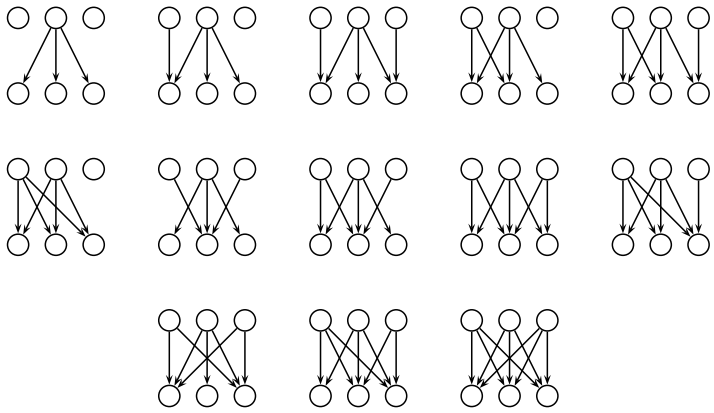


Figure 6.2: The thirteen prohibited subgraphs of the first kind.

in the first class, shown in Figure 6.2. Some graphs in the second class contain a graph from the first class as a subgraph, leaving seventeen isomorphically distinct graphs in the second class that do not contain a subgraph isomorphic to any member of the first class. These are represented in Figure 6.3. Consequently, thirty prohibited subgraphs are needed to specify the domain of the Brucker-Garey-Johnson Algorithm.

The first thing to note is that each prohibited subgraph itself contains a subgraph consisting of three nodes j_1, j_2 and j_3 with two order relations $j_1 \rightarrow j_2$ and $j_1 \rightarrow j_3$. This is the prohibited graph which defines the class of in-forests. Consequently all in-forests are in the domain of the Brucker-Garey-Johnson Algorithm, as was proven in [71]. Note, however, that many graphs that are not in-forests are also in the domain of the considered algorithm, indicating a broadening of the well known classical result. Additionally, to the author's knowledge this is the first time the domain of an algorithm has been completely specified.

6.2.2 Proving Sufficiency

The theorem below uses the results of Section 6.1 to demonstrate that a lack of the two prohibited subgraphs described in Subsection 6.2.1 is sufficient to guarantee that the Brucker-Garey-Johnson Algorithm will produce an optimal schedule.

Theorem 6.6 *For any instance of the $P|prec, pmtn|L_{max}$ problem where the precedence constraints do not contain a prohibited subgraph of the first or second kind, the Brucker-Garey-Johnson Algorithm produces an optimal schedule.*

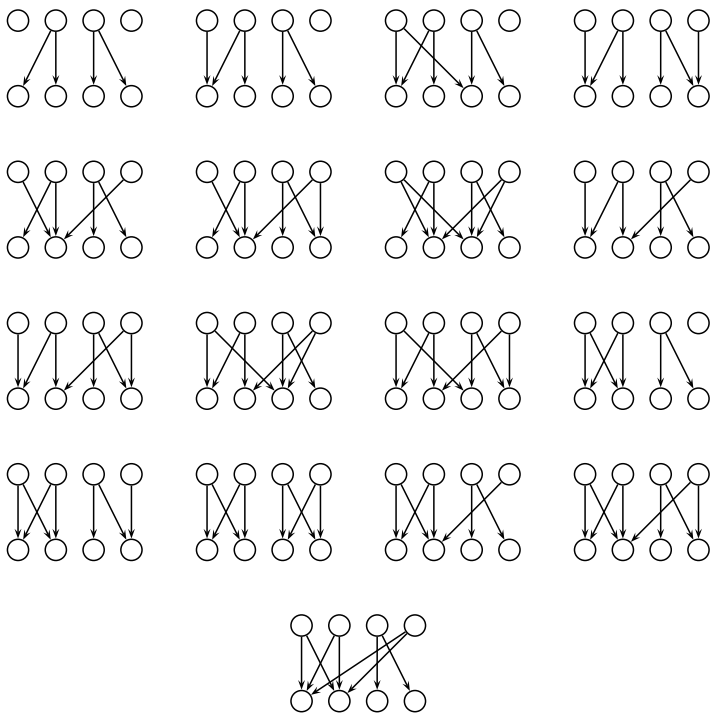


Figure 6.3: The seventeen prohibited subgraphs of the second kind that do not contain an induced subgraph of the first kind.

Proof. By Lemma 6.5 the non-optimal schedule must have some $0 \leq a < b < c < i^*$ such that $|H_b| < m$, $|H_a| > m$ and $|H_c| > m$.

Of all tasks $j \in H_b$ select as $\hat{\jmath}$ a task that precedes the largest number of tasks from H_c. Since

$$|H_c| \geq m + 1 \geq |H_b| + 2, \tag{6.6}$$

it follows from Lemma 6.1 that $|K_{\hat{\jmath}} \cap H_c| \geq 2$. Lemma 6.1 also implies that either $j = \hat{\jmath}$ or $j \to \hat{\jmath}$ for some $j \in H_a$, so at least one task $j \in H_a$ has $|K_j \cap H_c| \geq 2$. Select as $k_1 \in H_a$ the task with the largest $|K_{k_1} \cap H_c|$. From Lemma 4.11 and Corollary 6.4 a problem instance creating a non-optimal schedule must have $m \geq 2$, which implies $|H_c| \geq 3$ and $|H_a| \geq 3$. Consequently, if $|K_{k_1} \cap H_c| \geq 3$ then the problem instance contains a prohibited subgraph of the first kind, with k_1 as the key task.

On the other hand, if $|K_{k_1} \cap H_c| = 2$ this implies that $|K_{\hat{\jmath}} \cap H_c| = 2$. As a consequence of (6.6) there must exist $\hat{g} \in H_b \setminus \{\hat{\jmath}\}$ that precedes two tasks in $H_c \setminus K_{\hat{\jmath}}$. This implies that

$$|H_a| > m > |H_b| \geq |\{\hat{\jmath}, \hat{g}\}| = 2$$

and thus $|H_a| \geq 4$ and $m \geq 3$. Again from Lemma 6.1 either $j = \hat{g}$ or $j \to \hat{g}$ for some $j \in H_a \setminus \{k_1\}$. Let $k_2 \in H_a \setminus \{k_1\}$ be such a task. Since $|H_a| \geq 4$, $|K_{k_1} \cap H_c| = 2$, $|K_{k_2} \cap H_c| = 2$ and $K_{k_1} \cap K_{k_2} \cap H_c = \emptyset$, this problem instance contains a prohibited subgraph of the second kind, with k_1 and k_2 as the key tasks. ■

6.2.3 Proving Necessity

This subsection takes an arbitrary partial order containing a subgraph from one of the two prohibited classes and uses it to construct a problem instance for which the Brucker-Garey-Johnson Algorithm will not minimise the maximum lateness. It is able to do this by "removing" certain inconvenient tasks by allocating a large due date and a small processing time to them, leaving only the prohibited subgraph.

Consider any partial order P containing a subgraph Q isomorphic to a prohibited subgraph of either the first or the second kind, and select as X the set of nodes of Q. Select as B the corresponding set of key nodes. If Q is a prohibited subgraph of the first kind $|X| = 6$ and $|B| = 1$; if it is a prohibited subgraph of the second kind $|X| = 8$ and $|B| = 2$. Partition X into two sets: X_1 is composed of the successors of the key nodes, while X_0 is composed of the key nodes and all elements of X that are independent of the key nodes. By these definitions, X_0 corresponds to the "top row" of nodes depicted in Figure 6.1, X_1 corresponds to the "bottom row" of nodes,

and B corresponds to the shaded nodes. Note that $|X_0 \setminus B| = 2$, regardless of the prohibited subgraph Q under consideration.

Consider a problem instance φ defined as follows.

- The precedence constraints of φ are given by the partial order P;
- $m = |B| + 1$;
- all tasks $j \in N \setminus X$ have $p_j = \varepsilon$ and $d_j = 3 + 2n\varepsilon$, where $\varepsilon > 0$ is a constant that can be arbitrarily small;
- all tasks $j \in X \setminus B$ have $p_j = 1$;
- all tasks $j \in B$ have $p_j = d_j = 2$;
- all tasks $j \in X_0 \setminus B$ have $d_j = 1$; and
- all tasks $j \in X_1$ have $d_j = 3 + n\varepsilon$.

Since all but the tasks in X, i.e. the tasks corresponding to the prohibited subgraph Q, have been given an arbitrarily small processing time, schedules for the problem instance φ will be dominated by the processing of the tasks corresponding to the prohibited subgraph.

In the remainder of this subsection, bounds will be established on $L_{max}\left(\eta^{\mathscr{BGJ}}\right)$ and $L_{max}(\omega)$, demonstrating that $\eta^{\mathscr{BGJ}}$ is not an optimal schedule. This will be described using the notation

$$K(S) = \bigcup_{j \in S} K_j \qquad \text{and} \qquad J(S) = \bigcup_{j \in S} J_j,$$

for any set of tasks S.

Lemma 6.7 *For an optimal schedule ω for the problem instance φ as defined above,*

$$L_{max}(\omega) \leq n\varepsilon + \frac{2m-1}{m^2}.$$

Proof. Construct the schedule σ defined as follows. Schedule all tasks in $J(X_0)$ arbitrarily during the time interval $[0, n\varepsilon]$. Use the Modified McNaughton Algorithm to process to completion both tasks in $X_0 \setminus B$, and to process $1\frac{1}{m}$ units of each task in B, during the time interval

$$\left[n\varepsilon, n\varepsilon + 1 + \frac{2m-1}{m^2}\right],$$

giving

$$C_j(\sigma) - d_j \leq n\varepsilon + \frac{2m-1}{m^2} \qquad \text{for both } j \in X_0 \setminus B.$$

Schedule the remaining $\frac{m-1}{m}$ units of processing time of each task in B during the time interval

$$\left[n\varepsilon + 1 + \frac{2m-1}{m^2}, n\varepsilon + 2 + \frac{m-1}{m^2}\right],$$

giving

$$C_j(\sigma) - d_j \leq n\varepsilon + \frac{m-1}{m^2} \qquad \text{for each } j \in B.$$

Schedule all tasks in $J(X_1) \setminus (X_0 \cup J(X_0))$ arbitrarily during the time interval

$$\left[n\varepsilon + 2 + \frac{m-1}{m^2}, 2n\varepsilon + 2 + \frac{m-1}{m^2}\right].$$

Use the Modified McNaughton Algorithm to process to completion all tasks in X_1 during the time interval

$$\left[2n\varepsilon + 2 + \frac{m-1}{m^2}, 2n\varepsilon + 3 + \frac{2m-1}{m^2}\right],$$

giving

$$C_j(\sigma) - d_j \leq n\varepsilon + \frac{2m-1}{m^2} \qquad \text{for each } j \in X_1.$$

Finally, schedule all tasks in $N \setminus (X \cup J(X))$ arbitrarily during the time interval

$$\left[2n\varepsilon + 3 + \frac{2m-1}{m^2}, 3n\varepsilon + 3 + \frac{2m-1}{m^2}\right]$$

giving

$$C_j(\sigma) - d_j \leq n\varepsilon + \frac{2m-1}{m^2} \qquad \text{for each } j \in X_1.$$

Since the maximum lateness of σ is an upper bound on the optimal maximum lateness,

$$L_{max}(\omega) \leq L_{max}(\sigma) \leq n\varepsilon + \frac{2m-1}{m^2},$$

and the lemma is proven. ■

It now remains to establish a lower bound on $L_{max}\left(\eta^{\mathscr{BGJ}}\right)$, the maximum lateness of the schedule produced by the Brucker-Garey-Johnson Algorithm when it is applied to the problem instance φ. In order to do this it is necessary to study the values of μ's computed by the Brucker-Garey-Johnson Algorithm.

Recall that a chain of tasks is a set $S \subseteq N$ such that no two elements of S are independent. Define l_j to be the length of the longest chain that is a subset of K_j, i.e.

$$\max_{S \subseteq K_j : S \text{ is a chain}} \sum_{g \in S} p_g.$$

The following lemma, used in subsequent corollaries to derive bounds for many of the μ's computed by the Brucker-Garey-Johnson Algorithm, expresses a bound on μ_j in terms of l_j.

Lemma 6.8 *For any task j, either $\mu_j = -d_j$ or*

$$\mu_j \leq l_j - \min_{q \in K_j} d_q.$$

Proof. The lemma will be proven by induction on $|K_j|$. If $|K_j| = 0$, $\mu_j = -d_j$ and the lemma holds. Now suppose that there is some $k \geq 0$ for which the lemma holds when $|K_j| \leq k$, and consider the case where $|K_j| = k + 1$. Since the lemma holds if $\mu_j = -d_j$, (4.16) implies that only the case where $\mu_j = p_g + \mu_g$ for some $g \in K_j$ remains to be considered; for the remainder of the proof it will be assumed that this is the case.

The remainder of the proof involves the consideration of two cases, based on the value of μ_g.

- If $\mu_g = -d_g$,
$$\mu_j = p_g + \mu_g = p_g - d_g \leq l_j - \min_{q \in K_j} d_q,$$
and the lemma holds.

- On the other hand, if $\mu_g \neq -d_g$ it follows that that $\mu_g > -d_g$. By the inductive assumption and the fact that $K_g \subset K_j$, this means
$$\mu_j = p_g + \mu_g \leq p_g + l_g - \min_{q \in K_g} d_q \leq l_j - \min_{q \in K_j} d_q,$$
and the lemma holds in this case as well.

In both cases the assumption that the lemma holds for $|K_j| \leq k$ is sufficient to prove that the lemma also holds for $|K_j| = k + 1$. Since the lemma holds for $|K_j| = 0$, it has been proven by induction. ■

Corollary 6.9 *For any tasks j for which $K_j \cap X = \emptyset$, either $\mu_j = -d_j$ or*

$$\mu_j < -3 - n\varepsilon.$$

Corollary 6.10 *For all $j \in X_1$, $\mu_j = -d_j$.*

Corollary 6.11 *For any tasks j for which $(K_j \cap X) \subseteq X_1$, either $\mu_j = -d_j$ or $\mu_j < -2 - n\varepsilon$.*

Corollary 6.12 *For both $j \in X_0$, $\mu_j = -d_j$.*

Notice that Corollary 6.12 implies that all tasks in X_0 have the same priority at time 0. However, the tasks in $J(X_0)$ can disrupt the processing of the tasks in X_0, causing their priorities to diverge. Lemma 6.13 examines the processing of the tasks in $J(X_0)$, and Lemma 6.14 investigates the effect this has on the priorities of the tasks in X_0.

Lemma 6.13 *If $J(X_0) \neq \emptyset$, there exists a point of allocation $t_{\bar{\imath}}$ such that*

$$t_{\bar{\imath}} = \max_{q \in J(X_0)} C_q\left(\eta^{\mathscr{BGJ}}\right) < n\varepsilon.$$

Proof. Consider any $j \in J(X_0)$, any $g \in K_j \cap X_0$ and any i for which $j \in A_i$. Corollary 6.12 and (4.15) imply

$$\zeta_j\left(t_i, \eta^{\mathscr{BGJ}}\right) = p_j\left(t_i, \eta^{\mathscr{BGJ}}\right) + \mu_j > \mu_j \geq p_g + \mu_g = 0.$$

All the corollaries preceding this lemma demonstrate that, for any $q \in N \setminus J(X_0)$, $\zeta_j\left(t_i, \eta^{\mathscr{BGJ}}\right) \leq 0$.

If $|J(X_0) \cap A_i| < m$, Lemma 4.27 implies $(J(X_0) \cap A_i) \subseteq N_1^i$ and $C_j\left(\eta^{\mathscr{BGJ}}\right) \geq t_{i+1}$. On the other hand, if $|J(X_0) \cap A_i| \geq m$, Lemma 4.24 implies $(N_1^i \cup N_2^i) \subseteq (J(X_0) \cap A_i)$ and there exists a task $j' \in J(X_0) \cap A_i$ such that $C_{j'}\left(\eta^{\mathscr{BGJ}}\right) \geq t_{i+1}$. In either case, at least one task from $J(X_0)$ is being processed at every point in time during $[t_i, t_{i+1}]$, and at least one task from $J(X_0)$ is completed no earlier than t_{i+1}. Applying this to all i' for which $J(X_0) \cap A_{i'} \neq \emptyset$ proves the lemma. ■

Lemma 6.14 *If ε is sufficiently small, there exists a point of allocation $t_{\hat{\imath}}$ such that*

$$p_j\left(t_{\hat{\imath}}, \eta^{\mathscr{BGJ}}\right) = p_j - 1$$

for all $j \in X_0$.

Proof. If $J(X_0) = \emptyset$, the lemma holds because $N_2^0 = X_0$. If $J(X_0) \neq \emptyset$, Lemma 6.13 implies that

$$p_j - n\varepsilon < p_j\left(t_{\bar{\imath}}, \eta^{\mathscr{BGJ}}\right) \leq p_j \qquad \text{for all } j \in X_0. \tag{6.7}$$

For a sufficiently small value of ε, this implies $X_0 \subseteq A_{\bar{\imath}}$.

For any i for which $X_0 \subseteq A_i$ and

$$p_j\left(t_i, \eta^{\mathscr{BGJ}}\right) > p_j - 1 \qquad \text{for all } j \in X_0, \tag{6.8}$$

the corollaries of Lemma 6.8 imply that, for sufficiently small ε, and for all $j \in X_0$ and $g \in N \setminus (X_0 \cup J(X_0) \cup K(X_0))$,

$$\zeta_j\left(t_i, \eta^{\mathscr{BGJ}}\right) > -1 > \varepsilon - 2 - n\varepsilon > \zeta_g\left(t_i, \eta^{\mathscr{BGJ}}\right).$$

Hence, Lemma 4.24, (6.8) and the fact that $|X_0| > m$ imply $(N_1^i \cup N_2^i) \subseteq X_0$, i.e. all tasks that are independent of those in X_0, and thus might compete for processing time, always have strictly lower priority than all tasks in X_0.

By Lemma 4.7, for all $i \geq \bar{\imath}$ for which (6.8) holds,

$$|N_2^i \cap X_0| \geq |N_2^{\bar{\imath}} \cap X_0| + i - \bar{\imath}.$$

Hence, there exists some integer $i \in [\bar{\imath}, \bar{\imath} + |X_0|]$ such that either (6.8) does not hold, or $N_2^i = X_0$. In the latter case, the lemma is proven, so now the former case will be addressed.

Recall that Lemma 4.3 establishes an upper bound on the distance between consecutive points of allocation t_i and t_{i+1} in terms of ξ_i, the smallest difference in priority between tasks in different sets N_1^i, N_2^i and N_3^i. In this case, (6.7), (6.8) and Lemma 4.9 imply that $\xi_i \leq \xi_{\bar{\imath}} < n\varepsilon$, so Lemma 4.3 implies

$$t_{i+1} - t_i \leq mn\xi_i < mn^2\varepsilon.$$

If $t_i \leq 1$, (6.8) holds. Hence, for a sufficiently small value of ε, (6.8) holds for all i until $N_2^i = X_0$, and the lemma is proven. ■

Lemma 6.14 will now be used to establish a lower bound on the maximum lateness of the schedule produced by the Brucker-Garey-Johnson Algorithm when it is applies to the problem instance φ.

Lemma 6.15 *For a sufficiently small value of ε,*

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) \geq \frac{2}{m} - n\varepsilon.$$

Proof. Lemma 6.14 implies

$$p_j\left(t_i, \eta^{\mathscr{BGJ}}\right) = 1$$

for all $j \in B$, and also that

$$t_i \geq \frac{|X_0|}{m} = \frac{m+1}{m}.$$

Together, these imply that

$$C_j\left(\eta^{\mathscr{BGJ}}\right) \geq 2 + \frac{1}{m} \qquad \text{for all } j \in B.$$

Since every $g \in X_1$ is preceded by some $j \in B$, this implies that

$$C_q\left(\eta^{\mathscr{BGJ}}\right) \geq 3 + \frac{2}{m} \qquad \text{for some } q \in X_1.$$

As a consequence of

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) \geq C_q\left(\eta^{\mathscr{BGJ}}\right) - d_q,$$

the lemma is proven. ■

Finally, the theorem giving the necessary conditions for a partial order's inclusion in $\mathcal{D}^{\mathscr{BGJ}}$ can be stated.

Theorem 6.16 *For any partial order that contains a prohibited subgraph of the first or second kind there exists an assignment of m, p_j and d_j such that the Brucker-Garey-Johnson Algorithm produces a non-optimal schedule.*

Proof. By Lemma 6.7 and Lemma 6.15, for a sufficiently small value of ε,

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) \geq \frac{2}{m} - n\varepsilon > n\varepsilon + \frac{2m-1}{m^2} \geq L_{max}\left(\omega\right).$$

Hence, $\eta^{\mathscr{BGJ}}$ is not an optimal schedule, and the theorem is proven. ■

Theorem 6.6 and Theorem 6.16 combine to demonstrate that $\mathcal{D}^{\mathscr{BGJ}}$, the domain of the Brucker-Garey-Johnson Algorithm, is precisely the class of partial orders that do not include a prohibited subgraph of the first or second kind.

6.3 The Zinder-Roper Algorithm

This section analyses the domain of the Zinder-Roper Algorithm for the $P|prec, pmtn|L_{max}$ problem. An additional result is also obtained: recall that in Section 5.3 it was demonstrated that the Garey-Johnson Algorithm in a sense "dominates" the family $\mathfrak{F} \cap \mathfrak{S}$ of algorithms, as defined in Section 4.1, in terms of a particular measure of worst case performance. An analogous result is presented in this section, in which it is demonstrated that $\mathcal{D}^{\mathscr{A}} \subseteq \mathcal{D}^{\mathscr{ZR}}$ for all algorithms $\mathscr{A} \in \mathfrak{F}$. In other

words, this means that every precedence constraint graph for which any algorithm in $\mathfrak{F}$ guarantees an optimal solution is also guaranteed an optimal solution by the Zinder-Roper Algorithm. Unlike the analysis in Section 5.3, the requirement that the algorithm be stable, i.e. that it be a member of the family $\mathfrak{S}$, is not required.

As in the case of performance guarantee dominance, this is an unprecedented result which has important implications for the future of research. The combination of these two results in some sense implies that the family of algorithms $\mathfrak{F} \cap \mathfrak{S}$ that is investigated in this thesis can yield no further improvements: the best possible algorithms, in terms of solvable cases and worst case performance, are already known. Consequently, future research into improved algorithms must search outside this family.

6.3.1 The Class of Partial Orders

A description of the domain of the Zinder-Roper Algorithm involves the notion of a singular quartet, which is a set of tasks defined in terms of the precedence constraints $\rightarrow$. The definition of a singular quartet requires several concepts defined as follows.

A subset $S \subseteq N$ is splittable if the corresponding subgraph, i.e. the subgraph of $\rightarrow$ composed only of tasks from S, can be expressed as the series composition of two nonempty partial orders. In other words, it is splittable if S can be expressed as the union $S_1 \cup S_2$ where $S_1 \neq \emptyset$, $S_2 \neq \emptyset$, and $j \rightarrow g$ for all $j \in S_1$ and $g \in S_2$.

A set of tasks $Y \subseteq N$ will be known as a "transitive set" if and only if, for any $g \in Y$, any $q \in Y$ and any $j \in N$, the relation $g \rightarrow j \rightarrow q$ implies $j \in Y$. Any transitive set Y will be said to be "covered by" a set X, and X will be said to "cover" Y, if and only if

- the tasks in X are independent,
- $X \subseteq Y$, and
- $Y \subseteq (X \cup K(X))$,

i.e. Y is covered by the set of minimal elements of Y.

An "adequate transitive set" is defined to be any transitive set Y such that

- Y is not splittable and
- the width of Y is at least four.*

*This is a slight abuse of terminology, as a transitive set is simply a set of tasks, while the term "width" is properly applied only to partial orders. In order to simplify the exposition, the width of a set of tasks Y should be interpreted as the width of the suborder of $\rightarrow$ induced by Y.

A set X of two independent tasks will be known as a "singular pair" if and only if it covers at least one adequate transitive set. A set Z of four independent tasks will be known as a "singular quartet" if and only if there exists a singular pair X such that $X \subset Z$. In this section, the domain $\mathcal{D}^{\mathscr{ZR}}$ of the Zinder-Roper Algorithm will be proven to be the class of all partial orders that do not contain a singular quartet.

Example 6.17 *Recall the problem instance defined in Example 1.1 on page 17, and note in particular the precedence graph (i.e. partial order) depicted in Figure 1.1.* Define the set of tasks*

$$Y = \{2, 3, 4, 5, 7, 10, 11, 12, 14\}.$$

The set Y is a transitive set, while $\{2, 3, 5, 7\}$ is not, because $2 \to 4 \to 7$. The set $\{2, 10\}$ covers Y. Set $\{1, 10\}$ does not cover Y because $1 \notin Y$. Set $\{3, 4, 5, 10\}$ does not cover Y because

$$2 \notin (\{3, 4, 5, 10\} \cup K(\{3, 4, 5, 10\})).$$

Set $\{2, 3, 10\}$ does not cover Y because 2 and 3 are not independent.

Set Y is not splittable, which can be seen with the following reasoning. Suppose Y can be expressed as the union $S_1 \cup S_2$ where $S_1 \neq \emptyset$, $S_2 \neq \emptyset$, and $j \to g$ for all $j \in S_1$ and $g \in S_2$. If $2 \in S_2$, $j \to 2$ for all $j \in S_1$. Since no task in Y precedes 2, this would imply that $S_1 = \emptyset$, contradicting the definition of splittability. Hence, $2 \in S_1$, and since $2 \not\to 14$, $14 \in S_1$. Since no task in Y is preceded by both 2 and 14, $S_2 = \emptyset$, again contradicting the definition of splittability, and proving that Y is not splittable.

In contrast, the set $\{2, 3, 4, 5, 7\}$ is splittable; specifically, $\{2, 3, 4, 5, 7\}$ can be partitioned into the sets $S_1 = \{2\}$ and $S_2 = \{3, 4, 5, 7\}$. It can be seen that $K_2 \subseteq S_2$.

The width of set Y is 6, corresponding to the independent tasks 3, 4, 5, 11 and 12. This means that $\{2, 10\}$ is singular because

- *$\{2, 10\}$ covers Y; and*
- *Y is neither splittable, nor does it have width at most three.*

One interesting observation is that the version of the Zinder-Roper Algorithm used for the non-preemptive, UET task problem $P|prec, p_j = 1|L_{max}$ will solve all

*Note that, although this is an instance of $P|prec, pmtn, u_j|L_{max}$, i.e. $u_j > 1$ for some tasks j, the properties described above are properties of a partial order only – they do not depend on any other parameters of a problem instance.

problems where the precedence constraints are in the form of an interval order, while it can be seen that not all interval orders are in $\mathcal{D}^{\mathscr{ZR}}$ as defined above. This is a consequence of the additional freedom in the $P|prec, pmtn|L_{max}$ problem to change the processing times of tasks in order to create a problem instance that the algorithm will not solve.

6.3.2 Recognising the Prohibited Substructure

Although it is not a major focus of this thesis, it is useful to briefly consider how to determine if a given partial order $\prec$ is an element of $\mathcal{D}^{\mathscr{ZR}}$. A list of all sets of four independent tasks can be constructed in polynomial time by exhaustively testing all sets of four tasks. Once this has been done, each pair X of tasks within each set Y of four independent tasks can be examined to determine if there exists another set Z of four independent tasks where each task in $Z \setminus X$ is preceded by a task in X. This can also be done in polynomial time – specifically there are $O(n^4)$ sets of four independent tasks and thus $O(n^8)$ possible selections of X, Y and Z.

If sets X, Y and Z defined as above are found, the set

$$S = (X \cup K(X)) \cap (Z \cup J(Z))$$

is a transitive set; if no such X, Y and Z are found, $\prec \in \mathcal{D}^{\mathscr{ZR}}$. If any S defined above is not splittable, it follows that X and Y are singular sets, and thus $\prec \notin \mathcal{D}^{\mathscr{ZR}}$. If every set S defined above is splittable, $\prec \in \mathcal{D}^{\mathscr{ZR}}$.

Testing if S is splittable into S_1 and S_2 can be accomplished by considering the following observations. Firstly, if S is splittable, $X \in S_1$ and $Z \in S_2$, implying that all tasks in X must precede all tasks in Z. If if this is the case, the set S may or may not be splittable, and further testing is required. If it is not the case, set S can instantly be seen to be not splittable, and hence $\prec \notin \mathcal{D}^{\mathscr{ZR}}$.

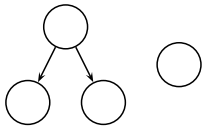
If there exists no task $j \in S \setminus X$ for which $|J_j \cap X| = 1$, the set S is splittable into X and $S \setminus X$, and the testing of S is completed. On the other hand, suppose there is such a task j and define

$$X' = (X \setminus J_j) \cup \{j\}.$$

If S is splittable, it follows that $j \in S_1$. Hence, S is splittable if and only if

$$S' = (X' \cup K(X')) \cap (Z \cup J(Z))$$

is also splittable.



M

Figure 6.4: A partial order on four elements, describing a new class of partial orders.

The test that was just applied to S can now be applied to the new set S'. Since $|S'| < |S|$, repeated application of this test will reveal whether or not S is splittable after a finite (and indeed polynomial in n) number of iterations.

The reasoning above also leads to the following theorem, relating to the computational complexity of different classes of precedence constraints, as considered in Chapter 3.1. Specifically, the class of M-free partial orders, where the partial order M is depicted in Figure 6.4, is considered, and it is demonstrated that it is a subclass of the domain $\mathcal{D}^{\mathscr{ZR}}$ of the Zinder-Roper Algorithm. According to [17], the class of M-free partial orders is in asymptotic growth family I – the slowest.

Theorem 6.18 *There exists no singular pair in any M-free partial order.*

Proof. Suppose there exists a singular pair X in the precedence constraints, and further suppose that Z is defined as above, i.e. Z is a set of four independent tasks, all of which are in X or preceded by at least one task in X. Consider the subgraph of the precedence constraints corresponding to $X \cup Z$ and note that there are nine isomorphically distinct possibilities, represented in Figure 6.5. Also in Figure 6.5, the nodes corresponding to a subgraph isomorphic to M are shaded in all graphs but the last.

In the final case, the algorithm described in this subsection will, after a finite number of iterations, find one of the other eight substructures somewhere within the subgraph induced by

$$S = (X \cup K(X)) \cap (Z \cup J(Z)).$$

As a consequence, if there exists a singular pair of tasks there also exists a subgraph isomorphic to M, and the theorem is proven. ■

The following corollary is derived from the theorem above and from the fact

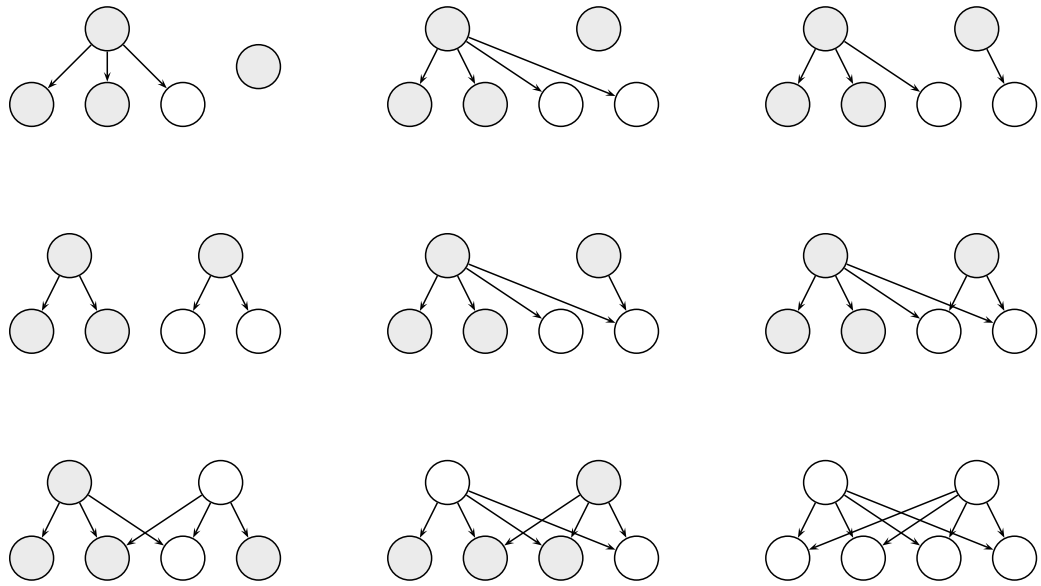


Figure 6.5: Possible subgraphs induced by a singular pair and four independent tasks.

that the partial order M has a subgraph isomorphic to the prohibited subgraph E describing the class of weak orders.

Corollary 6.19 *There exists no singular pair in any weak order.*

6.3.3 Proving Sufficiency

The aim of this subsection is to prove that the Zinder-Roper Algorithm is guaranteed to find an optimal schedule for $P|prec, pmtn|L_{max}$ if the partially ordered set of tasks does not contain a singular quartet. It will be accomplished in two stages. The first stage is a proof that if $\eta^{\mathscr{ZR}}$ does not minimise the criterion G, then the partially ordered set of tasks has a singular quartet. The second stage is a proof that if the Zinder-Roper Algorithm produces optimal schedules for all φ_j with criterion G, then the resultant schedule is also optimal for the instance of $P|prec, pmtn|L_{max}$.

The introduction of the sets H_i in Section 4.4 gives rise to a binary relation $\Rightarrow$ which can be described as follows: $g \Rightarrow j$ if and only if there exist $0 \leq i' < i'' < i^*$ and a sequence $q_{i'}$, ..., $q_{i''}$ such that

- the first element of this sequence is g and the last is j, i.e. $g = q_{i'}$ and $j = q_{i''}$;
- for each $i' \leq i \leq i''$, $q_i \in H_i$;

- for each $i' < i \leq i''$ either $q_{i-1} = q_i$ or $q_{i-1} \to q_i$.

The properties of the binary relation $\Rightarrow$ are examined in Lemma 6.20 and Lemma 6.21 below. In particular, Lemma 6.20 may be viewed as an extension of Corollary 4.28 and Lemma 6.1 to the new binary relation $\Rightarrow$.

Lemma 6.20 *For any schedule $\eta^{\mathscr{ZR}}$ produced by the Zinder-Roper Algorithm and which does not minimise the criterion G, for each $0 \leq i' < i'' < i^*$ and for any $j \in H_{i''}$, there exists $g \in H_{i'}$ such that $g \Rightarrow j$.*

Proof. This will be proven true by induction on $i'' - i'$. If $i'' - i' = 1$, the lemma follows directly from Lemma 6.1. Suppose that the lemma holds true for $i'' - i' = k$ for some $k \geq 1$, and consider the case where $i'' - i' = k+1$. The inductive assumption means that $q \Rightarrow j$ for some $q \in H_{i'+1}$.

If $q \in A_{i'}$ then

$$\zeta_q \left(t_{i'+1}, \eta^{\mathscr{ZR}} \right) \geq \zeta_q \left(t_{i'+2}, \eta^{\mathscr{ZR}} \right) \geq \rho^*$$

and so $q \in H_{i'}$ and the lemma is proven for $q = g$.

On the other hand, if $q \notin A_{i'}$ then Lemma 6.1 implies the existence of a task $g \in A_{i'}$ such that $g \to q$. Consequently (4.15) implies

$$\zeta_g \left(t_{i'+1}, \eta^{\mathscr{ZR}} \right) \geq \mu_g \geq p_q + \mu_q \geq \zeta_q \left(t_{i'+2}, \eta^{\mathscr{ZR}} \right) \geq \rho^*$$

and the lemma holds in this case as well. ■

The next lemma establishes an important property of the binary relation $\Rightarrow$. It is similar to Lemma 5.41: it proves a much stronger kind of non-precedence, but only for a restricted set of tasks. The statement of this lemma requires the following notation. For any $j \in H$, the set of all g such that $j \Rightarrow g$ will be denoted by $U(j)$. The lemma below demonstrates that, for all $j \in H_a$, there exists a task $g \in H_c$ such that $j \not\Rightarrow g$. This is proven by demonstrating that the rigid structure implied by the binary relation $\Rightarrow$ and the fact that $|H_i| \leq m$ for all $a < i < c$ constrains the value of μ_j in a way that contradicts the definition of i^* and x.

Lemma 6.21 *For any schedule $\eta^{\mathscr{ZR}}$ produced by the Zinder-Roper Algorithm and which does not minimise the criterion G,*

$$H_c \setminus U(j) \neq \emptyset$$

for any $j \in H_a$.

Proof. Suppose that $H_c \subseteq U(j)$ for some $j \in H_a$. Then, by the definition of $\Rightarrow$, for any $g \in H_c$, there exits a sequence $q_a(g)$, $q_{a+1}(g)$, ..., $q_c(g)$ such that

- the first element of this sequence is j and the last is g, i.e. $j = q_a(g)$ and $g = q_c(g)$;
- for each $a \leq i \leq c$, $q_i(g) \in H_i$;
- for each $a < i \leq c$ either $q_{i-1}(g) = q_i(g)$ or $q_{i-1}(g) \to q_i(g)$.

Hence, $H_c \subset K_j$. Furthermore, by Lemma 6.20, for any $h \in H_i$, where $c \leq i < i^*$, there exists $g' \in H_c$ such that $g' \Rightarrow h$. Therefore

$$\left(\bigcup_{i=c}^{i^*-1} H_i\right) \subseteq K_j \qquad \text{and} \qquad t_a < C_j\left(\eta^{\mathscr{ZR}}\right) \leq t_c.$$

On the other hand, the inequality $|H_i| \leq m$, which holds for all i such that $a < i < c$, implies that, for any $g \in H_c$, each $q_i(g) \in N_1^i$. Therefore, either $C_j\left(\eta^{\mathscr{ZR}}\right) < t_{a+1}$, or $C_j\left(\eta^{\mathscr{ZR}}\right) = t_i$ for some $a < i \leq c$. Let

$$\iota = \min\{i : i \in \mathbb{N} \wedge 0 \leq i < s \wedge j \notin A_i\}$$

i.e. $t_\iota = \max\left[C_j\left(\eta^{\mathscr{ZR}}\right), t_{a+1}\right]$. If $C_j\left(\eta^{\mathscr{ZR}}\right) < t_{a+1}$, then $\iota = a+1$ and $j \in N_2^a$. Hence, there exists a task $j' \in N_2^{\iota-1}$ such that

$$C_{j'}\left(\eta^{\mathscr{ZR}}\right) \geq t_\iota \qquad \text{and} \qquad \mu_j = \zeta_{j'}\left(t_\iota, \eta^{\mathscr{ZR}}\right). \tag{6.9}$$

If, on the other hand, $C_j\left(\eta^{\mathscr{ZR}}\right) \geq t_{a+1}$, it follows that $C_j\left(\eta^{\mathscr{ZR}}\right) = t_\iota$. Hence, (6.9) holds for $j' = j$. For the remainder of the proof, let j' be defined such that $j' \in A_{\iota-1}$ and (6.9) holds.

Recall that, when determining μ_j for a task j with $K_j \neq \emptyset$, a schedule η^j on the tasks in K_j is constructed by the Zinder-Roper Algorithm. As a consequence of the fact that $q_i(g) \in N_1^i$ for any $g \in H_c$ and any $a < i < c$, if $q_i(g) \neq j$ it follows that

$$p_{q_i(g)}\left(t_{i+1}, \eta^{\mathscr{ZR}}\right) \leq p_{q_i(g)}\left(t_{i+1} - \tau, \eta^j\right).$$

As a consequence of this, and the fact that $\left(\bigcup_{i=c}^{i^*-1} H_i\right) \subset K_j$, there exists a task $q \in \bigcup_{i=c}^{i^*-1} H_i$ such that

$$C_q\left(\eta^j\right) \geq t_{i^*} - t_\iota \qquad \text{and} \qquad \zeta_q\left(t_{i^*} - t_\iota, \eta^j\right) \geq \rho^*.$$

This, combined with (6.9), leads to

$$G\left(\eta^{\mathscr{ZR}}\right) \geq t_\iota + \zeta_{j'}\left(t_\iota, \eta^{\mathscr{ZR}}\right) = t_\iota + \mu_j \geq t_\iota + G\left(\eta^j\right)$$

$$\geq t_\iota + t_{i^*} - t_\iota + \zeta_q\left(t_{i^*} - t_\iota, \eta^j\right) \geq t_{i^*} + \rho^* = G\left(\eta^{\mathscr{ZR}}\right),$$

which contradicts the choice of x and t_{i^*}. ■

Lemma 6.20 and Lemma 6.21 state important properties of $\Rightarrow$. Specifically

- Lemma 6.20 demonstrated that every task $j \in H$ has a chain of high priority tasks leading to it through all earlier intervals; and
- Lemma 6.21 demonstrated that no task $j \in H_a$ has chains of high priority tasks leading, through all intervening intervals, to every task in H_c.

These properties will be used in Lemma 6.22 below to demonstrate that, if the Zinder-Roper Algorithm produces a schedule $\eta^{\mathscr{ZR}}$ that does not minimise the criterion G, then the precedence constraints of the corresponding problem instance contain a singular quartet. This lemma will then be used in Theorem 6.23 to establish a sufficient condition for the optimality of schedule $\eta^{\mathscr{ZR}}$. Specifically, if the precedence constraints do not contain a singular quartet, $\eta^{\mathscr{ZR}}$ is guaranteed to be optimal.

Lemma 6.22 *For any instance of the $P|prec, pmtn|L_{max}$ problem where the precedence constraints do not contain a singular quartet, the Zinder-Roper Algorithm produces a schedule which minimises the criterion G.*

Proof. By Lemma 6.5 the non-optimal schedule must have some $0 \leq a < b < c < i^*$ such that $|H_b| < m$, $|H_a| > m$ and $|H_c| > m$. Of all tasks $j \in H_b$ select as $\hat{\jmath}$ a task with the largest cardinality $|H_c \cap U(j)|$. The choice of $\hat{\jmath}$, Lemma 6.20 and the inequalities $|H_b| \leq m-1$ and $|H_c| \geq m+1$ imply

$$|H_c \cap U(\hat{\jmath})| \geq 2.$$

On the other hand, according to Lemma 6.20, there exists $j \in H_a$ such that $j \Rightarrow \hat{\jmath}$ and therefore

$$|H_c \cap U(j)| \geq 2.$$

Among all $j \in H_a$, select as k_1 a task with the largest cardinality $|H_c \cap U(j)|$. Then, taking into account Lemma 6.21,

$$2 \leq |H_c \cap U(k_1)| < |H_c|. \tag{6.10}$$

Two cases will now be considered, based on the value of $|H_c \cap U(k_1)|$. The reasoning in both cases, involving the definition of another task $k_2 \in H_a$, guarantees certain properties, (6.11) and (6.12), which will be used later in the proof.

- Suppose that $|H_c \cap U(k_1)| \geq 3$. Then, denote by k_2 any $j \in H_a$ such that

  $$(H_c \setminus U(k_1)) \cap U(j) \neq \emptyset;$$

  such a task j exists by virtue of (6.10) and Lemma 6.20.

- Suppose, on the other hand, that $|H_c \cap U(k_1)| = 2$. By Lemma 6.20, for any $q \in H_c \setminus U(k_1)$, there exists $g \in H_b$ such that $g \Rightarrow q$. Among all tasks $g \in H_b$, select as $\hat{g}$ a task with the largest cardinality $|(H_c \setminus U(k_1)) \cap U(g)|$. The choice of $\hat{g}$, Lemma 6.20 and the inequalities $|H_b| \leq m-1$ and $|H_c| \geq m+1$ imply

  $$|(H_c \setminus U(k_1)) \cap U(\hat{g})| = 2.$$

  According to Lemma 6.20, there exists $j \in H_a$ such that $j \Rightarrow \hat{g}$. Denote this task k_2.

Note that, in either case, the definitions of k_1 and k_2 imply

$$(H_c \setminus U(k_1)) \cap U(k_2) \neq \emptyset, \qquad (H_c \setminus U(k_2)) \cap U(k_1) \neq \emptyset \tag{6.11}$$

and

$$|H_c \cap (U(k_1) \cup U(k_2))| \geq 4. \tag{6.12}$$

It will now be proven that $\{k_1, k_2\}$ is a singular pair, as a preliminary step to demonstrating the existence of a singular quartet whenever the Zinder-Roper Algorithm fails to minimise G.

Define

$$B = \{k_1, k_2\} \qquad \text{and} \qquad W = H_c \cap (U(k_1) \cup U(k_2))$$

and consider the set

$$S = B \cup W \cup [K(B) \cap J(W)],$$

i.e. S that is comprised of the union

$$\{k_1, k_2\} \cup (H_c \cap U(k_1)) \cup (H_c \cap U(k_2))$$

and all tasks $g \in K_{k_1} \cup K_{k_2}$ such that

$$[(H_c \cap U(k_1)) \cup (H_c \cap U(k_2))] \cap K_g \neq \emptyset.$$

By virtue of (4.15), $S \subset \bigcup_{i=a}^{c} H_i$. Furthermore, S is a transitive set and $\{k_1, k_2\}$ covers S.

Suppose that $S = S_1 \cup S_2$ where $S_1 \neq \emptyset$, $S_2 \neq \emptyset$, and $j \to g$ for any $j \in S_1$ and $g \in S_2$, i.e. suppose S is splittable.

Since each H_i, with $a \leq i \leq c$, is comprised of independent tasks, each H_i has nonempty intersection with exactly one of the two sets S_1 and S_2. The definition of S_1 and S_2 implies that

$$B \subseteq S_1 \qquad \text{and} \qquad W \subseteq S_2.$$

Hence, for some i such that $a \leq i < c$,

$$(H_i \cap S) \subseteq S_1 \qquad \text{and} \qquad (H_{i+1} \cap S) \subseteq S_2.$$

Consider arbitrary

$$q_1 \in H_c \cap U(k_1) \qquad \text{and} \qquad q_2 \in (H_c \setminus U(k_1)) \cap U(k_2); \tag{6.13}$$

such tasks' existence is implied by (6.11). Then, by the definition of $\Rightarrow$, there exist $g_1 \in H_i$ and $g_2 \in H_{i+1}$ such that

$$k_1 \Rightarrow g_1 \Rightarrow q_1 \qquad \text{and} \qquad k_2 \Rightarrow g_2 \Rightarrow q_2.$$

On the other hand, by the choice of H_i and H_{i+1}, $g_1 \to g_2$, and therefore

$$k_1 \Rightarrow g_1 \Rightarrow g_2 \Rightarrow q_2$$

which, by the transitivity of the relation $\Rightarrow$, contradicts the fact that $q_2 \notin U(k_1)$, which follows from the definition (6.13). Hence, S is not splittable. As a consequence of (6.12) it follows that S has a width of at least four, and hence $\{k_1, k_2\}$ is a singular

pair. Since H_a is comprised of independent tasks, Corollary 5.46 implies

$$|H_a| \geq m + 1 \geq 4,$$

and this singular pair can be complemented by another two independent tasks which yields a singular quartet. Thus, if the Zinder-Roper Algorithm fails to minimise the criterion G, there must be a singular quartet. ■

Theorem 6.23 *For any instance of the $P|prec, pmtn|L_{max}$ problem where the precedence constraints do not contain a singular quartet, the Zinder-Roper Algorithm produces an optimal schedule.*

Proof. In the light of Lemma 6.22, in order to prove the theorem, it suffices to show that if the precedence constraints of an instance of the $P|prec, pmtn|L_{max}$ problem do not contain a singular quartet then $G(\sigma) = L_{max}(\sigma)$ for any schedule σ, i.e. the μ's are preserving. This property will be proven by induction on n.

It is obvious that the μ's are preserving when $n = 1$. Assume that, for some positive integer k, the μ's are preserving whenever $n \leq k$ and consider an arbitrary instance φ of $P|prec, pmtn|L_{max}$ with $n = k+1$. Since φ does not contain a singular quartet, all φ_j, i.e. all problem instances corresponding to the set of successors K_j of a task j as defined in (4.4), do not contain a singular quartet either. Taking into account the inductive assumption, Lemma 6.22 and the fact that all $|K_j| \leq k$, for any $j \in N$ such that $K_j \neq \emptyset$,

$$\mu_j = \max\left\{-d_j, G\left(\eta^j\right)\right\} = \max\left\{-d_j, L_{max}\left(\eta^j\right)\right\} = \max\left\{L_{max}\left(\omega^j\right), -d_j\right\}.$$

Hence Lemma 4.1 implies that the μ's are preserving whenever $n = k + 1$, and thus the lemma follows from Lemma 6.22. ■

Thus has a condition sufficient to ensure the optimality of $\eta^{\mathscr{ZR}}$ been established. Note the similarity to Theorem 6.6, where sufficient conditions for the optimality of $\eta^{\mathscr{BGJ}}$ were established.

6.3.4 Analysis of the Prohibited Substructure

In the beginning of this section it was stated that the domain $\mathcal{D}^{\mathscr{ZR}}$ of the Zinder-Roper Algorithm is a superclass of the domain $\mathcal{D}^{\mathscr{A}}$ of every algorithm $\mathscr{A} \in \mathfrak{F}$, and that $\mathcal{D}^{\mathscr{ZR}}$ is the class of all partial orders which do not contain a singular quartet. Put another way, it will be demonstrated in Subsection 6.3.5 that if the partially ordered set of tasks contains a singular quartet then, for any algorithm $\mathscr{A} \in \mathfrak{F}$,

there exist processing times p_j, due dates d_j and a number of machines m such that algorithm $\mathscr{A}$ fails to construct an optimal schedule for the corresponding instance of $P|prec, pmtn|L_{max}$. In order to prove this, the class of partial orders that contain a singular quartet, i.e. the class of partial orders that will be proven to lie outside the domain of every algorithm in $\mathfrak{F}$, will be examined in detail in this subsection. These results will then be employed in the following subsection to construct, for each algorithm $\mathscr{A} \in \mathfrak{F}$ and each partial order $\prec$ which contains a singular quartet, several instances of the $P|prec, pmtn|L_{max}$ problem; in each of these problem instances the precedence constraints are isomorphic to $\prec$, and it will then be proven that algorithm $\mathscr{A}$ must produce a non-optimal schedule for at least one of these instances.

This will be accomplished in a manner analogous to the proof, in Subsection 6.2.3, of the necessary conditions for optimality of the Brucker-Garey-Johnson Algorithm. Specifically, a prohibited substructure of the considered class of partial orders will be selected. In the case of the Brucker-Garey-Johnson Algorithm, this prohibited substructure is a prohibited subgraph of the first or second kind, as described in Subsection 6.2.1 and depicted in Figure 6.1, Figure 6.2 and Figure 6.3. In this section the structure selected is

- a singular quartet Q (including some singular pair $B \subset Q$) and
- a transitive set I such that
  - B covers I,
  - I has a width of at least four, and
  - I is splittable.

As in the case of the Brucker-Garey-Johnson Algorithm, all tasks outside this prohibited structure will be, in some sense, "eliminated" from consideration by assigning them arbitrarily small processing times ε and sufficiently large due dates. Tasks in the prohibited structure will then be given certain specific values of p_j and d_j which allow the non-optimality of any algorithm in $\mathfrak{F}$ to be demonstrated. Interestingly, this non-optimality will be demonstrated for $m = 3$, proving that the three machine problem $P3|prec, pmtn|L_{max}$, and hence also $P3|prec, pmtn, u_j|L_{max}$, cannot be solved by any algorithm in $\mathfrak{F}$. This is reminiscent of Corollary 5.33, which states that no algorithm in $\mathfrak{S}$ can solve the $P3|prec, pmtn, u_j|C_{max}$ problem. As stated in Section 5.3, this result can be easily adapted to demonstrate that no algorithm in $\mathfrak{S}$ can solve the $P3|prec, pmtn|C_{max}$ problem, i.e. the problem without malleable tasks. Together, these results demonstrate that no algorithm in the very broad family $\mathfrak{F} \cup \mathfrak{S}$ is able to solve the much-studied three machine problem, and directs future research away from all such algorithms.

Unlike the treatment of the Brucker-Garey-Johnson Algorithm, however, the selection of an arbitrary prohibited substructure is inadequate. It is necessary to constrain the selection of Q, the singular quartet under consideration. In particular, a partial order $\mapsto$ will be defined on singular quartets, and only such sets that are minimal in $\mapsto$ will be considered. To facilitate analysis of the precedence constraints between the tasks in $Q \cup I$, it is demonstrated that every singular pair B covers what is termed a "canonical transitive set", a transitive set with certain properties which will be exploited in subsequent proofs. Finally, the structure of a canonical transitive subset will be analysed. This analysis will proceed by defining an algorithm, called the Z-Algorithm, which accepts as an argument a canonical transitive set I, and produces a number of sets Z_i and Y_i^a of the tasks which comprise I. After the validity of the Z-Algorithm is demonstrated, several properties of the sets Z_i and Y_i^a that it gives as output are established. These sets, which in a sense characterise the structure of the precedence constraints, are subsequently used in Subsection 6.3.5 to specify the problem instances mentioned earlier. These problem instance will be used in Subsection 6.3.5 to demonstrate that, for any algorithm $\mathscr{A} \in \mathfrak{F}$ and any partial order $\prec$ which contains a singular quartet, $\prec \notin \mathcal{D}^{\mathscr{A}}$.

In this subsection the precedence constraints $\rightarrow$ common to the problem instances that will be defined in Subsection 6.3.5 are analysed, without reference to the values of p_j, d_j and m which are required to fully specify an instance of the $P|prec, pmtn|L_{max}$ problem. Only one property of $\rightarrow$ is assumed: $\rightarrow$ contains a singular quartet. Note that there may be many singular quartets defined by $\rightarrow$. Hence, let Υ be the set of all singular quartets for the precedence constraints under consideration. Several proofs appearing in Subsection 6.3.5 rely on properties which hold for some, but not all, singular quartets. In order to describe and analyse these sets, a binary relation $\mapsto$ will be defined on Υ as follows. For any $R \in \Upsilon$ and $S \in \Upsilon$, $R \mapsto S$ if $R \neq S$ and, for each $j \in R$, either $j \in S$ or $K_j \cap S \neq \emptyset$. Otherwise, $R \not\mapsto S$.

Lemma 6.24 *The binary relation $\mapsto$ is a partial order.*

Proof. In order to prove that $\mapsto$ is a partial order, it is necessary to prove that it is irreflexive, antisymmetric and transitive. Since $R \not\mapsto S$ if $R = S$, $\mapsto$ is irreflexive. If $R \mapsto S$, the fact that $R \neq S$ implies the existence of tasks $j \in R$ and $g \in S$ such that $j \rightarrow g$. Since all singular quartets are composed of independent tasks, $j \rightarrow g$ implies $g \notin R$ and $K_g \cap R = \emptyset$, and hence $S \not\mapsto R$. Consequently $\mapsto$ is antisymmetric.

Finally, suppose $R \mapsto S$ and $S \mapsto T$. Since $\mapsto$ is antisymmetric, $T \neq R$. Consider any task $j \in R$, and note that $R \mapsto S$ implies the existence of $g \in S$ such that either $j = g$ or $j \to g$. Similarly, $S \mapsto T$ implies the existence of $q \in T$ such that either $g = q$ or $g \to q$. Hence, the transitivity of $\to$ demonstrates that either $j = q$ or $j \to q$, and since j is an arbitrary element of R, it follows that $R \mapsto T$. Hence $\mapsto$ is transitive, and the lemma is proven. ■

As a consequence of this, there exists $Q \in \Upsilon$ such that there exists no $R \in \Upsilon$ satisfying $R \mapsto Q$, i.e. Q is a minimal element of Υ. Such a set Q will be termed a "minimal singular quartet." The lemma below states a property of minimal singular quartets; in fact it is the sole reason this concept was introduced.

Lemma 6.25 *If Q is a minimal singular quartet, $|K_j \cap Q| \neq 1$ for each $j \in N$.*

Proof. Suppose that there exists a task j such that

$$|K_j \cap Q| = 1, \tag{6.14}$$

and note that this implies that $j \notin Q \cup I$. Consider the set $R = (Q \setminus K_j) \cup \{j\}$ and note that $|R| = 4$. Consider any singular pair $B \subset Q$. If $K_j \cap B = \emptyset$, then $B \subset R$ and hence R is a singular quartet, i.e. $R \in \Upsilon$. By virtue of (6.14) and the fact that $(R \setminus \{j\}) \subset Q$, it follows that $R \mapsto Q$, contradicting the definition of Q as minimal.

On the other hand, suppose that $K_j \cap B \neq \emptyset$ and let I be an adequate transitive set covered by B. Define the set

$$P = \{j\} \cup \{g : g \in K_j \wedge K_g \cap I \neq \emptyset \wedge g \notin I\}, \tag{6.15}$$

the set $S = I \cup P$, and the set

$$C = (B \setminus K_j) \cup \{j\}.$$

It will now be proven that S is an adequate transitive subset, implying that C is a singular pair, further implying that R is a singular quartet.

First it will be proven that S is a transitive set. Consider any two tasks $g_1 \in S$ and $g_2 \in S \cap K_{g_1}$. Consider any q such that $g_1 \to q \to g_2$. By considering the two cases $g_2 \in I$ and $g_2 \in P$ below, it will be proven that $K_q \cap I \neq \emptyset$.

- If $g_2 \in I$, $(K_q \cap I) \supseteq \{g_2\} \neq \emptyset$.
- If $g_2 \in P$, by the definition (6.15) of the set P, $(K_q \cap I) \supseteq (K_{g_2} \cap I) \neq \emptyset$.

Two more cases, $g_1 \in I$ and $g_1 \in P$, are considered below, demonstrating that S is a transitive set.

- If $g_1 \in I$, the fact that I is a transitive set, along with $K_q \cap I \neq \emptyset$, implies $q \in I \subset S$.
- If $g_1 \in P$, (6.15) implies that $q \in K_{g_1} \subset K_j$. Either $q \in I \subset S$ or (6.15) implies $q \in P \subset S$.

Suppose that S is splittable, i.e. $S = S_1 \cup S_2$ where $S_1 \neq \emptyset$, $S_2 \neq \emptyset$ and $g \rightarrow q$ for all $g \in S_1$ and $q \in S_2$. Let j' be the sole member of $B \setminus K_j$. If $j' \in S_2$ there exists a task $g \in S_1$ and hence $g \rightarrow j'$. In the two cases below, it will be proven that this leads to a contradiction.

- If $g \in I$, j' is not a minimal element of I, which contradicts the fact that B covers I.
- If $g \in P$, either $j \rightarrow g \rightarrow j'$ or $j = g \rightarrow j'$. Both cases contradict the fact that $j' \in B \setminus K_j$.

Hence $j' \in S_1$. Since the tasks in B are independent, this implies that $B \subseteq S_1$. Since $B \cap K_j \neq \emptyset$, it follows that $j \in S_1$.

If $S_2 \cap I \neq \emptyset$, the fact that $(S_1 \cap I) \supseteq B \neq \emptyset$ implies that I is splittable, contradicting the fact that it is an adequate transitive set. Hence, $I \subset S_1$. This, combined with (6.15), implies that $P \subset S_1$. Since $S = (I \cup P) \subseteq S_1$, it follows that $S_2 = \emptyset$, a contradiction. Hence S is not splittable.

Since $I \subset S$, the fact that I is an adequate transitive set implies that the width of S is at least four. Combining this with the results above demonstrates that S is an adequate transitive subset. Since C covers S, C is a singular pair. Since $C \subset R$ and $|R| = 4$, R is a singular quartet, i.e. $R \in \Upsilon$. By virtue of (6.14) and the fact that $(R \setminus \{j\}) \subset Q$, it follows that $R \mapsto Q$, contradicting the definition of Q as minimal. ■

The concept of a minimal singular quartet is not used in the remainder of this subsection, but is used in the proof of Lemma 6.45 in Subection 6.3.5.

By definition, a singular pair covers at least one adequate transitive set. This set may not be unique. In order to examine certain kinds of transitive sets which have properties employed in subsequent analysis, some some new terminology will be introduced. A transitive set I will be called a "canonical transitive set" if and only if

- I is an adequate transitive set,
- I is covered by a singular pair B, and
- for each adequate transitive set S covered by B, $|I| \leq |S|$.

In other words, for any singular pair B there exist one or more adequate transitive sets covered by B. The number of elements in these sets may vary from one to the other; all those with the fewest elements are termed canonical transitive sets. Note that there may be more than one canonical transitive set for any given singular pair.

For any canonical transitive set I, define an "adequate subset of I" to be any set W such that

- $W \subseteq I$,
- the tasks in W are independent,
- $|W| \geq 4$, and
- there exists no task $j \in I \setminus W$ such that j is independent of all tasks in W.

A number of lemmas will now establish certain properties of canonical transitive sets and their adequate subsets.

Lemma 6.26 *For any canonical transitive set I and any adequate subset W of I, $I \cap K(W) = \emptyset$.*

Proof. If $I \cap K(W) \neq \emptyset$, then consider $S = I \setminus K(W)$. Suppose that S is splittable, i.e. $S = S_1 \cup S_2$, $S_1 \neq \emptyset$, $S_2 \neq \emptyset$ and $g \rightarrow j$ for all $g \in S_1$ and $j \in S_2$. Since the tasks in W are maximal elements of the set S, and since $S_2 \neq \emptyset$, $W \subseteq S_2$. This implies that $g \rightarrow j$ for all $g \in S_1$ and $j \in S_2 \cup (I \cap K(W))$, contradicting the definition of I as a canonical transitive set. Hence, S is not splittable. Since $|W| \geq 4$, $W \subset S$ and $|S| < |I|$, the assumption that $I \cap K(W) \neq \emptyset$ contradicts the definition of I. ■

Lemma 6.27 *For each canonical transitive set I there exists one adequate subset of I.*

Proof. By the definition of a canonical transitive subset, the width of I is at least four. By the definition of width, there exists a set $W \subseteq I$ of independent tasks, where $|W|$ is equal to the width of I. The existence of a task $j \in I \setminus W$ that is independent of all tasks in W contradicts the definition of width. Hence, W is an adequate subset of I.

Suppose that there exists W' such that

- W' is an adequate subset of I, and
- $W \neq W'$.

By Lemma 6.26 $I \cap K(W) = \emptyset = I \cap K(W')$, and hence the tasks in $W \cup W'$ are independent. Consequently, since $W \neq W'$ it follows that either $W \setminus W' \neq \emptyset$ or $W' \setminus W \neq \emptyset$. Both cases contradict the definitions of W and W' as adequate subsets of I. ■

Lemma 6.28 *For each canonical transitive set I the sole adequate subset W of I has cardinality four.*

Proof. Suppose that $|W| \geq 5$. If $W \subseteq K_j$ for each $j \in I \setminus W$, it follows that I is splittable, contradicting the definition of I. Hence, there exists at least one task $j \in I \setminus W$ such that $W \setminus K_j \neq \emptyset$.

If $|W \setminus K_j| \geq 3$, define the set $W_0 = \{j\} \cup (W \setminus K_j)$. A sequence of sets $W_0, W_1, \ldots, W_v$ will now be defined one by one. Suppose the set W_k has been defined but W_{k+1} has not. If there exists any task $g \in I \setminus W_k$ such that g is independent of all tasks in W_k, define $W_{k+1} = \{g\} \cup W_k$. If no such g exists, set $v = k$ and terminate the procedure; note that this is guaranteed to happen since $|N|$ is finite. These sets are all composed of independent tasks and have cardinality at least four. Hence W_v is an adequate subset of I, which contradicts Lemma 6.27.

If $2 \geq |W \setminus K_j| \geq 1$, consider $S = I \setminus \{g\}$, where g is an arbitrary element of $W \cap K_j$. By the assumption that $|W| \geq 5$, the width of S is at least four. Lemma 6.26 implies $I \cap K_g = \emptyset$, so the fact that I is a transitive set implies that S is also a transitive set.

Suppose that S is splittable, i.e. $S = S_1 \cup S_2$, $S_1 \neq \emptyset$, $S_2 \neq \emptyset$ and $q_1 \rightarrow q_2$ for all $q_1 \in S_1$ and $q_2 \in S_2$. Since $I \cap K(W) = \emptyset$ as a consequence of Lemma 6.26, this implies $(W \setminus \{g\}) \subseteq S_2$. On the other hand, since $W \setminus K_j \neq \emptyset$, $j \in S_2$. Since $j \rightarrow g$, it follows that $q_1 \rightarrow q_2$ for all $q_1 \in S_1$ and $q_2 \in S_2 \cup \{g\}$, I is splittable, a contradiction. Hence, S is not splittable.

Let B be the singular pair that covers I. Since $j \rightarrow g$, $g \notin B$ and hence B also covers S. Hence, S is an adequate transitive set covered by B, and the fact that $S \subset I$ contradicts the definition of I as a canonical transitive set. ■

The precedence structure between tasks in a canonical transitive set will now be analysed.

As stated earlier, this subsection demonstrates that if the partially ordered set of tasks contains a singular quartet then, for any algorithm in the considered family $\mathfrak{F}$ of

algorithms, there exist processing times p_j, due dates d_j and a number of machines m such that this algorithm fails to construct an optimal schedule for the corresponding instance of $P|prec, pmtn|L_{max}$. For any partial order $\prec$ which contains a singular quartet, this will be proven by constructing three problem instances with precedence constraints $\rightarrow$ isomorphic to $\prec$, and then demonstrating that no algorithm in $\mathfrak{F}$ can generate an optimal schedule for all of them. This thesis introduces the Z-Algorithm which plays an important role in the assignment of processing times p_j and due dates d_j to the tasks in these problem instances. This algorithm is applied to a canonical transitive set I covered by a singular pair B. It begins with some initial index $\underline{h}$, to be specified later.

The output of the algorithm is a sequence of sets of tasks, $Z_{\underline{h}}, Z_{\underline{h}+1}, \ldots, Z_{\bar{h}}$ for some index $\bar{h} > \underline{h}$. The value of $\bar{h} - \underline{h}$ depends on the precise structure of the precedence constraints between the tasks in I. Each Z_i is a subset of I, and each is composed of independent tasks. Each Z_i is partitioned by the Z-Algorithm into two nonempty sets, Y_i^0 and Y_i^1. An outline of the algorithm is given below, and it is specified rigorously on page 187.

The algorithm construct the sets Z_i one at a time, in increasing order of i. When each Z_i is defined, the sets Y_i^0 and Y_i^1 are also defined. Step 1 computes $Z_{\underline{h}}$. Each subsequent set Z_{i+1} is the result of a modification of Z_i. The modification that derives Z_{i+1} from Z_i takes one of two forms, specified in Step 2 and Step 3.

In order to describe these steps more fully, a definition is needed. For each index i and each task $j \in Z_i$, a set $R_{i,j}$, defined as

$$R_{i,j} = I \cap \left(\tilde{K}_j \setminus K(Z_i \setminus \{j\}) \right), \tag{6.16}$$

is computed. In words, this is the set of immediate successors of j which are in I and which are not successors of any other task in Z_i. In the case where $R_{i,j} = \emptyset$ the lemma below provides insight into the set K_j.

Lemma 6.29 *For any i and any two tasks $j_0 \neq j_1$ such that $Z_i = \{j_0, j_1\}$,*

$$R_{i,j_0} = \emptyset = R_{i,j_1} \qquad \textit{implies} \qquad I \cap K_{j_0} = I \cap K_{j_1}.$$

Proof. For either $a \in \{0,1\}^*$ the definition (6.16) gives

$$\emptyset \subseteq \left(I \cap \tilde{K}_{j_a} \right) \setminus \left(I \cap K_{j_{1-a}} \right) = I \cap \left(\tilde{K}_{j_a} \setminus K_{j_{1-a}} \right)$$

*In much of the rest of this chapter it is convenient to refer to two mathematical objects with indices 0 and 1. In order to refer to them generically, an arbitrary index $a \in \{0,1\}$ can be considered, and the objects referenced with indices a and $1-a$. In particular, this lemma considers tasks j_a and j_{1-a}; in later reasoning the sets Y_i^a and Y_i^{1-a} are considered.

$$= I \cap \left(\tilde{K}_{j_a} \setminus K(Z_i \setminus \{j_a\}) \right) = R_{i,j_a} = \emptyset.$$

Consequently

$$\left(I \cap \tilde{K}_{j_a} \right) \setminus \left(I \cap K_{j_{1-a}} \right) = \emptyset$$

and hence

$$I \cap \tilde{K}_{j_a} \subseteq I \cap K_{j_{1-a}}. \tag{6.17}$$

For any task $g \in I \cap K_{j_a}$ the fact that $j_a \in I$ and that I is a transitive set implies that either $g \in I \cap \tilde{K}_{j_a}$ or $q \to g$ for some $q \in I \cap \tilde{K}_{j_a}$. In either case $g \in I \cap K_{j_{1-a}}$ as a consequence of (6.17), implying

$$I \cap K_{j_a} \subseteq I \cap K_{j_{1-a}}. \tag{6.18}$$

Since (6.18) holds for both $a \in \{0, 1\}$ the lemma is proven. ■

In each iteration of the Z-Algorithm, Step 3 derives Z_{i+1} from Z_i by replacing a task $z \in Z_i$ with all tasks in $R_{i,z}$. By this process, each task in I will become a member of at least one set Z_i; this will be proven after the algorithm is rigorously specified. It will also be proven that when Step 3 is executed z will be selected so that $R_{i,z} \neq \emptyset$.

Sometimes it is necessary for an additional step to take place before such a task z can be selected. If this is the case, Z_{i+1} will be derived in a different way, specified by Step 2. If Step 2 determines Z_{i+1}, it does so by removing a task y in a manner that guarantees that Step 3 can then select some $z \in Z_{i+1}$ such that $R_{i+1,z} \neq \emptyset$. Hence, during some iterations of the Z-Algorithm Step 2 determines Z_{i+1} and Step 3 determines Z_{i+2}; during others, Step 2 does nothing while Step 3 determines Z_{i+1}.

After the execution of Step 3, Step 4 will terminate the algorithm if $|Z_i| = 4$, setting $\bar{h} = i$; otherwise it will initiate another iteration of Step 2 and Step 3. Since I is a canonical transitive set, Lemma 6.27 demonstrates that $Z_{\bar{h}}$ is the only adequate subset of I.

Note that for some steps of the Z-Algorithm it is not obvious that the operations they describe are valid. Before the Z-Algorithm can be analysed it must be demonstrated that it produces well-defined output after a finite number of steps. Specifically, the issues below must be addressed.

- It is not obvious that y can be selected as in Step 2.2 because Y_i^a might be empty. See Lemma 6.32 for the proof that this never happens.

The Z-Algorithm

1. Let the tasks in B be denoted arbitrarily as y_0 and y_1, then:

   1.1. set

      1.1.1. $Y^0_{\underline{h}} := \{y_0\}$,
      1.1.2. $Y^1_{\underline{h}} := \{y_1\}$ and
      1.1.3. $Z_{\underline{h}} := Y^0_{\underline{h}} \cup Y^1_{\underline{h}}$;

   1.2. set $i := \underline{h}$.

2. If $R_{i,j} = \emptyset$ for all $j \in Z_i$:

   2.1. select $a \in \{0, 1\}$ such that $|Y^a_i| \geq \left|Y^{1-a}_i\right|$;

   2.2. select $y \in Y^a_i$ such that $|I \cap K_y| \geq |I \cap K_j|$ for all $j \in Y^a_i$;

   2.3. set

      2.3.1. $Y^a_{i+1} := Y^a_i \setminus \{y\}$,
      2.3.2. $Y^{1-a}_{i+1} := Y^{1-a}_i$ and
      2.3.3. $Z_{i+1} := Y^0_{i+1} \cup Y^1_{i+1}$;

   2.4. increment i.

3. Regardless of whether Step 2 resulted in a new value of i:

   3.1. select $z \in Z_i$ such that $R_{i,z} \neq \emptyset$;

   3.2. select $a \in \{0, 1\}$ such that $z \in Y^a_i$;

   3.3. set

      3.3.1. $Y^a_{i+1} := (Y^a_i \setminus \{z\}) \cup R_{i,z}$,
      3.3.2. $Y^{1-a}_{i+1} := Y^{1-a}_i$ and
      3.3.3. $Z_{i+1} := Y^0_{i+1} \cup Y^1_{i+1}$;

   3.4. increment i.

4. If $|Z_i| = 4$ set $\bar{h} := i$ and terminate; otherwise go to Step 2.

- It is not obvious that z can be selected as in Step 3.1 because it might be the case that $R_{i,j} = \emptyset$ for all $j \in Z_i$. See Lemma 6.33 for the proof that this never happens.

- It is not obvious that the algorithm terminates as in Step 4. See Lemma 6.39 for the proof that this always happens.

The proof of validity proceeds by induction. Since several lemmas are required, the inductive assumptions will be stated here and the lemmas will demonstrate that, if these assumptions hold for some $i = \bar{\imath}$, then either the algorithm terminates with $\bar{h} = \bar{\imath}+1$ or the assumptions hold for $i = \bar{\imath}+1$. The inductive assumptions are that, for some integer $\bar{\imath} \geq \underline{h}$ and each integer $i \in [\underline{h}, \bar{\imath}]$:

1. $|Z_i| \leq 3$;

2. $Y_i^0 \neq \emptyset \neq Y_i^1$;

3. if Y_i^0, Y_i^1 and Z_i were defined according to Step 2.3 there exists at least one $j \in Z_i$ for which $R_{i,j} \neq \emptyset$;

4. if Y_i^0, Y_i^1 and Z_i were not defined according to Step 2.3* there exists no task $j \in I \setminus Z_i$ such that j is independent of every task in Z_i;

5. $Y_i^0 \cap Y_i^1 = \emptyset$; and

6. the tasks in Z_i are independent.

Note that Assumption 1 implies that the algorithm has not terminated, and hence that Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are also defined by the Z-Algorithm. Also note that Assumption 2 implies that it is possible to select a task y as in Step 2.2, and that Assumption 3 implies that it is possible to select a task z as in Step 3.1. Hence these two inductive assumptions imply that no invalid operation is executed during the computation of Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1}.

The case of $\bar{\imath} = \underline{h}$ is rather trivial; consequently the proof in this case is not made explicit.

Lemma 6.30 *The inductive assumptions hold for $\bar{\imath} = \underline{h}$.*

Proof. Since $\bar{\imath} = \underline{h}$, $\underline{h}$ is the only value of i for which the inductive assumptions must be proven to hold. The lemma follows from Step 1 of the Z-Algorithm, and the fact that B is a singular pair. ■

*This means that either Y_i^0, Y_i^1 and Z_i were defined according to Step 3.3 or $i = \underline{h}$, in which case Y_i^0, Y_i^1 and Z_i were defined according to Step 1.1.

The proof that the inductive assumptions propagate to subsequent values of i is divided into several lemmas below. Lemma 6.31 and Lemma 6.34 are of general utility, addressing the precedence constraints of tasks in Z_i. Aside from these two lemmas, the inductive assumptions are considered one by one:

- Lemma 6.32 considers Assumption 2;
- Lemma 6.33 considers Assumption 3;
- Lemma 6.35 considers Assumption 4;
- Lemma 6.36 considers Assumption 5; and
- Lemma 6.37 considers Assumption 6.

There is no lemma considering Assumption 1 because this is the assumption that the algorithm has not terminated for any $i \leq \bar{\imath}$; on the contrary, Lemma 6.39 demonstrates that the Z-Algorithm will always terminate after a finite number of steps.

Lemma 6.31 *If the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$, and if $Y_{\bar{\imath}}^0$, $Y_{\bar{\imath}}^1$ and $Z_{\bar{\imath}}$ are not defined according to Step 2.3,*

$$I \cap K(Z_{\bar{\imath}}) \neq \emptyset.$$

Proof. Suppose the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$, and that $Y_{\bar{\imath}}^0$, $Y_{\bar{\imath}}^1$ and $Z_{\bar{\imath}}$ are not defined according to Step 2.3. Lemma 6.26, Lemma 6.27 and Lemma 6.28 demonstrate the existence of a set $W \subseteq I$ such that

$$|W| = 4 \qquad \text{and} \qquad I \cap K(W) = \emptyset.$$

The fact that $I \cap K(W) = \emptyset$ implies that $W \cap J(Z_{\bar{\imath}}) = \emptyset$, and since Assumption 1 gives

$$|W| = 4 > 3 \geq |Z_{\bar{\imath}}|$$

it follows that

$$W \setminus (Z_{\bar{\imath}} \cup J(Z_{\bar{\imath}})) \neq \emptyset.$$

By Assumption 4

$$W \subseteq I \subseteq J(Z_{\bar{\imath}}) \cup Z_{\bar{\imath}} \cup K(Z_{\bar{\imath}}).$$

Consequently

$$\emptyset \subset W \cap K(Z_{\bar{\imath}}) \subseteq I \cap K(Z_{\bar{\imath}}),$$

and the lemma is proven. ■

Lemma 6.32 *If the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$,*

$$Y^0_{\bar{\imath}+1} \neq \emptyset \neq Y^1_{\bar{\imath}+1}.$$

Proof. Suppose the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$. By Assumption 1, sets $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined. Two cases will be considered based on how they differ from $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$.

- Suppose $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined according to Step 2.3, i.e. there exists a task $y \in Z_{\bar{\imath}}$ such that $Z_{\bar{\imath}+1} = Z_{\bar{\imath}} \setminus \{y\}$. Select $a \in \{0,1\}$ such that $y \in Y^a_{\bar{\imath}}$. Step 2.3.2 and Assumption 2 together imply that

  $$Y^{1-a}_{\bar{\imath}+1} = Y^{1-a}_{\bar{\imath}} \neq \emptyset.$$

  It will now be proven by contradiction that $Y^a_{\bar{\imath}+1} \neq \emptyset$. If $Y^a_{\bar{\imath}+1} = \emptyset$, Step 2.3.1 implies that $Y^a_{\bar{\imath}} = \{y\}$. By Step 2.1 and Assumption 2, it follows that

  $$1 \leq \left|Y^{1-a}_{\bar{\imath}}\right| \leq |Y^a_{\bar{\imath}}| = 1.$$

  Let j be the sole task in $Y^{1-a}_{\bar{\imath}}$. Since the task y was selected, it follows from Step 2 that $R_{\bar{\imath},j} = \emptyset = R_{\bar{\imath},y}$, and hence Lemma 6.29 gives

  $$I \cap K_y = I \cap K_j. \tag{6.19}$$

  Define

  $$S_1 = I \cap (J(Z_{\bar{\imath}}) \cup Z_{\bar{\imath}}) \qquad \text{and} \qquad S_2 = I \cap K(Z_{\bar{\imath}}).$$

  Since the fact that task y was selected when defining $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ implies that $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$ were not defined according to Step 2.3, Assumption 4 implies that $I = S_1 \cup S_2$ and Lemma 6.31 implies that $S_2 \neq \emptyset$. Consequently, it is possible to select two tasks $g_1 \in S_1$ and $g_2 \in S_2$. From (6.19) it follows that $g_1 \to g_2$, and so I is splittable, a contradiction. Hence, $\{y\} \subset Y^a_{\bar{\imath}}$ and as a consequence

  $$Y^a_{\bar{\imath}+1} = Y^a_{\bar{\imath}} \setminus \{y\} \neq \emptyset.$$

  Thus, the lemma has been proven in the case where a task y is selected as in Step 2.3.

- On the other hand, suppose $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are not defined according to Step 2.3, i.e. there exists a task $z \in Z_{\bar{\imath}}$ such that $Z_{\bar{\imath}+1} = (Z_{\bar{\imath}} \setminus \{z\}) \cup R_{\bar{\imath},z}$, as in Step 3.3. Select $a \in \{0,1\}$ such that $z \in Y^a_{\bar{\imath}}$. Step 3.3.2 and Assumption 2 together imply that

$$Y^{1-a}_{\bar{\imath}+1} = Y^{1-a}_{\bar{\imath}} \neq \emptyset.$$

Two cases are now considered, based on how $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$ were defined. In both cases it is demonstrated that $R_{\bar{\imath},z} \neq \emptyset$, and

$$\emptyset \subset R_{\bar{\imath},z} \subseteq Y^a_{\bar{\imath}+1}.$$

  - If $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$ were defined according to Step 2.3, Assumption 3 and Step 3.1 imply $R_{\bar{\imath},z} \neq \emptyset$.
  - On the other hand, if $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$ were not defined according to Step 2.3, Step 2 and Step 3.1 imply $R_{\bar{\imath},z} \neq \emptyset$.

Hence, the lemma holds in both cases. ■

Lemma 6.33 *If the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$, and if $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined according to Step 2.3, there exists at least one $j \in Z_{\bar{\imath}+1}$ for which $R_{\bar{\imath}+1,j} \neq \emptyset$.*

Proof. Suppose the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$, and that $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined according to Step 2.3, i.e. there exists a task $y \in Z_{\bar{\imath}}$ such that $Z_{\bar{\imath}+1} = Z_{\bar{\imath}} \setminus \{y\}$. Select $a \in \{0,1\}$ such that $y \in Y^a_{\bar{\imath}}$.

Lemma 6.32 and Step 2.3.1 together imply

$$\emptyset \subset Y^a_{\bar{\imath}+1} = Y^a_{\bar{\imath}} \setminus \{y\}.$$

If $|Y^a_{\bar{\imath}}| \geq 3$ or $\left|Y^{1-a}_{\bar{\imath}}\right| \geq 2$ then, by Assumption 2 and Assumption 5,

$$|Z_{\bar{\imath}}| = |Y^a_{\bar{\imath}}| + \left|Y^{1-a}_{\bar{\imath}}\right| \geq 4,$$

contradicting Assumption 1. Hence $|Y^a_{\bar{\imath}}| = 2$ and $\left|Y^{1-a}_{\bar{\imath}}\right| = 1$. Let j be the sole task in $Y^{1-a}_{\bar{\imath}}$ and let g be the sole task in $Y^a_{\bar{\imath}+1}$. This implies that $Z_{\bar{\imath}} = \{y, j, g\}$ and $Z_{\bar{\imath}+1} = \{j, g\}$.

Suppose that $R_{\bar{\imath}+1,j} = \emptyset = R_{\bar{\imath}+1,g}$, i.e. suppose that the lemma does not hold. As a consequence, Lemma 6.29 gives

$$I \cap K_j = I \cap K_g. \tag{6.20}$$

Since $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined according to Step 2.3, Step 2 implies that $R_{\bar{\imath},y} = \emptyset$, so by (6.20) it follows that

$$I \cap K_y \subseteq I \cap K_j = I \cap K_g,$$

and by Step 2.2

$$I \cap K_y = I \cap K_j = I \cap K_g. \tag{6.21}$$

Since $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined according to Step 2.3, $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$ were not defined according to Step 2.3, and Lemma 6.31 implies that $I \cap K(Z_{\bar{\imath}}) \neq \emptyset$. This, combined with (6.21) demonstrates that I can be partitioned into two nonempty sets

$$S_1 = I \cap (J(Z_{\bar{\imath}}) \cup Z_{\bar{\imath}}) \qquad \text{and} \qquad S_2 = I \cap K(Z_{\bar{\imath}}),$$

and every task in S_1 precedes every task in S_2, i.e. I is splittable. This contradicts the definition of I as a canonical transitive set and so the lemma is proven. ■

Lemma 6.34 *For any i such that Y^0_{i+1}, Y^1_{i+1} and Z_{i+1} are defined according to Step 2.3,*

$$I \cap K_y \subseteq K(Z_{i+1})$$

where $y \in Z_i \setminus Z_{i+1}$.

Proof. Suppose that Y^0_{i+1}, Y^1_{i+1} and Z_{i+1} are defined according to Step 2.3, i.e. there exists a task $y \in Z_i$ such that $Z_{i+1} = Z_i \setminus \{y\}$. Hence

$$\emptyset = R_{i,y} = I \cap \left(\tilde{K}_y \setminus K(Z_i \setminus \{y\})\right) = \left(I \cap \tilde{K}_y\right) \setminus K(Z_{i+1}),$$

and consequently

$$I \cap \tilde{K}_y \subseteq K(Z_{i+1}). \tag{6.22}$$

For any task $g \in I \cap K_y$ the fact that $y \in I$ and that I is a transitive set implies that either $g \in I \cap \tilde{K}_y$ or $q \to g$ for some $q \in I \cap \tilde{K}_y$. In either case $g \in K(Z_{i+1})$ as a consequence of (6.22), and the lemma is proven. ■

Lemma 6.35 *If the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$, and if $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are not defined according to Step 2.3, there exists no task $j \in I \setminus Z_{\bar{\imath}+1}$ such that j is independent of every task in $Z_{\bar{\imath}+1}$.*

Proof. Suppose the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$, and that $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are not defined according to Step 2.3. Since $\bar{\imath} \geq \underline{h}$, this implies

that they are defined according to Step 3.3, i.e. $Z_{\bar{\imath}+1}$ is formed by replacing a task $z \in Z_{\bar{\imath}}$ with the tasks in $R_{\bar{\imath},z}$. Consider any task $j \in I$. Several cases will now be considered, based on the relationship between j and the tasks in $Z_{\bar{\imath}}$.

- If $j \in J(Z_{\bar{\imath}} \setminus \{z\})$, $j \in Z_{\bar{\imath}} \setminus \{z\}$ or $j \in K(Z_{\bar{\imath}} \setminus \{z\})$, the fact that $(Z_{\bar{\imath}} \setminus \{z\}) \subseteq Z_{\bar{\imath}+1}$ implies $j \in J(Z_{\bar{\imath}+1})$, $j \in Z_{\bar{\imath}+1}$ or $j \in K(Z_{\bar{\imath}+1})$ respectively.

- If $j \in J_z \cup \{z\}$, the fact that $R_{\bar{\imath},z} \subseteq I \cap K_z$ and $R_{\bar{\imath},z} \neq \emptyset$ (by Assumption 3 and Step 2) implies $j \in J(Z_{\bar{\imath}+1})$.

- Suppose that $j \in K_z \setminus K(Z_{\bar{\imath}} \setminus \{z\})$ and consider the following two cases.

  - If $j \in \tilde{K}_z$ the definition (6.16) of $R_{\bar{\imath},z}$ and the fact that $j \notin K(Z_{\bar{\imath}} \setminus \{z\})$ together imply that $j \in R_{\bar{\imath},z} \subseteq Z_{\bar{\imath}+1}$.
  - If $j \notin \tilde{K}_z$ it follows that

$$j \in I \cap K\left(\tilde{K}_z\right) \subseteq I \cap \left(K\left(R_{\bar{\imath},z}\right) \cup K\left(Z_{\bar{\imath}} \setminus \{z\}\right)\right) = I \cap K(Z_{\bar{\imath}+1}).$$

- Finally, suppose that

$$j \notin J(Z_{\bar{\imath}}) \cup Z_{\bar{\imath}} \cup K(Z_{\bar{\imath}}), \tag{6.23}$$

  i.e. suppose that j is not in $Z_{\bar{\imath}}$ and is independent of every task in $Z_{\bar{\imath}}$. This supposition, combined with Assumption 4, implies that $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$ were defined according to Step 2.3, i.e. $Z_{\bar{\imath}} = Z_{\bar{\imath}-1} \setminus \{y\}$ for some task $y \in Z_{\bar{\imath}-1}$. As a consequence $\bar{\imath} > \underline{h}$ and the sets $Y^0_{\bar{\imath}-1}$, $Y^1_{\bar{\imath}-1}$ and $Z_{\bar{\imath}-1}$ were not defined according to Step 2.3. By Assumption 4, this observation implies

$$j \in J(Z_{\bar{\imath}-1}) \cup Z_{\bar{\imath}-1} \cup K(Z_{\bar{\imath}-1}). \tag{6.24}$$

  Combining (6.23) and (6.24) leads to the following cases.

  - If $j \in J(Z_{\bar{\imath}-1} \setminus \{y\})$, $j \in Z_{\bar{\imath}-1} \setminus \{y\}$ or $j \in K(Z_{\bar{\imath}-1} \setminus \{y\})$, the fact that $Z_{\bar{\imath}} = Z_{\bar{\imath}-1} \setminus \{y\}$ implies $j \in J(Z_{\bar{\imath}})$, $j \in Z_{\bar{\imath}}$ or $j \in K(Z_{\bar{\imath}})$ respectively, contradicting (6.23).
  - Suppose that $j \in J_y \cup \{y\}$. Note that $z \in Z_{\bar{\imath}-1} \setminus \{y\}$ and, because $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$ were defined according to Step 2.3, Step 2 implies that

$$\emptyset = R_{\bar{\imath}-1,z} = I \cap \left(\tilde{K}_z \setminus K(Z_{\bar{\imath}-1} \setminus \{z\})\right) = I \cap \left(\tilde{K}_z \setminus K((Z_{\bar{\imath}} \cup \{y\}) \setminus \{z\})\right)$$

$$= I \cap \left(\tilde{K}_z \setminus K(Z_{\bar{\imath}} \setminus \{z\})\right) \setminus K_y = R_{\bar{\imath},z} \setminus K_y$$

and hence $R_{\bar{\imath},z} \subseteq K_y$. Because this, Assumption 3 and Step 3 together imply that

$$\emptyset \subset R_{\bar{\imath},z} \subseteq Z_{\bar{\imath}+1} \cap K_y,$$

it follows that

$$j \in J_y \cup \{y\} \subseteq J(Z_{\bar{\imath}+1} \cap K_y) \subseteq J(Z_{\bar{\imath}+1}).$$

– Finally, suppose that $j \in K_y$. From Lemma 6.34 it follows that

$$j \in I \cap K_y \subseteq K(Z_{\bar{\imath}}),$$

again contradicting (6.23).

Hence, in all cases $j \in I$ implies $j \in J(Z_{\bar{\imath}+1})$, $j \in Z_{\bar{\imath}+1}$ or $j \in K(Z_{\bar{\imath}+1})$. ■

Lemma 6.36 *If the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$,*

$$Y^0_{\bar{\imath}+1} \cap Y^1_{\bar{\imath}+1} = \emptyset.$$

Proof. Suppose the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$. By Assumption 1, sets $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined. Two cases will be considered based on how they differ from $Y^0_{\bar{\imath}}$, $Y^1_{\bar{\imath}}$ and $Z_{\bar{\imath}}$.

- Suppose $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are defined according to Step 2.3, i.e. there exists a task $y \in Z_{\bar{\imath}}$ such that $Z_{\bar{\imath}+1} = Z_{\bar{\imath}} \setminus \{y\}$. By Assumption 5, Step 2.3.1 and Step 2.3.2

  $$Y^0_{\bar{\imath}+1} \cap Y^1_{\bar{\imath}+1} \subseteq Y^0_{\bar{\imath}} \cap Y^1_{\bar{\imath}} = \emptyset$$

  so the lemma holds in this case.

- On the other hand, suppose $Y^0_{\bar{\imath}+1}$, $Y^1_{\bar{\imath}+1}$ and $Z_{\bar{\imath}+1}$ are not defined according to Step 2.3, i.e. there exists a task $z \in Z_{\bar{\imath}}$ such that $Z_{\bar{\imath}+1} = (Z_{\bar{\imath}} \setminus \{z\}) \cup R_{\bar{\imath},z}$, as in Step 3.3. Select $a \in \{0, 1\}$ such that $z \in Y^a_{\bar{\imath}}$. By Step 3.3.1,

  $$Y^a_{\bar{\imath}+1} = (Y^a_{\bar{\imath}} \setminus \{z\}) \cup R_{\bar{\imath},z} \subseteq Y^a_{\bar{\imath}} \cup K_z. \tag{6.25}$$

  By Assumption 5 and Step 3.3.2

  $$Y^a_{\bar{\imath}} \cap Y^{1-a}_{\bar{\imath}+1} = Y^a_{\bar{\imath}} \cap Y^{1-a}_{\bar{\imath}} = \emptyset. \tag{6.26}$$

By Assumption 6, Step 3.3.2 and the fact that $z \in Y_{\bar{\imath}}^{a}$

$$K_z \cap Y_{\bar{\imath}+1}^{1-a} = K_z \cap Y_{\bar{\imath}}^{1-a} = \emptyset. \tag{6.27}$$

Combining (6.25), (6.26) and (6.27) demonstrates that the lemma holds in this case as well.

Hence the lemma holds in both cases. ■

Lemma 6.37 *If the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$, the tasks in $Z_{\bar{\imath}+1}$ are independent.*

Proof. Suppose the inductive assumptions hold for some integer $\bar{\imath} \geq \underline{h}$. By Assumption 1, sets $Y_{\bar{\imath}+1}^{0}$, $Y_{\bar{\imath}+1}^{1}$ and $Z_{\bar{\imath}+1}$ are defined. Two cases will be considered based on how they differ from $Y_{\bar{\imath}}^{0}$, $Y_{\bar{\imath}}^{1}$ and $Z_{\bar{\imath}}$.

- Suppose $Y_{\bar{\imath}+1}^{0}$, $Y_{\bar{\imath}+1}^{1}$ and $Z_{\bar{\imath}+1}$ are defined according to Step 2.3, i.e. there exists a task $y \in Z_{\bar{\imath}}$ such that $Z_{\bar{\imath}+1} = Z_{\bar{\imath}} \setminus \{y\}$. Since $Z_{\bar{\imath}+1} \subset Z_{\bar{\imath}}$ the lemma holds in this case as a consequence of Assumption 6.

- On the other hand, suppose $Y_{\bar{\imath}+1}^{0}$, $Y_{\bar{\imath}+1}^{1}$ and $Z_{\bar{\imath}+1}$ are not defined according to Step 2.3, i.e. there exists a task $z \in Z_{\bar{\imath}}$ such that $Z_{\bar{\imath}+1} = (Z_{\bar{\imath}} \setminus \{z\}) \cup R_{\bar{\imath},z}$, as in Step 3.3. Select $a \in \{0,1\}$ such that $z \in Y_{\bar{\imath}}^{a}$. By Assumption 6 the tasks in $Z_{\bar{\imath}} \setminus \{z\}$ are independent. Since $R_{\bar{\imath},z} \subseteq \tilde{K}_z$ the tasks in $R_{\bar{\imath},z}$ are independent, and Assumption 6 implies that

  $$(Z_{\bar{\imath}} \setminus \{z\}) \cap K(R_{\bar{\imath},z}) \subseteq (Z_{\bar{\imath}} \setminus \{z\}) \cap K_z = \emptyset.$$

  Finally, by the definition (6.16) of $R_{\bar{\imath},z}$ it follows that

  $$(Z_{\bar{\imath}} \setminus \{z\}) \cap J(R_{\bar{\imath},z}) = \emptyset,$$

  so the lemma holds in this case as well.

Hence the lemma holds in both cases. ■

The following corollary combines the results of Lemma 6.30, Lemma 6.32, Lemma 6.33, Lemma 6.35, Lemma 6.36 and Lemma 6.37. It demonstrates that the Z-Algorithm can always select a task y as in Step 2.2, and a task z as in Step 3.1.

Corollary 6.38 *The inductive assumptions hold for every integer $\bar{\imath} \geq \underline{h}$ such that $|Z_{\bar{\imath}}| \leq 3$.*

Lemma 6.39 *The Z-Algorithm terminates after a finite number of iterations.*

Proof. For each integer $i \geq \underline{h}$ for which the sets Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are defined, either Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are defined according to Step 2.3, in which $Z_{i+1} = Z_i \setminus \{y\}$ for some $y \in Z_i$ and hence

$$I \cap K(Z_i) \supseteq I \cap K(Z_i \setminus \{y\}) = K(Z_{i+1});$$

or Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are defined according to Step 3.3, in which case $Z_{i+1} = (Z_i \setminus \{z\}) \cup R_{i,z}$ for some $z \in Z_i$ and hence

$$I \cap K(Z_i) \supset I \cap K((Z_i \setminus \{z\}) \cup R_{i,z}) = I \cap K(Z_{i+1}).$$

Thus, in either case, $I \cap K(Z_{i+1}) \subseteq I \cap K(Z_i)$. Furthermore, the latter case gives $I \cap K(Z_{i+1}) \subset I \cap K(Z_i)$, and this holds for at least half of the sets Z_{i+1}. Since $|I|$ is finite there exists some finite integer $i > \underline{h}$ for which $I \cap K(Z_i) = \emptyset$.

Lemma 6.26, Lemma 6.27 and Lemma 6.28 demonstrate the existence of a set $W \subseteq I$ such that

$$|W| = 4 \qquad \text{and} \qquad I \cap K(W) = \emptyset.$$

By Corollary 6.38 and Lemma 6.35, Assumption 4 holds for all i. Consequently whenever Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are defined according to Step 3.3,

$$W \subseteq Z_{i+1} \cup K(Z_{i+1}). \tag{6.28}$$

By Lemma 6.34, (6.28) also holds whenever Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are defined according to Step 2.3. Since there exists some finite integer $i > \underline{h}$ for which $I \cap K(Z_i) = \emptyset$, $Z_i = W$ and the Z-Algorithm terminates. ■

This completes the proof of the validity of the Z-Algorithm.

The lemmas below describe important properties of the Z-Algorithm. These properties are used in the construction of schedules in Lemma 6.63 and Theorem 6.75.

Lemma 6.40 *All $j \in I$ have $j \in Z_i$ for some integer $i \in \left[\underline{h}, \bar{h}\right]$.*

Proof. Suppose there exists a task $j \in I$ such that $j \notin Z_i$ for all $\underline{h} \leq i \leq \bar{h}$. Select the largest i such that $j \in K(Z_i)$ – it is possible to make such a selection due to Lemma 6.30 and Lemma 6.39. Since $j \notin K(Z_{i+1})$, there exists some $g \in Z_i \setminus Z_{i+1}$

such that $g \to j$. Two cases will now be considered based on how Z_{i+1} differs from Z_i.

- Suppose Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are defined according to Step 2.3, i.e. there exists a task $y \in Z_i$ such that $Z_{i+1} = Z_i \setminus \{y\}$ and hence $g = y$. However, Lemma 6.34 implies
$$j \in I \cap K_g \subseteq K(Z_{i+1}),$$
contradicting the definition of j.

- On the other hand, suppose Y_{i+1}^0, Y_{i+1}^1 and Z_{i+1} are defined according to Step 3.3, i.e. there exists a task $z \in Z_i$ such that $Z_{i+1} = (Z_i \setminus \{z\}) \cup R_{i,z}$ and hence $g = z$. The fact that $j \notin K(Z_i \setminus \{z\})$ and $j \in K_z$ implies that either $j \in R_{i,z}$, in which case $j \in Z_{i+1}$, or $j \in K(R_{i,z})$, in which case $j \in K(Z_{i+1})$. Both of these cases contradict the selection of j.

Consequently, the lemma holds in both cases. ■

Lemma 6.41 *For every integer $i \in \left[\underline{h}+1, \bar{h}\right]$, for each $a \in \{0,1\}$ and for each $j \in Y_i^a$,*
$$j \in Y_{i-1}^a \cup K(Y_{i-1}^a).$$

Proof. Consider any i, a and j as defined in the statement of the lemma. If $j \in Y_{i-1}^a$ the lemma holds, so suppose $j \notin Y_{i-1}^a$. Two cases will now be considered based on how Z_i differs from Z_{i-1}.

- Suppose Y_i^0, Y_i^1 and Z_i are defined according to Step 2.3, i.e. there exists a task $y \in Z_{i-1}$ such that $Z_i = Z_{i-1} \setminus \{y\}$. Since $Z_i \subset Z_{i-1}$ this contradicts the selection of j.

- On the other hand, suppose Y_i^0, Y_i^1 and Z_i are defined according to Step 3.3, i.e. there exists a task $z \in Z_{i-1}$ such that $Z_i = (Z_{i-1} \setminus \{z\}) \cup R_{i-1,z}$ and hence $j \in R_{i-1,z} \subseteq K_z$.

Consequently, the lemma holds. ■

Lemma 6.42 *For every integer $i \in \left[\underline{h}+1, \bar{h}\right]$, for each $a \in \{0,1\}$ and for each $j \in Y_i^a$,*
$$j \notin Y_{i-1}^{1-a} \cup K\left(Y_{i-1}^{1-a}\right).$$

Proof. Consider any i, a and j as defined in the statement of the lemma. If $j \in Y_{i-1}^a$ the lemma holds as a consequence of Corollary 6.38 and Lemma 6.32, so suppose $j \notin Y_{i-1}^a$. Two cases will now be considered based on how Z_i differs from Z_{i-1}.

- Suppose Y_i^0, Y_i^1 and Z_i are defined according to Step 2.3, i.e. there exists a task $y \in Z_{i-1}$ such that $Z_i = Z_{i-1} \setminus \{y\}$. Since $Z_i \subset Z_{i-1}$ this contradicts the selection of j.

- On the other hand, suppose Y_i^0, Y_i^1 and Z_i are defined according to Step 3.3, i.e. there exists a task $z \in Z_{i-1}$ such that $Z_i = (Z_{i-1} \setminus \{z\}) \cup R_{i-1,z}$ and hence (6.16) implies

$$\begin{aligned} j \in R_{i-1,z} &= I \cap \left(\tilde{K}_z \setminus K(Z_{i-1} \setminus \{z\})\right) \\ &\subseteq I \cap \left(\tilde{K}_z \setminus K(Y_{i-1}^{1-a})\right) = \left(I \cap \tilde{K}_z\right) \setminus K(Y_{i-1}^{1-a}). \end{aligned}$$

  Since $z \in Y_{i-1}^a$ and $z \to j$, Corollary 6.38 implies $j \notin Y_{i-1}^{1-a}$.

Consequently, the lemma holds. ■

Lemma 6.43 *For every integer $i \in \left[\underline{h}+1, \bar{h}\right]$ and for each $a \in \{0,1\}$, the tasks in $Y_{i-1}^{1-a} \cup Y_i^a$ are independent.*

Proof. Consider any i and a as defined in the statement of the lemma, and consider any $j \in Y_i^a$. Two cases will now be considered based on how Z_i differs from Z_{i-1}.

- Suppose Y_i^0, Y_i^1 and Z_i are defined according to Step 2.3, i.e. there exists a task $y \in Z_{i-1}$ such that $Z_i = Z_{i-1} \setminus \{y\}$. Since $Z_i \subset Z_{i-1}$ the lemma holds in this case as a consequence of

$$Y_{i-1}^{1-a} \cup Y_i^a \subseteq Y_{i-1}^{1-a} \cup Y_{i-1}^a = Z_{i-1} \tag{6.29}$$

  and Corollary 6.38.

- On the other hand, suppose Y_i^0, Y_i^1 and Z_i are defined according to Step 3.3, i.e. there exists a task $z \in Z_{i-1}$ such that $Z_i = (Z_{i-1} \setminus \{z\}) \cup R_{i-1,z}$. If $z \in Y_{i-1}^{1-a}$, (6.29) holds and the lemma follows from Corollary 6.38. On the other hand, if $z \in Y_i^a$

$$Y_{i-1}^{1-a} \cup Y_i^a = Y_i^{1-a} \cup Y_i^a = Z_i$$

  and the lemma follows from Corollary 6.38 and Lemma 6.37.

Consequently, the lemma holds in both cases. ■

Lemma 6.44 *For every integer* $i \in \left[\underline{h}+1, \bar{h}-1\right]$ *and for each* $a \in \{0,1\}$,

$$\left|Y_{i-1}^{1-a} \cup Y_i^a\right| \leq 3.$$

Proof. As a consequence of Lemma 6.43, $Y_{i-1}^{1-a} \cup Y_i^a$ is a set of independent tasks, and is a subset of I. Since I is a canonical transitive set, Lemma 6.26, Lemma 6.27 and Lemma 6.28 imply that

- $\left|Y_{i-1}^{1-a} \cup Y_i^a\right| \leq 4$ and
- if $\left|Y_{i-1}^{1-a} \cup Y_i^a\right| = 4$ it follows that $Y_{i-1}^{1-a} \cup Y_i^a = Z_{\bar{h}}$.

Suppose $\left|Y_{i-1}^{1-a} \cup Y_i^a\right| = 4$. For all $j \in Z_{\bar{h}}$, $I \cap K_j = \emptyset$ and hence $Y_{i-1}^{1-a} \cup Y_i^a = Z_{\bar{h}}$ implies $Y_{i-1}^{1-a} = Y_i^{1-a}$. Consequently $i = \bar{h}$, and the lemma follows. ■

Now that some properties of minimal singular quartets, canonical transitive sets and the Z-Algorithm have been established, they will be used in the construction of three problem instances in the following subsection.

6.3.5 Dominance of Solvable Cases

Consider an arbitrary algorithm $\mathscr{A} \in \mathfrak{F}$, and suppose that the domain of this algorithm contains some particular partial order $\prec$ containing a singular quartet – it will later be proven that no such algorithm exists as this would lead to a contradiction. From this point, it will be assumed that algorithm $\mathscr{A}$ is being used to assign all values of μ, and that the precedence constraints of all considered problem instances are isomorphic to $\prec$.

By Lemma 6.24 the poset $\prec$ contains at least one minimal singular quartet, and by definition there is at least one corresponding canonical transitive set. The Z-Algorithm will now be applied to some canonical transitive set given by $\prec$ to construct several problem instances, φ^1, φ^2 and φ^3. It will then be demonstrated that algorithm $\mathscr{A}$ cannot minimise the maximum lateness for all three instances. Since the only properties of $\mathscr{A}$ and $\prec$ that are used to prove this are that $\mathscr{A} \in \mathfrak{F}$ and that $\prec$ contains a singular quartet, it follows that $\mathcal{D}^{\mathscr{A}} \subseteq \mathcal{D}^{\mathscr{XR}}$. The considered instances have much in common, described below.

Select as Z_0 some minimal singular quartet, and let $B \subset Z_0$ be a singular pair. Select a canonical transitive set I covered by B, and apply the Z-Algorithm to I, with the initial index $\underline{h} = 1$. Note that this generates sets $Z_1, Z_2, \ldots, Z_{\bar{h}}$, and results in $Z_1 = B$ and $|Z_{\bar{h}}| = 4$.

Denoting

$$Z = \bigcup_{i=0}^{\bar{h}} Z_i = Z_0 \cup I,$$

the values of processing times and due dates common to φ^1, φ^2 and φ^3 will now be specified. For all tasks $j \in N \setminus Z$, $d_j = 5\bar{h} + 1$ and $p_j = \varepsilon$, where $\varepsilon > 0$ may be made arbitrarily small. The purpose of this is to eliminate all tasks not in Z, i.e. not in the prohibited structure being examined. In order to avoid tedious repetition of the phrase "for sufficiently small $\varepsilon > 0$" in the statements of the dozens of lemmas and corollaries in this subsection it will hereafter be assumed that ε is "sufficiently small" for every desired purpose. It will only be stated when the fact that ε can be made arbitrarily small is used explicitly in a proof.

For all tasks $j \in Z \setminus Z_0$, let

$$d_j = 4\bar{h} + \max\left\{i : 1 \leq i \leq \bar{h} \wedge j \in Z_i\right\},$$

i.e. $4\bar{h}$ plus the index of the last Z_i of which j is an element; and

$$p_j = \left|\left\{i : 1 \leq i \leq \bar{h} \wedge j \in Z_i\right\}\right|,$$

i.e. the number of sets Z_i of which j is an element. For both $j \in Z_0 \setminus Z_1$, $p_j = d_j = 1$. In all problem instances $m = 3$. As a consequence of these assignments, the only difference between the problems will be the processing time and due date of the two tasks composing Z_1.

Several definitions are required before any specific problem instances are presented. For any $\alpha \in \{1, 2, 3\}$, denote by $\eta_\alpha^{\mathscr{A}}$ the schedule produced by the application of algorithm $\mathscr{A}$ to the problem instance φ^α. Note that the assumption that $\prec$ is in the domain of $\mathscr{A}$ implies that $\eta_\alpha^{\mathscr{A}}$ is an optimal schedule for the problem instance φ^α. If $J(Z_0) \neq \emptyset$, define

$$\bar{t}_\alpha = \max_{j \in J(Z_0)} C_j\left(\eta_\alpha^{\mathscr{A}}\right),$$

otherwise $\bar{t}_\alpha = 0$. In words, $\bar{t}_\alpha$ is the earliest point in time during schedule $\eta_\alpha^{\mathscr{A}}$ at which no predecessor of any task in the minimal singular quartet Z_0 is available. Further define

$$\hat{t}_\alpha = \max_{j \in Z_0 \setminus Z_1} C_j\left(\eta_\alpha^{\mathscr{A}}\right).$$

In words, $\hat{t}_\alpha$ is the earliest point in time during schedule $\eta_\alpha^{\mathscr{A}}$ at which no task that

is both

- in the minimal singular quartet Z_0 and
- not in the singular pair Z_1

is available.

Consider the points of allocation $t_0, t_1, \ldots$ of schedule $\eta_\alpha^{\mathscr{A}}$ and define

$$R_\alpha = \left\{ i : i \in \mathbb{N} \wedge \bar{t}_\alpha \leq t_i < \hat{t}_\alpha \right\}.$$

This is the set of every index i for which, at the corresponding point of allocation t_i in the schedule $\eta_\alpha^{\mathscr{A}}$,

- no task in $J(Z_0)$ is available and
- at least one task in $Z_0 \setminus Z_1$ is available.

Select

$$\bar{\imath}_\alpha = \min R_\alpha \qquad \text{and} \qquad \hat{\imath}_\alpha = \max R_\alpha,$$

i.e. $\hat{t}_\alpha \in (t_{\hat{\imath}_\alpha}, t_{\hat{\imath}_\alpha+1}]$ and either $\bar{t}_\alpha = t_0 = 0 = t_{\bar{\imath}_\alpha}$ or $\bar{t}_\alpha \in (t_{\bar{\imath}_\alpha-1}, t_{\bar{\imath}_\alpha}]$. Corollary 6.46 establishes that $R_\alpha \neq \emptyset$, and hence $\bar{\imath}_\alpha$ and $\hat{\imath}_\alpha$ are well defined. The examination of these indices of points of allocation facilitates subsequent reasoning about the processing of the tasks in $J(Z_0)$ and the tasks in $Z_0 \setminus Z_1$. This is particularly useful since each considered task has equal processing times and due dates in the three problem instances φ^1, φ^2 and φ^3.

Lemma 6.45 *For any $\alpha \in \{1, 2, 3\}$, there exists a point of allocation $t_i \in [\bar{t}_\alpha, n\varepsilon)$ for schedule $\eta_\alpha^{\mathscr{A}}$.*

Proof. If $\bar{t}_\alpha = 0$ then

$$t_{\bar{\imath}_\alpha} = \bar{t}_\alpha = 0 < n\varepsilon$$

and the lemma holds.

Now consider the case where $\bar{t}_\alpha > 0$ and define

$$i = \min \left\{ i' : i' \in \mathbb{N} \wedge t_i \geq \bar{t}_\alpha \right\}.$$

For any time t in schedule $\eta_\alpha^{\mathscr{A}}$, if any task $j \in J(Z_0)$ has $C_j\left(\eta_\alpha^{\mathscr{A}}\right) > t$, the selection of Z_0 as a minimal singular quartet, along with Lemma 6.25, implies that at least two tasks in Z_0 are not yet available for processing. Since

$$|Z_0| - 2 = 2 < 3 = m,$$

the definition of Algorithm P implies that there is at least one task from $N \setminus Z$ being processed at time t, implying

$$t_{i-1} \leq \sum_{j \in N \setminus Z} \left(p_j - p_j \left(t_{i-1}, \eta_\alpha^{\mathscr{A}} \right) \right). \tag{6.30}$$

Note that the definition of $\bar{t}_\alpha$ implies that it is possible to select a task

$$g \in \left(N_1^{i-1} \cup N_2^{i-1} \right) \cap J(Z_0).$$

Two cases are now considered.

- If $g \in N_1^{i-1}$, it follows from (6.30) that

$$t_i = t_{i-1} + p_g \left(t_{i-1}, \eta_\alpha^{\mathscr{A}} \right) \leq p_g + \sum_{j \in N \setminus (Z \cup \{g\})} \left(p_j - p_j \left(t_{i-1}, \eta_\alpha^{\mathscr{A}} \right) \right)$$

$$\leq \sum_{j \in N \setminus Z} p_j = |N \setminus Z| \, \varepsilon < n\varepsilon.$$

- On the other hand, if $g \in N_2^{i-1}$, it follows from (6.30) that

$$t_i \leq t_{i-1} + \sum_{j \in N_2^{i-1}} \left(p_j \left(t_{i-1}, \eta_\alpha^{\mathscr{A}} \right) - p_j \left(t_i, \eta_\alpha^{\mathscr{A}} \right) \right)$$

$$\leq t_{i-1} + \sum_{j \in N_2^{i-1}} \min \left\{ p_g \left(t_{i-1}, \eta_\alpha^{\mathscr{A}} \right), p_j \left(t_{i-1}, \eta_\alpha^{\mathscr{A}} \right) \right\}$$

$$\leq \sum_{j \in N \setminus Z} p_j + \sum_{j \in Z \setminus K_g} p_g \left(t_{i-1}, \eta_\alpha^{\mathscr{A}} \right) \leq \sum_{j \in N \setminus (Z \cap K_g)} \varepsilon < n\varepsilon.$$

Hence the lemma holds in both cases. ■

Corollary 6.46 *It holds that* $R_\alpha \neq \emptyset$.

Proof. This follows from the preceding lemma and the fact that $\hat{t}_\alpha \geq 1$. ■

Corollary 6.47 *For any schedule* σ *for problem instance* φ^α, *and all* $j \in Z_0$,

$$p_j - n\varepsilon < p_j \left(t_{\bar{\imath}_\alpha}, \eta_\alpha^{\mathscr{A}} \right) \leq p_j.$$

For the points of allocation t_i for $\eta_\alpha^{\mathscr{A}}$, define the set

$$S_\alpha = \left\{ i : i \in R_\alpha \wedge (N_1^i \cup N_2^i) \setminus Z \neq \emptyset \right\},$$

i.e. S_α is a set of indices of points of allocation for the schedule $\eta_\alpha^{\mathscr{A}}$, specifically the subset of R_α for which a task not in Z is processed. This definition will be used in the lemma below.

Lemma 6.48 *For the points of allocation t_i for $\eta_\alpha^{\mathscr{A}}$,*

$$\sum_{i \in S_\alpha} (t_{i+1} - t_i) < n^3 \varepsilon.$$

Proof. Note that, for any $i \in S_\alpha$, there exists a task $g \in (N_1^i \cup N_2^i) \setminus Z$. Hence, by Lemma 4.10,

$$t_{i+1} - t_i \leq n p_g = n\varepsilon.$$

As a consequence, Corollary 4.8 implies

$$\sum_{i \in S_\alpha} (t_{i+1} - t_i) \leq |S_\alpha| n\varepsilon < sn\varepsilon \leq n^3 \varepsilon$$

and the lemma follows. ■

The preceding results provide insight into the processing of tasks in the early intervals of $\eta_\alpha^{\mathscr{A}}$. They will be used throughout the rest of this subsection when analysing these schedules.

The first problem instance considered is φ^1. Recall that, for all three considered problem instances the only values not already specified are the processing times and due dates of the tasks in Z_1. Let φ^1 be defined such that they have a value of one, i.e. both $j \in Z_1$ have $p_j = d_j = 1$. Until stated otherwise, points of allocation t_i, sets of available tasks A_i and sets of tasks receiving equal processing time N_a^i will be considered to be those values associated with schedule $\eta_1^{\mathscr{A}}$. The following lemmas establish the fact that any algorithm that produces an optimal schedule for φ^1 must assign at least three of the four tasks in Z_0 an equal value of μ_j.

Lemma 6.49 demonstrates that all tasks in $Z \setminus Z_0$ are unimportant to the maximum lateness – that, in effect, it is trivial to schedule them so that they do not increase the maximum lateness beyond what is obtained by consideration of only the tasks in Z_0. Lemma 6.50 uses this result to establish the existence of a schedule σ^1 with a particular maximum lateness. Since it is assumed that $\eta_1^{\mathscr{A}}$ is an optimal schedule for φ^1, Lemma 6.50 establishes an upper bound on $L_{max}\left(\eta_1^{\mathscr{A}}\right)$. This is used in Lemma 6.51 to determine how the tasks in Z_0 are scheduled in $\eta_1^{\mathscr{A}}$. Finally, this result is used in Lemma 6.53 to derive properties of the μ's assigned by algorithm

$\mathscr{A}$ to the tasks in Z_0. These properties are used in the analysis of φ^2.

Lemma 6.49 *For any schedule σ for φ^1 such that, at all times in*

$$\left[0, \max_{j \in N} C_j(\sigma)\right]$$

at least one machine is processing a task,

$$L_{max}(\sigma) = \max_{j \in Z_0} C_j(\sigma) - 1.$$

Proof. Since $d_j = 1$ for all $j \in Z_0$,

$$\max_{j \in Z_0} C_j(\sigma) - 1 \leq L_{max}(\sigma).$$

For adequately small ε,

$$\max_{j \in N \setminus Z_0} (C_j(\sigma) - d_j) \leq \max_{j \in Z_0} C_j(\sigma) + \sum_{j \in N \setminus Z_0} p_j - \min_{j \in N \setminus Z_0} d_j$$

$$< \max_{j \in Z_0} C_j(\sigma) + \sum_{1 \leq i \leq \bar{h}} |Z_i| + n\varepsilon - \left(4\bar{h} + 2\right)$$

$$< \max_{j \in Z_0} C_j(\sigma) + 4\bar{h} + n\varepsilon - (4\bar{h} + 2) < \max_{j \in Z_0} C_j(\sigma) - 1 \leq L_{max}(\sigma).$$

Consequently, $L_{max}(\sigma) = C_j(\sigma) - d_j$ for some $j \in Z_0$, and the lemma is proven. ■

Define

$$t'_1 = t_{\bar{\imath}_1} + \frac{1}{3} \sum_{g \in Z_0} p_g \left(t_{\bar{\imath}_1}, \eta_1^{\mathscr{A}}\right).$$

Lemma 6.50 *There exists a schedule σ^1 for problem instance φ^1 for which*

$$L_{max}\left(\sigma^1\right) = t'_1 - 1.$$

Proof. If $\varepsilon > 0$ is adequately small, by Corollary 6.47,

$$\max_{g \in Z_0} \left[p_g \left(t_{\bar{\imath}_1}, \eta_1^{\mathscr{A}}\right)\right] \leq 1 \leq \frac{4}{3}(1 - n\varepsilon) \leq t'_1 - t_{\bar{\imath}_1}.$$

Hence, McNaughton's Algorithm can create a schedule σ^1 that is identical to $\eta_1^{\mathscr{A}}$ on the interval $[0, t_{\bar{\imath}_1}]$, and which schedules the remaining processing time of all tasks $j \in Z_0$ during $[t_{\bar{\imath}_1}, t'_1]$. By Lemma 6.49, the remaining tasks may be scheduled in σ^1 so that

$$L_{max}\left(\sigma^1\right) = t'_1 - 1,$$

and the lemma is proven. ■

Lemma 6.51 *In schedule $\eta_1^{\mathscr{A}}$, all machines at all times in $\left[t_{\bar{\imath}_1}, \hat{t}_1\right]$ are processing tasks from Z_0, and $C_j\left(\eta_1^{\mathscr{A}}\right) \leq \hat{t}_1$ for all $j \in Z_0$.*

Proof. If, for any $j \in Z_0$, $C_j\left(\eta_1^{\mathscr{A}}\right) > t'_1$, it follows from Lemma 6.50 that

$$L_{max}\left(\eta_1^{\mathscr{A}}\right) > t'_1 - 1 = L_{max}\left(\sigma^1\right),$$

contradicting the assumption that $\eta_1^{\mathscr{A}}$ is optimal. Hence, in schedule $\eta_1^{\mathscr{A}}$, there is at least one task from $Z_0 \setminus Z_1$ completed at time t'_1 and thus $t'_1 = \hat{t}_1$. Consequently, in schedule $\eta_1^{\mathscr{A}}$, all machines at all times in $\left[t_{\bar{\imath}_1}, \hat{t}_1\right]$ are processing tasks from Z_0. ■

Corollary 6.52 *It holds that*

$$\frac{4}{3} \leq \hat{t}_1 = \max_{j \in Z_0} C_j\left(\eta_1^{\mathscr{A}}\right) \leq t_{\bar{\imath}_1} + \frac{4}{3} < n\varepsilon + \frac{4}{3}.$$

Lemma 6.53 *When applied to φ^1, algorithm $\mathscr{A}$ assigns*

$$\mu_j = \min_{g \in Z_0} \mu_g$$

for at least three tasks $j \in Z_0$.

Proof. Recall that $\hat{t}_1 \in (t_{\hat{\imath}_1}, t_{\hat{\imath}_1+1}]$ and consider sets $N_1^{\hat{\imath}_1}$, $N_2^{\hat{\imath}_1}$ and $N_3^{\hat{\imath}_1}$. Two cases will now be considered: either there is a task $q \in Z_0$ for which $C_q\left(\eta_1^{\mathscr{A}}\right) \leq t_{\hat{\imath}_1}$, or there is no such task.

- Suppose that no task $q \in Z_0$ has $C_q\left(\eta_1^{\mathscr{A}}\right) \leq t_{\hat{\imath}_1}$, i.e. $Z_0 \subseteq A_{\hat{\imath}_1}$. Two sub-cases will now be considered: that of a task in $Z_0 \cap N_1^{\hat{\imath}_1}$, and that of a task in $Z_0 \cap N_3^{\hat{\imath}_1}$.

  - Consider a task $j \in N_1^{\hat{\imath}_1}$. Lemma 6.51 implies that $g \in N_3^i$ for all $g \in A_i \setminus Z$ and all $i \in R_1 \setminus \{\hat{\imath}_1\}$. Hence no task is completed during $(t_{\bar{\imath}_1}, t_{\hat{\imath}_1}]$, so Lemma 4.7 implies that $j \in N_1^i$ for all $i \in R_1$. Thus, by (4.7), Lemma 6.45 and Corollary 6.52,

    $$p_j \geq \sum_{i : j \in N_1^i} (t_{i+1} - t_i) \geq t_{\hat{\imath}_1+1} - t_{\bar{\imath}_1} \geq \hat{t}_1 - t_{\bar{\imath}_1} > \frac{4}{3} - n\varepsilon.$$

    For a small enough value of $\varepsilon > 0$, this is a contradiction of the definition of φ^1, for which $p_j = 1$.

– If, on the other hand, $j \in N_3^{\hat{\imath}_1}$, $j \in N_3^i$ for all $i \in R_1$ by the same reasoning as the previous sub-case. Thus, by (4.9) and Corollary 6.47, for a small enough value of $\varepsilon > 0$,

$$0 < p_j - n\varepsilon < p_j\left(t_{\bar{\imath}_1}, \eta_1^{\mathscr{A}}\right) = p_j\left(t_{\hat{\imath}_1+1}, \eta_1^{\mathscr{A}}\right).$$

This implies $C_j\left(\eta_1^{\mathscr{A}}\right) > t_{\hat{\imath}_1+1}$, contradicting Lemma 6.51.

As a consequence of the two sub-cases above, $Z_0 \subseteq N_2^{\hat{\imath}_1}$. Hence, Corollary 6.52 and the definition of N_2 together imply, for all $j \in Z_0$ and $g \in Z_0$,

$$\mu_j = p_j\left(t_{\hat{\imath}_1}, \eta_1^{\mathscr{A}}\right) + \mu_j - \left(p_j\left(t_{\hat{\imath}_1}, \eta_1^{\mathscr{A}}\right) - p_j\left(t_{\hat{\imath}_1+1}, \eta_1^{\mathscr{A}}\right)\right)$$

$$= p_g\left(t_{\hat{\imath}_1}, \eta_1^{\mathscr{A}}\right) + \mu_g - \left(p_g\left(t_{\hat{\imath}_1}, \eta_1^{\mathscr{A}}\right) - p_g\left(t_{\hat{\imath}_1+1}, \eta_1^{\mathscr{A}}\right)\right) = \mu_g.$$

Thus, the lemma is proven in the case where there is no task $q \in Z_0$ such that $C_q\left(\eta_1^{\mathscr{A}}\right) \leq t_i$.

- On the other hand, suppose $C_q\left(\eta_1^{\mathscr{A}}\right) \leq t_{\hat{\imath}_1}$ for some $q \in Z_0$ and select i' such that $C_q\left(\eta_1^{\mathscr{A}}\right) \in (t_{i'}, t_{i'+1}]$. Now consider sets $N_1^{i'}$, $N_2^{i'}$ and $N_3^{i'}$, and observe that either $q \in N_1^{i'}$ or $q \in N_2^{i'}$. Since Lemma 6.51 implies

$$C_g\left(\eta_1^{\mathscr{A}}\right) = \max_{j \in Z_0} C_j\left(\eta_1^{\mathscr{A}}\right)$$

for three $g \in Z_0$, no task is completed during $(t_{\bar{\imath}_1}, t_{i'}]$, and thus $Z_0 \subseteq A_{i'}$. Consequently, because $|Z_0| = 4 > 3 = m$, there exists a task $v \in Z_0 \setminus \{q\}$ such that $v \in N_2^{i'} \cup N_3^{i'}$. Hence, by Lemma 4.4,

$$\mu_q = p_q\left(t_{i'+1}, \eta_1^{\mathscr{A}}\right) + \mu_q \geq p_v\left(t_{i'+1}, \eta_1^{\mathscr{A}}\right) + \mu_v > \mu_v. \tag{6.31}$$

As before, Lemma 6.51 implies that $g \in N_3^i$ for all $g \in A_i \setminus Z$ and all $i \in R_1 \setminus \{\hat{\imath}_1\}$. Consequently, no task is completed during $(t_{\bar{\imath}_1}, t_{i'}]$, and $(Z_0 \setminus \{q\}) \subseteq N_1^i$ for all $i' \leq i < \hat{\imath}_1$. Again, two sub-cases will be considered: that of a task in $(Z_0 \setminus \{q\}) \cap N_1^{i'}$, and that of a task in $(Z_0 \setminus \{q\}) \cap N_3^{i'}$.

– For any $j \in Z_0 \setminus \{q\}$, if $j \in N_1^{i'}$, Lemma 4.7 implies that $j \in N_1^i$ for all i for which $\bar{\imath}_1 \leq i \leq i'$. As a consequence of Lemma 6.51, the three tasks in $Z_0 \setminus \{q\}$ are all processed continuously during $\left[t_{i'+1}, \hat{t}_1\right]$. Hence, by (4.7),

Lemma 6.45 and Corollary 6.52,

$$p_j \geq \sum_{\bar{\imath}\leq i\leq i'} (t_{i+1} - t_i) + p_j\left(t_{i'+1}, \eta_1^{\mathscr{A}}\right)$$

$$\geq t_{i'+1} - t_{\bar{\imath}_1} + \hat{t}_1 - t_{i'+1} = \hat{t}_1 - t_{\bar{\imath}_1} > \frac{4}{3} - n\varepsilon,$$

a contradiction for sufficiently small ε as in the previous case.

– If $j \in N_3^{i'}$, Lemma 4.7 implies $j \in N_3^i$ for all i such that $\bar{\imath}_1 \leq i \leq i'$. Since

$$t_{i'+1} \geq C_q\left(\eta_1^{\mathscr{A}}\right) \geq p_q = 1,$$

it follows that, by Corollary 6.52, $\hat{t}_1 - t_{i'+1} < n\varepsilon + \frac{1}{3}$. Consequently, for a small enough value of $\varepsilon > 0$,

$$0 < p_j\left(t_{\bar{\imath}_1}, \eta_1^{\mathscr{A}}\right) - n\varepsilon - \frac{1}{3} = p_j\left(t_{i'+1}, \eta_1^{\mathscr{A}}\right) - n\varepsilon - \frac{1}{3}$$

$$= p_j\left(\hat{t}_1, \eta_1^{\mathscr{A}}\right) + \hat{t}_1 - t_{i'+1} - n\varepsilon - \frac{1}{3} < p_j\left(\hat{t}_1, \eta_1^{\mathscr{A}}\right),$$

contradicting Corollary 6.52 as in the previous case.

Therefore $(Z_0 \setminus \{q\}) \subseteq N_2^{i'}$ and, for all $j \in Z_0 \setminus \{q\}$ and $g \in Z_0 \setminus \{q\}$,

$$p_j\left(t_{i'+1}, \eta_1^{\mathscr{A}}\right) + \mu_j = p_g\left(t_{i'+1}, \eta^{\mathscr{A}}\right) + \mu_g. \tag{6.32}$$

By Lemma 6.51,

$$p_j\left(t_{i'+1}, \eta_1^{\mathscr{A}}\right) = p_g\left(t_{i'+1}, \eta^{\mathscr{A}}\right) = \hat{t}_1 - t_{i'+1}$$

which implies, along with (6.32), $\mu_j = \mu_g$. All that is required is to prove that $\mu_q \geq \mu_j = \mu_g$, which follows directly from (6.31).

Thus, the lemma is proven in both cases. ■

The value of $\min_{j\in Z_0} \mu_j$ produced when algorithm $\mathscr{A}$ is applied to φ^1 will appear in the other problem instances considered. Denote this value $\bar{\mu}$.

The preceding lemma may now be used in the examination of a different problem instance, φ^2, where the tasks $j \in Z_1$ have

$$p_j = d_j = 4\bar{h} - 1 + |\{i : 0 \leq i \leq \bar{h} \wedge j \in Z_i\}|,$$

i.e. one for every Z_i of which j is an element, except for Z_1, which contributes $4\bar{h}$. Note that the only differences between φ^1 and φ^2 are the due dates and processing times of the two tasks in Z_1. Hence, for all tasks $j \in Z \setminus Z_1$, d_j and φ_j (see 4.4 on page 66) remain unchanged between these two problem instances. By the definition of the family $\mathfrak{F}$, the value of μ_j assigned by algorithm $\mathscr{A} \in \mathfrak{F}$ will also not change between these two problem instances.

Until stated otherwise, points of allocation t_i, sets of available tasks A_i and sets of tasks receiving equal processing time N_a^i will be considered to be those values associated with schedule $\eta_2^{\mathscr{A}}$. The lemmas below demonstrate that algorithm $\mathscr{A}$ gives at most one of the two tasks $j \in Z_1$ a value of μ_j which is, in a manner defined rigorously below, close to $1 + \bar{\mu} - p_j$.

This result will then be used in the consideration of the third problem instance, φ^3, for which any algorithm in $\mathfrak{F}$ can only produce an optimal schedule by assigning a value of μ_j that is close to $1 + \bar{\mu} - p_j$. This result demonstrates that the algorithm $\mathscr{A}$ cannot produce an optimal schedule for all problem instances for which the precedence constraints are isomorphic to $\prec$. Since $\prec$ is an arbitrary partial order containing a singular quartet it follows that a singular quartet is sufficient to ensure that the partial order is not in the domain of any algorithm in $\mathfrak{F}$. Since $\mathscr{LR} \in \mathfrak{F}$ this, combined with Theorem 6.23, demonstrates both that the $\mathcal{D}^{\mathscr{LR}}$ is precisely the class of partial orders that do not contain singular quartets, and that $\mathcal{D}^{\mathscr{A}} \subseteq \mathcal{D}^{\mathscr{LR}}$ for all $\mathscr{A} \in \mathfrak{F}$.

As before, the proof will proceed by first constructing a schedule (σ^2 in Lemma 6.54 below) and then comparing its maximum lateness to that of an optimal schedule. By the assumption that $\prec \in \mathcal{D}^{\mathscr{A}}$ the optimal maximum lateness is equal to $L_{max}\left(\eta_2^{\mathscr{A}}\right)$, and thus gives information on the μ's generated by algorithm $\mathscr{A}$ for the problem instance φ^2.

Lemma 6.54 *There exists a schedule σ^2 for φ^2 such that*

$$L_{max}\left(\sigma^2\right) = n\varepsilon + \frac{5}{9}.$$

Proof. Construct σ^2 as follows. Schedule all tasks in $J(Z_0)$ arbitrarily during the interval $[0, n\varepsilon]$. Use the Modified McNaughton Algorithm to schedule $\frac{2}{3}$ of a unit of both tasks in Z_1 and $\frac{1}{3}$ of a unit of both tasks in $Z_0 \setminus Z_1$ during the interval

$$\left[n\varepsilon, n\varepsilon + \frac{2}{3}\right].$$

Again use the Modified McNaughton Algorithm to schedule $\frac{2}{3}$ of a unit of processing

from each of the four tasks in Z_0 during

$$\left[n\varepsilon + \frac{2}{3}, n\varepsilon + \frac{14}{9}\right].$$

This will complete both tasks in $Z_0 \setminus Z_1$, and hence

$$C_j\left(\sigma^2\right) - d_j \leq n\varepsilon + \frac{5}{9} \qquad \text{for both } j \in Z_0 \setminus Z_1. \tag{6.33}$$

Schedule $4\bar{h} - \frac{1}{3}$ units of both tasks in Z_1 during

$$\left[n\varepsilon + \frac{14}{9}, n\varepsilon + 4\bar{h} + \frac{11}{9}\right].$$

By the definition of a transitive set, all $j \in J(Z_i) \setminus Z$ for any $1 \leq i \leq \bar{h}$ have

$$J_j \cap \bigcup_{i=1}^{\bar{h}} Z_i = \emptyset,$$

i.e. any task j which precedes a task in Z is not itself preceded by any task in $Z \setminus (Z_0 \setminus Z_1)$. Thus, schedule all such tasks j during this interval as well. This will complete the sole task in $Z_1 \setminus Z_2$, giving

$$C_j\left(\sigma^2\right) - d_j = n\varepsilon + \frac{2}{9} \qquad \text{for the sole task } j \in Z_1 \setminus Z_2. \tag{6.34}$$

By Lemma 6.27 and the definition of a canonical transitive set, $|Z_i| \leq 3$ if $2 \leq i < \bar{h}$. Hence, for each $2 \leq i < \bar{h}$, schedule one unit of all $j \in Z_i$ during

$$\left[n\varepsilon + 4\bar{h} + i - \frac{7}{9}, n\varepsilon + 4\bar{h} + i + \frac{2}{9}\right].$$

For all $2 \leq i \leq \bar{h}$ and all $j \in Z_i$

$$p_j\left(n\varepsilon + 4\bar{h} + i - \frac{7}{9}, \sigma^2\right) = |\{i' : i \leq i' \leq \bar{h} \wedge j \in Z_{i'}\}|.$$

Thus, for any $2 \leq i < \bar{h}$

$$C_j\left(\sigma^2\right) - d_j = n\varepsilon + \frac{2}{9} \qquad \text{for the sole task } j \in Z_i \setminus Z_{i+1}. \tag{6.35}$$

Use the Modified McNaughton Algorithm to schedule all four tasks in $Z_{\bar{h}}$ for one

unit during

$$\left[n\varepsilon + 5\bar{h} - \frac{7}{9}, n\varepsilon + 5\bar{h} + \frac{5}{9}\right],$$

resulting in

$$C_j\left(\sigma^2\right) - d_j \leq n\varepsilon + \frac{5}{9} \qquad \text{for all } j \in Z_{\bar{h}}. \tag{6.36}$$

Note that (6.36) is an equality for three of the tasks in $Z_{\bar{h}}$.

Finally, schedule all remaining tasks, i.e. all tasks that are not in Z and do not precede any task in Z, arbitrarily during

$$\left[n\varepsilon + 5\bar{h} + \frac{5}{9}, 2n\varepsilon + 5\bar{h} + \frac{5}{9}\right]$$

giving

$$C_j\left(\sigma^2\right) - d_j \leq 2n\varepsilon - \frac{4}{9} \qquad \text{for all } j \in N \setminus Z. \tag{6.37}$$

The lemma follows from (6.33), (6.34), (6.35), (6.36) and (6.37), and the fact that (6.36) is an equality for three of the tasks in $Z_{\bar{h}}$. ■

As stated earlier, for all tasks $j \in Z \setminus Z_1$, d_j and $\varphi_j{}^*$ are the same in problem instance φ^1 and φ^2, and hence the definition of family $\mathfrak{F}$ implies that algorithm $\mathscr{A} \in \mathfrak{F}$ will produce the same value of μ_j for both problem instances. Consequently, Lemma 6.53 implies the existence of a task $j \in Z_0 \setminus Z_1$ for which $\mu_j = \bar{\mu}$ when algorithm $\mathscr{A}$ is applied to φ^2.

Some definitions will now be introduced for use in the analysis of schedule $\eta_2^{\mathscr{A}}$. Select $w \in Z_0 \setminus Z_1$ such that

$$\mu_w = \bar{\mu} = 1 + \bar{\mu} - p_w;$$

in subsequent lemmas, the processing of w will be compared to the processing of the other three tasks in Z_0. Task v is defined to be the remaining task in $Z_0 \setminus Z_1$, i.e. $Z_0 \setminus Z_1 = \{v, w\}$. Note that Lemma 6.53 implies that $\mu_v \geq \mu_w = \bar{\mu}$. A set $Z' \subseteq Z_0$ will also be defined. Specifically, if

$$p_v\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_v \leq 1 + \bar{\mu} + \frac{1}{25n^3} \tag{6.38}$$

define $Z' = Z_0$, otherwise define $Z' = Z_0 \setminus \{v\}$.

Lemma 6.55 below establishes that, if both tasks $j \in Z_1$ have μ_j "close" to

*Recall that φ_j is a tuple containing all information about the successors of a task j, and if $K_j \neq \emptyset$ by definition φ_j is a problem instance. It is important to keep in mind the difference between subinstances of a given problem instances, denoted by φ with a subscript, and specific example problem instances used in the analysis in this section, denoted by φ with a superscript.

$1 + \bar{\mu} - p_j$, then at the point of allocation $t_{\bar{\imath}_2}$, i.e. the earliest point of allocation in schedule $\eta_2^{\mathscr{A}}$ at which all predecessors of tasks in Z_0 have been completed, all tasks in Z' will have priorities that are "close". This result will then be used in Lemma 6.56 to demonstrate that, if both tasks $j \in Z_1$ have μ_j "close" to $1 + \bar{\mu} - p_j$, all tasks in Z' will converge in priority and eventually be united in some set N_2^i. Since analysing the membership of tasks in the sets N_1^i, N_2^i and N_3^i is a convenient way to characterise their processing, this result is later used to establish relations on the μ's of the tasks in Z_0 which determine how they are processed in the schedule $\eta_2^{\mathscr{A}}$.

Lemma 6.55 *When algorithm $\mathscr{A}$ is applied to φ^2, if*

$$|1 + \bar{\mu} - p_j - \mu_j| \leq \frac{1}{25n^3} \tag{6.39}$$

for both tasks $j \in Z_1$, for any $g \in Z'$ and $q \in Z'$,

$$\left|p_g\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_g - p_q\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) - \mu_q\right| < \frac{1}{12n^3}.$$

Proof. Note that (6.39) holds in the case of $j \in Z_0 \setminus Z_1$ where $\mu_j = \bar{\mu}$. If only one task $j \in Z_0 \setminus Z_1$ has $\mu_j = \bar{\mu}$, Lemma 6.53 implies that $\mu_v > \mu_w = \bar{\mu}$. Hence, if $v \in Z'$, (6.38), (6.39), Corollary 6.47 and $p_v = 1$ together imply that, for any $j \in Z' \setminus \{v\}$, and for a small enough value of $\varepsilon > 0$,

$$p_j\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_j - \frac{1}{12n^3} \leq p_j + \mu_j - \frac{1}{12n^3} < p_j + \mu_j - \frac{1}{25n^3} - n\varepsilon$$

$$\leq 1 + \bar{\mu} - n\varepsilon \leq p_v - n\varepsilon + \mu_v < p_v\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_v \leq 1 + \bar{\mu} + \frac{1}{25n^3}$$

$$\leq p_j + \mu_j + \frac{2}{25n^3} < p_j\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + n\varepsilon + \mu_j + \frac{2}{25n^3} < p_j\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_j + \frac{1}{12n^3}.$$

On the other hand, regardless of whether $v \in Z'$, consider any $g \in Z' \setminus \{v\}$ and $q \in Z' \setminus \{v\}$. Similar to the previous case,

$$p_q\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_q - \frac{1}{12n^3} \leq p_q + \mu_q - \frac{1}{12n^3} < p_q + \mu_q - \frac{1}{25n^3} - \frac{1}{24n^3}$$

$$\leq 1 + \bar{\mu} - \frac{1}{24n^3} < 1 + \bar{\mu} - \frac{1}{25n^3} - n\varepsilon \leq p_g - n\varepsilon + \mu_g$$

$$< p_g\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_g \leq p_g + \mu_g \leq 1 + \bar{\mu} + \frac{1}{25n^3} \leq p_q + \mu_q + \frac{2}{25n^3}$$

$$< p_q\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + n\varepsilon + \mu_q + \frac{2}{25n^3} < p_q\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_q + \frac{1}{12n^3}.$$

Consequently the lemma is proven. ■

Lemma 6.56 *When algorithm $\mathscr{A}$ is applied to φ^2, if (6.39) holds for both tasks $j \in Z_1$, there exists $i \in R_2$ such that $t_i < \frac{1}{4}$ and $Z' \subseteq N_2^i$.*

Proof. Select $g \in Z'$ and $q \in Z'$ such that

$$p_g\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_g = \max_{j \in Z'} \left[p_j\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_j\right]$$

and

$$p_q\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_q = \min_{j \in Z'} \left[p_j\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_j\right].$$

Furthermore, select as i' the smallest i such that

$$Z' \subseteq N_a^i \text{ for some } a \in \{1, 2\} \quad \text{or} \quad \min_{j \in Z_0} \left[C_j\left(\eta_2^{\mathscr{A}}\right)\right] \leq t_i. \tag{6.40}$$

For $i = \hat{\imath}_2 + 1$, (6.40) holds, so this selection is always possible. For any $i \in R_2$ such that $i < i'$ it will now be proven that $t_{i+1} - t_i < \frac{1}{4n^2}$ by considering the two cases below.

- If $Z' \subseteq N_3^i$ then there exists a task $j \in N_1^i \cup N_2^i$, and since $A_i \cap Z = Z_0$ it follows that $j \notin Z$. Hence, Lemma 4.10 implies that, for an adequately small value of $\varepsilon > 0$,
$$t_{i+1} - t_i \leq np_j = n\varepsilon < \frac{1}{4n^2}.$$

- On the other hand, if $Z' \setminus N_3^i \neq \emptyset$ it follows from the definition of i' that
$$Z' \cap N_e^i \neq \emptyset \qquad \text{and} \qquad Z' \cap N_f^i \neq \emptyset \qquad \text{for } e \neq f.$$
In this case Corollary 4.5 implies $g \in N_a^i$ and $q \in N_b^i$ for $a \neq b$. Hence, by Lemma 4.3, Lemma 4.9 and Lemma 6.55,
$$t_{i+1} - t_i < mn\xi_i \leq mn\left(p_g\left(t_i, \eta_2^{\mathscr{A}}\right) + \mu_g - p_q\left(t_i, \eta_2^{\mathscr{A}}\right) - \mu_q\right)$$
$$\leq mn\left(p_g\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_g - p_q\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) - \mu_q\right) < \frac{mn}{12n^3} = \frac{1}{4n^2}.$$

Thus, $t_{i+1} - t_i < \frac{1}{4n^2}$ in either case.

As a consequence of this, Corollary 4.8 implies

$$t_{i'} - t_{\bar{\imath}_2} = \sum_{i=\bar{\imath}_2}^{i'-1} (t_{i+1} - t_i) < \frac{i' - \bar{\imath}_2}{4n^2} \leq \frac{s}{4n^2} \leq \frac{1}{4}.$$

For a small enough value of $\varepsilon > 0$ and any $j \in Z_0$, Lemma 6.45 implies

$$C_j\left(\eta_2^{\mathscr{A}}\right) \geq 1 > \frac{1}{4} + n\varepsilon > \frac{1}{4} + t_{\bar{i}_2} > t_{i'}.$$

Therefore $Z_0 \subseteq A_{i'}$ and thus $Z' \subseteq N_a^i$ for some $a \in \{1, 2\}$. All that remains is to prove that $a = 2$.

If $Z' = Z_0$, $|Z'| = 4 > 3 = m$ demonstrates that Z' cannot be a subset of any N_1, and consequently $Z' \subseteq N_2^{i'}$. On the other hand, if $Z' \neq Z_0$, (6.38) does not hold. Thus, for all $j \in Z'$,

$$p_v\left(t_{\bar{i}_2}, \eta_2^{\mathscr{A}}\right) + \mu_v > 1 + \bar{\mu} + \frac{1}{25n^3} \geq p_j + \mu_j \geq p_j\left(t_{\bar{i}_2}, \eta_2^{\mathscr{A}}\right) + \mu_j.$$

By Corollary 4.5, this implies

$$p_v\left(t_{i'}, \eta_2^{\mathscr{A}}\right) + \mu_v \geq p_j\left(t_{i'}, \eta_2^{\mathscr{A}}\right) + \mu_j.$$

Since $|Z_0| = 4 > 3 = m$, it follows that $Z' \subseteq N_2^{i'}$. ■

Lemma 6.57 and Lemma 6.58 below are dedicated to establishing that certain tasks are available at time $t_{\hat{i}_2}$ in schedule $\eta_2^{\mathscr{A}}$. These results are then combined with Lemma 6.56 and used to study the processing of the tasks in Z_0 in the schedule $\eta_2^{\mathscr{A}}$ by considering their membership in the sets $N_1^{\hat{i}_2}$, $N_2^{\hat{i}_2}$ and $N_3^{\hat{i}_2}$, and thus establishing relations on their μ's.

Lemma 6.57 *It holds that* $Z_1 \subseteq A_{\hat{i}_2}$.

Proof. By Corollary 6.46 all predecessors of the tasks in Z_1 are completed before $t_{\hat{i}_2}$. Hence, if $j \notin A_{\hat{i}_2}$ for some task $j \in Z_1$, it follows that $C_j\left(\eta_2^{\mathscr{A}}\right) \leq t_{\hat{i}_2}$. In this case, there exists a task $g \in Z_0 \setminus Z_1$ such that, by Lemma 6.54 and for adequately small $\varepsilon > 0$,

$$L_{max}\left(\eta_2^{\mathscr{A}}\right) \geq C_g\left(\eta_2^{\mathscr{A}}\right) - d_g = \hat{t}_2 - 1 > t_{\hat{i}_2} - 1$$

$$\geq C_j\left(\eta_2^{\mathscr{A}}\right) - 1 \geq p_j - 1 \geq 4\bar{h} \geq 8 > n\varepsilon + \frac{5}{9} = L_{max}\left(\sigma^2\right),$$

contradicting the assumption that $\prec \in \mathcal{D}^{\mathscr{A}}$, i.e. that $\eta_2^{\mathscr{A}}$ is an optimal schedule. ■

Lemma 6.58 *When algorithm* $\mathscr{A}$ *is applied to* φ^2*, if (6.39) holds for both tasks* $j \in Z_1$, $w \in A_{\hat{i}_2}$.

Proof. Recall that $w \in Z_0 \setminus Z_1$ is a task for which $\mu_w = \bar{\mu}$. Further recall that there exists a task $j \in Z_0 \setminus Z_1$ such that $C_j\left(\eta_2^{\mathscr{A}}\right) > t_{\hat{i}_2}$, and since Corollary 6.46

demonstrates that the predecessors of j are all completed by time $t_{\hat{\imath}_2}$ it follows that $j \in A_{\hat{\imath}_2}$.

Select an index ι such that $C_v\left(\eta_2^{\mathscr{A}}\right) \in (t_\iota, t_{\iota+1}]$, i.e.

$$\iota = \max\{i : v \in A_i\}.$$

If $p_w\left(t_\iota, \eta_2^{\mathscr{A}}\right) > 0$ it follows that either

- $C_w\left(\eta_2^{\mathscr{A}}\right) \leq t_{\iota+1}$, in which case $w \in A_\iota$ and hence $\hat{\imath}_2 = \iota$ and $\{v, w\} \subseteq A_{\hat{\imath}_2}$; or
- $C_w\left(\eta_2^{\mathscr{A}}\right) > t_{\iota+1}$, in which case $w \in A_{\iota+1}$ implying $\hat{\imath}_2 > \iota$ and hence $w \in A_{\hat{\imath}_2}$.

Thus in order to prove the lemma it is sufficient to establish that $p_w\left(t_\iota, \eta_2^{\mathscr{A}}\right) > 0$. This is accomplished by considering the following two cases.

- Suppose that (6.38) holds. In this case $v \in Z'$ and by Lemma 6.56 there exists $i \in R_2$ such that $t_i < \frac{1}{4}$ and $Z_0 = Z' \subseteq N_2^i$. By the definition of N_2,

  $$p_w\left(t_i, \eta_2^{\mathscr{A}}\right) + \bar{\mu} = p_w\left(t_i, \eta_2^{\mathscr{A}}\right) + \mu_w = p_v\left(t_i, \eta_2^{\mathscr{A}}\right) + \mu_v \geq p_v\left(t_i, \eta_2^{\mathscr{A}}\right) + \bar{\mu} \quad (6.41)$$

  and hence $p_w\left(t_i, \eta_2^{\mathscr{A}}\right) \geq p_v\left(t_i, \eta_2^{\mathscr{A}}\right)$. By Lemma 4.9 both tasks will receive the same amount of processing in each interval until one is completed and hence

  $$p_w\left(t_\iota, \eta_2^{\mathscr{A}}\right) = p_w\left(t_i, \eta_2^{\mathscr{A}}\right) - \left(p_v\left(t_i, \eta_2^{\mathscr{A}}\right) - p_v\left(t_\iota, \eta_2^{\mathscr{A}}\right)\right) \geq p_v\left(t_\iota, \eta_2^{\mathscr{A}}\right) > 0.$$

- On the other hand, suppose that (6.38) does not hold. In this case

  $$p_v\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_v > 1 + \bar{\mu} + \frac{1}{25n^3} > p_w + \mu_w \geq p_w\left(t_{\bar{\imath}_2}, \eta_2^{\mathscr{A}}\right) + \mu_w. \quad (6.42)$$

  Define the set

  $$R_2' = \{i : i \in R_2 \setminus \{0\} \wedge \{v, w\} \subseteq A_{i-1}\}.$$

  Given the above definition, Corollary 4.5 implies that for any $i \in R_2'$

  $$p_v\left(t_i, \eta_2^{\mathscr{A}}\right) + \mu_v \geq p_w\left(t_i, \eta_2^{\mathscr{A}}\right) + \mu_w. \quad (6.43)$$

  If (6.43) is an equality for some $i \in R_2'$ then (6.41) holds, and the same reasoning used in the previous case demonstrates that $p_w\left(t_{\hat{\imath}_2}, \eta_2^{\mathscr{A}}\right) \geq p_v\left(t_{\hat{\imath}_2}, \eta_2^{\mathscr{A}}\right)$.

  Suppose, on the other hand, that (6.43) is not an equality for any $i \in R_2'$. In this case, for any $i \in R_2'$ either $(N_1^i \cup N_2^i) \setminus Z_0 \neq \emptyset$ or $v \in N_1^i$. Hence, by

Lemma 6.45 and Lemma 6.48,

$$C_v\left(\eta_2^{\mathscr{A}}\right) \leq t_{\bar{\imath}_2} + p_v + \sum_{i \in S_2}(t_{i+1} - t_i) < 1 + (n + n^3)\varepsilon. \tag{6.44}$$

By Lemma 6.56, $Z' \subseteq N_2^{i'}$ for some i' such that $t_{i'} < \frac{1}{4}$. Consequently, for any $i \in R_2'$ such that $i \geq i'$ if $(N_1^i \cup N_2^i) \setminus Z_0 \neq \emptyset$ it follows from Lemma 4.9 that $N_1^i = \{v\}$ and $N_2^i = Z' = Z_0 \setminus \{v\}$. A lower bound on the remaining processing time of w at time t_ι can be derived by noting that, during any interval $[t_i, t_{i+1}]$ where $Z' = N_2^i$, task w is processed for

$$\frac{m - |N_1^i|}{|N_2^i|}(t_{i+1} - t_i) = \frac{m - |\{v\}|}{|Z'|}(t_{i+1} - t_i) = \frac{2}{3}(t_{i+1} - t_i)$$

units of time. With this insight, Lemma 6.48, (6.44) and the fact that $t_{i'} < \frac{1}{4}$ then imply that, for an adequately small value of $\varepsilon > 0$,

$$\begin{aligned} p_w\left(t_\iota, \eta_2^{\mathscr{A}}\right) &\geq p_w - t_{i'} - \sum_{i \in S_2}(t_{i+1} - t_i) - \frac{2}{3}\left(t_\iota - t_{i'} - \sum_{i \in S_2}(t_{i+1} - t_i)\right) \\ &= 1 - \frac{1}{3}t_{i'} - \frac{1}{3}\sum_{i \in S_2}(t_{i+1} - t_i) - \frac{2}{3}t_\iota \\ &> 1 - \frac{1}{12} - \frac{1}{3}n^3\varepsilon - \frac{2}{3}(1 + (n + n^3)\varepsilon) = \frac{1}{4} - \frac{2}{3}n\varepsilon - n^3\varepsilon > 0. \end{aligned}$$

Hence, $p_w\left(t_\iota, \eta_2^{\mathscr{A}}\right) > 0$ in both cases, and the lemma holds. ■

This information gleaned in the previous two lemmas regarding the elements of $A_{\hat{\imath}_2}$ will now be employed. Combined with Lemma 6.56 these results are used in the lemma below to derive a relationship between tasks' priorities at time $t_{\hat{\imath}_2+1}$. Subsequent lemmas will use this result to establish constraints on the μ's algorithm $\mathscr{A}$ must compute in order for $\eta_2^{\mathscr{A}}$ to be optimal.

Lemma 6.59 *When algorithm $\mathscr{A}$ is applied to φ^2, if (6.39) holds for both tasks $j \in Z_1$, $p_j\left(t_{\hat{\imath}_2+1}, \eta_2^{\mathscr{A}}\right) + \mu_j = \bar{\mu}$ for both tasks $j \in Z_1$.*

Proof. By Lemma 6.56, $Z' \subseteq N_2^{i'}$ for some i' such that $t_{i'} < \frac{1}{4}$. This i' will be used throughout the rest of this proof.

By Lemma 4.9, for all $i \geq i'$ such that $Z' \subseteq A_i$, and all $j \in Z'$ and $g \in Z'$,

$$p_j\left(t_i, \eta_2^{\mathscr{A}}\right) + \mu_j = p_g\left(t_i, \eta_2^{\mathscr{A}}\right) + \mu_g.$$

By Lemma 6.58, $w \in A_{\hat{\imath}_2}$, so Lemma 6.57 implies

$$p_j\left(t_{\hat{\imath}_2+1}, \eta_2^{\mathscr{A}}\right) + \mu_j = p_w\left(t_{\hat{\imath}_2+1}, \eta_2^{\mathscr{A}}\right) + \mu_w = \bar{\mu}$$

for both $j \in Z_1$. ■

Define

$$\psi = \max_{j \in Z_1}\left(p_j - p_j\left(t_{\hat{\imath}_2+1}, \eta_2^{\mathscr{A}}\right)\right).$$

In words, ψ is the largest amount of any task $j \in Z_1$ that is processed before $t_{\hat{\imath}_2+1}$ in schedule $\eta_2^{\mathscr{A}}$. This will be employed in the Lemma 6.60 and Lemma 6.62 which address how the tasks are processed, and Lemma 6.63 which establishes an important property that the μ's must obey in order for $\eta_2^{\mathscr{A}}$ to be optimal.

Lemma 6.60 *When algorithm $\mathscr{A}$ is applied to φ^2, if (6.39) holds for both tasks $j \in Z_1$, for all integers ι for which $1 \le \iota < \bar{h}$ and all $g \in Z_\iota$,*

$$p_g\left(t_{\hat{\imath}_2+1} + 4\bar{h} + \iota - \psi, \eta_2^{\mathscr{A}}\right) \ge \left|\left\{i : \iota < i \le \bar{h} \wedge g \in Z_i\right\}\right|$$

and

$$C_g\left(\eta_2^{\mathscr{A}}\right) \ge t_{\hat{\imath}_2+1} + 4\bar{h} + \iota - \psi.$$

Proof. This lemma will be proven by induction on ι. For both $g \in Z_1$,

$$\begin{aligned} p_g\left(t_{\hat{\imath}_2+1}, \eta_2^{\mathscr{A}}\right) &= 4\bar{h} - 1 + \left|\left\{i : 0 \le i \le \bar{h} \wedge g \in Z_i\right\}\right| - \left(p_g - p_g\left(t_{\hat{\imath}_2+1}, \eta_2^{\mathscr{A}}\right)\right) \\ &\ge 4\bar{h} - 1 + \left|\left\{i : 0 \le i \le \bar{h} \wedge g \in Z_i\right\}\right| - \psi \\ &= 4\bar{h} + 1 - \psi + \left|\left\{i : 1 < i \le \bar{h} \wedge g \in Z_i\right\}\right|. \end{aligned}$$

Thus

$$p_g\left(t_{\hat{\imath}_2+1} + 4\bar{h} + 1 - \psi, \eta_2^{\mathscr{A}}\right) \ge \left|\left\{i : 1 < i \le \bar{h} \wedge g \in Z_i\right\}\right|$$

and

$$C_g\left(\eta_2^{\mathscr{A}}\right) \ge t_{\hat{\imath}_2+1} + 4\bar{h} + 1 - \psi.$$

Consequently the lemma has been proven in the case of $\iota = 1$.

Now suppose that the lemma has been proven for all $\iota < k$ for some k for which $2 \le k < \bar{h}$ and consider the case where $\iota = k$. Select any $g \in Z_k$ and observe that, by Lemma 6.41, either $g \in Z_{k-1}$ or there exists a task $q \in Z_{k-1} \cap J_g$. These two cases will be considered below.

- If $g \in Z_{k-1}$ the inductive assumption implies

$$p_g\left(t_{\hat{\imath}_2+1} + 4\bar{h} + k - \psi, \eta_2^{\mathscr{A}}\right) \geq p_g\left(t_{\hat{\imath}_2+1} + 4\bar{h} + k - 1 - \psi, \eta_2^{\mathscr{A}}\right) - 1$$

$$\geq \left|\left\{i : k-1 < i \leq \bar{h} \wedge g \in Z_i\right\}\right| - 1 = \left|\left\{i : k < i \leq \bar{h} \wedge g \in Z_i\right\}\right|$$

and

$$C_g\left(\eta_2^{\mathscr{A}}\right) \geq t_{\hat{\imath}_2+1} + 4\bar{h} + k - 1 - \psi + p_g\left(t_{\hat{\imath}_2+1} + 4\bar{h} + k - 1 - \psi, \eta_2^{\mathscr{A}}\right)$$

$$\geq t_{\hat{\imath}_2+1} + 4\bar{h} + k - 1 - \psi + \left|\left\{i : k-1 < i \leq \bar{h} \wedge g \in Z_i\right\}\right|$$

$$= t_{\hat{\imath}_2+1} + 4\bar{h} + k - \psi + \left|\left\{i : k < i \leq \bar{h} \wedge g \in Z_i\right\}\right| \geq t_{\hat{\imath}_2+1} + 4\bar{h} + k - \psi.$$

  Hence, the lemma holds for $\iota = k$ in this case.

- On the other hand, if there exists a task $q \in Z_{k-1} \cap J_g$, Lemma 6.37 implies $q \notin Z_k$. Consequently, by the inductive assumption,

$$C_q\left(\eta_2^{\mathscr{A}}\right) \geq t_{\hat{\imath}_2+1} + 4\bar{h} + k - 1 - \psi.$$

  Hence,

$$p_g\left(t_{\hat{\imath}_2+1} + 4\bar{h} + k - \psi, \eta_2^{\mathscr{A}}\right) \geq p_g - 1 = \left|\left\{i : k < i \leq \bar{h} \wedge g \in Z_i\right\}\right|$$

  and

$$C_g\left(\eta_2^{\mathscr{A}}\right) \geq C_q\left(\eta_2^{\mathscr{A}}\right) + p_g \geq C_q\left(\eta_2^{\mathscr{A}}\right) + 1 \geq t_{\hat{\imath}_2+1} + 4\bar{h} + k - \psi.$$

  Hence the lemma holds for $\iota = k$ in this case as well.

Thus, by induction on ι, the lemma holds. ■

Lemma 6.61 *When algorithm $\mathscr{A}$ is applied to φ^2, if (6.39) holds for both tasks $j \in Z_1$,*

$$t_{\hat{\imath}_2+1} \geq \frac{4}{3} - \frac{2}{75n^3}.$$

Proof. Let the two tasks in Z_1 be g and q. By (6.39), Lemma 6.59 and the definition of $\hat{\imath}_2$,

$$t_{\hat{\imath}_2+1} \geq \frac{1}{m} \sum_{j \in Z_0} \left(p_j - p_j\left(t_{\hat{\imath}_2+1}, \eta_2^{\mathscr{A}}\right)\right)$$

$$= \frac{1}{m}\left(\sum_{j\in Z_1}\left(p_j - p_j\left(t_{i_2+1}, \eta_2^{\mathscr{A}}\right)\right) + \sum_{j\in Z_0\setminus Z_1} p_j\right)$$

$$= \frac{1}{m}\left(p_g - p_g\left(t_{i_2+1}, \eta_2^{\mathscr{A}}\right) + p_q - p_q\left(t_{i_2+1}, \eta_2^{\mathscr{A}}\right) + 2\right)$$

$$= \frac{1}{m}(p_g - (\bar{\mu} - \mu_g) + p_q - (\bar{\mu} - \mu_q) + 2)$$

$$\geq \frac{1}{m}\left(p_g - \left(p_g - 1 + \frac{1}{25n^3}\right) + p_q - \left(p_q - 1 + \frac{1}{25n^3}\right) + 2\right)$$

$$= \frac{1}{3}\left(4 - \frac{2}{25n^3}\right) = \frac{4}{3} - \frac{2}{75n^3},$$

and the lemma holds. ■

Lemma 6.62 *When algorithm $\mathscr{A}$ is applied to φ^2, if (6.39) holds for both tasks $j \in Z_1$,*

$$\psi \leq 1 + \frac{1}{25n^3}.$$

Proof. As a consequence of (6.39) and Lemma 6.59, there exists a task $g \in Z_1$ for which

$$\psi = p_g - p_g\left(t_{i_2+1}, \eta_2^{\mathscr{A}}\right) = p_g - (\bar{\mu} - \mu_g)$$

$$\leq p_g - \left(p_g - 1 - \frac{1}{25n^3}\right) = 1 + \frac{1}{25n^3},$$

and the lemma is proven. ■

Lemma 6.63 *When algorithm $\mathscr{A}$ is applied to φ^2, (6.39) holds for at most one of the two tasks $j \in Z_1$.*

Proof. Suppose this is not the case, i.e. suppose that

$$|1 + \bar{\mu} - p_j - \mu_j| \leq \frac{1}{25n^3}$$

for both $j \in Z_1$ and hence Lemma 6.60 applies. Select $g \in Z_1$ such that $p_g - p_g\left(t_{i_2+1}, \eta_2^{\mathscr{A}}\right) = \psi$ and select $a \in \{0, 1\}$ such that $g \in Y_1^a$ in the construction of $Z_1, Z_2, \ldots, Z_{\bar{h}}$.

Since $|Z_{\bar{h}}| = 4$, Lemma 6.60 implies

$$\max_{j\in Z_{\bar{h}}} C_j\left(\eta_2^{\mathscr{A}}\right) \geq t_{i_2+1} + 5\bar{h} - 1 - \psi + \frac{1}{m}\sum_{j\in Z} p_j\left(t_{i_2+1} + 5\bar{h} - 1 - \psi, \eta_2^{\mathscr{A}}\right)$$

$$\geq t_{\hat{\imath}_2+1} + 5\bar{h} - 1 - \psi + \frac{4}{3} = t_{\hat{\imath}_2+1} + 5\bar{h} - \psi + \frac{1}{3}$$

and hence

$$L_{max}\left(\eta_2^{\mathscr{A}}\right) \geq t_{\hat{\imath}_2+1} + 5\bar{h} - \psi + \frac{1}{3} - 5\bar{h} \geq t_{\hat{\imath}_2+1} - \psi + \frac{1}{3}.$$

By Lemma 6.61 and Lemma 6.62, this gives, for adequately large $\varepsilon > 0$,

$$L_{max}\left(\eta_2^{\mathscr{A}}\right) \geq t_{\hat{\imath}_2+1} - \psi + \frac{1}{3} \geq \frac{4}{3} - \frac{2}{75n^3} - \left(1 + \frac{1}{25n^3}\right) + \frac{1}{3} = \frac{2}{3} - \frac{1}{15n^3}$$

$$> \frac{2}{3} - \frac{1}{15} = \frac{9}{15} = \frac{2}{45} + \frac{5}{9} > n\varepsilon + \frac{5}{9}.$$

However, by Lemma 6.54 there exists a schedule σ^2 for φ^2 for which

$$L_{max}\left(\sigma^2\right) = n\varepsilon + \frac{5}{9} < L_{max}\left(\eta_2^{\mathscr{A}}\right),$$

contradicting the assumption that $\eta_2^{\mathscr{A}}$ is an optimal schedule. ■

The above lemma implies the existence of $\bar{\jmath} \in Z_1$ such that

$$|1 + \bar{\mu} - p_{\bar{\jmath}} - \mu_{\bar{\jmath}}| > \frac{1}{25n^3} \tag{6.45}$$

when algorithm $\mathscr{A}$ is applied to φ^2. The establishment of this result is the reason for the consideration of the problem instance φ^2. By the definition of the family $\mathfrak{F}$ it follows that algorithm $\mathscr{A} \in \mathfrak{F}$ will produce the same value of $\mu_{\bar{\jmath}}$ in any problem instance for which $d_{\bar{\jmath}}$ and $\varphi_{\bar{\jmath}}$ have the same values as in problem instance φ^2. Such a problem instance, φ^3, will now be specified, and it will be proven that (6.45) results in a non-optimal schedule $\eta_3^{\mathscr{A}}$, contradicting the assumption that $\mathscr{A}$ produces an optimal schedule for all problem instances with precedence constraints isomorphic to $\prec$. Since $\mathscr{A}$ is an arbitrary algorithm in $\mathfrak{F}$ and $\prec$ is an arbitrary partial order containing a singular quartet this serves to demonstrate that $\mathcal{D}^{\mathscr{A}} \subseteq \mathcal{D}^{\mathscr{ZR}}$ for every algorithm $\mathscr{A} \in \mathfrak{F}$.

Let problem instance φ^3 have

$$p_{\bar{\jmath}} = d_{\bar{\jmath}} = 4\bar{h} - 1 + |\{i : 0 \leq i \leq \bar{h} \wedge \bar{\jmath} \in Z_i\}|$$

and $p_j = d_j = 1$ for $j \in Z_1 \setminus \{\bar{\jmath}\}$. Note that the only differences between φ^2 and φ^3 are the due dates and processing times of one of the two tasks in Z_1. Hence,

for all tasks $j \in Z \setminus \{\bar{j}\}$, d_j and φ_j remain unchanged between these two problem instances. By the definition of the family $\mathfrak{F}$, the value of μ_j assigned by algorithm $\mathscr{A} \in \mathfrak{F}$ will also not change between these two problem instances.

Until stated otherwise, points of allocation t_i, sets of available tasks A_i and sets of tasks receiving equal processing time N_a^i will be considered to be those values associated with schedule $\eta_3^{\mathscr{A}}$.

Define

$$t_3' = t_{\bar{i}_3} + \frac{1}{3}\left(p_{\bar{j}}\left(t_{\bar{i}_3}, \eta_3^{\mathscr{A}}\right) - (p_{\bar{j}} - 1) + \sum_{g \in Z_0 \setminus \{\bar{j}\}} p_g\left(t_{\bar{i}_3}, \eta_3^{\mathscr{A}}\right)\right),$$

i.e. t_3' is a lower bound on the time at which all tasks $j \in Z_0$ have been processed for at least one unit of time. This value will be used in the lemma below to establish the existence of a schedule with a specific maximum lateness. This will subsequently be compared to $L_{max}\left(\eta_3^{\mathscr{A}}\right)$ to establish the non-optimality of $\eta_3^{\mathscr{A}}$.

In several subsequent proofs the processing of the tasks in $Z \setminus Z_0$ is considered. Recall that, when the sets $Z_1, Z_2, \ldots, Z_{\bar{h}}$ were specified by the Z-Algorithm, each Z_i for $1 \leq i \leq \bar{h}$ was specified to be the union of two disjoint, non-empty sets Y_i^0 and Y_i^1. From this point until the end of the section let $a \in \{0, 1\}$ be defined such that $\bar{j} \in Y_1^a$ in the construction of $Z_1, Z_2, \ldots, Z_{\bar{h}}$. When examining the problem instances φ^1 and φ^2 there was no need to consider the sets Y_i^0 and Y_i^1 separately. Now, however, task $\bar{j}$ has a very large processing time while the other task in Z_1 has a processing time of one unit, which has important consequences for the processing of subsequent tasks.

In particular, Lemma 6.41 implies that for each task $g \in Y_i^a$ there exists a chain C of tasks such that $\{g, \bar{j}\} \subseteq C$ and $C \cap Y_{i'}^a \neq \emptyset$ for each integer $i' \in [1, i]$. In words, this means that there is a chain of tasks from $\bar{j}$ to g which has an element in every $Y_{i'}^a$ between the two tasks. This limits when the tasks in Y_i^a can be processed in any schedule for problem instance φ^3. On the other hand, Lemma 6.42 demonstrates that no such chain exists for any task in Y_i^{1-a}.* Consequently the processing of tasks in Y_i^{1-a} is less restricted than the processing of tasks in Y_i^a. This fact will be used throughout the remainder of this section.

Lemma 6.64 *There exists a schedule σ^3 for problem instance φ^3 for which*

$$L_{max}\left(\sigma^3\right) = t_3' - 1.$$

*It is important to recall that a task $q \in Y_i^{1-a}$ may be preceded by $\bar{j}$. In fact, it may be the case that $J_q \cap K_{\bar{j}} \cap Z_{i'} \neq \emptyset$ for each integer $i' \in [1, i]$. The key distinction is that there is no single chain with an element in each $Z_{i'}$.

Proof. The proof of this lemma is similar to that of Lemma 6.51. First, let σ^3 be identical to $\eta_3^{\mathscr{A}}$ on the interval $[0, t_{\bar{\imath}_3}]$. If $\varepsilon > 0$ is adequately small, by Corollary 6.47 and the fact that $t'_3 \geq \frac{4}{3}$,

$$\max_{g \in Z_0 \setminus \{\bar{\jmath}\}} \left[p_g \left(t_{\bar{\imath}_3}, \eta_3^{\mathscr{A}} \right) \right] \leq 1 < \frac{4}{3}(1 - n\varepsilon) < t'_3 - t_{\bar{\imath}_3}.$$

Hence, use McNaughton's Algorithm to schedule all $j \in Z_0$ for

$$p_j \left(t_{\bar{\imath}_3}, \eta_3^{\mathscr{A}} \right) - (p_j - 1)$$

units of processing time during $[t_{\bar{\imath}_3}, t'_3]$. This results in $C_j\left(\sigma^3\right) \leq t'_3$ for all $j \in Z_0 \setminus \{\bar{\jmath}\}$, and consequently

$$C_j\left(\sigma^3\right) - d_j \leq t'_3 - 1.$$

Consider any i for which $1 \leq i < \bar{h} - 1$. By Lemma 6.43, all tasks in $Y_i^a \cup Y_{i+1}^{1-a}$ are independent of one another. By Lemma 6.44, $|Y_i^a \cup Y_{i+1}^{1-a}| \leq 3$.

First considering $i = 1$, schedule during the interval $\left[t'_3, t'_3 + 4\bar{h}\right]$, $4\bar{h}$ units of $\bar{\jmath}$, one unit of each task in Y_2^{1-a} and, as in Lemma 6.54, all remaining tasks $j \in N \setminus Z$ such that $K_j \cap Z \neq \emptyset$. If $\bar{\jmath} \notin Z_2$, this results in

$$C_{\bar{\jmath}}\left(\sigma^3\right) - d_{\bar{\jmath}} = t'_3 + 4\bar{h} - (4\bar{h} + 1) = t'_3 - 1.$$

For $1 < i < \bar{h} - 1$, schedule one unit of each task in $Y_i^a \cup Y_{i+1}^{1-a}$ during the interval

$$\left[t'_3 + 4\bar{h} + i - 2, t'_3 + 4\bar{h} + i - 1\right].$$

For all $j \in Y_i^a \setminus Y_{i+1}^a$, this results in

$$C_j\left(\sigma^3\right) - d_j = t'_3 + 4\bar{h} + i - 1 - \left(4\bar{h} + i\right) = t'_3 - 1.$$

Similarly, for all $j \in Y_{i+1}^{1-a} \setminus Y_{i+2}^{1-a}$,

$$C_j\left(\sigma^3\right) - d_j = t'_3 + 4\bar{h} + i - 1 - \left(4\bar{h} + i + 1\right) = t'_3 - 2.$$

Now two cases will be considered.

- First consider the case where $|Y_{\bar{h}-1}^a \cup Y_{\bar{h}}^{1-a}| = 4$. By the definition of a canonical transitive set I there is only one subset of I containing four independent tasks. Hence, $|Y_{\bar{h}-1}^a \cup Y_{\bar{h}}^{1-a}| = 4$ implies $Y_{\bar{h}-1}^a \cup Y_{\bar{h}}^{1-a} = Z_{\bar{h}}$. By Lemma 6.37 all tasks

in $Z_{\bar{h}}$ are independent. Since, for all $j \in Z$,

$$p_j\left(t_3' + 5\bar{h} - 3, \sigma^3\right) = \left|\left\{i : \bar{h} - 1 \leq i \leq \bar{h} \wedge j \in Z_i\right\}\right|,$$

it follows that

$$p_j\left(t_3' + 5\bar{h} - 3, \sigma^3\right) \leq 2$$

and by Lemma 6.36 and Lemma 6.32,

$$\sum_{j \in Z} p_j\left(t_3' + 5\bar{h} - 3, \sigma^3\right) = \left|Y_{\bar{h}-1}^a\right| + |Z_{\bar{h}}| \leq |Z_{\bar{h}-1}| - 1 + |Z_{\bar{h}}| \leq 6.$$

Therefore, use McNaughton's Algorithm to schedule the remaining processing of all tasks in $Z_{\bar{h}}$ during the interval

$$\left[t_3' + 5\bar{h} - 3, t_3' + 5\bar{h} - 1\right].$$

For all $j \in Z_{\bar{h}}$, this gives

$$C_j\left(\sigma^3\right) - d_j \leq t_3' + 5\bar{h} - 1 - 5\bar{h} = t_3' - 1.$$

- Alternatively, if $|Y_{\bar{h}-1}^a \cup Y_{\bar{h}}^{1-a}| \leq 3$, schedule one unit of each task $j \in Y_{\bar{h}-1}^a \cup Y_{\bar{h}}^{1-a}$ during the interval

$$\left[t_3' + 5\bar{h} - 3, t_3' + 5\bar{h} - 2\right].$$

By Lemma 6.36 and Lemma 6.32,

$$\sum_{j \in Z} p_j\left(t' + 5\bar{h} - 2, \sigma^3\right) = |Y_{\bar{h}}^a| \leq |Z_{\bar{h}}| - 1 = 3.$$

Therefore, schedule all tasks in $Y_{\bar{h}}^a$ during the interval

$$\left[t_3' + 5\bar{h} - 2, t_3' + 5\bar{h} - 1\right].$$

This results in

$$C_j\left(\sigma^3\right) - d_j \leq t_3' - 1$$

for all $j \in Y_{\bar{h}-1}^a \cup Z_{\bar{h}}$.

Finally, schedule all remaining tasks in $N \setminus Z$ arbitrarily during the interval

$$\left[t_3' + 5\bar{h} - 1, n\varepsilon + t_3' + 5\bar{h} - 1\right].$$

Combining the results above, it follows that $L_{max}(\sigma^3) = t'_3 - 1$. ■

The above lemma, combined with the assumption of the optimality of $\eta_3^{\mathscr{A}}$, is used in the lemma below to establish results regarding the processing of tasks in $\eta_3^{\mathscr{A}}$. It will then be demonstrated that this in fact contradicts the assumption of the optimality of $\eta_3^{\mathscr{A}}$.

Lemma 6.65 *In schedule $\eta_3^{\mathscr{A}}$, all machines at all times during $\left[t_{\bar{\imath}_3}, \hat{t}_3\right]$ are processing tasks from Z_0, and*

$$p_j\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = p_j - 1$$

for all $j \in Z_0$.

Proof. If $C_j\left(\eta_3^{\mathscr{A}}\right) > t'_3$ for any $j \in Z_0 \setminus \{\bar{\jmath}\}$, it follows from Lemma 6.64 that

$$L_{max}\left(\eta_3^{\mathscr{A}}\right) > t'_3 - 1 = L_{max}\left(\sigma^3\right),$$

contradicting the assumption that $\eta_3^{\mathscr{A}}$ is optimal. Similarly, if $p_{\bar{\jmath}}\left(t'_3, \eta_3^{\mathscr{A}}\right) > p_{\bar{\jmath}} - 1$, Lemma 6.41 implies the existence of a task $q \in Y_{\bar{h}}^a$ for which

$$\begin{aligned} C_q\left(\eta_3^{\mathscr{A}}\right) - d_q &\geq t'_3 + p_{\bar{\jmath}}\left(t'_3, \eta_3^{\mathscr{A}}\right) + \left|\left\{i : 2 \leq i \leq \bar{h} \wedge \bar{\jmath} \notin Z_i\right\}\right| - 5\bar{h} \\ &> t'_3 + p_{\bar{\jmath}} - 1 + \left|\left\{i : 2 \leq i \leq \bar{h} \wedge \bar{\jmath} \notin Z_i\right\}\right| - 5\bar{h} \\ &= t'_3 + 4\bar{h} - 2 + \left|\left\{i : 0 \leq i \leq \bar{h} \wedge \bar{\jmath} \in Z_i\right\}\right| + \left|\left\{i : 0 \leq i \leq \bar{h} \wedge \bar{\jmath} \notin Z_i\right\}\right| - 5\bar{h} \\ &= t'_3 + 4\bar{h} - 2 + \bar{h} + 1 - 5\bar{h} = t'_3 - 1 = L_{max}\left(\sigma^3\right), \end{aligned}$$

also contradicting the assumption that $\eta_3^{\mathscr{A}}$ is optimal.

As a consequence, the definition of t'_3 implies that, in schedule $\eta_3^{\mathscr{A}}$, all machines at all times in $[t_{\bar{\imath}_3}, t'_3]$ are processing tasks from Z_0. Since $m = 3$, at least two tasks $j \in Z_0 \setminus \{\bar{\jmath}\}$ have $C_j\left(\eta_3^{\mathscr{A}}\right) = t'_3$, and at least one such task must be an element of $Z_0 \setminus Z_1$. By the definition of $\hat{t}_3$ as the largest completion time of the two tasks in $Z_0 \setminus Z_1$, $t'_3 = \hat{t}_3$ and the lemma follows. ■

Corollary 6.66 *It holds that*

$$\frac{4}{3} \leq \hat{t}_3 = \max_{j \in Z_0 \setminus \{\bar{\jmath}\}} C_j\left(\eta_3^{\mathscr{A}}\right) \leq t_{\bar{\imath}_3} + \frac{4}{3} < n\varepsilon + \frac{4}{3}.$$

Corollary 6.67 *At most one $j \in Z_0$ has $C_j\left(\eta_3^{\mathscr{A}}\right) < \hat{t}_3$.*

Recall that $\hat{\imath}_3$ is defined such that

$$\max_{j \in Z_0 \setminus Z_1} C_j\left(\eta_3^{\mathscr{A}}\right) = \hat{t}_3 \in (t_{\hat{\imath}_3}, t_{\hat{\imath}_3+1}].$$

Corollary 6.67 will be used later to divide the analysis of the schedule $\eta_3^{\mathscr{A}}$ into two cases: either one task from Z_0 is completed no later than time $t_{\hat{\imath}_3}$, or no task is. To facilitate the consideration of these two cases, select as i'_3 the largest i such that $Z_0 \subseteq A_i$, i.e. some task $j \in Z_0$ has $C_j\left(\eta_3^{\mathscr{A}}\right) \in (t_{i'_3}, t_{i'_3+1}]$. By Lemma 6.45 and Corollary 6.66,

$$t_{\bar{\imath}_3} \leq t_{i'_3} \leq t_{\hat{\imath}_3} < \hat{t}_3 < n\varepsilon + \frac{4}{3}. \tag{6.46}$$

The lemma below, demonstrating that no tasks are completed during $\left(t_{\bar{\imath}_3}, t_{i'_3}\right]$, will be used later in conjunction with Lemma 4.7, which establishes how tasks move between sets N_1, N_2 and N_3 if no tasks are completed.

Lemma 6.68 *No $j \in N$ has $C_j\left(\eta_3^{\mathscr{A}}\right) \in \left(t_{\bar{\imath}_3}, t_{i'_3}\right]$.*

Proof. For all $j \in N \setminus Z_0$, Lemma 6.65 demonstrates that j is not processed during $\left(t_{\bar{\imath}_3}, \hat{t}_3\right)$. From (6.46), $t_{i'_3} < \hat{t}_3$ and the lemma holds for tasks not in Z_0.

On the other hand, by the definition of i'_3, all $j \in Z_0$ have $C_j\left(\eta_3^{\mathscr{A}}\right) > t_{i'_3}$, so the lemma holds for all tasks. ■

By Lemma 6.65, all three tasks from $Z_0 \setminus \{\bar{\jmath}\}$ are completed no later than time $\hat{t}_3$. As a consequence, if $i'_3 = \hat{\imath}_3$ all three tasks from $Z_0 \setminus \{\bar{\jmath}\}$ are completed during $(t_{\hat{\imath}_3}, t_{\hat{\imath}_3+1}] = (t_{i'_3}, t_{i'_3+1}]$. Lemma 6.69 below establishes that if $i'_3 = \hat{\imath}_3$ then $\hat{t}_3$ is in some sense "close" to $t_{\hat{\imath}_3+1}$. Since $t_{\hat{\imath}_3+1}$ is the earliest point in time at which any tasks in $Z \setminus Z_0$ will be processed, this result is used to link Corollary 6.66, which provides bounds on the value of $\hat{t}_3$, to the processing of the tasks in $Z \setminus Z_0$. The case of $i'_3 < \hat{\imath}_3$ is considered in Lemma 6.70, which establishes how the remaining three tasks in Z_0 are processed after one has been completed.

Lemma 6.69 *If $i'_3 = \hat{\imath}_3$, $t_{\hat{\imath}_3+1} - \hat{t}_3 < n\varepsilon$.*

Proof. If $i'_3 = \hat{\imath}_3$, $Z_0 \subseteq A_{\hat{\imath}_3}$. Since any $j \in Z \setminus Z_0$ is preceded by at least one task in Z_1, it follows that

$$A_{\hat{\imath}_3} \subseteq ((N \setminus Z) \cup Z_0).$$

Lemma 6.65 implies that three tasks from Z_0 are processed at time $\hat{t}_3$, and consequently

$$\left|Z_0 \cap \left(N_1^{\hat{\imath}_3} \cup N_2^{\hat{\imath}_3}\right)\right| \geq 3. \tag{6.47}$$

If $\left(N_1^{\hat{\imath}_3} \cup N_2^{\hat{\imath}_3}\right) \subseteq Z_0$, (6.47) implies that some task $j \in Z_0 \backslash Z_1$ has $C_j\left(\eta_3^{\mathscr{A}}\right) = t_{i'_3+1}$ and hence $t_{i'_3+1} = \hat{t}_3$. On the other hand, if

$$\left(N_1^{\hat{\imath}_3} \cup N_2^{\hat{\imath}_3}\right) \setminus Z_0 \neq \emptyset,$$

there exists a task $g \in A_{\hat{\imath}_3} \setminus Z$. Lemma 4.10 then implies that

$$t_{\hat{\imath}_3+1} - \hat{t}_3 < t_{\hat{\imath}_3+1} - t_{\hat{\imath}_3} \leq np_g = n\varepsilon.$$

Hence the lemma holds in both cases. ■

Lemma 6.70 *If $i'_3 < \hat{\imath}_3$, $|A_{i'_3+1} \cap Z_0| = 3$ and each task in $A_{i'_3+1} \cap Z_0$ is processed at all times during $\left[t_{i'_3+1}, \hat{t}_3\right]$ in schedule $\eta_3^{\mathscr{A}}$.*

Proof. If $i'_3 < \hat{\imath}_3$, $t_{i'_3+1} \leq t_{\hat{\imath}_3} < \hat{t}_3$. Hence, Corollary 6.67 implies that $|A_{i'_3+1} \cap Z_0| = 3$. Since $m = 3$, the lemma follows from Lemma 6.65. ■

The consideration of $\eta_3^{\mathscr{A}}$ so far has focused on deriving results on the processing of tasks in Z_0 and the arrangement of points of allocation from the assumption that $\eta_3^{\mathscr{A}}$ is optimal. In order to more effectively reason about Algorithm P, several lemmas below will address the membership of the sets $N_1^{i'_3}$, $N_2^{i'_3}$ and $N_3^{i'_3}$. Lemma 6.71 establishes an upper bound on the completion times of tasks in $Z_0 \cap N_1^{i'_3}$, which is used in Lemma 6.72 to establish that $N_1^{i'_3}$ contains at most one task. Subsequently Lemma 6.73 demonstrates that $Z_0 \cap N_3^{i'_3} = \emptyset$, and Lemma 6.74 demonstrates that $\bar{\jmath} \in N_2^{i'_3}$. Since the evolution of the membership of N_1, N_2 and N_3 as Algorithm P progresses is well understood, these results are all used in Theorem 6.75 to prove that $\eta_3^{\mathscr{A}}$ is not optimal, at long last establishing that, for any algorithm $\mathscr{A} \in \mathfrak{F}$ and any partial order $\prec$ which contains a singular quartet, it is impossible to guarantee optimality for all problem instances with precedence constraints isomorphic to $\prec$.

Lemma 6.71 *For each task $j \in Z_0 \cap N_1^{i'_3}$, $C_j\left(\eta_3^{\mathscr{A}}\right) \leq t_{i'_3+1}$.*

Proof. Consider a task $j \in Z_0 \cap N_1^{i'_3}$. By Lemma 6.68, no tasks are completed during $\left(t_{\bar{\imath}_3}, t_{i'_3}\right]$ and hence, by Lemma 4.7, $j \in N_1^i$ for all i such that $\bar{\imath}_3 \leq i \leq i'_3$. Thus, j is processed continuously during $[t_{\bar{\imath}_3}, t_{i'_3+1}]$.

From (6.46), either $\hat{t}_3 \leq t_{i'_3+1}$ or $i'_3 < \hat{\imath}_3$. In the latter case, if $C_j\left(\eta_3^{\mathscr{A}}\right) > t_{i'_3+1}$, Lemma 6.70 states that j is processed continuously during $\left[t_{i'_3+1}, \hat{t}_3\right]$. Hence, regardless of whether $\hat{t}_3 \leq t_{i'_3+1}$ or $i'_3 < \hat{\imath}_3$, if $C_j\left(\eta_3^{\mathscr{A}}\right) > t_{i'+1}$, j is processed continuously during $\left[t_{\bar{\imath}_3}, \hat{t}_3\right]$. Consequently, Lemma 6.45 and Corollary 6.66 imply that, for a

small enough value of $\varepsilon > 0$,

$$p_j\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = p_j\left(t_{\bar{\imath}_3}, \eta_3^{\mathscr{A}}\right) - \left(\hat{t}_3 - t_{\bar{\imath}_3}\right) < p_j - \frac{4}{3} + n\varepsilon < p_j - 1, \tag{6.48}$$

contradicting Lemma 6.65. ■

Lemma 6.72 *If $N_1^{i'_3} \neq \emptyset$, $\left|N_1^{i'_3}\right| = 1$ and $t_{i'_3+1} < \hat{t}_3$.*

Proof. Consider any $j \in N_1^{i'_3}$. By Lemma 6.65, $j \in Z_0$. If $\hat{t}_3 \leq t_{i'_3+1}$ then $i'_3 = \hat{\imath}_3$ and hence Lemma 4.7 and Lemma 6.68 imply that j is processed continuously during $\left[t_{\bar{\imath}_3}, \hat{t}_3\right]$. As in the previous lemma, this leads to (6.48), contradicting Lemma 6.65.

Consequently, by (6.46) and Lemma 6.71,

$$C_j\left(\eta_3^{\mathscr{A}}\right) \leq t_{i'_3+1} < \hat{t}_3.$$

The lemma then follows from Corollary 6.67. ■

Lemma 6.73 *It holds that $Z_0 \cap N_3^{i'_3} = \emptyset$.*

Proof. Suppose that there exists a task $j \in Z_0 \cap N_3^{i'_3}$. As in Lemma 6.71, Lemma 6.68 states that no tasks are completed during $\left(t_{\bar{\imath}_3}, t_{i'_3}\right]$ and thus Lemma 4.7 implies $j \in N_3^i$ for all i such that $\bar{\imath}_3 \leq i \leq i'_3$. If $i'_3 = \hat{\imath}_3$, Corollary 6.47 implies that, for a small enough value of $\varepsilon > 0$,

$$p_j\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = p_j\left(t_{\bar{\imath}_3}, \eta_3^{\mathscr{A}}\right) > p_j - n\varepsilon > p_j - 1,$$

which contradicts Lemma 6.65.

On the other hand, if $i'_3 \neq \hat{\imath}_3$ it follows from (6.46) that $t_{i'_3+1} \leq t_{\hat{\imath}_3} < \hat{t}_3$. In this case, Corollary 6.47 and Corollary 6.66 together imply, for a small enough value of $\varepsilon > 0$,

$$\begin{aligned} p_j\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) &\geq p_j\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) - \left(\hat{t}_3 - t_{i'_3+1}\right) = p_j\left(t_{\bar{\imath}_3}, \eta_3^{\mathscr{A}}\right) - \hat{t}_3 + t_{i'_3+1} \\ &> p_j - n\varepsilon - \left(n\varepsilon + \frac{4}{3}\right) + \min_{g \in Z_0} C_g\left(\eta_3^{\mathscr{A}}\right) \geq p_j - 2n\varepsilon - \frac{4}{3} + \min_{g \in Z_0} p_g \\ &= p_j - 2n\varepsilon - \frac{1}{3} > p_j - 1, \end{aligned}$$

which contradicts Lemma 6.65. ■

Lemma 6.74 *It holds that $\bar{\jmath} \in N_2^{i'_3}$.*

Proof. Lemma 6.73 demonstrates that $\bar{\jmath} \notin N_3^{i'_3}$. If $\bar{\jmath} \in N_1^{i'_3}$, Lemma 6.72 implies $t_{i'_3+1} < \hat{t}_3$ and consequently, by Lemma 6.71 and Corollary 6.66,

$$8 \leq 4\bar{h} \leq p_{\bar{\jmath}} \leq C_{\bar{\jmath}}\left(\eta_3^{\mathscr{A}}\right) \leq t_{i'_3+1} < \hat{t}_3 < n\varepsilon + \frac{4}{3},$$

a contradiction for a small enough value of $\varepsilon > 0$. Hence the lemma is proven. ■

Theorem 6.75 *For each partial order $\prec$ which contains a singular quartet, there exists no algorithm $\mathscr{A} \in \mathfrak{F}$ that can solve all problem instances where the precedence constraints are isomorphic to $\prec$.*

Proof. Recall that the definition of $\mathfrak{F}$ implies that $\mu_j = \mu^{\mathscr{A}}(d_j, \varphi_j)$ for all $j \in N$, where $\mu^{\mathscr{A}}$ is a function defining algorithm $\mathscr{A}$. For all j such that $j \neq \bar{\jmath}$ and $j \not\to \bar{\jmath}$, d_j and p_j are the same in φ^1 and φ^3, and consequently φ_j remains the same as well. This implies that μ_j also remains the same between these two problem instances. From Lemma 6.53 and the definition of $\mathfrak{F}$, at least two of the tasks $j \in Z_0 \setminus \{\bar{\jmath}\}$ have $\mu_j = \bar{\mu}$ and the third has $\mu_j \geq \bar{\mu}$. Additionally, by (6.45) and the definition of φ^3,

$$1 + \bar{\mu} - p_{\bar{\jmath}} - \frac{1}{25n^3} > \mu_{\bar{\jmath}} \qquad \text{or} \qquad \mu_{\bar{\jmath}} > 1 + \bar{\mu} - p_{\bar{\jmath}} + \frac{1}{25n^3}.$$

By Lemma 6.72 and Lemma 6.73, at least three of the tasks in Z_0 are in $N_2^{i'_3}$, and consequently $\mu_g = \bar{\mu}$ for some $g \in N_2^{i'_3} \setminus \{\bar{\jmath}\}$. Because $g \neq \bar{\jmath}$, Lemma 6.65 implies

$$p_g\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = p_g - 1 = 0. \tag{6.49}$$

By Lemma 6.74 $\bar{\jmath} \in N_2^{i'_3}$. Hence, the definition of N_2 implies

$$p_{\bar{\jmath}}\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \mu_{\bar{\jmath}} = p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \mu_g = p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \bar{\mu}. \tag{6.50}$$

Two cases are now considered.

- If $i'_3 = \hat{\imath}_3$, it follows that $t_{i'_3+1} = t_{\hat{\imath}_3+1} \geq \hat{t}_3$ and hence, by (6.49),

  $$0 = p_g\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) \geq p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) \geq 0.$$

  Hence, by (6.50) and Lemma 6.65,

  $$p_{\bar{\jmath}} - 1 + \mu_{\bar{\jmath}} = p_{\bar{\jmath}}\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) + \mu_{\bar{\jmath}} \geq p_{\bar{\jmath}}\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \mu_{\bar{\jmath}} = p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \bar{\mu} = \bar{\mu}$$

and, taking into account Lemma 6.69,

$$p_{\bar{\jmath}} - 1 + \mu_{\bar{\jmath}} = p_{\bar{\jmath}}\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) + \mu_{\bar{\jmath}} < p_{\bar{\jmath}}\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + n\varepsilon + \mu_{\bar{\jmath}}$$

$$= p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \bar{\mu} + n\varepsilon = \bar{\mu} + n\varepsilon.$$

Rearranging the two inequalities above gives

$$1 + \bar{\mu} - p_{\bar{\jmath}} \leq \mu_{\bar{\jmath}} < 1 + \bar{\mu} - p_{\bar{\jmath}} + n\varepsilon. \tag{6.51}$$

- On the other hand, if $i'_3 \neq \hat{\imath}_3$ it follows from (6.46) that $t_{i'_3+1} \leq t_{\hat{\imath}_3} < \hat{t}_3$. Since at least two tasks $q \in Z_0 \setminus \{\bar{\jmath}\}$ have $\mu_q = \bar{\mu}$, Corollary 6.67 implies that task g may be selected such that $C_g\left(\eta_3^{\mathscr{A}}\right) \geq \hat{t}_3 > t_{i'_3+1}$.

  This implies $\{\bar{\jmath}, g\} \subseteq A_{i'_3+1}$, so by Lemma 6.70,

$$p_{\bar{\jmath}}\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) - p_{\bar{\jmath}}\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) - p_g\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = \hat{t}_3 - t_{i'_3+1}.$$

  Combining this with (6.49) and (6.50) gives

$$p_{\bar{\jmath}}\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) + \mu_{\bar{\jmath}} = p_{\bar{\jmath}}\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \mu_{\bar{\jmath}} - p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + p_g\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right)$$

$$= p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + \bar{\mu} - p_g\left(t_{i'_3+1}, \eta_3^{\mathscr{A}}\right) + p_g\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = \bar{\mu}.$$

  Applying Lemma 6.65 gives

$$\mu_{\bar{\jmath}} = \bar{\mu} - p_{\bar{\jmath}}\left(\hat{t}_3, \eta_3^{\mathscr{A}}\right) = 1 + \bar{\mu} - p_{\bar{\jmath}}. \tag{6.52}$$

For a small enough value of $\varepsilon > 0$, both (6.51) and (6.52) contradict the definition of φ^3, which specifically selected $\bar{\jmath}$ so that (6.45) holds. Since the Lemmas guaranteeing the existance of such a task depended only on the assumption that the partial order $\prec$ is in the domain of algorithm $\mathscr{A}$, it follows that this assumption does not hold. Since $\prec$ is an arbitrary partial order containing a singular quartet, and $\mathscr{A}$ is an arbitrary member of the family $\mathfrak{F}$, the theorem is proven. ■

A summary of the long chain of reasoning leading to this conclusion follows.

- First, the assumption was made that an arbitrary partial order $\prec$ containing a singular quartet is in the domain of an arbitrary algorithm $\mathscr{A} \in \mathfrak{F}$. Most proofs throughout this subsection rely on this assumption.

- Under the condition that all $j \in Z_0$ have $p_j = d_j = 1$, Lemma 6.53 demonstrates that at least three of these tasks have a common value of μ_j, denoted $\bar{\mu}$, and that this is the smallest μ_j among the four tasks in Z_0.
- By the definition of the family $\mathfrak{F}$, the algorithm $\mathscr{A}$ cannot "see" the processing times and due dates of the other three tasks in Z_0 when determining the value of μ_g for some $g \in Z_0$. Consequently, for $\mathscr{A}$ to guarantee an optimal solution in the event that $p_g = d_g = 1$, it must always assign a value to μ_g as if $p_j = d_j = 1$ for all $j \in Z_0$, regardless of whether this is actually the case.
- An alternative case is considered where
  - $p_j = d_j = 1$ for both $j \in Z_0 \setminus Z_1$; and
  - the two tasks in Z_1 have processing times and due dates such that, if they are given identical μ's, the processing of the tasks in each Z_i will take place in parallel.

  The former condition ensures that the insight of Lemma 6.53 constrains the values of μ's for both $j \in Z_0 \setminus Z_1$. The latter condition implies that, as demonstrated in Lemma 6.63, if each of the two tasks $j \in Z_1$ have μ_j which is in some sense "close to" $1 + \bar{\mu} - p_j$, the four tasks in $Z_{\bar{h}}$ will be processed in parallel, ensuring a non-optimal schedule. Hence there exists some task $\bar{\jmath} \in Z_1$ with a value of $\mu_{\bar{\jmath}}$ that is not "close to" $1 + \bar{\mu} - p_{\bar{\jmath}}$.
- As before, algorithm $\mathscr{A}$ must always assign the same value of $\mu_{\bar{\jmath}}$, regardless of the processing times and due dates of the other three tasks in Z_0.
- A final case is considered where
  - $p_j = d_j = 1$ for all three $j \in Z_0 \setminus \{\bar{\jmath}\}$; and
  - $\bar{\jmath}$ has the same processing time and due date as in the previous case.

  These two conditions ensure that the insights of Lemma 6.53 and Lemma 6.63 apply in this final case, constraining the values of the μ's. It is then demonstrated in Theorem 6.75 that an optimal schedule requires a value of $\mu_{\bar{\jmath}}$ that is "close to" $1 + \bar{\mu} - p_{\bar{\jmath}}$, and hence that it is impossible for an algorithm from $\mathfrak{F}$ to guarantee optimality for all problem instances precedence constraints isomorphic to $\prec$.

The domain of the Zinder-Roper Algorithm has now been established.

Corollary 6.76 *The class $\mathcal{D}^{\mathscr{ZR}}$ is composed of all partial orders that do not contain a singular quartet.*

6.4 The Garey-Johnson Algorithm

Based on the work in the previous section, the reasoning below proves the domain of the Garey-Johnson Algorithm is a subclass of the domain of the Zinder-Roper Algorithm and a superclass of the domain of the Brucker-Garey-Johnson Algorithm.

6.4.1 The Class of Partial Orders

Using the definition of a singular pair from Section 6.3, the domain of the Garey-Johnson Algorithm, denoted $\mathcal{D}^{\mathscr{GJ}}$, can be defined as follows.

Recall that a pair of independent tasks is singular if and only if there exists a subgraph corresponding to a transitive set of the original partially ordered set of tasks, for which this pair is a cover, that has width at least four and is not splittable. Further define a singular task as being a task j for which there exists a singular pair S such that $S \subseteq K_j$. Recall that a singular quartet is a set of four independent tasks that contains a singular pair, and further recall that this is the prohibited structure for $\mathcal{D}^{\mathscr{ZR}}$. Define a pseudo-singular set of tasks as being a set of at least two independent tasks that does not contain a singular pair of tasks or a singular task.

The domain of the Garey-Johnson Algorithm may then be defined to be the class of all partial orders that do not contain any pseudo-singular quartets.

Notice that, since every singular quartet is also pseudo-singular, $\mathcal{D}^{\mathscr{GJ}} \subset \mathcal{D}^{\mathscr{ZR}}$.

Additionally, consider the two classes of prohibited subgraphs for the Brucker-Garey-Johnson Algorithm, defined in Subsection 6.2.1. Every singular task is a task that precedes a set of three independent tasks. This means that every partial order with a set of four indepedent tasks that contains a singular task also contains a prohibited graph of the first kind (specified in Figure 6.2). For every singular pair that does not contain a task that precedes a set of three independent tasks, both tasks in this singular pair each precede a set of two independent tasks that are independent of one another. Thus, every partial order with a set of four indepedent tasks that contains a singular pair also contains a prohibited graph of the first kind or the second kind (the latter of which is specified in Figure 6.3). Thus,

$$\mathcal{D}^{\mathscr{BGJ}} \subset \mathcal{D}^{\mathscr{GJ}} \subset \mathcal{D}^{\mathscr{ZR}},$$

establishing an ordering of the primary approximation algorithms considered in this thesis.

Thus, Theorem 5.28 establishes the peculiar result that the Garey-Johnson Algo-

rithm dominates the family $\mathfrak{F}$ of algorithms in terms of performance guarantee, while this algorithm's domain is a strict subclass of the domain of the Zinder-Roper Algorithm. It is not known if the Zinder-Roper Algorithm is also dominant in terms of performance guarantee, but consider the algorithm that constructs $\eta^{\mathscr{GJ}}$ (the schedule produced by the Garey-Johnson Algorithm) and $\eta^{\mathscr{ZR}}$ (the schedule produced by the Zinder-Roper Algorithm), and selects the best of the two. Although this algorithm may not be a member of $\mathfrak{F}$ it does have both a performance guarantee and a set of solvable precedence constraints that dominates all algorithms in $\mathfrak{F}$, and it is polynomial. It is quite surprising that such an algorithm exists.

6.4.2 Proving Sufficiency

In this subsection it will be proven that a sufficient condition for the optimality of the Garey-Johnson Algorithm is that the precedence constraints contain no pseudo-singular quartets. Recall that b is the largest $i < i^*$ for which $|H_i| < m$, c is the smallest $i > b$ for which $|H_i| > m$ and a is the largest $i < b$ for which $|H_i| > m$. The next lemma will prove a result about the successors of the tasks in H_b.

Lemma 6.77 *For any non-optimal schedule $\eta^{\mathscr{GJ}}$ created by the Garey-Johnson Algorithm, for all $j \in H_{\hat{\imath}}$ and all $b \leq \hat{\imath} < c$,*

$$\bigcup_{i=\hat{\imath}+1}^{i^*-1} H_i \setminus K_j \neq \emptyset.$$

Proof. Consider any task $j \in H_{\hat{\imath}}$ such that $H_i \subseteq K_j$ for all $b \leq \hat{\imath} < i < i^*$ and note that this implies that $C_j\left(\eta^{\mathscr{GJ}}\right) = t_{\hat{\imath}+1}$. Since $x \in H_{i^*-1}$, from (4.17) and (4.27)

$$a_j = \min_{g \in K_j} \mu_g \leq \mu_x \leq \rho^* = p_x\left(t_{i^*}, \eta^{\mathscr{GJ}}\right) + \mu_x \leq \max_{g \in K_j}\left(\frac{p_g}{u_g} + \mu_g\right) = b_j.$$

Since $|H_i| \geq m$ implies $N_1^i \cup N_2^i \subseteq H_i$,

$$p_g\left(t_{i^*}, \eta^{\mathscr{GJ}}\right) + \mu_g \geq \rho^* \qquad \text{for all} \qquad g \in \bigcup_{i=\hat{\imath}+1}^{i^*-1} H_i.$$

Hence, it follows from (4.18), (4.19) and the definition of H_i that

$$\mu_j \geq \rho^* + \frac{1}{m}\sum_{g \in K_j} \beta_g(\rho^*) \geq \rho^* + \frac{1}{m} \sum_{g \in \bigcup_{i=\hat{\imath}+1}^{i^*-1} H_i} \min\left[p_g, \max\left\{0, p_g - \rho^* + \mu_g\right\}\right]$$

$$\geq \rho^* + \frac{1}{m} \sum_{g \in \bigcup_{i=i+1}^{i^*-1} H_i} \min \left[p_g, \max \left\{ 0, p_g - \left(p_g \left(t_{i^*}, \eta^{\mathscr{GJ}} \right) + \mu_g \right) + \mu_g \right\} \right]$$

$$= \rho^* + \frac{1}{m} \sum_{g \in \bigcup_{i=i+1}^{i^*-1} H_i} \left(p_g - p_g \left(t_{i^*}, \eta^{\mathscr{GJ}} \right) \right) = \rho^* + t_{i^*} - t_{i+1}.$$

Consequently

$$C_j \left(\eta^{\mathscr{GJ}} \right) + \mu_j \geq t_{i+1} + (\rho^* + t_{i^*} - t_{i+1}) = t_{i^*} + \rho^* = G \left(\eta^{\mathscr{GJ}} \right),$$

contradicting the selection of t_{i^*}. ■

Corollary 6.78 *For any non-optimal schedule $\eta^{\mathscr{GJ}}$ created by the Garey-Johnson Algorithm, $|H_b| \geq 2$.*

Proof. If $|H_b| = 1$ then, for $j \in H_b$, Lemma 6.1 implies $H_i \subseteq \{j\} \cup K_j$ for all $i > b$ and since Corollary 5.15 implies $|H_{b+1}| \geq m \geq 3$, it follows that $C_j(\eta) = t_{b+1}$. Consequently $H_i \subseteq K_j$ for all $b < i < i^*$ and the result follows from Lemma 6.77. ■

Lemma 6.79 *For any non-optimal schedule $\eta^{\mathscr{GJ}}$ constructed by the Garey-Johnson Algorithm, there exists a singular pair $B = \{b_1, b_2\}$ such that*

$$B \subset \bigcup_{i=b}^{c} H_i.$$

Proof. From Lemma 4.14, the μ's are preserving, and consequently if $\eta^{\mathscr{GJ}}$ is not optimal for the criterion L_{max} it is also not optimal for the criterion G. Therefore, a, b and c, as defined in Section 6.1, exist.

Among all $j \in H_{c-1}$ select as g the task with the largest value of $|K_j \cap H_c|$. Since $|H_c| > m$ and $|H_{c-1}| \leq m$, it follows from Lemma 6.1 that $|K_g \cap H_c| \geq 2$.

First consider the case where $|K_g \cap H_c| \geq 3$. By Lemma 6.77 there exists a task $q \in H_c \setminus K_g$. In a similar manner to the proof of Lemma 6.22, consider the set

$$S = \{g, q\} \cup (K_g \cap H_c).$$

Note that S is a non-splittable transitive set of N, with width at least four, for which $\{g, q\}$ is a cover, i.e. $\{g, q\}$ is a singular pair.

On the other hand, consider the case where $|K_g \cap H_c| = 2$. If there exists a task $j \in H_{c-1}$ such that $|(K_j \cap H_c) \setminus K_g| = 2$, select q as such a task and consider the set

$$S = \{g, q\} \cup (K_g \cap H_c) \cup (K_q \cap H_c) .$$

Note that, similar to the previous case, S is a non-splittable transitive set of N, with width four, for which $\{g, q\}$ is a cover, i.e. $\{g, q\}$ is a singular pair.

Alternatively, consider the case where $|K_g \cap H_c| = 2$ and there does not exist a task $j \in H_{c-1}$ such that $|(K_j \cap H_c) \setminus K_g| = 2$. In this case, $|H_b| < m$ and $|H_c| > m$ imply the existence of some i such that $b \leq i < c-1$ and there exists a task $j \in H_i$ for which $|K_j \cap H_{i+1}| \geq 2$. Select as $\hat{\imath}$ the largest such i and select as q such a task j.

Observe that, for all i such that $\hat{\imath} < i < c-1$, for each $j \in H_i$ either $j \in H_{i+1}$ or $|K_j \cap H_{i+1}| = 1$. Two cases will now be considered below: either $q \rightarrow g$ or $q \not\rightarrow g$. Further observe that, for each $j \in H_{c-1} \setminus \{g\}$, either $j \in H_c \setminus K_g$ or $(K_j \cap H_c) \setminus K_g = 1$.

In the case where $q \rightarrow g$, Lemma 6.77 implies the existance of a task $v \in H_{\hat{\imath}+1} \setminus K_q$. Now consider the set

$$S = \{q, v\} \cup \left((K_q \cup K_v) \cap \left(\bigcup_{i=\hat{\imath}+1}^{c} H_i \right) \right) .$$

Again S is a non-splittable transitive set of N, with width at least four, for which $\{q, v\}$ is a cover, i.e. $\{q, v\}$ is a singular pair.

Finally, consider the case where $q \not\rightarrow g$. In this case, consider the set

$$S = \{g, q\} \cup (K_g \cap H_c) \cup \left(K_q \cap \left(\bigcup_{i=\hat{\imath}+1}^{c} H_i \right) \right) .$$

As before, S is a non-splittable transitive set of N, with width at least four, for which $\{g, q\}$ is a cover, i.e. $\{g, q\}$ is a singular pair. ■

The following theorem proves a sufficient condition for the optimality of the Garey-Johnson Algorithm.

Theorem 6.80 *For any instance of $P|prec, pmtn|L_{max}$ where the precedence constraints do not contain a pseudo-singular quartet, the Garey-Johnson Algorithm produces an optimal schedule.*

Proof. Consider a singular pair $B = \{b_1, b_2\}$ as defined in Lemma 6.79, and consider any transitive non-splittable subset S, with width at least four, of N for which B is a cover. If $B \subseteq K_j$ for any $j \in H_a$, j is a singular task. Recall that

Corollary 5.15 demonstrated that Garey-Johnson Algorithm produces an optimal schedule when $m = 2$, and thus,

$$|H_a| \geq m + 1 \geq 4. \tag{6.53}$$

Hence, this singular task j can be complemented by another three independent tasks, yielding a pseudo-singular quartet.

On the other hand, if no such j exists, Lemma 6.1 implies the existence of $j_1 \in H_a$ and $j_2 \in H_a$ such that

- either $j_1 \to b_1$ or $j_1 = b_1$, and
- either $j_2 \to b_2$ or $j_2 = b_2$.

Consider the set

$$T = \{j_1, j_2\} \cup S \cup (K_{j_1} \cap J_{b_1}) \cup (K_{j_2} \cap J_{b_2})$$

and note that T is a transitive set of N, with width at least four, and $\{j_1, j_2\}$ is a cover of T.

Suppose that $T = S_1 \cup S_2$ where $S_1 \neq \emptyset$, $S_2 \neq \emptyset$, and $j \to g$ for any $j \in S_1$ and $g \in S_2$, i.e. suppose T is splittable. The non-splittability of S, along with the definition of S_1 and S_2, implies that

$$\{j_1, j_2\} \subseteq S_1 \qquad \text{and} \qquad S \subseteq S_2.$$

However, the fact that $j_1 \not\to b_2$ and $j_2 \not\to b_1$ contradicts the definition of S_1 and S_2, so T is not splittable. Thus $\{j_1, j_2\}$ is a singular pair, and (6.53) implies that these tasks can be complemented by another two independent tasks which yields a singular (and thus also pseudo-singular) set of cardinality four. ■

6.4.3 Proving Necessity

This subsection is devoted to proving that, whenever a pseudo-singular quartet is present, an optimal schedule from the Garey-Johnson Algorithm cannot be guaranteed. In the case where there exists a singular quartet, i.e. a set of four independent tasks that contains, as a subset, a singular pair, then the non-optimality of the Garey-Johnson Algorithm follows from Theorem 6.75.

This leaves only the case where there is a set of cardinality four containing a singular task, but no singular pair. A problem instance φ will now be specified

that demonstrates that optimality cannot be guaranteed when the Garey-Johnson Algorithm schedules tasks with precedence constraints containing a pseudo-singular quartet.

Consider an arbitrary partial order $\prec$ that contains at least one pseudo-singular quartet. Let Q be a pseudo-singular subset, let $k_1 \in Q$ be a singular task, let $B \in K_{k_1}$ be a singular pair and let I be a canonical transitive set of N for which B is a cover. For some $\underline{h} \geq 3$, the definition of k_1 implies the existance of a chain of tasks

$$k_1 \rightarrow k_2 \rightarrow \cdots \rightarrow k_{\underline{h}-2}$$

such that

- $k_i \in \tilde{K}_{k_{i-1}}$ for any i for which $1 < i < \underline{h} - 1$ and
- $B \subseteq \tilde{K}_{k_{\underline{h}-2}}$.

Define $Z_0 = Q$, $Z_i = \{k_i\}$ for all i such that $1 \leq 1 < \underline{h} - 1$, let $Z_{\underline{h}-1} = B$ and use the Z-Algorithm to construct from I the sets Z_i for all i such that $\underline{h} \leq i \leq \bar{h}$. Note that this implies that $Z_{\underline{h}-1} = B = Z_{\underline{h}}$. Define $H = \mathbb{N} \cap [0, \bar{h}]$ and $Z = \cup_{i \in H} Z_i$.

Problem instance φ will now be defined as follows. The precedence contraints are defined by the partial order $\prec$. As before, let all tasks $j \in N \setminus Z$ have $p_j = \varepsilon$ and $d_j = \bar{h} + 1$. It will be shown that, for adequately small $\varepsilon > 0$, these tasks will have in some sense a negligible effect on the resulting schedule. Indeed, throughout the reasoning below, it will be assumed that $\varepsilon > 0$ is adequately small, without explicitly stating this assumption at every turn. Similar to before, for all $j \in N \setminus Z$, let

$$p_j = |\{i : i \in H \wedge j \in Z_i\}| \qquad \text{and} \qquad d_j = \max\{i : i \in H \wedge j \in Z_i\}.$$

Finally, let $m = 3$.

The following lemmas prove results regarding the μ's generated by the Garey-Johnson Algorithm for the problem instance φ. In particular, it will be proven that $\mu_j = -d_j$ for all $j \in Z$. Throughout this section, it will be assumed that the Garey-Johnson Algorithm is being applied to the problem instance φ.

Lemma 6.81 *For all $j \in N \setminus Z$ such that $K_j \cap Z = \emptyset$, $\mu_j < -\bar{h} - \frac{2}{3}$.*

Proof. This lemma will be proven by induction. For all $j \in N \setminus Z$, if $K_j = \emptyset$ then

$$\mu_j = -\bar{h} - 1 < -\bar{h} - \frac{2}{3}.$$

Now consider any $j \in N \setminus Z$ such that $K_j \neq \emptyset$ and $K_j \cap Z = \emptyset$. For all $g \in K_j$, assume that

$$-\bar{h} - 1 \leq \mu_g \leq \varepsilon |K_g| \left(1 + \frac{|K_g| + 1}{2m}\right) - \bar{h} - 1. \tag{6.54}$$

Note that (6.54) holds for all $g \in N \setminus Z$ for which $K_g = \emptyset$.

Recall that the value of μ_j assigned by the Garey-Johnson algorithm depends on a_j and b_j defined in equation (4.17). Given the inductive assumption (6.54), (4.17) and the fact that $|K_g| < |K_j|$ for all $g \in K_j$ imply

$$-\bar{h} - 1 \leq \min_{g \in K_j} \mu_j = a_j < b_j = \max_{g \in K_j} (p_g + \mu_g)$$

$$\leq \varepsilon + \varepsilon(|K_j| - 1)\left(1 + \frac{|K_j|}{2m}\right) - \bar{h} - 1.$$

Recall that, according to (4.19), the μ's computed by the Garey-Johnson Algorithm depend, in part, on the function $\beta_g(\rho)$ as defined in (4.18). Hence, as a consequence of the preceding bounds on a_j and b_j, (4.18) and (4.19) imply

$$-\bar{h} - 1 = -d_j \leq \mu_j = \max\left\{-d_j, \max_{a_j \leq \rho \leq b_j}\left(\rho + \frac{1}{m}\sum_{g \in K_j} \beta_g(\rho)\right)\right\}$$

$$= \max\left\{-\bar{h} - 1, \max_{a_j \leq \rho \leq b_j}\left(\rho + \frac{1}{m}\sum_{g \in K_j} \min\left[p_g, \max\{0, p_g + \mu_g - \rho\}\right]\right)\right\}$$

$$\leq \max\left\{-\bar{h} - 1, b_j + \frac{1}{m}\sum_{g \in K_j} p_g\right\}$$

$$\leq \max\left\{-\bar{h} - 1, \varepsilon + \varepsilon(|K_j| - 1)\left(1 + \frac{|K_j|}{2m}\right) - \bar{h} - 1 + \frac{1}{m}\sum_{g \in K_j} p_g\right\}$$

$$= \varepsilon + \varepsilon(|K_j| - 1)\left(1 + \frac{|K_j|}{2m}\right) - \bar{h} - 1 + \frac{|K_j|}{m}\varepsilon$$

$$= \varepsilon\left(1 + |K_j| - 1 + \frac{|K_j|^2 - |K_j|}{2m} + \frac{|K_j|}{m}\right) - \bar{h} - 1$$

$$= \varepsilon |K_j| \left(1 + \frac{|K_j| + 1}{2m}\right) - \bar{h} - 1.$$

Consequently, (6.54) holds for all $g \in N \setminus Z$ such that $K_g \cap Z = \emptyset$, and thus the lemma holds for adequately small ε. ■

Corollary 6.82 *For all $j \in N \setminus Z$ such that $K_j \cap Z = \emptyset$, and all $\rho > -\bar{h} - \frac{1}{2}$, $\beta_j(\rho) = 0$.*

Proof. By Lemma 6.81 and (4.18), for an adequately small $\varepsilon > 0$,

$$\beta_j(\rho) = \min\left[p_j, \max\{0, p_j + \mu_j - \rho\}\right] \leq \min\left[p_j, \max\left\{0, p_j - \bar{h} - \frac{2}{3} + \bar{h} + \frac{1}{2}\right\}\right]$$

$$= \min\left[p_j, \max\left\{0, p_j - \frac{1}{6}\right\}\right] = \min\left[p_j, 0\right] = 0,$$

and the corollary follows. ■

Lemma 6.83 *For all $j \in Z_{\bar{h}}$, $\mu_j = -d_j = -\bar{h}$.*

Proof. If $K_j = \emptyset$, the Garey-Johnson Algorithm sets $\mu_j = -d_j$, so the lemma holds. For the remainder of this proof it will be assumed that $K_j \neq \emptyset$.

Note that, from (4.19) and Lemma 6.81,

$$-\bar{h} - 1 \leq \min_{g \in K_j} \mu_j = a_j < b_j = \max_{g \in K_j} (p_g + \mu_g) < \varepsilon - \bar{h} - \frac{2}{3}.$$

Therefore, for an adequately small value of $\varepsilon > 0$, (4.18) and (4.19) imply

$$-\bar{h} = -d_j \leq \mu_j = \max\left\{-d_j, \max_{a_j \leq \rho \leq b_j}\left(\rho + \frac{1}{m}\sum_{g \in K_j} \beta_g(\rho)\right)\right\}$$

$$\leq \max\left\{-d_j, b_j + \frac{1}{m}\sum_{g \in K_j} p_g\right\} \leq \max\left\{-\bar{h}, \varepsilon - \bar{h} - \frac{2}{3} + \frac{n}{m}\varepsilon\right\} = -\bar{h}.$$

Hence, the lemma holds in this case as well. ■

For all $j \in N$ such that $j \in Z$ or $J_j \cap Z \neq \emptyset$, define

$$\iota(j) = \max\left\{i : i \in H \wedge (j \in Z_i \vee J_j \cap Z_i \neq \emptyset)\right\}.$$

In words, $\iota(j)$ is the index of the last Z_i that contains either j or a predecessor of j. Note that, for all $j \in Z$, $d_j = \iota(j)$. Note that this implies that $\iota(j) \in H$.

Define

$$X = \{j : J_j \cap Z \neq \emptyset \wedge K_j \cap Z \neq \emptyset\},$$

i.e. X is the set of all tasks in $N \setminus Z$ which both precede and succeed tasks in Z. This set can be seen as the set of all tasks in $N \setminus Z$ that can affect the value of μ_j for any $j \in Z$, and which have not yet been considered in Lemma 6.81. This set of tasks is the subject of several lemmas below.

Lemma 6.84 *For any $j \in X$, $\iota(j) < \underline{h} - 1$.*

Proof. Suppose this is not the case, i.e. suppose there exists some $j \in X$ such that $\iota(j) \geq \underline{h} - 1$. Recall that the sets $Z_{\underline{h}-1} = Z_{\underline{h}}, Z_{\underline{h}+1}, \ldots, Z_{\bar{h}}$ are constructed by the Z-Algorithm from a canonical transitive set I of N. Recall also that $Z_0 = Q$ is a pseudo-singular set of four tasks. Consequently, the fact that $\iota(j) \geq \underline{h} - 1$ implies that there is some $g \in I$ for which $g \rightarrow j$.

On the other hand, the definition of X implies that there is some $q \in Z$ for which $j \rightarrow q$. By the definition of the problem instance φ, the tasks comprising set $Z \setminus I$ are a chain of tasks $k_1 \rightarrow k_2 \rightarrow \cdots \rightarrow k_{\underline{h}-2}$ and the three tasks in $Z_0 \setminus \{k_1\}$. Since $I \subseteq K_{k_{\underline{h}-2}} \subseteq K_{k_1}$, it follows that $j \in K_{k_1}$ and hence $q \in K_{k_1}$. If $q \in Z_0 \setminus \{k_1\}$ then $Z_0 = Q$ is not a set of independent tasks, contradicting the initial selection of Q as a pseudo-singular set.

As a consequence of the preceding reasoning, $q \in I$. Since $g \rightarrow j \rightarrow q$, by the definition of a transitive set of N, $j \in I$ and hence $j \in Z$. This contradicts the definition of X, and so the lemma is proven. ■

Lemma 6.85 *For all $j \in X$, $K_j \cap Z_{\iota(j)+1} = \emptyset$.*

Proof. As before, for all $i \in H$ such that $1 \leq i \leq \underline{h} - 2$, there exists a task k_i such that $Z_i = \{k_i\}$ and $Z_{i+1} \subseteq \tilde{K}_{k_i}$, where $\tilde{K}_q$ is the set of immediate successors of task q. Additionally, recall that $k_1 \in Z_0$. As a consequence of Lemma 6.84, $\iota(j) \leq \underline{h} - 2$. Two cases are considered below.

- If $\iota(j) \geq 1$, $j \in K_{k_{\iota(j)}}$, and thus $Z_{\iota(j)+1} \subseteq \tilde{K}_{k_{\iota(j)}}$. If $K_j \cap Z_{\iota(j)+1} \neq \emptyset$, this implies the existance of a task $g \in Z_{\iota(j)+1}$ such that $k_{\iota(j)} \rightarrow j \rightarrow g$, contradicting the fact that g is an immediate successor of $k_{\iota(j)}$.

- If $\iota(j) = 0$, there exists some task $g \in Z_0$ such that $g \rightarrow j$. If $K_j \cap Z_{\iota(j)+1} \neq \emptyset$, this implies $j \rightarrow k_1$. This implies that $g \rightarrow k_1$, contradicting the fact that $\{g, k_1\} \subset Z_0$ and that Z_0 is a set of independent tasks.

Hence, in both cases j precedes no tasks in $Z_{\iota(j)+1}$. ■

Lemma 6.86 *For any $j \in Z$ such that $\iota(j) \geq \underline{h} - 1$, and any ρ such that $\iota(j) \leq -\rho \leq \bar{h}$,*

$$\sum_{g \in K_j} \beta_g(\rho) = \sum_{g \in K_j \cap Z} \beta_g(\rho).$$

Proof. For any task $g \in K_j \setminus Z$, the definition of X implies either $g \in X$ or $K_g \cap Z = \emptyset$. However,

$$\iota(g) \geq \iota(j) \geq \underline{h} - 1$$

implies, along with Lemma 6.84, that $g \notin X$ and hence $K_g \cap Z = \emptyset$. Hence the lemma follows from Corollary 6.82. ■

As has been stated earlier, it will be proven that $\mu_j = -d_j$ for all $j \in Z$. The proof will proceed by induction on $\iota(j)$, in descending order. There are two inductive assumptions: for some $\hat{\imath} \in H$,

$$\mu_j = -\iota(j) \text{ for all } j \in Z \text{ such that } \iota(j) \geq \hat{\imath} \tag{6.55}$$

and

$$\mu_j < -\iota(j) - \frac{2}{3} \text{ for all } j \in X \text{ such that } \iota(j) \geq \hat{\imath}. \tag{6.56}$$

Lemma 6.83 proves that (6.55) for $\hat{\imath} = \bar{h}$, and Lemma 6.84 demonstrates that (6.56) holds vacuously for all $\hat{\imath} \in H$ such that $\hat{\imath} \geq \underline{h} - 1$.

For all $j \in Z$, define

$$P_j = \{i : i \in H \wedge j \in Z_i\}$$

and note that

$$P_j = \{i : i \in H \wedge d_j - p_j < i \leq d_j\}. \tag{6.57}$$

Define $\varrho = \lfloor -\rho \rfloor$, and for all $i \in H$ define

$$\delta(i) = \begin{cases} 1 & \text{if } i \leq \varrho \\ -\rho - \varrho & \text{if } i = \varrho + 1 \\ 0 & \text{otherwise} \end{cases}.$$

Additionally, in order to avoid the consideration of unnecessary cases, let $Z_{\bar{h}+1} = \emptyset$.

Lemma 6.87 *For any $j \in Z$ and any ρ such that $0 \leq -\rho \leq \bar{h}$, if (6.55) holds for $\hat{\imath} = \iota(j)$,*

$$\beta_j(\rho) = \sum_{i \in P_j} \delta(i).$$

Proof. For any $j \in Z$, if $\hat{\imath} = \iota(j)$, the first inductive assumption (6.55) implies $\iota(j) = d_j = -\mu_j$. Hence, by the definition (4.18) of $\beta_j(\rho)$,

$$\beta_j(\rho) = \min\left[p_j, \max\left\{0, p_j - d_j - \rho\right\}\right]. \tag{6.58}$$

Two cases will now be considered based on the value of d_j.

- If $d_j \leq -\rho$, due to the integrality of d_j, $d_j \leq \varrho$. Consequently, (6.57) implies

  $$\emptyset \subseteq \{i : i \in P_j \wedge i > \varrho\} \subseteq \{i : i \in P_j \wedge i > d_j\} = \emptyset$$

  and hence

  $$\{i : i \in P_j \wedge i > \varrho\} = \emptyset.$$

  Additionally, since $d_j \leq -\rho$,

  $$p_j - d_j - \rho \geq p_j > 0.$$

  As a consequence of both of these facts, along with (6.58),

  $$\begin{aligned}\beta_j(\rho) &= \min\left[p_j, p_j - d_j - \rho\right] = p_j = |P_j| \\ &= \sum_{i:i\in P_j \wedge i \leq \varrho} 1 = \sum_{i:i\in P_j \wedge i \leq \varrho} \delta(i) \\ &= \sum_{i:i\in P_j \wedge i \leq \varrho} \delta(i) + \sum_{i:i\in P_j \wedge i > \varrho} \delta(i) = \sum_{i \in P_j} \delta(i).\end{aligned}$$

- On the other hand, consider the case where $d_j > -\rho$. It follows from this that $d_j \geq \varrho$ and

  $$p_j - d_j - \rho < p_j.$$

  Hence, from (6.58),

  $$\beta_j(\rho) = \max\left\{0, p_j - d_j - \rho\right\}. \tag{6.59}$$

  Two sub-cases are considered below, also based on the value of d_j.

  - If $d_j > p_j - \rho$, it follows that

    $$p_j - d_j - \rho < 0,$$

    and thus (6.59) implies $\beta_j(\rho) = 0$. Additionally, from $d_j > p_j - \rho$ and the

integrality of $d_j - p_j$, it follows that

$$d_j - p_j \geq \varrho + 1.$$

As a consequence of this and (6.57),

$$\emptyset \subseteq \{i : i \in P_j \wedge i \leq \varrho + 1\} \subseteq \{i : i \in P_j \wedge i \leq d_j - p_j\} = \emptyset$$

and hence

$$\{i : i \in P_j \wedge i \leq \varrho + 1\} = \emptyset.$$

It follows that

$$\beta_j(\rho) = 0 = \sum_{i:i\in P_j \wedge i \leq \varrho+1} \delta(i) + \sum_{i:i\in P_j \wedge i > \varrho+1} \delta(i) = \sum_{i\in P_j} \delta(i).$$

– Finally, if $p_j - \rho \geq d_j > -\rho$, it follows from (6.59) that

$$\beta_j(\rho) = p_j - d_j - \rho.$$

Additionally, $d_j > -\rho$ and the integrality of d_j and $d_j - p_j$ imply

$$d_j - p_j < d_j - p_j + 1 \leq \varrho + 1 \leq d_j.$$

Consequently, (6.57) implies

$$\{i : i \in P_j \wedge i \leq \varrho + 1\} = \{i : i \in H \wedge d_j - p_j < i \leq d_j \wedge i \leq \varrho + 1\}$$

$$= \{i : i \in H \wedge d_j - p_j < i \leq \varrho + 1\} \neq \emptyset.$$

Hence, $j \in Z_{\varrho+1}$ and

$$\sum_{i:i\in P_j \wedge i \leq \varrho} \delta(i) = \sum_{i:i\in P_j \wedge i \leq \varrho} 1 = p_j - d_j + \varrho.$$

As a consequence of (6.59) and the above reasoning,

$$\beta_j(\rho) = p_j - d_j - \rho = p_j - d_j + \varrho + (-\rho - \varrho)$$

$$= \sum_{i:i\in P_j \wedge i \leq \varrho} \delta(i) + \delta(\varrho + 1) + \sum_{i:i\in P_j \wedge i > \varrho+1} \delta(i) = \sum_{i\in P_j} \delta(i).$$

Thus the lemma is proven for all values of d_j. ■

Corollary 6.88 *For any $j \in X \cup Z$ and any ρ such that $0 \leq -\rho \leq \bar{h}$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$,*

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) = \sum_{i=\iota(j)+1}^{\varrho} |K_j \cap Z_i| - (\rho + \varrho)|K_j \cap Z_{\varrho+1}|.$$

Proof. Since all $g \in K_j \cap Z$ have $\iota(g) > \iota(j)$, when the first inductive assumption (6.55) holds for $\hat{\imath} = \iota(j) + 1$, Lemma 6.87 applies for all tasks $g \in K_j \cap Z$. Hence,

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) = \sum_{i \in H} \delta(i)|K_j \cap Z_i| = \sum_{i=\iota(j)+1}^{\varrho} |K_j \cap Z_i| + (-\rho - \varrho)|K_j \cap Z_{\varrho+1}|$$

and the corollary is proven. ■

Note that, by the definition of ϱ, it follows that $-1 < \rho + \varrho \leq 0$. This is important to keep in mind when examining the following lemmas.

Lemma 6.89 *For any $j \in Z$ such that $\iota(j) \geq \underline{h} - 1$, and any ρ such that $\iota(j) \leq -\rho \leq \bar{h}$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$,*

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \leq -m(\rho + \iota(j)).$$

Proof. Recall that, by Lemma 6.32 and Lemma 6.42, any $j \in Z$ such that $\iota(j) \geq \underline{h} - 1$ has $(K_j \cap Z_{\iota(j)+1}) \subset Z_{\iota(j)+1}$, i.e. no task $j \in Z \setminus Z_{\bar{h}}$ will precede all tasks in the next set $Z_{\iota(j)+1}$,* implying

$$|K_j \cap Z_{\iota(j)+1}| < |Z_{\iota(j)+1}|. \tag{6.60}$$

Five cases are considered below based on the values of $\iota(j)$ and ρ. When examining these cases, n

- If $\iota(j) = -\rho = \bar{h}$ then $\iota(j) = \varrho = \bar{h}$ and therefore Corollary 6.88 implies

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) = -(\rho + \varrho)|K_j \cap Z_{\bar{h}+1}| = 0 = -m(\rho + \iota(j)).$$

- On the other hand, if $\iota(j)+1 = -\rho = \bar{h}$ then $\iota(j) = \varrho - 1 = \bar{h} - 1$ and therefore

*Recall that $Z_{\bar{h}+1} = \emptyset$. This definition reduced the number of special cases that must be considered.

(6.60) and Corollary 6.88 imply

$$\sum_{g\in K_j\cap Z}\beta_g(\rho)=|K_j\cap Z_{\iota(j)+1}|=|K_j\cap Z_{\bar h}|\le 3=-m(\rho+\iota(j)).$$

- Alternatively, if $\iota(j)+1<-\rho=\bar h$ then $\iota(j)+2\le\varrho=\bar h$ and therefore (6.60) and Corollary 6.88 imply

$$\begin{aligned}\sum_{g\in K_j\cap Z}\beta_g(\rho)&\le|K_j\cap Z_{\iota(j)+1}|+\sum_{i=\iota(j)+2}^{\bar h-1}|Z_i|+|Z_{\bar h}|\\&\le 2+m\left(\bar h-\iota(j)-2\right)+4=-m(\rho+\iota(j)).\end{aligned}$$

- Additionally, if $\iota(j)+1\le-\rho<\bar h$ then $\iota(j)<\varrho\le\bar h-1$ and therefore (6.60) and Corollary 6.88 imply

$$\begin{aligned}\sum_{g\in K_j\cap Z}\beta_g(\rho)&\le|K_j\cap Z_{\iota(j)+1}|+\sum_{i=\iota(j)+2}^{\varrho}|Z_i|-(\rho+\varrho)|Z_{\varrho+1}|\\&\le 2+m\left(\varrho-\iota(j)-1\right)-4(\rho+\varrho)\\&=(2-\rho-\varrho)+m\left(\varrho-\iota(j)-1\right)-m(\rho+\varrho)\\&=(2-\rho-\varrho)+m\left(-\rho-\iota(j)-1\right)\\&<m+m\left(-\rho-\iota(j)-1\right)=-m(\rho+\iota(j)).\end{aligned}$$

- Finally, if $\iota(j)\le-\rho<\iota(j)+1\le\bar h$ then $\iota(j)=\varrho$, and therefore (6.60) and Corollary 6.88 imply

$$\sum_{g\in K_j\cap Z}\beta_g(\rho)=-(\rho+\varrho)(|Z_{\varrho+1}|-1)\le 3(-\rho-\varrho)=-m(\rho+\iota(j)).$$

Hence, in all five cases the lemma is proven. ■

Corollary 6.90 *For any $j\in Z$ such that $\iota(j)\ge\underline h-1$, and any ρ such that $\iota(j)\le-\rho\le\bar h$, if (6.55) holds for $\hat\imath=\iota(j)+1$,*

$$\sum_{g\in K_j}\beta_g(\rho)\le-m(\rho+\iota(j)).$$

Proof. This follows from the preceding lemma and Lemma 6.86. ■

Lemma 6.91 *For any $j \in Z$ such that $\iota(j) < \underline{h} - 1$, and any ρ such that $\iota(j) + \frac{1}{2} < -\rho \leq \bar{h}$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$,*

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) < -m(\rho + \iota(j)) - \frac{1}{2}.$$

Proof. If (6.55) holds for $\hat{\imath} = \iota(j) + 1$, Corollary 6.88 applies. If $\iota(j) < \underline{h} - 1$ then the construction of $K_0, K_1, \ldots, K_{\underline{h}-2}$ in the problem instance φ implies

$$|K_j \cap Z_{\iota(j)+1}| = |Z_{\iota(j)+1}| < m \qquad \text{and} \qquad |K_j \cap Z_{\iota(j)+2}| = |Z_{\iota(j)+2}| < m. \tag{6.61}$$

This will be used in the four cases considered below.

- If $\rho = -\bar{h}$ then $\iota(j) + 3 \leq \varrho = -\bar{h}$ and therefore (6.61) and Corollary 6.88 imply

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \leq |Z_{\iota(j)+1}| + |Z_{\iota(j)+2}| + \sum_{i=\iota(j)+3}^{\bar{h}-1} |Z_i| + |Z_{\bar{h}}|$$

$$\leq 2 + 2 + m(\bar{h} - \iota(j) - 3) + 4 = -m(\rho + \iota(j)) - 1 < -m(\rho + \iota(j)) - \frac{1}{2}.$$

- On the other hand, if $\iota(j) + 2 \leq -\rho < \bar{h}$, then $\iota(j) \leq \varrho - 2$ and therefore (6.61) and Corollary 6.88 imply

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \leq |Z_{\iota(j)+1}| + |Z_{\iota(j)+2}| + \sum_{i=\iota(j)+3}^{\varrho} |Z_i| - (\rho + \varrho)|Z_{\varrho+1}|$$

$$\leq 2 + 2 + m(\varrho - \iota(j) - 2) - 4(\rho + \varrho)$$

$$= 1 - \rho - \varrho - m(\rho + \iota(j) + 1) < -m(\rho + \iota(j)) - \frac{1}{2}.$$

- Alternatively, if $\iota(j) + 1 \leq -\rho < \iota(j) + 2$, then $\iota(j) = \varrho - 1$ and therefore (6.61) and Corollary 6.88 imply

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \leq |Z_{\iota(j)+1}| - (\rho + \varrho)|Z_{\iota(j)+2}|$$

$$\leq 2 - 2(\rho + \varrho) = -2(\rho + \iota(j)) < -m(\rho + \iota(j)) - \frac{1}{2}.$$

- Finally, if $\iota(j) + \frac{1}{2} < -\rho < \iota(j) + 1$ then $\iota(j) = \varrho$ and therefore (6.61) and

Corollary 6.88 imply

$$\sum_{g\in K_j\cap Z}\beta_g(\rho)\le -(\rho+\varrho)|Z_{\iota(j)+1}|\le -2(\rho+\iota(j))$$

$$\le -3(\rho+\iota(j))+\rho+\iota(j)<-m(\rho+\iota(j))-\frac{1}{2}.$$

Hence, in all four cases the lemma is proven. ■

Observe that both inductive assumptions are used in the following lemma.

Lemma 6.92 *For any $j\in Z$ such that $\iota(j)<\underline{h}-1$, and any ρ such that $\iota(j)\le -\rho\le\bar{h}$, if (6.55) holds for $\hat{\imath}=\iota(j)+1$ and (6.56) holds for $\hat{\imath}=\iota(j)$,*

$$\sum_{g\in K_j}\beta_g(\rho)<-m(\rho+\iota(j)).$$

Proof. In the case where $\iota(j)+\frac{1}{2}\le -\rho\le\bar{h}$ Lemma 6.91 applies due to the first inductive assumption (6.55). Hence, for a small enough value of $\varepsilon>0$,

$$\sum_{g\in K_j}\beta_g(\rho)=\sum_{g\in K_j\cap Z}\beta_g(\rho)+\sum_{g\in K_j\setminus Z}\beta_g(\rho)<-m(\rho+\iota(j))-\frac{1}{2}+\sum_{g\in K_j\setminus Z}p_g$$

$$<n\varepsilon-m(\rho+\iota(j))-\frac{1}{2}<-m(\rho+\iota(j)).$$

On the other hand, if $\iota(j)\le -\rho<\iota(j)+\frac{1}{2}$, $\iota(j)=\varrho$. In this case, the second inductive assumption (6.56) implies that, for all $g\in K_j\cap X$, and for a small enough $\varepsilon>0$,

$$\beta_g(\rho)=\min\left[p_g,\max\{0,p_g-\rho+\mu_g\}\right]$$

$$\le\min\left[p_g,\max\left\{0,p_g+\left(\iota(j)+\frac{1}{2}\right)-\left(\iota(g)+\frac{2}{3}\right)\right\}\right]$$

$$=\min\left[p_g,\max\left\{0,p_g+\iota(j)-\iota(g)-\frac{1}{6}\right\}\right]$$

$$\le\min\left[p_g,\max\left\{0,p_g-\frac{1}{6}\right\}\right]=\min\left[p_g,0\right]=0.$$

This, combined with Corollary 6.82, implies that $\beta_g(\rho)=0$ for all $g\in K_j\setminus Z$. Hence, Corollary 6.88 implies

$$\sum_{g\in K_j}\beta_g(\rho)=\sum_{g\in K_j\cap Z}\beta_g(\rho)\le -(\rho+\varrho)|Z_{\varrho+1}|\le -2(\rho+\varrho)$$

$$= -2(\rho + \iota(j)) < -m(\rho + \iota(j))$$

and the lemma is proven in this case as well. ■

Corollary 6.93 *For any $j \in Z$, and any ρ such that $\iota(j) \leq -\rho \leq \bar{h}$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$ and (6.56) holds for $\hat{\imath} = \iota(j)$,*

$$\sum_{g \in K_j} \beta_g(\rho) \leq -m(\rho + \iota(j)).$$

Proof. This follows from Corollary 6.90 and Lemma 6.92. ■

Lemma 6.94 *For any $j \in Z$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$ and (6.56) holds for $\hat{\imath} = \iota(j)$,*

$$\mu_j \leq \max\left\{-d_j, \max_{-\bar{h}-1 \leq \rho \leq -\iota(j)} \left(\rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho)\right)\right\}.$$

Proof. Recall that in the definition (4.19) of the μ's produced by the Garey-Johnson Algorithm, the variable ρ in the inner maximum is constrained by a_j and b_j, which are defined in (4.17). The two inductive assumptions, along with (4.19), imply

$$a_j = \min_{g \in K_j} \mu_g \geq -\max_{g \in K_j} d_g \geq -\bar{h} - 1.$$

Additionally, observe that Lemma 6.81 and the second inductive assumption (6.56) imply that, for all $j \in K_j \setminus Z$,

$$\mu_g < -\iota(g) - \frac{2}{3}.$$

As a consequence of this and the first inductive assumption (6.55),

$$\begin{aligned} b_j = \max_{g \in K_j}(p_g + \mu_g) &= \max\left\{\max_{g \in K_j \cap Z}(p_g + \mu_g), \max_{g \in K_j \setminus Z}(p_g + \mu_g)\right\} \\ &\leq \max\left\{\max_{g \in K_j \cap Z}(p_g - \iota(g)), \max_{g \in K_j \setminus Z}\left(p_g - \iota(g) - \frac{2}{3}\right)\right\}. \end{aligned} \tag{6.62}$$

For any $g \in K_j \cap Z$, (6.57) and the first inductive assumption (6.55) imply

$$p_g - \iota(g) = p_g - d_g = |\{i : i \in H \wedge g \in Z_i\}| - \max\{i : i \in H \wedge g \in Z_i\}$$

$$= -\max\{i : i \in H \wedge g \notin Z_i\} \leq -\max\{i : i \in H \wedge j \in Z_i\} = -\iota(j).$$

For any $g \in K_j \setminus Z$ and adequately small $\varepsilon > 0$,

$$\varepsilon - \iota(g) - \frac{2}{3} \leq \varepsilon - \iota(j) - \frac{2}{3} < -\iota(j).$$

Combining the two relations above with (6.62) gives

$$-\bar{h} - 1 \leq a_j < b_j \leq -\iota(j).$$

The lemma then follows from this and the definition (4.19) of the μ's produced by the Garey-Johnson Algorithm. ■

Notice that the range of ρ in the statement of the preceding lemma is somewhat larger than that covered by Corollary 6.93: in particular the case where $\rho \in \left[-\bar{h} - 1, -\bar{h}\right)$ must be considered separately. The next lemma facilitates the consideration of the case where $\rho < -\bar{h}$ using Corollary 6.93.

Lemma 6.95 *For any $j \in X \cup Z$, and any $\rho < -\bar{h}$,*

$$\rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) < \rho + \frac{1}{m} \sum_{g \in K_j \cap Z} \beta_g\left(-\bar{h}\right) + \frac{n}{m}\varepsilon.$$

Proof. For all $g \in K_j$, the definition (4.18) of $\beta_g(\rho)$ implies

$$\beta_g(\rho) = \min[p_g, \max\{0, p_g - \rho + \mu_g\}] \leq p_g.$$

Consequently, for all $g \in K_j \setminus Z$ it follows that $\beta_g(\rho) \leq p_g = \varepsilon$, and for all $g \in K_j \cap Z$, (4.18) and (4.19) imply

$$\beta_g(\rho) \leq p_g = \min\left[p_g, \max\left\{0, p_g + \bar{h} - d_g\right\}\right]$$

$$\leq \min\left[p_g, \max\left\{0, p_g + \bar{h} + \mu_g\right\}\right] = \beta_g\left(-\bar{h}\right).$$

From this it can be seen that

$$\sum_{g \in K_j} \beta_g(\rho) = \sum_{g \in K_j \cap Z} \beta_g(\rho) + \sum_{g \in K_j \setminus Z} \beta_g(\rho) < \sum_{g \in K_j \cap Z} \beta_g\left(-\bar{h}\right) + n\varepsilon$$

and the lemma follows. ■

Lemma 6.96 *For any $j \in Z$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$ and (6.56) holds for*

$\hat{\imath} = \iota(j)$,

$$\mu_j \leq \max\left\{-d_j, \max_{-\bar{h}-\frac{1}{2}<\rho\leq-\iota(j)}\left(\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho)\right)\right\}.$$

Proof. If $-\bar{h}-1 \leq \rho \leq -\bar{h}-\frac{1}{2}$, Corollary 6.93 and Lemma 6.95 imply that, for a large enough value of $\varepsilon > 0$,

$$\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho) < -\bar{h} - \frac{1}{2} + \frac{1}{m}\sum_{g\in K_j\cap Z}\beta_g\left(-\bar{h}\right) + \frac{n}{m}\varepsilon$$

$$\leq -\bar{h} - \frac{1}{2} - \left(-\bar{h} + \iota(j)\right) + \frac{n}{m}\varepsilon = \frac{n}{m}\varepsilon - \iota(j) - \frac{1}{2} < -\iota(j) = -d_j.$$

Consequently,

$$\max\left\{-d_j, \max_{-\bar{h}-1\leq\rho\leq-\iota(j)}\left(\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho)\right)\right\}$$

$$= \max\left\{-d_j, \max_{-\bar{h}-\frac{1}{2}<\rho\leq-\iota(j)}\left(\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho)\right)\right\}$$

and the lemma follows from Lemma 6.94. ■

Lemma 6.97 *For all $j \in X \cup Z$, if $\rho \leq -\bar{h}$, $\beta_j(\rho) = p_j$.*

Proof. From the definition (4.19) of the μ's produced by the Garey-Johnson Algorithm, if $j \in Z$,

$$\mu_j \geq -d_j \geq -\bar{h} \geq \rho.$$

On the other hand, if $j \in X$, (4.15) and the definition of X imply

$$\mu_j \geq \max_{g\in K_j}\mu_g \geq -\min_{g\in K_j} d_g \geq -\bar{h} \geq \rho.$$

Thus, in either case, the definition (4.18) of $\beta_j(\rho)$ implies

$$\beta_j(\rho) = \min\left[p_j, \max\{0, p_j - \rho + \mu_j\}\right] = \min\left[p_j, p_j - \rho + \mu_g\right] = p_j,$$

and the lemma holds. ■

Lemma 6.98 *For any $j \in Z$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$ and (6.56) holds for*

$\hat{\imath} = \iota(j)$,

$$\mu_j \leq \max\left\{-d_j, \max_{-\bar{h}\leq\rho\leq-\iota(j)}\left(\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho)\right)\right\}.$$

Proof. If $-\bar{h} - \frac{1}{2} < \rho < -\bar{h}$, Corollary 6.82 and Lemma 6.97 imply

$$\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho) = \rho + \frac{1}{m}\sum_{g\in K_j\cap(X\cup Z)} p_g$$

$$< -\bar{h} + \frac{1}{m}\sum_{g\in K_j\cap(X\cup Z)} p_g = -\bar{h} + \frac{1}{m}\sum_{g\in K_j\cap(X\cup Z)}\beta_g\left(-\bar{h}\right).$$

Consequently,

$$\max\left\{-d_j, \max_{-\bar{h}-\frac{1}{2}<\rho\leq-\iota(j)}\left(\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho)\right)\right\}$$

$$= \max\left\{-d_j, \max_{-\bar{h}\leq\rho\leq-\iota(j)}\left(\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho)\right)\right\}$$

and the lemma follows from Lemma 6.96. ■

Lemma 6.99 *For any $j \in Z$, if (6.55) holds for $\hat{\imath} = \iota(j) + 1$ and (6.56) holds for $\hat{\imath} = \iota(j)$, $\mu_j = -\iota(j)$.*

Proof. Mindful of the bound in the statement of Lemma 6.98, consider any ρ such that $-\bar{h} \leq \rho \leq -\iota(j)$. From Corollary 6.93,

$$\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho) \leq \rho - (\rho + \iota(j)) = -\iota(j).$$

Consequently, Lemma 6.98 and the fact that $d_j = \iota(j)$ imply

$$-\iota(j) \leq \mu_j \leq \max\left\{-\iota(j), \max_{-\bar{h}\leq\rho\leq-\iota(j)}\left(\rho + \frac{1}{m}\sum_{g\in K_j}\beta_g(\rho)\right)\right\} = -\iota(j)$$

and the lemma follows. ■

The preceding lemma demonstrates that, for some $k \geq 0$, if (6.55) holds for $\hat{\imath} = k + 1$ and (6.56) holds for $\hat{\imath} = k$, the first inductive assumption (6.55) holds for $\hat{\imath} = k$. The proof of this required the second inductive assumption (6.56), so it remains to be proven that this assumption holds for $\hat{\imath} = k - 1$.

Lemma 6.100 *For any $j \in X$ and any ρ such that $\iota(j)+1 \le -\rho \le \bar{h}$, if (6.55) and (6.56) hold for $\hat{\imath} = \iota(j)+1$,*

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \le -m(\rho + \iota(j) + 1).$$

Proof. Since (6.55) holds for $\hat{\imath} = \iota(j)+1$, Corollary 6.88 applies to all tasks $g \in K_j \cap Z$ with $\iota(g) \ge \iota(j)+1$. Additionally, by Lemma 6.85, $K_j \cap Z_{\iota(j)+1} = \emptyset$. Hence,

$$|K_j \cap Z_{\iota(j)+1}| = 0 \qquad \text{and} \qquad |K_j \cap Z_{\iota(j)+2}| \le |Z_{\iota(j)+2}| < m. \tag{6.63}$$

Since (6.63) implies (6.61) which was used in the proof of Lemma 6.91, and since Corollary 6.88 applies to all tasks $g \in K_j \cap Z$ as in the proof of Lemma 6.91, this lemma may be proved in a similar manner, by considering the three cases below.

- If $\rho = -\bar{h}$ then $\iota(j)+3 \le \varrho = -\bar{h}$ and therefore (6.63) and Corollary 6.88 imply

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \le |Z_{\iota(j)+2}| + \sum_{i=\iota(j)+3}^{\bar{h}-1} |Z_i| + |Z_{\bar{h}}|$$

$$\le 2 + m(\bar{h} - \iota(j) - 3) + 4 = -m(\rho + \iota(j) + 1).$$

- On the other hand, if $\iota(j)+2 \le -\rho < \bar{h}$, then $\iota(j) \le \varrho - 2$ and therefore (6.61) and Corollary 6.88 imply

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \le |Z_{\iota(j)+2}| + \sum_{i=\iota(j)+3}^{\varrho} |Z_i| - (\rho + \varrho)|Z_{\varrho+1}|$$

$$\le 2 + m(\varrho - \iota(j) - 2) - 4(\rho + \varrho)$$

$$= 2 - \rho - \varrho - m(\rho + \iota(j) + 2) < -m(\rho + \iota(j) + 1).$$

- Finally, if $\iota(j)+1 \le -\rho < \iota(j)+2$, then $\iota(j) = \varrho - 1$ and therefore (6.61) and Corollary 6.88 imply

$$\sum_{g \in K_j \cap Z} \beta_g(\rho) \le -(\rho + \varrho)|Z_{\iota(j)+2}|$$

$$\le -2(\rho + \varrho) = -2(\rho + \iota(j) + 1) < -m(\rho + \iota(j) + 1).$$

Hence, in all three cases the lemma is proven. ■

Lemma 6.101 *For any $j \in X$ and any ρ such that $\iota(j)+1 \le -\rho \le \bar{h}+1$, if (6.55) and (6.56) hold for $\hat{\imath} = \iota(j)+1$,*

$$\rho + \frac{1}{m}\sum_{g\in K_j} \beta_g(\rho) < -\iota(j) - \frac{3}{4}.$$

Proof. If $-\bar{h}-1 \le \rho < -\bar{h}$, for a small enough value of $\varepsilon > 0$,

$$\rho + \frac{1}{m}\sum_{g\in K_j} \beta_g(\rho) \le \rho + \frac{1}{m}\sum_{g\in K_j} p_g = \rho + \frac{1}{m}\left(\sum_{g\in K_j\cup Z} p_g + \sum_{g\in K_j\setminus Z} p_g\right)$$

$$< -\bar{h} + \frac{1}{m}\left(\sum_{i=\iota(j)+2}^{\bar{h}} |Z_i| + n\varepsilon\right)$$

$$= -\bar{h} + \frac{1}{m}\left(|Z_{\iota(j)+2}| + \sum_{i=\iota(j)+3}^{\bar{h}-1} |Z_i| + |Z_{\bar{h}}| + n\varepsilon\right)$$

$$\le -\bar{h} + \frac{1}{m}\big(2 + m(\bar{h} - \iota(j) - 3) + 4 + n\varepsilon\big) = \frac{n}{m}\varepsilon - \iota(j) - 1 < -\iota(j) - \frac{3}{4}.$$

On the other hand, if $-\bar{h} \le \rho \le -\iota(j) - 1$, Lemma 6.100 implies, for a small enough value of $\varepsilon > 0$,

$$\rho + \frac{1}{m}\sum_{g\in K_j} \beta_g(\rho) = \rho + \frac{1}{m}\left(\sum_{g\in K_j\cup Z} \beta_g(\rho) + \sum_{g\in K_j\setminus Z} \beta_g(\rho)\right)$$

$$\le \rho - (\rho + \iota + 1) + \frac{n}{m}\varepsilon < -\iota(j) - \frac{3}{4}.$$

Hence, the lemma holds in both cases. ∎

Lemma 6.102 *For any $j \in X$ for which all $g \in K_j \cap X$ have $\iota(g) > \iota(j)$, and any ρ such that $\iota(j)+1 \le -\rho \le \bar{h}+1$, if (6.55) and (6.56) hold for $\hat{\imath} = \iota(j)+1$, $\mu_j \le -\iota(j) - \frac{3}{4}$.*

Proof. By the definition (4.17) of a_j and b_j,

$$a_j = \min_{g\in K_j} \mu_g \ge -\max_{g\in K_j} d_g \ge -\bar{h} - 1.$$

Furthermore, by Lemma 6.85 and the inductive assumptions, for an adequately small

value of $\varepsilon > 0$,

$$b_j = \max_{g \in K_j} (p_g + \mu_g) = \max \left\{ \max_{g \in K_j \cap Z} (p_g + \mu_g), \max_{g \in K_j \setminus Z} (p_g + \mu_g) \right\}$$

$$\leq \max \left\{ \max_{g \in K_j \cap Z} ((\iota(g) - \iota(j) - 1) - \iota(g)), \max_{g \in K_j \setminus Z} \left(\varepsilon - \iota(g) - \frac{2}{3} \right) \right\}$$

$$\leq \max \left\{ -\iota(j) - 1, \varepsilon - \iota(j) - 1 - \frac{2}{3} \right\} = -\iota(j) - 1.$$

Consequently, by the definition (4.19) of the μ's produced by the Garey-Johnson Algorithm, and by Lemma 6.101,

$$\mu_j \leq \max \left\{ -\bar{h} - 1, \max_{-\bar{h}-1 \leq \rho \leq -\iota(j)-1} \left(\rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) \right) \right\}$$

$$\leq \max \left\{ -\bar{h} - 1, -\iota(j) - \frac{3}{4} \right\} \leq -\iota(j) - \frac{3}{4}$$

and the lemma holds. ■

For each task $j \in X$, define

$$W_j = \{g : g \in K_j \cap X \wedge \iota(g) = \iota(j)\}.$$

Lemma 6.103 *For any $j \in X$, if (6.55) and (6.56) hold for $\hat{\imath} = \iota(j) + 1$, $\mu_j < -\iota(j) - \frac{2}{3}$.*

Proof. This lemma will be proven in similar manner to Lemma 6.81, specifically by induction. First consider a task $j \in X$ such that all $g \in K_j \cap X$ have $\iota(g) > \iota(j)$, i.e. $W_j = \emptyset$. By Lemma 6.102, $\mu_j \leq -\iota(j) - \frac{3}{4}$.

Now consider any $j \in X$ such that there exists a task $g \in K_j \cap X$ for which $\iota(g) = \iota(j)$. For all $g \in K_j$ assume that

$$\mu_g \leq \varepsilon |W_g| - \iota(j) - \frac{3}{4}. \tag{6.64}$$

Note that (6.64) holds for all $g \in X$ such that all $W_g = \emptyset$.

Now consider a task $j \in X$ such that $W_j \neq \emptyset$. Given the inductive assumption (6.64), (4.17) and the fact that $|W_g| < |W_j|$ for all $g \in W_j$ imply

$$-\bar{h} - 1 \leq \min_{g \in K_j} \mu_j = a_j < b_j = \max_{g \in K_j} (p_g + \mu_g)$$

$$= \max_{g \in W_j} (p_g + \mu_g) \leq \varepsilon |W_j| - \iota(j) - \frac{3}{4}.$$

If $-\iota(j) - 1 < \rho \leq b_j$, it follows from the bound on b_j above that $\iota(j) = \varrho$. From this, Lemma 6.85 and Corollary 6.88 it follows that

$$\rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) = \rho \leq b_j.$$

If, on the other hand, $-\bar{h} - 1 \leq \rho \leq -\iota(j) - 1$, Lemma 6.101 states that

$$\rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) < -\iota(j) - \frac{3}{4}.$$

Thus, for all $a_j < \rho \leq b_j$

$$\rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) \leq \varepsilon |W_j| - \iota(j) - \frac{3}{4}.$$

Hence, for a small enough value of $\varepsilon > 0$,

$$\mu_j = \max \left\{ -d_j, \max_{a_j \leq \rho \leq b_j} \left(\rho + \frac{1}{m} \sum_{g \in K_j} \beta_g(\rho) \right) \right\}$$

$$\leq \max \left\{ -\bar{h} - 1, \varepsilon |W_j| - \iota(j) - \frac{3}{4} \right\}$$

$$= \varepsilon |W_j| - \iota(j) - \frac{3}{4}.$$

Consequently, (6.54) holds for all $g \in N \setminus Z$ such that $K_g \cap Z = \emptyset$, and thus the lemma holds for adequately small $\varepsilon > 0$. ■

Lemma 6.104 *When the Garey-Johnson Algorithm is applied to the problem instance φ, $\mu_j = -d_j$ for all $j \in Z$.*

Proof. The lemma will be proved by induction. First note that, as stated earlier, (6.55) and (6.56) hold for $\hat{\imath} = \bar{h}$. Consider the case where, for some $k \leq \bar{h}$, (6.55) and (6.56) hold for $\hat{\imath} = k$. In this case, Lemma 6.103 demonstrates that (6.56) holds for $\hat{\imath} = k - 1$.

Now consider the case where, for some $k \leq \bar{h}$, (6.55) holds for $\hat{\imath} = k$ and (6.56) holds for $\hat{\imath} = k - 1$. In this case, Lemma 6.99 demonstrates that (6.55) holds for $\hat{\imath} = k - 1$. Hence, the lemma is proven by induction. ■

Theorem 6.105 *For any partial order that does not contain a pseudo-singular quartet there exists an assignment of m, p_j and d_j such that the Garey-Johnson Algorithm produces a non-optimal schedule.*

Proof. Consider the schedule σ defined as follows:

- all tasks in $N \setminus (Q \cup K(Q))$ are processed to completion during $[0, n\varepsilon]$;
- one unit of each task in $Q \setminus \{k_1\}$, and $\frac{4}{3}$ units of task k_1, are processed during

$$\left[n\varepsilon, n\varepsilon + \frac{13}{9}\right],$$

  completing all tasks in $Q \setminus \{k_1\}$;
- all tasks in $K(Q) \setminus K_{k_1}$, as well as task k_1, are processed to completion during

$$\left[n\varepsilon + \frac{13}{9}, n\varepsilon + \frac{19}{9}\right],$$

  which is possible for adequately small ε;
- for all i such that $1 < i < \underline{h} - 1$, all tasks in $K_{k_{i-1}} \setminus K_{k_i}$, including task k_i, are processed to completion during

$$\left[n\varepsilon + i + \frac{1}{9}, n\varepsilon + i + \frac{10}{9}\right],$$

  which again is possible for adequately small ε;
- each task in $B = Z_{\underline{h}-1} = Z_{\underline{h}}$ is processed for two units during

$$\left[n\varepsilon + \underline{h} - 1 + \frac{1}{9}, n\varepsilon + \underline{h} + \frac{10}{9}\right],$$

  and all tasks in $K_{\underline{h}-2} \setminus K(B)$ are processed to completion during this interval, which is possible for adequately small ε because $|B| = 2 < m$;
- as a consequence of the fact that I is a transitive set of N, it follows that $K_j \cap Z = \emptyset$ for all $j \in K(Z_{\underline{h}}) \setminus Z$, and since I is canonical it follows that for all i such that $\underline{h} < i < \bar{h}$, $|Z_i| \leq 3 = m$ and consequently each task in Z_i is processed for one unit during

$$\left[n\varepsilon + i + \frac{1}{9}, n\varepsilon + i + \frac{10}{9}\right];$$

- since $|Z_{\bar{h}}| = 4$, all tasks in $Z_{\bar{h}}$ are processed for one unit during

$$\left[n\varepsilon + \bar{h} + \frac{1}{9}, n\varepsilon + \bar{h} + \frac{13}{9}\right],$$

  completing them all;

- finally, all tasks in $K(B) \setminus Z$ are processed to completion during

$$\left[n\varepsilon + \bar{h} + \frac{13}{9}, 2n\varepsilon + \bar{h} + \frac{13}{9}\right].$$

The maximum lateness of σ may be calculated as follows:

- for a large enough value of δ, all tasks $j \in N \setminus Z$ have $C_j(\sigma) - d_j \leq n\varepsilon + \frac{13}{9}$;
- all tasks $j \in Q \setminus \{k_1\}$ have $C_j(\sigma) - d_j = C_j(\sigma) \leq n\varepsilon + \frac{13}{9}$;
- for all i such that $1 \leq i < \bar{h}$ and all $j \in Z_i \setminus Z_{i+1}$,

$$C_j(\sigma) - d_j = n\varepsilon + i + \frac{10}{9} - i = n\varepsilon + \frac{10}{9};$$

- and finally, for all tasks $j \in Z_{\bar{h}}$,

$$C_j(\sigma) - d_j \leq n\varepsilon + \bar{h} + \frac{13}{9} - \bar{h} = n\varepsilon + \frac{13}{9},$$

  and this is an equality for three such tasks.

Since the maximum lateness of σ provides an upper bound on the optimal maximum lateness,

$$L_{max}(\omega) \leq L_{max}(\sigma) = n\varepsilon + \frac{13}{9}. \tag{6.65}$$

On the other hand, consider the schedule $\eta^{\mathscr{GJ}}$ produced by the Garey-Johnson Algorithm. Lemma 6.104 demonstrates that $\mu_j = -d_j$ for all $j \in Z$. Consequently (4.15) implies that $\mu_g \geq p_j + \mu_j$ for all $j \in Z$ and $g \in J(Q)$. Thus,

$$\bar{t} = \max_{j \in J(Q)} C_j\left(\eta^{\mathscr{GJ}}\right) \leq n\epsilon$$

and the tasks in Q have priority at time $\bar{t}$ differing by at most $n\varepsilon$. Lemma 4.7 then implies that, for an adequately small value of $\varepsilon > 0$, $C_{k_1}\left(\eta^{\mathscr{GJ}}\right) \geq 2\frac{1}{3}$. Taking into account Lemma 6.41, it follows that, for all i such that $1 < i < \bar{h}$ and all $j \in Z_i \setminus Z_{i+1}$

$$C_j\left(\eta^{\mathscr{GJ}}\right) \geq i + \frac{4}{3}.$$

Finally, since $|Z_{\bar{h}}| = 4$, the above reasoning implies that

$$\max_{j \in Z_{\bar{h}}} C_j \left(\eta^{\mathscr{GJ}} \right) \geq \bar{h} + \frac{5}{3}.$$

This gives

$$L_{max} \left(\eta^{\mathscr{GJ}} \right) \geq \frac{5}{3}$$

and so, for an adequately small value of ε, (6.65) contradicts the optimality of $\eta^{\mathscr{GJ}}$.

■

This concludes the analysis of the domains of approximation algorithms. The concept of an algorithm's domain is a powerful one, providing insight into the nature and strength of an approximation algorithm. It is to be hoped that the techniques developed for this chapter will be applied to other approximation algorithms in the future.

Chapter 7

An Optimal Algorithm

Although the scheduling problems considered in this thesis are known to be NP-hard, it may be desired to find an optimal solution anyway. Some small problems are easily solved in a fairly small amount of time, and others have special structure that enables a quick solution. Considered in this chapter is a linear programming method to solve the $P|prec, pmtn, u_j|L_{max}$ problem. This approach was presented at the ASOR Recent Advances 2015 [113], and was prepared in collaboration with Yakov Zinder. For much simpler, but somewhat similar, uses of a linear programs to solve scheduling problems, see [73], [100] and particularly [35].

7.1 The Linear Program

The general idea behind the method described in this section is to suppose that some oracle has granted, for some particular instance φ of the problem $P|prec, pmtn, u_j|L_{max}$, knowledge of the order in which tasks are completed in an optimal schedule ω for φ. It turns out that this knowledge is sufficient to allow the determination of an optimal schedule in polynomial time. Without loss of generality, suppose that the tasks have been renumbered, from 1 to n, in order of their completion times (with ties being broken arbitrarily), i.e.

$$C_1(\omega) \leq C_2(\omega) \leq \cdots \leq C_n(\omega).$$

In order to simplify the exposition in this chapter, let $C_0(\sigma) = 0$ for all schedules σ. Note that 0 is not a task in the considered instance of the problem $P|prec, pmtn, u_j|L_{max}$, i.e. $0 \notin N$.

Once this renumbering has taken place, sets A_j and B_j are defined for all $j \in N$ as follows:

- A_j is the set of all tasks available for processing after $C_{j-1}(\sigma)$ but before

$C_j(\sigma)$,* i.e.

$$A_j = \{g : g \in N \wedge g \geq j \wedge (\forall q \in J_g : q < j)\};$$

- and B_j is the set of all tasks g for which j is available during $[C_{g-1}(\sigma), C_g(\sigma))$, i.e.

$$B_j = \{g : g \in N \wedge j \in A_g\}.$$

The key to the process of transforming this ordering of tasks into an optimal schedule is the Modified McNaughton Algorithm, as described in Subsection 4.2.1. If the completion times are known, along with the amount

$$p_j(C_{i-1}(\omega), \omega) - p_j(C_i(\omega), \omega) \tag{7.1}$$

which each task j is processed in each interval $[C_{i-1}(\omega), C_i(\omega)]$ between two consecutive completion times, a schedule can be easily constructed by applying the McNaughton Algorithm first to the interval $[0, C_1(\omega)]$ (i.e. the interval $[C_0(\omega), C_1(\omega)]$), then to the interval $[C_1(\omega), C_2(\omega)]$, and so on, until the entire schedule has been constructed.

In order to bridge the gap between

- merely knowing the order of completion times, as supplied by the oracle, and
- knowing the values $C_j(\omega)$ of these completion times and the amount (7.1) of each task processed between them, as required to construct a schedule as described above,

a linear program, Linear Program 7.1, can be used. This linear program is similar to some existing publications, for example [53].

Analogous to the reasoning behind the notation $C_0(\sigma)$, the constant $C_0 = 0$ is defined for use in this linear program. The variables of Linear Program 7.1 are

- L_{max}, the maximum lateness,
- C_j for all $j \in N$, the completion time of task j, and
- $p_{j,i}$ for all $j \in N$ and $i \in B_j$, the amount of task j which is processed during $[C_{i-1}, C_i]$.

All variables are non-negative except for L_{max}. Note that L_{max} and C_j are variables of the linear program, while $L_{max}(\sigma)$ and $C_j(\sigma)$ are values associated with the

*For the sake of convenience of notation this is somewhat inconsistent with the usage in Chapter 4: there, A_i was the set of tasks available during $[t_i, t_{i+1})$, while here it is the set of tasks available during $[C_{i-1}(\sigma), C_i(\sigma))$.

schedule σ. Similarly, note that $p_{j,i}$ is a variable of the linear program, while p_j is a parameter of a problem instance.

$$\begin{aligned}
&\min L_{max} && && (7.2a)\\
&\text{s.t. } L_{max} \geq C_j - d_j && \forall j \in N && (7.2b)\\
&\quad C_{j-1} \leq C_j && \forall j \in N && (7.2c)\\
&\quad \sum_{j \in A_i} p_{j,i} \leq m(C_i - C_{i-1}) && \forall i \in N && (7.2d)\\
&\quad 0 \leq p_{j,i} \leq u_j(C_i - C_{i-1}) && \forall j \in N,\ \forall i \in B_j && (7.2e)\\
&\quad \sum_{i \in B_j} p_{j,i} = p_j && \forall j \in N && (7.2f)
\end{aligned}$$

Linear Program 7.1: Linear program for solving the $P|prec, pmtn, u_j|L_{max}$ problem, given an order of completion times.

It can be seen that there are $O(n^2)$ variables defined above, and a cursory examination of Linear Program 7.1 demonstrates that there are also $O(n^2)$ constraints. Consequently (see [65] and [63]) this linear program may be solved in an amount of time polynomial in the number of tasks n. It is also worth noting that this linear program is quite sparse: only (7.2e) defines $O(n^2)$ constraints, while there are $O(n)$ constraints not defined by this inequality, and each constraint defined by (7.2e) has only three variables with non-zero coefficients: $p_{j,i}$, C_i and C_{i-1}.

A simple explanation of the linear program follows:

- the objective function (7.2a) simply minimises the maximum lateness, as is the goal of the $P|prec, pmtn, u_j|L_{max}$ problem;
- (7.2b) constrains the maximum lateness to be not less than the lateness of any individual task, and since it is being minimised and there are no other constraints involving L_{max}, this leads to

  $$L_{max} = \max_{j \in N} (C_j - d_j),$$

  the traditional definition of maximum lateness;
- (7.2c) constrains the tasks to being completed in the specified order;
- the McNaughton constraints (4.5) are described by (7.2d) and (7.2e); and finally
- (7.2f) specifies the constraint that tasks are completed only after they have received enough processing.

For the purpose of demonstrating that Linear Program 7.1 does indeed produce an optimal schedule, a modified scheduling problem, here denoted $P|prec, pmtn, u_j, ord|L_{max}$, will be considered. Specifically, $P|prec, pmtn, u_j, ord|L_{max}$ is defined to be the original $P|prec, pmtn, u_j|L_{max}$ problem subject to the additional constraint that

$$C_1(\sigma) \leq C_2(\sigma) \leq \cdots \leq C_n(\sigma). \tag{7.3}$$

Let Σ^{ord} denote the set of all feasible schedules for the considered instance of the $P|prec, pmtn, u_j, ord|L_{max}$ problem, and similarly define Σ to be set of all feasible schedules for the considered instance of the $P|prec, pmtn, u_j|L_{max}$ problem. Using these definitions, the following lemma is specified, and will be used in Theorem 7.2 to justify Linear Program 7.1.

Lemma 7.1 *For each $\sigma \in \Sigma^{ord}$, there exists a feasible solution to Linear Program 7.1 such that $L_{max} = L_{max}(\sigma)$.*

Proof. Use the following process to turn any schedule $\sigma \in \Sigma^{ord}$ into a feasible solution to Linear Program 7.1:

- let $L_{max} = L_{max}(\sigma)$,
- let $C_j = C_j(\sigma)$ for all $j \in N$, and
- let $p_{j,i} = p_j(C_{i-1}(\sigma), \sigma) - p_j(C_i(\sigma), \sigma)$ for all $j \in N$ and all $i \in B_j$.

The constraints of Linear Program 7.1 will now be considered in turn:

- (7.2b) holds by the definition of maximum lateness,
- (7.2c) holds as a consequence of the additional constraint (7.3) imposed upon the $P|prec, pmtn, u_j, ord|L_{max}$ problem for which σ is a feasible schedule,
- (7.2d) holds because each machine can process at most one task at a time,
- (7.2e) holds because any task j can be processed on at most u_j machines simultaneously, and
- (7.2f) holds by the definition of completion time.

Consequently, any schedule $\sigma \in \Sigma^{ord}$ is transformed by the above process into a feasible solution for Linear Program 7.1. ■

Theorem 7.2 *An optimal solution of Linear Program 7.1 can be transformed in polynomial time to a schedule $\varpi \in \Sigma$ for which*

$$L_{max}(\varpi) \leq \min_{\sigma \in \Sigma^{ord}} L_{max}(\sigma).$$

Proof. The following procedure creates a schedule ϖ from an optimal solution to Linear Program 7.1. It is something like the reverse of the transformation used in Lemma 7.1.*

1. Set $i = 1$,
2. use the McNaughton Algorithm to schedule $p_{j,i}$ units of task j during $[C_{i-1}, C_i]$ for all j,
3. if $i = n$ terminate the procedure, otherwise increment i and go to Step 2.

The McNaughton Algorithm may be used here since (7.2d) and (7.2e) imply that (4.5) holds each time it is called.

As a consequence of (7.2f), $p_j(C_j, \varpi) = 0$ and thus $C_j(\varpi) \leq C_j$. This and (7.2b) together imply $L_{max} \geq L_{max}(\varpi)$, and hence the theorem follows from Lemma 7.1. ■

Corollary 7.3 *Given an oracle which specifies the order of completion times of some optimal schedule ω for the problem $P|prec, pmtn, u_j|L_{max}$, this problem can be solved in polynomial time.*

The above corollary suggests an algorithm for deriving an optimal schedule for any instance φ of the problem $P|prec, pmtn, u_j|L_{max}$, the specification of which requires the following definitions. A total order P on the set N is a partial order for which, given any pair of tasks $j \in N$ and $g \in N$, precisely one of the three properties below holds:

$$j \; P \; g, \qquad g \; P \; j, \qquad \text{or} \qquad j = g.$$

In other words, a total order is a partial order for which no two distinct tasks are independent of one another. For any partial order P on the set of tasks N, let $\Pi(P)$ denote the set of linear extensions of P, i.e. the set of all total orders $\prec$ such that, for all $j \in N$ and $g \in N$, $j \prec g$ if $j \; P \; g$. For any total order $\prec$, denote by $\varphi(\prec)$ the problem instance created by renumbering the tasks of φ from 1 to n according

*This procedure does not precisely reverse the transformation, as idle machines will be treated differently by McNaughton's Algorithm. In particular, (7.3) may not hold for $\sigma = \varpi$, and so it is not necessarily true that $\varpi \in \Sigma^{ord}$.

to their order in $\prec$. Given these definitions, an algorithm for finding an optimal schedule for the problem instance φ, with precedence constraints defined by the partial order $\rightarrow$, is described below.

Exhaustive Optimal Algorithm

1. for each total order $\prec \in \Pi(\rightarrow)$:

   1.1. apply Linear Program 7.1 to problem instance $\varphi(\prec)$ and determine the optimal solution,

   1.2. use the algorithm in the proof of Theorem 7.2 to construct a schedule $\varpi(\prec)$ from this optimal solution,

2. select a total order $\prec \in \Pi(\rightarrow)$ for which

$$L_{max}(\varpi(\prec)) = \min_{\prec' \in \Pi(\rightarrow)} L_{max}(\varpi(\prec'))$$

   and return the schedule $\varpi(\prec)$.

A key insight derived from the algorithm above is that the $P|prec, pmtn, u_j|L_{max}$ problem (and thus also the more frequently studied $P|prec, pmtn|L_{max}$ and $P|prec, pmtn|C_{max}$ problems) may in some sense be decomposed into two parts:

- a "continuous" part amenable to linear programming and solvable in polynomial time, and
- a "discrete" part involving a search for an optimal ordering of tasks, which is NP-hard.

For any given order of completion times, an optimal schedule may be generated in polynomial time. Unfortunately, the completion times may be ordered according to any linear extension of the precedence constraints, and in general this is not polynomial in n, see for example [18]. As is often the case, a similar result holds for the $P|prec, p_j = 1|L_{max}$ problem: if the order of completion times in an optimal schedule is known, a simple list scheduling algorithm will construct an optimal schedule in polynomial time, while discovering such a list is NP-hard in general.

7.1.1 Total Weighted Tardiness

Another well known objective function, total weighted tardiness, can be minimised using the same procedure. Suppose that, for each task $j \in N$, a weight w_j is given

as part of the problem instance. The total weighted tardiness of a schedule σ is then defined as

$$\sum_{j \in N} w_j T_j(\sigma)$$

where

$$T_j(\sigma) = \max\{0, C_j(\sigma) - d_j\}$$

is the tardiness of task j in schedule σ. Note that, as stated in the introduction, when $d_j = 0$ and $w_j = 1$ for all $j \in N$ this reduces to the total completion time

$$\sum_{j \in N} C_j(\sigma).$$

This is an even more difficult problem than the maximum lateness problem. In [40] it was demonstrated that the $P2|chains, pmtn, |\sum C_j$ problem is NP-hard – unlike the maximum lateness problem the minimisation of the total completion time is NP-hard even on two machines and with precedence constraints in the form of chains.*

Linear Program 7.1 may be adapted in a simple way in order to minimise the total weighted tardiness, i.e. in order to solve the $P|prec, pmtn, u_j|\sum w_j T_j$ problem. Specifically,

- new non-negative variables $T_j \geq 0$ for all $j \in N$ are introduced,
- the objective function (7.2a) is replaced with

$$\sum_{j \in N} w_j T_j,$$

- and the set of equations (7.2b) is replaced with

$$T_j \geq C_j - d_j \text{ for all } j \in N. \tag{7.4}$$

Since L_{max} is replaced with n new variables, and since the n constraints defined by (7.2b) are replaced by n new constraints defined by (7.4), the resulting linear program is still polynomial in n.

The proof of the validity of this linear program is substantially similar to Lemma 7.1 and Theorem 7.2. For the remainder of this chapter it will be assumed that the objective function under consideration is the maximum lateness.

*A partial order is said to be composed of chains if and only if it is both an in-forest and an out-forest.

7.2 An Improvement

Linear Program 7.1 is simple to understand, but it is also quite inefficient if many different task orderings are being searched. In order to reduce the number of cases, some tasks may be combined.

One specific curiosity is the case where there are no precedence constraints. Even though this is the simplest case, and is solved by even the Brucker-Garey-Johnson Algorithm (see Theorem 6.6 or [71]), there is an intractably large number (specifically $n!$) of linear extensions of $\rightarrow$. A simplifying observation is to note that there are only two reasons the completion time of a task j matters when constructing a schedule σ:

- it may determine the maximum lateness, i.e. $L_{max}(\sigma) = C_j(\sigma) - d_j$, and
- it determines the time at which the immediate successors of j, the tasks comprising $\tilde{K}_j$, become available.

Consequently, if $d_j = d_g$ and $K_j = K_g$, tasks j and g can be given a single, amalgamated completion time in the resulting modification of Linear Program 7.1. Furthermore, in the more general case where $d_j \geq d_g$ and $K_j \subseteq K_g$, although the tasks cannot be treated as having a single completion time, it can be demonstrated that there is no benefit to considering the case where $C_g > C_j$.

These observations can be used to demonstrate that not all linear extensions of $\rightarrow$ need to be checked in order to find an optimal schedule. Denote by $\hat{\rightarrow}$ the partial order on N such that, for any $j \in N$ and $g \in N$,

- $j \hat{\rightarrow} g$ if $j \rightarrow g$;
- $j \hat{\rightarrow} g$ if $d_g \geq d_j$ and $K_g \subset K_j$;
- $j \hat{\rightarrow} g$ if $d_g > d_j$ and $K_g = K_j$;
- $j \hat{\rightarrow} g$ if $d_g = d_j$, $K_g = K_j$ and $j < g$; and
- $j \not\hat{\rightarrow} g$ otherwise.

Note that in the case where $d_g = d_j$ and $K_g = K_j$, the tasks are ordered according to their indices in the original problem instance φ. Since $\hat{\rightarrow}$ is an extension of $\rightarrow$, it follows that $|\Pi(\hat{\rightarrow})| \leq |\Pi(\rightarrow)|$. It will be proven that only the linear extensions of $\hat{\rightarrow}$ need to be checked in order to find an optimal schedule.

For all $j \in N$, define

$$\Psi_j = \{g : g \in N \wedge d_j \geq d_g \wedge K_j \subseteq K_g\}$$

and, for any schedule $\sigma \in \Sigma$, define

$$\psi_j(\sigma) = \max_{g \in \Psi_j} C_g(\sigma).$$

Note that $j \in \Psi_j$ and so $\psi_j(\sigma) \geq C_j(\sigma)$. The set Ψ_j can be viewed as the set of all tasks which, in some sense, dominate task j. The desirable property of these values is that at least one linear extension of $\hat{\to}$ arranges tasks in non-decreasing order of $\psi_j(\sigma)$. This follows from the lemma below.

Lemma 7.4 *If $j \hat{\to} g$, $\psi_j(\sigma) \leq \psi_g(\sigma)$ for any schedule $\sigma \in \Sigma$.*

Proof. This proof will be broken up into two cases, specifically if $j \hat{\to} g$ either $j \to g$ or $d_g \geq d_j$ and $K_g \subseteq K_j$.

- If $j \to g$, it follows that $q \to g$ for any $q \in \Psi_j$, and hence

$$\psi_j(\sigma) = \max_{q \in \Psi_j} C_q(\sigma) < \max_{q \in \Psi_j} C_g(\sigma) = C_g(\sigma) \leq \psi_g(\sigma).$$

- If $d_g \geq d_j$ and $K_g \subseteq K_j$, it follows that $\Psi_j \subseteq \Psi_g$, and hence

$$\psi_j(\sigma) = \max_{q \in \Psi_j} C_q(\sigma) \leq \max_{q \in \Psi_g} C_q(\sigma) = \psi_g(\sigma).$$

Consequently the lemma is proven in both cases. ■

The proof of the theorem below is accomplished by transforming an arbitrary schedule $\sigma \in \Lambda$ into a feasible solution of Linear Program 7.1.

Theorem 7.5 *For some $\prec \in \Pi(\hat{\to})$, the schedule $\varpi(\prec)$ is optimal for the $P|prec, pmtn, u_j|L_{max}$ problem.*

Proof. Consider any schedule $\sigma \in \Sigma$. Renumber the tasks according to any linear extension of $\hat{\to}$ such that, after the renumbering,

$$\psi_1(\sigma) \leq \psi_2(\sigma) \leq \cdots \leq \psi_n(\sigma). \tag{7.5}$$

Lemma 7.4 demonstrates that this is always possible.

Now apply the following process:

- let $L_{max} = L_{max}(\sigma)$,
- let $C_j = \psi_j(\sigma)$ for all $j \in N$, and

- let $p_{j,i} = p_j(\psi_{i-1}(\sigma), \sigma) - p_j(\psi_i(\sigma), \sigma)$ for all $j \in N$ and all $i \in B_j$.

The constraints of Linear Program 7.1 will now be considered in turn:

- (7.2b) holds for all $j \in N$ because there exists a $g \in \Psi_j$ such that $C_g(\sigma) = \psi_j(\sigma)$, and hence

$$L_{max} = L_{max}(\sigma) \geq C_g(\sigma) - d_g \geq C_g(\sigma) - d_j = \psi_j(\sigma) - d_j = C_j - d_j;$$

- (7.2c) holds as a consequence of (7.5);

- (7.2d) holds because each machine can process at most one task at a time;

- (7.2e) holds because a task j can be processed on at most u_j machines simultaneously; and

- (7.2f) holds by the definition of completion time, the fact that

$$C_j = \psi_j(\sigma) \geq C_j(\sigma),$$

and because, for all $g \in J_j$ there exists a $q \in J_j$ such that

$$C_g = \psi_g(\sigma) = C_q(\sigma).$$

Consequently, any feasible schedule for any instance of $P|prec, pmtn, u_j|L_{max}$ corresponds to a feasible solution to Linear Program 7.1 where the tasks have been renumbered according to a linear extension of $\hat{\rightarrow}$. Thus, the theorem follows from Theorem 7.2 and the fact that $L_{max} = L_{max}(\sigma)$. ■

Example 7.6 *Recall that Example 1.1 on page 17 introduced a problem instance with fifteen tasks. The above reasoning allows several optimisations:*[*]

- *since* $K_5 = K_6 = K_7 = K_{11} = K_{14} = K_{15} = \emptyset$*, the variables* C_j *corresponding to all of these tasks can be ordered according to their due dates;*

- *since* $d_6 = d_7 = d_{14} = d_{15} = 15$ *all four tasks may be assumed to have equal values of* C_j*;*

- *since* $d_5 = d_{11} = 11$ *both tasks may be assumed to have equal values of* C_j*, and this combined value is not greater than the* C_j *corresponding to tasks 6, 7, 14 and 15; and*

[*]For the sake of simplicity, the tasks have not been renumbered in this exposition.

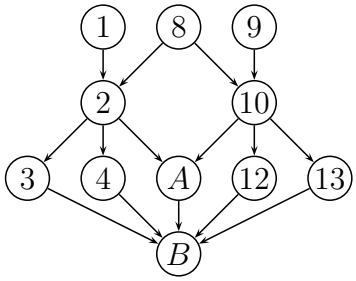


Figure 7.1: The new partial order $\hat{\rightarrow}$, where $A = \{5, 11\}$ and $B = \{6, 7, 14, 15\}$.

- *since* $K_2 \subset K_8$, $K_{10} \subset K_8$ *and* $d_8 = 7 < 11 = d_2 = d_{10}$ *it may be assumed that the* C_j *corresponding to task 8 is not greater than that corresponding to tasks 2 and 10.*

The above constraints on the ordering of $C_1, \ldots, C_n$*, combined with the constraints imposed directly by the precedence constraints, imply the partial order* $\hat{\rightarrow}$ *represented in Figure 7.1.*

While there are 13,122,000 linear extensions of the precedence constraints $\rightarrow$*, there are only 3168 linear extensions of* $\hat{\rightarrow}$*.*

The reasoning in this section provides fundamental insight into the reason for the difficulty of scheduling in the presence of different kinds of precedence constraints. In particular it helps explain Table 3.1, which catalogues the state of current knowledge of computational complexity of different subproblems. In this authors opinion, the key insight of this section is that independent tasks with same successors reduce the complexity of solving the problem, at least using the methods laid out in this chapter. As discussed earlier, weak orders are composed of "layers" of tasks, all with the same successors; in-forests are constructed from multiple sets of tasks, each with identical successors; partial orders with limited width have limited numbers of independent tasks. All can be solved in polynomial time. Out-forests, series-parallel orders and bipartite orders can all contain many independent tasks, each with a different sets of successors.

All of this is speculation, and falls far short of a general theory of the computational complexity of subproblems of $P|prec, pmtn, u_j|L_{max}$. Nevertheless, it seems likely to this author that a more general theory in this area awaits discovery.

Chapter 8

Integer Preemptions

This chapter presents polynomial-time algorithms and analyses their worst-case performance for the following scheduling problem, which will be denoted $P|prec, int - pmtn|L_{max}$. The results of this chapter were originally published in [123] in collaboration with Yakov Zinder. A finite set of tasks (jobs, operations) $N = \{1, 2, \ldots, n\}$ is to be processed on $m \geq 1$ identical parallel machines (processors). The processing of tasks begins at time $t = 0$. Each machine can process only one task at a time, and each task can be processed by only one machine at a time. Since all machines are identical, it is not important what machine actually processes a task, provided that the number of tasks, processed simultaneously, does not exceed m.

The order in which tasks can be processed is restricted by precedence constraints – an irreflexive, antisymmetric and transitive relation on N. If task j precedes task g, denoted $j \to g$, then task g cannot be processed until task j has been completed. In this case, g is called a successor of j and j is a predecessor of g. Task j is an immediate predecessor of task g if $j \to g$ and there is no $q \in N$ such that $j \to q \to g$. Task g is an immediate successor of j if j is an immediate predecessor of g.

The processing of a task can commence only at an integer point in time. At any integer point in time, the processing of a task can be interrupted and resumed later at another integer point in time. The processing time of $j \in N$ will be denoted p_j and is an integer. In other words, p_j is the integer number of time units that j should accumulate in order to be completed.

The constraints on the points in time where preemptions can occur permits more accurate modeling of various practical situations. For example, it permits modeling of the discrete nature of CPU clock cycles. More accurate models with limited preemptions have been subject to increasing attention in recent years, for example [23].

A schedule σ specifies, for each task j, a sequence of time units where j is

processed. The right end of the last of these time units is denoted by $C_j(\sigma)$ and is referred to as the completion time of j in schedule σ. The goal is to minimise the maximum lateness

$$L_{max}(\sigma) = \max_{j \in N}\{C_j(\sigma) - d_j\},$$

where d_j is the integer due time (due date) by which it is desired to complete j.

If all $p_j = 1$, then no preemptions are involved and the considered scheduling problem becomes the classical maximum lateness problem with unit execution time (UET) tasks. In the standard three-field notation (see for example [20] or [89]) the latter problem is denoted by $P|prec, p_j = 1|L_{max}$. It is well-known that $P|prec$, $p_j = 1|L_{max}$ is NP-hard in the strong sense, as proven in [109].

The problem with integer preemptions can be reduced to $P|prec, p_j = 1|L_{max}$ by replacing each $j \in N$ by a chain of p_j UET tasks. Unfortunately, this reduction cannot be accomplished in an amount of time polynomial in n. Therefore, the known polynomial-time approximation algorithms for $P|prec, p_j = 1|L_{max}$ are not directly applicable. The relaxation of the restriction that tasks can commence processing only at integer points in time and that the preemptions and resumptions of processing can occur only at integer points in time gives the well-known maximum lateness problem with preemptions, denoted $P|prec, pmtn|L_{max}$, examined in earlier chapters.

The problems $P|prec, p_j = 1|L_{max}$ and $P|prec, pmtn|L_{max}$ are closely related. Thus, the algorithms in [21], [45] and [119], which were developed for the $P|prec$, $p_j = 1|L_{max}$ problem, were modified in [71], [55] and [120] for the problem $P|prec$, $pmtn|L_{max}$. The core of all these modifications is an algorithm which schedules tasks according to their priorities, whereas the idea of the method of calculating priorities was borrowed from the algorithms for $P|prec, p_j = 1|L_{max}$. Two version of this core scheduling algorithm for $P|prec, pmtn|L_{max}$ can be found in [71] and [105].

The chapter presents a new polynomial-time algorithm for $P|prec, pmtn|L_{max}$ which can be viewed as an amalgamation of the algorithms in [71] and [105]. This new algorithm is an alternative to the algorithms in [71] and [105], although in this chapter its role is limited to be a subroutine (first phase) in another polynomial-time algorithm for the problem with integer preemptions, denoted $P|prec, int - pmtn$ $|L_{max}$. The latter algorithm will be referred to as Algorithm IP. The chapter also presents modifications, for the case of integer preemptions, of the algorithms originally described in [21], [45] and [119] for $P|prec, p_j = 1|L_{max}$. All three modifications are based on Algorithm IP and are characterised by their worst-case performance guarantees.

The structure of the chapter is as follows. Section 8.1 describes Algorithm IP,

which constructs a schedule using the tasks' priorities. The computation of tasks' priorities requires that each task j should be assigned a certain parameter μ_j. Unlike previous chapters, μ_j is, in this chapter, restricted to integer values only, for reasons that are fundamental to the nature of Algorithm IP. This algorithm can be used with the Brucker-Garey-Johnson, Garey-Johnson and Zinder-Roper Algorithms already described.

Section 8.2 provides insight into the structure of these schedules and any other schedules which may be constructed by algorithms with certain similar properties. This in similar to Section 4.4 for the continuous case. Finally, Section 8.3 presents a summary of the results. As these results and their derivations are similar to those presented in Chapter 5, only the outlines are presented.

8.1 Algorithm IP

All considerd algorithms are comprised of two components – the method of calculating μ's and the method of constructing a schedule using $p_j(t, \sigma) + \mu_j$ as a priority of task j in schedule σ at time t. Algorithm IP, described in this section, schedules tasks according to their priorities, where the tasks with higher priority have preference over the tasks with lower priority in the allocation of machine times.

The allocation of machine times depends on the tasks which are simultaneously available for processing, because these tasks compete for machine time. Therefore, Algorithm IP constructs the resulting schedule, which will be denoted η, section by section, where each section is a partial schedule specifying processing between two certain points in time. The end of each section is caused by the completion of one of the tasks available for processing. Once the machine times allocated to the tasks in a section are determined, the actual section (part of η) is constructed by the McNaughton's Algorithm, originally presented in [81].

More specifically, let integers $T_0 < \cdots < T_k$ be the points in time specifying the sections of η. The first point of the first section is $T_0 = 0$ – the commencement of the tasks' processing. Denote by A_i the set of all tasks available for processing at time T_i, i.e. the set of all tasks that have not been completed by T_i and which do not have uncompleted predecessors at T_i. The last point T_k designates the completion of all tasks, and therefore $A_k = \emptyset$. Furthermore, Algorithm IP constructs sections in such manner that, for each $0 \leq i < k$,

- $A_i \neq A_{i+1}$ and
- only tasks from A_i are processed in the time interval $[T_i, T_{i+1}]$.

Note that these T_i differ from the points of allocation for Algorithm P, where the set of available tasks does not necessarily change. Indeed, this is a key propety of Algorithm IP that allows the easy scheduling of tasks subject to integer preemptions. The point in time that corresponds to the end of a section is determined by the event that the set of the tasks available for processing has changed as a result of the completion of at least one task.

The allocation of machine times to the tasks in each A_i (or equivalently, the construction of each section) is a two-phase procedure. The first phase is the allocation of machine times to the tasks in A_i under the relaxation that the commencement of processing of a task, preemptions of this processing and its resumptions, can occur at any points in time which may not necessarily be integer. This process is similar to one or more iterations of Algorithm P. The second phase is a transformation of the machine times, assigned at the first phase, into integers. This allows the construction of a section of η by McNaughton's Algorithm, where each task can commence its processing only at an integer point in time and where the processing of any task can be preempted and resumed only at integer points in time. After the conclusion of the second phase, McNaughton's Algorithm is executed, constructing this section of the schedule η.

Algorithm IP

1. Initialise $i := 0$ and $T_0 := 0$.
2. Compute the set of available tasks A_i, then:
   - 2.1. perform the first phase, which relaxes the integrality constraints then essentially performs iterations of Algorithm P until at least one task is completed;
   - 2.2. perform the second phase, which transforms the (potentially) non-integer values produced by the first phase into integers; and
   - 2.3. schedule the results using McNaughton's Algorithm.
3. If all tasks have been completed, set $k := i + 1$ and terminate the algorithm. Otherwise, increment i and go to Step 2.

For each section, the allocation of machine times to the tasks in the corresponding A_i during the first phase is accomplished in several stages. Each stage is associated with a point of allocation, each of which is determined in the same manner as the points of allocation in Algorithm P.

8.1.1 The First Phase

Let $t_0 < \cdots < t_u$ be the points of allocation for the section of η that specifies the processing in the time interval $[T_i, T_{i+1}]$. Hence, $t_0 = T_i$ is an integer, whereas t_u is not necessarily equal to T_{i+1}, because t_u may not be an integer. Let v_j^r be the amount of machine time allocated to $j \in A_i$ at t_r. The allocation of machine times v_j^r at any point of allocation t_r where $0 \leq r < u$, is analogous to that of Algorithm P.

The points of allocation are determined consecutively starting with $t_0 = T_i$. When the point of allocation t_r is found, the total amount of machine time, allocated to $j \in A_i$ in the time interval $[t_0, t_r]$, is

$$V_j^r = \begin{cases} 0 & \text{if } r = 0 \\ \sum_{s=0}^{r-1} v_j^s & \text{if } 1 \leq r \leq u \end{cases}.$$

The quantity $p_j(T_i, \eta) + \mu_j - V_j^r$ is the priority that j would have at t_r if, after the point in time T_i, the tasks in A_i are processed under the relaxation allowing the commencement, interruption and resumption of tasks' processing at any points in time, not only those which are integer. Since η is a result of the transformation of all V_g^u into integers, $p_j(T_i, \eta) + \mu_j - V_j^r$ is not the priority of j at t_r. This quantity will instead be referred to as the phase-one priority or simply p1-priority.

After finding the point of allocation t_r, the next point of allocation t_{r+1} (or, equivalently, the value of $t_{r+1} - t_r$) and the amount of machine time v_j^r allocated to each task $j \in A_i$ in the interval $[t_r, t_{r+1}]$ are determined as follows. The set A_i is partitioned into three sets, N_r^1, N_r^2 and N_r^3. The set N_r^1 (which may be empty) is chosen as the largest subset of A_i with cardinality not greater than m such that, for any j which is in this set and for any $g \in A_i$ which is not in this set,

$$p_j(T_i, \eta) + \mu_j - V_j^r > p_g(T_i, \eta) + \mu_g - V_g^r.$$

Then, the set N_r^2 is the empty set if $|N_r^1| = m$, and otherwise is the set of all tasks $j \in (A_i \setminus N_r^1)$, which have equal p1-priorities $p_j(T_i, \eta) + \mu_j - V_j^r$ and this collective p1-priority is the largest among all tasks in $A_i \setminus N_r^1$. All tasks $j \in A_i$, which are not in $N_r^1 \cup N_r^2$, form N_r^3. Therefore, N_r^3 also can be empty. It is easy to see that this partition of A_i is unique, that $N_r^1 \cup N_r^2 \neq \emptyset$, and that if $N_r^2 \neq \emptyset$, then $|N_r^1 \cup N_r^2| > m$.

Let $\Delta = t_{r+1} - t_r$. At the point of allocation t_r, the tasks in N_r^1 have higher p1-priorities than other tasks in A_i. Therefore, each $j \in N_r^1$ (if there are any) is allocated the largest possible amount of machine time in the interval $[t_r, t_{r+1}]$, that

is $v_j^r = \Delta$. This imposes the following restriction on Δ:

$$\min_{j \in N_r^1} \left[p_j \left(T_i, \eta \right) - V_j^r \right] \geq \Delta. \tag{8.1}$$

The tasks constituting N_r^3 have lower p1-priorities than any other tasks in A_i, and therefore are not allocated any machine time at t_r, i.e. $v_j^r = 0$ for all $j \in N_r^3$.

All tasks in N_r^2 have the same p1-priority, and therefore are allocated equal machine time at t_r. Since the total machine time available in the time interval $[t_r, t_{r+1}]$ is $m\Delta$, and since $|N_r^1|\Delta$ units of this machine time have been allocated to the tasks in N_r^1, each task $j \in N_r^2$ (if there are any) is allocated at t_r

$$v_j^r = \frac{m - |N_r^1|}{|N_r^2|} \Delta$$

units of machine time. The allocation of machine time to the tasks in N_r^2, assuming this set is non-empty, imposes the following restriction on Δ:

$$\min_{j \in N_r^2} \left[p_j \left(T_i, \eta \right) - V_j^r \right] \geq \frac{m - |N_r^1|}{|N_r^2|} \Delta. \tag{8.2}$$

By virtue of $|N_r^1| \leq m$ and $|N_r^1 \cup N_r^2| > m$, and the fact that $N_r^2 \neq \emptyset$ implies $|N_r^1| \neq m$, for any $j \in N_r^2$ it follows that

$$0 < v_j^r < \Delta. \tag{8.3}$$

In order to preserve the relationship between the p1-priorities of the tasks in the sets N_r^1, N_r^2 and N_r^3, the p1-priority of any task in N_r^2 should not become at t_{r+1} greater than the p1-priority of a task in N_r^1. If $N_r^1 \neq \emptyset \neq N_r^2$ this imposes the following restriction on Δ:

$$\min_{j \in N_r^1} \left[p_j \left(T_i, \eta \right) + \mu_j - V_j^r \right] - \Delta \geq p_g \left(T_i, \eta \right) + \mu_g - V_g^r - \frac{m - |N_r^1|}{|N_r^2|} \Delta, \tag{8.4}$$

where g is an arbitrary task from N_r^2. Similarly, the p1-priority of any task in N_r^3 should not become at t_{r+1} greater than the p1-priority of a task in $N_r^1 \cup N_r^2$. If $N_r^3 \neq \emptyset$ this imposes the following restrictions on Δ:

$$p_g \left(T_i, \eta \right) + \mu_g - V_g^r - \frac{m - |N_r^1|}{|N_r^2|} \Delta \geq \max_{j \in N_r^3} \left[p_j \left(T_i, \eta \right) + \mu_j - V_j^r \right] \tag{8.5}$$

in the case of $N_r^2 \neq \emptyset$, where g is an arbitrary task from N_r^2, and

$$\min_{j \in N_r^1} \left[p_j \left(T_i, \eta \right) + \mu_j - V_j^r \right] - \Delta \geq \max_{j \in N_r^3} \left[p_j \left(T_i, \eta \right) + \mu_j - V_j^r \right] \tag{8.6}$$

in the case of $N_r^1 \neq \emptyset$.

Algorithm IP computes the largest value of Δ that satisfies (8.1), (8.2), (8.4), (8.5), and (8.6), and sets

$$t_{r+1} = t_r + \Delta.$$

If, for the chosen value of Δ, both (8.1) and (8.2) are strict inequalities, then

$$p_j \left(T_i, \eta \right) - V_j^r - v_j^r > 0, \qquad \text{for all } j \in A_i, \tag{8.7}$$

i.e. the allocation at t_r has not changed the set A_i. In this case, Algorithm IP commences the allocation of machine time at t_{r+1}, using exactly the same procedure as for t_r. However, if, for the chosen value of Δ, at least one of these two inequalities becomes an equality, the allocation at t_r has changed the set A_i. In this case, t_{r+1} becomes the last point of allocation (denoted t_u) in the first phase, and Algorithm IP proceeds to the second phase.

The values passed by the first phase to the second phase are:

- the final point of allocation t_u for the section currently being considered and
- the total amount of processing V_j^u allocated to each task $j \in A_i$.

The second phase will then transform these values into integers. See Algorithm IP – First Phase for a summary of the preceding description.

Before proceeding to a description of the second phase, several lemmas and corollaries are necessary in order to understand the nature of the output of the first phase. Due to their similarity to the results in Section 4.2, they are presented without proofs. The role of these lemmas and corollaries is explained below. As was stated earlier, a section is comprised of several points of allocation. At each point of allocation t_r, the set of all available tasks is partitioned into three sets N_r^1, N_r^2 and N_r^3. Lemma 8.1 specifies how this partition of tasks evolves from one point of allocation to another within each section. This lemma is of the greatest importance and is used in the proofs of several subsequent lemmas, in particular Lemma 8.4 and Lemma 8.11.

Lemma 8.1 also leads directly to Corollary 8.2 and Corollary 8.3. As was described earlier, the first phase calculates V_j^u for all $j \in A_i$. These values may not be integers, so the second phase of Algorithm IP transforms them into integers.

Algorithm IP – First Phase

1. Begin the first phase by setting r, the index of the current point of allocation t_r, equal to zero.
2. Set t_0, the first point of allocation for this section, equal to T_i.
3. Set V_j^r, the total machine time allocated to task j at all points of allocation $t_s < t_r$ in the current section, to zero.
4. Compute the p1-priorities $p_j(T_i, \eta) + \mu_j - V_j^r$ of all tasks $j \in A_i$, and use these values to determine the sets N_r^1, N_r^2 and N_r^3.
5. Determine the next point of allocation $t_{r+1} := t_r + \Delta$ by selecting the largest value of Δ such that (8.1), (8.2), (8.4), (8.5) and (8.6) are satisfied.
6. Set $V_j^{r+1} = V_j^r + \Delta$ for all $j \in N_r^1$, set $V_j^{r+1} := V_j^r + \frac{m - |N_r^1|}{|N_r^2|}\Delta$ for all $j \in N_r^2$ and set $V_j^{r+1} := V_j^r$ for all $j \in N_r^3$.
7. If (8.1) or (8.2) is an equality, set $u := r+1$ and end the first phase; otherwise, increment r and go to Step 5.

Corollary 8.2 provides insight into how this transformation is made for all tasks $j \in N_{u-1}^1$. Corollary 8.3 demonstrates that, if $N_{u-1}^2 \neq \emptyset$, then the values V_j^u can be transformed into integers in such a way that all machines are busy during the time interval $[T_i, T_{i+1}]$.

From any point in time, each task must be processed not more than its remaining processing time. In order to ensure this, Lemma 8.4 establishes a bound on V_j^r, for all $r \in \{0, 1, \ldots, u-1\}$ and $j \in A_i$, in terms of $p_j(T_i, \eta)$. The subsequent Corollary 8.5 proves the relation $\lceil V_j^u \rceil \leq p_j(T_i, \eta)$ for all $j \in A_i$, and this is used in establishing the validity of the second phase of Algorithm IP.

As described earlier, Algorithm IP processes tasks with high p1-priorities in preference to tasks with low p1-priorities. The processing of each task reduces its p1-priority, so the rate at which a task's p1-priority decreases for tasks with higher p1-priority is not less than the rate for tasks with lower p1-priority. Lemma 8.6 proves that the method of allocation of processing times adopted in the first phase of Algorithm IP does not violate the relation between tasks in terms of their p1-priorities. More specifically, if at some point of allocation one task has p1-priority not less than that of another task, then this relation holds for all subsequent points of allocation within the section.

The result, stated in Lemma 8.6, is used in the proof of Lemma 8.11. More specifically, Lemma 8.11 is concerned with two points of allocation, t_u and t_{u-1},

which are the last and second last points of allocation in the considered section of η, respectively. Lemma 8.11 uses an implication of Lemma 8.6 to the p1-priorities which the tasks, constituting the sets N^1_{u-1}, N^2_{u-1} and N^3_{u-1}, have at the last point of allocation t_u. In order to facilitate reading, this implication is stated separately as Corollary 8.7 of Lemma 8.6.

Lemma 8.1 *For any* $0 \leq r < s < u$,

- $N^1_s \subseteq N^1_r$,
- $N^1_r \subseteq N^1_s \cup N^2_s$,
- $N^2_r \subseteq N^2_s$ *and*
- $N^3_s \subseteq N^3_r$.

Corollary 8.2 *For any* $j \in N^1_{u-1}$, $V^u_j = t_u - T_i$.

Corollary 8.3 *If* $N^2_{u-1} \neq \emptyset$, *then*

$$\sum_{j \in N^1_{u-1}} V^u_j + \sum_{j \in N^2_{u-1}} V^u_j = m(t_u - T_i).$$

Lemma 8.4 *For all tasks* $j \in A_i$ *and all* $r < u$, $v^r_j \leq p_j\left(T_i, \eta\right) - V^r_j$.

Corollary 8.5 *For all tasks* $j \in A_i$, $\left\lceil V^u_j \right\rceil \leq p_j\left(T_i, \eta\right)$.

Lemma 8.6 *For any* $r \in \{0, \ldots, u-1\}$ *and any pair of tasks* $j \in A_i$ *and* $g \in A_i$ *such that*

$$p_j\left(T_i, \eta\right) + \mu_j - V^r_j \geq p_g\left(T_i, \eta\right) + \mu_g - V^r_g,$$

it follows that

$$p_j\left(T_i, \eta\right) + \mu_j - V^{r+1}_j \geq p_g\left(T_i, \eta\right) + \mu_g - V^{r+1}_g.$$

Corollary 8.7 *For any* $a \in \{1, 2\}$ *and any* $b \in \{2, 3\}$ *such that* $a < b$, *and for any two tasks* $j \in N^a_{u-1}$ *and* $g \in N^a_{u-1}$ *it follows that*

$$p_j\left(T_i, \eta\right) + \mu_j - V^u_j \geq p_g\left(T_i, \eta\right) + \mu_g - V^u_g.$$

8.1.2 The Second Phase

As a consequence of the first phase, when $r = u-1$ the choice of Δ makes either (8.1) or (8.2) an equality, and possibly makes both equalities. Since $t_0 = T_i$, the left end point of the time interval $[t_0, t_u]$ is an integer. In contrast, the right end point t_u and all V_j^u are not necessarily integer; hence, the second phase of the section construction is a transformation of t_u into an integer T_{i+1} and each V_j^u into an integer v_j. The method of this transformation depends on what inequality, (8.1) or (8.2), becomes an equality. The transformation deals only with the tasks that constitute the set $j \in N_{u-1}^1 \cup N_{u-1}^2$, because Lemma 8.1 implies that, for any $j \in N_{u-1}^3$, $V_j^u = 0$.

In both of the two cases considered below, there is some $a \in \{1, 2\}$ for which V_j^u is an integer for all $j \in N_{u-1}^a$. Since the goal of the second phase of Algorithm IP is to transform the (potentially) non-integer values of V_j^u into integers, the manner in which the second phase proceeds is determined by which of the two cases holds, i.e. which of the two inequalities (8.1) or (8.2) is an equality. The two cases are summarised below.

- Lemma 8.8 addresses the case when (8.1) is an equality. This lemma proves that, in the case when (8.1) is an equality, V_j^u is an integer for all $j \in N_{u-1}^1$. Lemma 8.8 justifies the following assignments made by Algorithm IP in the case when (8.1) is an equality:
  - $T_{i+1} = t_u$,
  - $v_g = V_g^u$ for each $g \in N_{u-1}^1$ and
  - either $v_g = \lceil V_g^u \rceil$ or $v_g = \lfloor V_g^u \rfloor$ for each $g \in N_{u-1}^2$.
- Lemma 8.9 addresses the case when (8.2) is an equality. This lemma proves that, in the case when (8.2) is an equality, V_j^u is an integer for all $j \in N_{u-1}^2$. Lemma 8.9 justifies the following assignments made by Algorithm IP in the case when (8.2) is an equality:
  - $T_{i+1} = \lfloor t_u \rfloor$,
  - $v_g = \lfloor V_g^u \rfloor$ for each $g \in N_{u-1}^1$ and
  - either $v_g = V_g^u$ or $v_g = V_g^u - 1$ for each $g \in N_{u-1}^2$. The determination of the number of tasks for which $v_g = V_g^u - 1$ is based on Lemma 8.10. Lemma 8.10 shows how to compute the number of tasks for which $v_g = V_g^u - 1$ in order to precisely fill the section between T_i and T_{i+1}.

If both (8.1) and (8.2) are equalities, Lemma 8.8 and Lemma 8.9 combine to demonstrate that t_u and all V_j^u are integers. As a consequence, Algorithm IP assignes

$T_{i+1} = t_u$ and $v_g = V_g^u$ for each $g \in N_{u-1}^1 \cup N_{u-1}^2$. These assignments are compatible with both cases above, but in practice it is convenient to use the same assignments for the case where (8.1) and (8.2) are equalities as is used in the case where only (8.1) is an equality.

Note that Corollary 8.5 demonstrates that, for all $j \in N_{u-1}^1 \cup N_{u-1}^2$,

$$v_g \leq \left\lceil V_g^u \right\rceil \leq p_j\left(T_i, \eta\right).$$

As a consequence of this, no task j will be assigned more than $p_j\left(T_i, \eta\right)$ units of processing for this section of the schedule η. The case where (8.1) is an equality will be considered first.

Lemma 8.8 *If, when $r = u - 1$, the choice of Δ makes (8.1) an equality, then V_j^u is an integer for all $j \in N_{u-1}^1$, and t_u is an integer also.*

Proof. When (8.1) is an equality, there is a task $j_1 \in N_{u-1}^1$ such that

$$p_{j_1}\left(T_i, \eta\right) - V_{j_1}^{u-1} = \Delta. \tag{8.8}$$

Taking into account that, for Δ chosen at t_{u-1}, $v_{j_1}^{u-1} = \Delta$ and that

$$V_{j_1}^u = V_{j_1}^{u-1} + v_{j_1}^{u-1},$$

(8.8) implies

$$p_{j_1}\left(T_i, \eta\right) = V_{j_1}^u.$$

Hence, by virtue of Corollary 8.2, for any $j \in N_{u-1}^1$,

$$p_{j_1}\left(T_i, \eta\right) = V_{j_1}^u = V_j^u = t_u - T_i.$$

Since both T_i and $p_{j_1}\left(T_i, \eta\right)$ are integer, t_u is an integer, as is V_j^u for all tasks $j \in N_{u-1}^1$. ■

As a consequence of the above lemma, in the case where the choice of Δ makes (8.1) an equality, Algorithm IP sets $T_{i+1} = t_u$ and $v_j = V_j^u$ for all $j \in N_{u-1}^1$.

If $N_{u-1}^2 = \emptyset$, then according to Lemma 8.1 no tasks, except those that are in N_{u-1}^1, are allocated machine time at the first phase, and therefore the transformation into integers is completed. If $N_{u-1}^2 \neq \emptyset$ then, by Corollary 8.3,

$$\sum_{j \in N_{u-1}^2} \lfloor V_j^u \rfloor \leq m(T_{i+1} - T_i) - \sum_{j \in N_{u-1}^1} V_j^u \leq \sum_{j \in N_{u-1}^2} \lceil V_j^u \rceil.$$

Since $v_j = V_j^u$, for all $j \in N_{u-1}^1$, Algorithm IP chooses

$$m(T_{i+1} - T_i) - \sum_{j \in N_{u-1}^1} v_j - \sum_{j \in N_{u-1}^2} \lfloor V_j^u \rfloor$$

tasks j from N_{u-1}^2 and for each of them sets $v_j = \lceil V_j^u \rceil$. For each remaining j in N_{u-1}^2, Algorithm IP sets $v_j = \lfloor V_j^u \rfloor$. There are many ways in which this selection can be made, but for the purposes of this chapter let the tasks be selected such that for any $j \in N_{u-1}^2$ that is selected and $g \in N_{u-1}^2$ that is not, $j < g$.

This concludes the transformation when the choice of Δ makes (8.1) an equality. As a result of this transformation, t_u is transformed into T_{i+1} and each V_j^u, $j \in N_{u-1}^1 \cup N_{u-1}^2$, is transformed into v_j, where T_{i+1} and all v_j are integer. Taking into account (8.3), which holds for any $0 \leq r < u$, it is easy to see that

$$v_j \leq T_{i+1} - T_i, \text{ for all } j \in N_{u-1}^1 \cup N_{u-1}^2, \text{ and } \sum_{j \in N_j^1 \cup N_j^2} v_j \leq m(T_{i+1} - T_i). \quad (8.9)$$

Lemma 8.9 *If, when $r = u - 1$, the choice of Δ makes (8.2) an equality, then V_j^u is an integer for all $j \in N_{u-1}^2$.*

Proof. When (8.2) is an equality, there is some $j_2 \in N_{u-1}^2$ such that, for Δ chosen at t_{u-1},

$$p_{j_2}(T_i, \eta) - V_{j_2}^{u-1} = \frac{m - |N_r^1|}{|N_r^2|}\Delta. \quad (8.10)$$

and consequently

$$v_{j_2}^{u-1} = \frac{m - |N_r^1|}{|N_r^2|}\Delta.$$

Because

$$V_{j_2}^u = V_{j_2}^{u-1} + v_{j_2}^{u-1},$$

(8.10) gives

$$p_{j_2}(T_i, \eta) = V_{j_2}^u. \quad (8.11)$$

Since $p_{j_2}(T_i, \eta)$ is an integer, this implies $V_{j_2}^u$ is an integer also. Since all $j \in N_{u-1}^2$ have the same p1-priority at time t_{u-1}, and all have the same value of v_j^{u-1} this implies they have the same p1-priority at time t_u. Since all μ_j are integers as well this common p1-priority must be an integer, further implying that V_j^u is an integer for all $j \in N_{u-1}^2$. ■

Task j_2 in the preceding lemma is used in this portion of Algorithm IP – specifically, it is a task designated for completion in the current section. It may be selected

in a number of different ways but for the purposes of this chapter, select j_2 such that (8.10) holds for j_2 and for no $g \in N^2_{u-1}$ such that $g < j_2$.

If t_u is an integer, then by Corollary 8.2, V^u_j is an integer for all $j \in N^1_{u-1}$. Therefore, in the case of integer t_u, the transformation concludes by setting $T_{i+1} = t_u$ and $v_j = V^u_j$ for all $j \in N^1_{u-1} \cup N^2_{u-1}$. It is obvious that T_{i+1} and all v_j, $j \in N^1_{u-1} \cup N^2_{u-1}$, obtained as a result of this transformation, satisfy (8.9).

If t_u is not an integer, then Algorithm IP sets $T_{i+1} = \lfloor t_u \rfloor$ and $v_j = \lfloor V^u_j \rfloor$ for all $j \in N^1_{u-1}$.

Lemma 8.10 *If t_u is not an integer, then*

$$\sum_{j \in N^2_{u-1}} (V^u_j - 1) < \left(m - \left|N^1_{u-1}\right|\right)(\lfloor t_u \rfloor - T_i) < \sum_{j \in N^2_{u-1}} V^u_j.$$

Proof. By Corollary 8.2 and Corollary 8.3,

$$\sum_{j \in N^2_{u-1}} V^u_j = \left(m - \left|N^1_{u-1}\right|\right)(t_u - T_i)$$

which, by virtue of the inequality $\left|N^2_{u-1}\right| > m - \left|N^1_{u-1}\right|$, gives

$$\sum_{j \in N^2_{u-1}} (V^u_j - 1) = \sum_{j \in N^2_{u-1}} V^u_j - \left|N^2_{u-1}\right| = \left(m - \left|N^1_{u-1}\right|\right)(t_u - T_i) - \left|N^2_{u-1}\right|$$

$$< \left(m - \left|N^1_{u-1}\right|\right)(t_u - T_i) - \left(m - \left|N^1_{u-1}\right|\right) = \left(m - \left|N^1_{u-1}\right|\right)(t_u - 1 - T_i)$$

$$< \left(m - \left|N^1_{u-1}\right|\right)(\lfloor t_u \rfloor - T_i) < \left(m - \left|N^1_{u-1}\right|\right)(t_u - T_i) = \sum_{j \in N^2_{u-1}} V^u_j$$

and the lemma is proven. ■

So, if t_u is not an integer, Algorithm IP chooses a subset of N^2_{u-1}, which does not contain j_2 and has cardinality

$$\sum_{j \in N^2_{u-1}} V^u_j - \left(m - \left|N^1_{u-1}\right|\right)(\lfloor t_u \rfloor - T_i),$$

and sets $v_j = V^u_j - 1$ for every task in this subset. For each $j \in N^2_{u-1}$ that do not belong to the selected subset, Algorithm IP sets $v_j = V^u_j$. Again, this selection may be made in a number of ways. For the purposes of this chapter let any task $j \in N^2_{u-1}$ that is selected, and any task $g \in N^2_{u-1}$ that is not, have the relation $j < g$ unless $g = j_2$. It is easy to see that T_{i+1} and all v_j, $j \in N^1_{u-1} \cup N^2_{u-1}$, obtained as a result of this transformation, satisfy (8.9).

A complete summary of the second phase is given in Algorithm IP – Second Phase.

Algorithm IP – Second Phase

1. If (8.1) is an equality:

   1.1. set $T_{i+1} := t_u$;

   1.2. set $v_g := V_g^u$ for all $g \in N_{u-1}^1$;

   1.3. set $v_g := \lceil V_g^u \rceil$ for

$$m(T_{i+1} - T_i) - \sum_{j \in N_{u-1}^1} v_j - \sum_{j \in N_{u-1}^2} V_j^u$$

   tasks $g \in N_{u-1}^2$ and set $v_g := \lfloor V_g^u \rfloor$ for the remaining tasks $g \in N_{u-1}^2$.

2. If (8.1) is not an equality:

   2.1. set $T_{i+1} := \lfloor t_u \rfloor$;

   2.2. set $v_g := \lfloor V_g^u \rfloor$ for all $g \in N_{u-1}^1$;

   2.3. set $v_g := V_g^u - 1$ for

$$\sum_{j \in N_{u-1}^2} V_j^u - \left(m - \left|N_{u-1}^1\right|\right) \left(\lfloor t_u \rfloor - T_i\right)$$

   tasks $g \in N_{u-1}^2$ and set $v_g := V_g^u$ for the remaining tasks $g \in N_{u-1}^2$.

3. Set $v_g := 0$ for all tasks $g \in N_{u-1}^3$.

After the execution of the second phase, McNaughton's Algorithm is executed.

The priorities of tasks which are used in the first phase of Algorithm IP and which are referred to as p1-priorities may differ from the priorities of tasks in the schedule η. According to Corollary 8.7, at the point of allocation t_u, the p1-priority of any task in N_{u-1}^1 is not less than that of any task in $N_{u-1}^2 \cup N_{u-1}^3$, and the p1-priority of any task in N_{u-1}^2 is not less than that of any task in N_{u-1}^3. The following Lemma 8.11 proves that, although the priorities of tasks in the schedule η at time T_{i+1} may differ from the p1-priorities at point of allocation t_u, the above relation between the priorities of tasks from different sets remains unchanged.

Furthermore, according to Lemma 8.6, at the point of allocation t_u, all tasks in N_{u-1}^2 have the same p1-priorities. This property does not hold anymore after the transformation into the schedule with integer preemptions. Lemma 8.12 proves

that, although these new priorities may be different at the point in time T_{i+1} for different tasks in N_{u-1}^2, this difference cannot be greater than one. This important property is further used in Lemma 8.13.

Lemma 8.11 *For any $j_1 \in N_{u-1}^1$, $j_2 \in N_{u-1}^2$ and $j_3 \in N_{u-1}^3$,*

$$p_{j_1}(T_{i+1}, \eta) + \mu_{j_1} \geq p_{j_3}(T_{i+1}, \eta) + \mu_{j_3}, \tag{8.12}$$

$$p_{j_2}(T_{i+1}, \eta) + \mu_{j_2} \geq p_{j_3}(T_{i+1}, \eta) + \mu_{j_3}, \tag{8.13}$$

$$p_{j_1}(T_{i+1}, \eta) + \mu_{j_1} \geq p_{j_2}(T_{i+1}, \eta) + \mu_{j_2}. \tag{8.14}$$

Proof. By virtue of Lemma 8.1, $V_j^u = 0$ for all $j \in N_{u-1}^3$. Therefore,

$$p_j(T_i, \eta) = p_j(T_{i+1}, \eta)$$

for all $j \in N_{u-1}^3$.

If the choice of Δ when r is $u-1$ makes (8.1) an equality, then, for all $j \in N_{u-1}^1$, V_j^u is an integer. Hence, by Corollary 8.7, (8.12) is given by

$$p_{j_1}(T_{i+1}, \eta) + \mu_{j_1} = p_{j_1}(T_i, \eta) + \mu_{j_1} - v_{j_1} = p_{j_1}(T_i, \eta) + \mu_{j_1} - V_{j_1}^u$$

$$\geq p_{j_3}(T_i, \eta) + \mu_{j_3} - V_{j_3}^u = p_{j_3}(T_i, \eta) + \mu_{j_3} = p_{j_3}(T_{i+1}, \eta) + \mu_{j_3}.$$

Furthermore, Corollary 8.7 demonstrates that

$$p_{j_2}(T_i, \eta) + \mu_{j_2} - V_{j_2}^u \geq p_{j_3}(T_i, \eta) + \mu_{j_3} - V_{j_3}^u,$$

and since $p_{j_2}(T_i, \eta)$, μ_{j_2}, $p_{j_3}(T_i, \eta)$, μ_{j_3} are integer and $V_{j_3}^u = 0$, it follows that

$$p_{j_2}(T_i, \eta) + \mu_{j_2} - \lceil V_{j_2}^u \rceil \geq p_{j_3}(T_i, \eta) + \mu_{j_3} - V_{j_3}^u = p_{j_3}(T_{i+1}, \eta) + \mu_{j_3}. \tag{8.15}$$

Hence, by virtue of

$$p_{j_2}(T_{i+1}, \eta) + \mu_{j_2} = p_{j_2}(T_i, \eta) + \mu_{j_2} - v_{j_2}$$

$$\geq p_{j_2}(T_i, \eta) + \mu_{j_2} - \lceil V_{j_2}^u \rceil \geq p_{j_3}(T_{i+1}, \eta) + \mu_{j_3},$$

(8.15) gives (8.13).

Finally, Corollary 8.7 demonstrates that

$$p_{j_1}(T_i, \eta) + \mu_{j_1} - V_{j_1}^u \geq p_{j_2}(T_i, \eta) + \mu_{j_2} - V_{j_2}^u,$$

and since $V_{j_1}^u$, $p_{j_1}(T_i,\eta)$, $p_{j_2}(T_i,\eta)$, μ_{j_1} and μ_{j_2} are integer, it follows that

$$p_{j_1}(T_i,\eta)+\mu_{j_1}-V_{j_1}^u \geq p_{j_2}(T_i,\eta)+\mu_{j_2}-\left\lfloor V_{j_2}^u \right\rfloor, \tag{8.16}$$

Hence, by virtue of

$$p_{j_1}(T_{i+1},\eta)+\mu_{j_1}=p_{j_1}(T_i,\eta)+\mu_{j_1}-v_{j_1}=p_{j_1}(T_i,\eta)+\mu_{j_1}-V_{j_1}^u$$

$$\geq p_{j_2}(T_i,\eta)+\mu_{j_2}-\lfloor V_{j_2}^u \rfloor \geq p_{j_2}(T_i,\eta)+\mu_{j_2}-v_{j_2}=p_{j_2}(T_{i+1},\eta)+\mu_{j_2},$$

(8.16) gives (8.14).

If on the other hand, the choice of Δ when r is $u-1$ makes (8.2) an equality, then, for all $j \in N_{u-1}^2$, V_j^u is an integer and $V_j^u - 1 \leq v_j \leq V_j^u$. Hence, taking into account that $V_{j_3}^u = 0$,

$$p_{j_2}(T_{i+1},\eta)+\mu_{j_2}=p_{j_2}(T_i,\eta)+\mu_{j_2}-v_{j_2} \geq p_{j_2}(T_i,\eta)+\mu_{j_2}-V_{j_2}^u$$

$$\geq p_{j_3}(T_i,\eta)+\mu_{j_3}-V_{j_3}^u=p_{j_3}(T_{i+1},\eta)+\mu_{j_3}$$

which gives (8.13). Furthermore, $V_{j_3}^u = 0$ implies that

$$p_{j_1}(T_{i+1},\eta)+\mu_{j_1}=p_{j_1}(T_i,\eta)+\mu_{j_1}-v_{j_1}=p_{j_1}(T_i,\eta)+\mu_{j_1}-\lfloor V_{j_1}^u \rfloor$$

$$\geq p_{j_1}(T_i,\eta)+\mu_{j_1}-V_{j_1}^u \geq p_{j_3}(T_i,\eta)+\mu_{j_3}-V_{j_3}^u=p_{j_3}(T_{i+1},\eta)+\mu_{j_3}$$

and therefore (8.12) .

Finally, since $V_{j_2}^u$, $p_{j_1}(T_i,\eta)$, $p_{j_2}(T_i,\eta)$, μ_{j_1} and μ_{j_2} are integer,

$$p_{j_1}(T_i,\eta)+\mu_{j_1}-V_{j_1}^u \geq p_{j_2}(T_i,\eta)+\mu_{j_2}-V_{j_2}^u$$

implies

$$p_{j_1}(T_i,\eta)+\mu_{j_1}-\lfloor V_{j_1}^u \rfloor \geq p_{j_2}(T_i,\eta)+\mu_{j_2}-(V_{j_2}^u-1)$$

and consequently, by virtue of $V_j^u - 1 \leq v_j \leq V_j^u$,

$$p_{j_1}(T_{i+1},\eta)+\mu_{j_1}=p_{j_1}(T_i,\eta)+\mu_{j_1}-v_{j_1}$$

$$=p_{j_1}(T_i,\eta)+\mu_{j_1}-\lfloor V_{j_1}^u \rfloor \geq p_{j_2}(T_i,\eta)+\mu_{j_2}-(V_{j_2}^u-1)$$

$$\geq p_{j_2}(T_i,\eta)+\mu_{j_2}-v_{j_2}=p_{j_2}(T_{i+1},\eta)+\mu_{j_2}$$

which gives (8.14). ■

Lemma 8.12 *For any $j \in N_{u-1}^2$ and $g \in N_{u-1}^2$,*

$$|p_j(T_{i+1}, \eta) + \mu_j - p_g(T_{i+1}, \eta) - \mu_g| \leq 1, \tag{8.17}$$

and the inequality

$$p_j(T_{i+1}, \eta) + \mu_j > p_g(T_{i+1}, \eta) + \mu_g \tag{8.18}$$

implies $C_j(\eta) > T_{i+1}$.

Proof. Let j and g be any two tasks in N_{u-1}^2. The proof is based on the equality

$$p_j(T_i, \eta) + \mu_j - V_j^u = p_g(T_i, \eta) + \mu_g - V_g^u \tag{8.19}$$

and the transformation conducted at the second phase of the construction of the considered section of η. If

$$p_j(T_{i+1}, \eta) + \mu_j = p_g(T_{i+1}, \eta) + \mu_g,$$

then the lemma holds. So, assume that j and g satisfy (8.18).

Assume that the choice of Δ when r is $u-1$ makes (8.1) an equality. Then, by virtue of the fact that μ_j, μ_g, $p_j(T_i, \eta)$ and $p_g(T_i, \eta)$ are integer, (8.18) and (8.19) imply that both, V_j^u and V_g^u, are not integer and that

$$v_j = \lfloor V_j^u \rfloor \qquad \text{and} \qquad v_g = \lceil V_g^u \rceil,$$

which in turn leads to $C_j(\eta) > T_{i+1}$. Furthermore, once again taking into account (8.19) and the fact that μ_j, μ_g, $p_j(T_i, \eta)$ and $p_g(T_i, \eta)$ are integer,

$$p_j(T_{i+1}, \eta) + \mu_j = p_j(T_i, \eta) + \mu_j - \lfloor V_j^u \rfloor = p_g(T_i, \eta) + \mu_g - \lfloor V_g^u \rfloor$$

$$= p_g(T_i, \eta) + \mu_g - \left(\lceil V_g^u \rceil - 1\right) = p_g(T_{i+1}, \eta) + \mu_g + 1$$

which gives (8.17).

Assume that the choice of Δ when r is $u-1$ makes (8.2) an equality. Then, V_j^u and V_g^u are integer and (8.19) implies that

$$v_j = V_j^u - 1 \qquad \text{and} \qquad v_g = V_g^u.$$

This again gives $C_j(\eta) > T_{i+1}$. Furthermore, taking into account (8.19),

$$p_j(T_{i+1}, \eta) + \mu_j = p_j(T_i, \eta) + \mu_j - (V_j^u - 1)$$

$$= p_g(T_i, \eta) + \mu_g - V_g^u + 1 = p_g(T_{i+1}, \eta) + \mu_g + 1$$

which again gives (8.17). ■

8.1.3 Methods for Assigning μ's

The methods of calculating μ's for the case of integer preemptions can be taken from the continuous case (almost) without modification.

- The Brucker-Garey-Johnson Algorithm can use precisely the same μ's. Since all due dates and processing times are integers, this results in all μ's being integers as well. As before, these μ's are modified due dates (as defined in Section 4.4), and (4.15) holds.

- Even with integer due dates and processing times, the μ's produced by the continuous version of the Garey-Johnson Algorithm may not be integers. Hence, when scheduling tasks in the presence of integer preemptions, the μ's should be rounded up to the nearest integer, i.e. (4.19) is replaced with

$$\mu_j := \max\left\{-d_j, \max_{a_j \le \rho \le b_j}\left(\rho + \left\lceil \frac{1}{m}\sum_{g \in K_j} \beta_g(\rho)\right\rceil\right)\right\}.$$

  Since tasks will only ever be completed at integer points in time, these rounded μ's are modified due dates, and (4.15) holds.

- When calculating μ's according to the Zinder-Roper Algorithm, the only adaptation necessary is to use Algorithm IP, rather than Algorithm P, to produce the schedule η^j in (4.24). As with the continuous case, (4.15) holds but the resulting μ's are not necessarily modified due dates.

These algorithms have many of the same properties as their continuous counterparts, and the proofs of these properties are similar, and hence have been omitted. As a consequence of these similarities the results for integer preemptions are quite similar to the results for arbitrary preemptions.

8.2 Schedule Structure

This section describes a structure found in the schedules produced by Algorithm IP for a wide variety of μ assignment algorithms. These results will be used to prove a performance guarantee for the Zinder-Roper Algorithm.

As has been described in Section 8.1, in constructing each section of the schedule η, Algorithm IP uses as a subroutine an algorithm for the problem $P|prec, pmtn|L_{max}$, where preemptions are not limited to integer points in time. The analyses of algorithms for this relaxed problem are based on certain properties of points of allocation (see for example [120]). This section demonstrates that the points in time specifying sections of η also permit a similar analysis.

As in Section 8.1, denote by A_i the set of all tasks available for processing at the beginning of the section, specified by the point in time T_i (the point in time when the section starts) and the point in time T_{i+1} (the point in time when the section ends). Consider the partition of A_i into three disjoint sets A_i^1, A_i^2 and A_i^3, obtained as follows. Each section of η is constructed using points of allocation $t_0 < t_1 < \cdots < t_u$. For any section $[T_i, T_{i+1}]$, define

$$A_i^1 = N_{u-1}^1, \qquad A_i^2 = N_{u-1}^2 \qquad \text{and} \qquad A_i^3 = N_{u-1}^3,$$

where $t_0 < t_1 < \cdots < t_u$ are the points of allocation of $[T_i, T_{i+1}]$.

Consider all points in time t such that there exists j for which

$$C_j(\eta) \geq t \qquad \text{and} \qquad t + p_j(t, \eta) + \mu_j = G(\eta).$$

As an example, one such t is the completion time of any task $j \in N$ which satisfies

$$C_j(\eta) + \mu_j = G(\eta).$$

Among all t with the above mentioned property, select the smallest and denote it τ. It is easy to see that if $\tau = 0$, then η is an optimal schedule. In what follows, it is assumed that $T_{i^*-1} < \tau \leq T_{i^*}$, where T_{i^*-1} and T_{i^*} are the endpoints of some section of η.

Lemma 8.13 *The set $A_{i^*-1}^1$ is empty and there exists $x \in A_{i^*-1}^2$ such that*

$$T_{i^*} + p_x(T_{i^*}, \eta) + \mu_x = G(\eta).$$

Proof. If $\tau = T_{i^*}$ then the lemma is proven by the definition of τ.

Suppose this is not the case, i.e. suppose that $\tau \in (T_{i^*-1}, T_{i^*})$. Consider j such that

$$C_j(\eta) \geq \tau \qquad \text{and} \qquad \tau + p_j(\tau, \eta) + \mu_j = G(\eta).$$

This implies

$$G(\eta) = \tau + p_j(\tau, \eta) + \mu_j \leq C_j(\eta) + \mu_j \leq G(\eta),$$

and hence $p_j(\tau,\eta) = C_j(\eta) - \tau$. As a consequence of this, by Lemma 8.1, $j \in A^1_{i^*-1} \cup A^2_{i^*-1}$. If $j \in A^1_{i^*-1}$, then, by Corollary 8.2,

$$p_j(\tau,\eta) = p_j(T_{i^*-1},\eta) - (\tau - T_{i^*-1})$$

and therefore

$$T_{i^*-1} + p_j(T_{i^*-1},\eta) + \mu_j = \tau + p_j(\tau,\eta) + \mu_j = G(\eta),$$

which contradicts the selection of τ. Hence, $j \in A^2_{i^*-1}$.

If $C_j(\eta) \geq T_{i^*}$, the relation $p_j(\tau,\eta) = C_j(\eta) - \tau$ implies the statement of the lemma. Alternatively, suppose that this is not the case, i.e. suppose that $C_j(\eta) < T_{i^*}$. In this case, Lemma 8.12 implies the existence of some $g \in A^2_{i^*-1}$ for which

$$p_g(T_{i^*},\eta) + \mu_g \geq p_j(T_{i^*},\eta) + \mu_j.$$

For any such g, it follows that

$$G(\eta) = C_j(\eta) + \mu_j = C_j(\eta) + p_j(T_{i^*},\eta) + \mu_j$$

$$\leq C_j(\eta) + p_g(T_{i^*},\eta) + \mu_g < T_{i^*} + p_g(T_{i^*},\eta) + \mu_g \leq G(\eta),$$

a contradiction. ■

As in the case of arbitrary preemptions, select some task x as specified in the statement of the above lemma. However, in contrast to consideration of arbitrary preemptions, the priority of task x at time T_{i^*} will not always be used as the cutoff for "high priority" tasks, the processing of which is examined in subsequent proofs; rather, two cases are considered.

- In the first case, all tasks $j \in A^2_{i^*-1}$ have $p_j(T_{i^*},\eta) + \mu_j \geq p_x(T_{i^*},\eta) + \mu_x$. In this case, let
$$\phi = p_x(T_{i^*},\eta) + \mu_x.$$

- Alternatively, if this is not the case, let
$$\phi = p_x(T_{i^*},\eta) + \mu_x - 1.$$

Note that the latter case implies, by Lemma 8.12, that $C_x(\eta) > T_{i^*}$. Furthermore, in either case Lemma 8.12 implies that all tasks $j \in A^2_{i^*-1}$ have $p_j(T_{i^*},\eta) + \mu_j \geq \phi$.

For all $1 \leq i \leq i^*$, define the set

$$M_i = \{j : p_j(T_i, \eta) + \mu_j \geq \phi \wedge p_j(T_i, \eta) < p_j(T_{i-1}, \eta)\}.$$

In words, this means that the set M_i contains all tasks processed during the section $[T_{i-1}, T_i]$ which also have priority at T_i not less than ϕ. If

$$\sum_{j \in M_i} (p_j(T_{i-1}, \eta) - p_j(T_i, \eta)) = m(T_i - T_{i-1}),$$

the section is said to be complete. Otherwise it is said to be incomplete. Observe that the fact that $A^2_{i^*-1} \neq \emptyset$ and that all tasks $j \in A^2_{i^*-1}$ have $p_j(T_{i^*}, \eta) + \mu_j \geq \phi$ implies that $[T_{i^*-1}, T_{i^*}]$ is complete and that $|M_{i^*}| \geq m$.

The inclusion in the definition of M_i of the requirement that task j must be processed during $[T_{i-1}, T_i]$ is necessary to establish Lemma 8.14 below, the integer preemptions equivalent of Corollary 4.28. The specification of a "high priority" task as having a priority not less than ϕ is necessary to prove the subsequent Lemma 8.15, the integer preemptions equivalent of Lemma 4.27.

Lemma 8.14 *If all μ's obey (4.15), for all $1 \leq i' \leq i \leq i^*$ and all $j \in M_i$, if $[T_{i'-1}, T_{i'}]$ is incomplete there exists a task $g \in M_{i'} \cap A^1_{i'-1}$ such that either $g = j$ or $g \to j$.*

Proof. For any $1 < i \leq i^*$, consider a task $j \in M_i$ and consider a section $[T_{i'-1}, T_{i'}]$ with $1 \leq i' < i$. By the definition of the set $A_{i'-1}$ of available tasks, there exists some $g \in A_{i'-1}$ such that either $g = j$ or $g \to j$.

If $g \in A^1_{i'-1}$, the lemma follows from (4.15) and Corollary 8.2. Otherwise, either $g \in A^2_{i'-1}$ or $g \in A^3_{i'-1}$. In either case, for all q for which $p_q(T_{i'-1}, \eta) > p_q(T_{i'}, \eta)$, Lemma 8.11, Lemma 8.12, (4.15) and the fact that $j \in M_i$ imply

$$p_q(T_{i'}, \eta) + \mu_q \geq p_g(T_{i'}, \eta) + \mu_g - 1 \geq p_j(T_{i-1}, \eta) + \mu_j - 1 \geq p_j(T_i, \eta) + \mu_j \geq \phi,$$

and thus $[T_{i'-1}, T_{i'}]$ is complete. ■

Lemma 8.15 *If all μ's obey (4.15), for all $1 \leq i \leq i^*$, if $[T_{i-1}, T_i]$ is incomplete there exists a task in M_i that is processed at all times during $[T_{i-1}, T_i]$.*

Proof. First consider the case where $i < i^*$. Since $x \in M_{i^*}$, Lemma 8.14 establishes the existence of some $g \in M_i \cap A^1_{i-1}$. By Corollary 8.2 this implies that g is processed at all times during $[T_{i-1}, T_i]$, so the lemma holds.

On the other hand, suppose $i = i^*$. For any $g \in A^2_{i^*-1}$ the definition of ϕ implies that $p_g(T_i, \eta) + \mu_g \geq \phi$. Hence, if $A^2_{i^*-1} \neq \emptyset$ all tasks that are processed during $[T_{i^*-1}, T_{i^*}]$ are in M_{i^*}. By Corollary 8.3 and the definition of the second phase of Algorithm IP, $A^2_{i^*-1} \neq \emptyset$ implies that all machines are busy at all times during $[T_{i^*-1}, T_{i^*}]$. Consequently $[T_{i^*-1}, T_{i^*}]$ is complete if $A^2_{i^*-1} \neq \emptyset$ and hence the lemma holds in this case.

If $A^2_{i^*-1} = \emptyset$ it follows that $A^1_{i^*-1} \neq \emptyset$ and thus $x \in A^1_{i^*-1}$. Since $p_x(T_{i^*}, \eta) + \mu_x \geq \phi$, $x \in M_{i^*} \cap A^1_{i^*-1}$ and Corollary 8.2 implies that x is processed at all times during $[T_{i^*-1}, T_{i^*}]$, so the lemma holds in all cases. ■

The preceding reasoning demonstrates that many of the properties of a schedule constructed by Algorithm P have rough equivalents for schedules constructed by Algorithm IP. As established by Corollary 8.2, Corollary 8.3, Lemma 8.11 and Lemma 8.12, each section $[T_i, T_{i+1}]$ produced by Algorithm IP gives rise to three sets of tasks, A^1_i, A^2_i and A^3_i, with the following properties.

- Every task in A^1_i is processed at all times during $[T_i, T_{i+1}]$ and has priority at time T_{i+1} not less than that of all tasks in $A^2_i \cup A^3_i$.
- If $A^2_i \neq \emptyset$ all machines are busy at all times during $[T_i, T_{i+1}]$.
- Every task in A^3_i is not processed during $[T_i, T_{i+1}]$ and has priority at time T_{i+1} not more than that of all tasks in $A^1_i \cup A^2_i$.

In these respects, A^1_i, A^2_i and A^3_i correspond to the sets N^i_1, N^i_2 and N^i_3 that arise from a time interval $[t_i, t_{i+1}]$ in a schedule produced by Algorithm P.

Furthermore, the priority ϕ introduced in the analysis of Algorithm IP corresponds roughly to the priority ρ^* introduced in the analysis of Algorithm P. In addition, the set M_{i+1} for Algorithm IP and the set H_i for Algorithm P allow the definition of complete and incomplete sections/intervals with the two properties below.

- During a section/interval that is complete, all machines at all times process "high priority" tasks.
- There is a chain of "high priority" tasks, which precedes task x, such that at every time during any incomplete section/interval a task from that chain is being processed.

There are two significant* differences as detailed below.

*The fact that M_i is indexed by the rightmost point of the corresponding section, while H_i is indexed by the leftmost point of the corresponding interval, arises from historical convention and ease of use.

- Each task in M_i is processed during the corresponding section, because this property is used in the proof of Lemma 8.14. In contrast it is not necessarily true that each task in H_i is processed during the corresponding interval, because inclusion of these tasks is necessary to prove Lemma 6.3.

- As Lemma 8.12 established, the tasks in A_i^2 might have priorities at time T_{i+1} that differ by one. This discrepancy, not present in the case of arbitrary preemptions, has a modest effect on the performance guarantees that are established in this chapter.

8.3 Results

As the proofs of the performance guarantees of the considered algorithms are very similar to the corresponding proofs in Chapter 5, the proof for the Brucker-Garey-Johnson Algorithm will be outlined, while the Garey-Johnson and Zinder-Roper Algorithms will not have proofs given. As specified earlier, $p_{min} = \min_{j \in N} p_j$, $d_{max} = \max_{j \in N} d_j$ and l is the length of the longest chain of tasks.

Theorem 8.16 *Given an instance of the $P|prec, int - pmtn|L_{max}$ problem, for any schedule $\eta^{\mathscr{BGJ}}$ produced by the Brucker-Garey-Johnson Algorithm and any optimal schedule ω,*

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) - L_{max}\left(\omega\right) \leq \begin{cases} 0 & \text{if } l = 1 \\ \dfrac{m-1}{m}(l - p_{min} + 1) & \text{otherwise} \end{cases}$$

for all combinations of $l \geq p_{min} \geq 1$ and $m \geq 1$.

Proof. As in earlier proofs, let c be the total length of complete sections and w be the total length of incomplete sections, and observe that $l = 1$ implies that there are no precedence constraints, and hence $w = 0$ and $\eta^{\mathscr{BGJ}}$ is optimal.

There are two cases to consider, based on the value of ϕ. Since

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) = G\left(\eta^{\mathscr{BGJ}}\right) = T_{i^*} + p_x\left(T_{i^*}, \eta\right) + \mu_x$$

either

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) = c + w + \phi \qquad \text{or} \qquad L_{max}\left(\eta^{\mathscr{BGJ}}\right) = c + w + 1 + \phi.$$

Recall that, in the latter case, task x is processed for at least one time unit after time T_{i^*}, meaning there is an extra unit of "high priority" processing, beyond what

is implied by the complete and incomplete sections. A lower bound on $L_{max}(\omega) = G(\omega)$ can be derived from a lower bound on the earliest that all "high priority" processing can be finished. Hence,

$$L_{max}(\omega) \geq c + \frac{w}{m} + \phi \qquad \text{or} \qquad L_{max}(\omega) \geq c + \frac{w+1}{m} + \phi$$

and subtracting gives

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) - L_{max}(\omega) \leq \frac{m-1}{m} w$$

or

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) - L_{max}(\omega) \leq \frac{m-1}{m}(w+1).$$

Since, as stated earlier, $|M_{i^*}| \geq m$, this implies that either $\eta^{\mathscr{BGJ}}$ is an optimal schedule or the chain of "high priority" tasks, the existence of which is implied by Lemma 8.14, precedes an additional task that is an element of the set M_{i^*}. As a consequence of this,

$$l \geq w + p_{min};$$

substituting gives

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) - L_{max}(\omega) \leq \frac{m-1}{m}(l - p_{min})$$

or

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) - L_{max}(\omega) \leq \frac{m-1}{m}(l - p_{min} + 1)$$

and the theorem is proven. ■

Corollary 8.17 *Given an instance of the $P|prec, int - pmtn|L_{max}$ problem, for any schedule $\eta^{\mathscr{BGJ}}$ produced by the Brucker-Garey-Johnson Algorithm and any optimal schedule ω,*

$$L_{max}\left(\eta^{\mathscr{BGJ}}\right) \leq \left(2 - \frac{1}{m}\right) L_{max}(\omega) + \left(1 - \frac{1}{m}\right)(d_{max} - p_{min} + 1)$$

for all combinations of d_{max}, $l \geq p_{min} \geq 1$ and $m \geq 1$.

The performance guarantees presented below can be derived in the same way as the corresponding results for the case of arbitrary preemptions.

Theorem 8.18 *Given an instance of the $P|prec, int - pmtn|L_{max}$ problem, for any*

schedule $\eta^{\mathscr{GJ}}$ *produced by the Garey-Johnson Algorithm and any optimal schedule* ω*,*

$$L_{max}\left(\eta^{\mathscr{GJ}}\right) - L_{max}\left(\omega\right) \leq \begin{cases} 0 & \text{if } m \leq 2 \text{ or } l = 1 \\ \dfrac{m-2}{m}(l - p_{min} + 1) & \text{otherwise} \end{cases}$$

for all combinations of $l \geq p_{min} \geq 1$ *and* $m \geq 1$*.*

Corollary 8.19 *Given an instance of the* $P|prec, int - pmtn|L_{max}$ *problem, for any schedule* $\eta^{\mathscr{GJ}}$ *produced by the Garey-Johnson Algorithm and any optimal schedule* ω*,*

$$L_{max}\left(\eta^{\mathscr{GJ}}\right) \leq \begin{cases} 0 & \text{if } m \leq 2 \\ \left(2 - \dfrac{2}{m}\right) L_{max}\left(\omega\right) + \left(1 - \dfrac{1}{m}\right)(d_{max} - p_{min} + 1) & \text{otherwise} \end{cases}$$

for all combinations of d_{max}*,* $l \geq p_{min} \geq 1$ *and* $m \geq 1$*.*

Theorem 8.20 *Given an instance of the* $P|prec, int - pmtn|L_{max}$ *problem, for any schedule* $\eta^{\mathscr{ZR}}$ *produced by the Zinder-Roper Algorithm and any optimal schedule* ω*,*

$$L_{max}\left(\eta^{\mathscr{ZR}}\right) \leq \begin{cases} 0 & \text{if } m \leq 2 \\ \left(2 - \dfrac{2}{m}\right) L_{max}\left(\omega\right) + \left(1 - \dfrac{1}{m}\right)(d_{max} - p_{min} + 1) & \text{otherwise} \end{cases}$$

for all combinations of d_{max}*,* $l \geq p_{min} \geq 1$ *and* $m \geq 1$*.*

If all $p_j = 1$, the $P|prec, int - pmtn|L_{max}$ problem reduces to the much studied $P|prec, p_j = 1|L_{max}$ problem, for which there exist many performance guarantees. It is worth noting that, if $p_{min} = 1$, the performance guarantees established in this chapter are in many cases quite close to existing results for the unit execution time (UET) task versions of the considered scheduling algorithms.

Conclusion

This thesis has examined, from a variety of perspectives, several related problems of scheduling tasks with preemptions and precedence constraints on identical parallel machines to minimise the maximum lateness. The limits of current knowledge in this area have been significantly advanced, prompting new questions to be answered in future research.

Summary

Chapter 3 considered the computational complexity of several subproblems defined by precedence constraints limited to particular classes of partial orders, and attempts were made to link these results with the asymptotic growth rates of these classes as the number of nodes (tasks) increases. The complexity results expanded on well known existing results from [71] and elsewhere, while the results regarding the asymptotic growth rates of poset classes summarised some existing results from order theory.

Chapter 4 considers several specific approximation algorithms and observes the common elements between them, defining families of algorithms based on these commonalities and elucidating some common properties of these algorithms. These algorithms (and families of algorithms) are then examined in Chapter 5 and Chapter 6.

In Chapter 5, a number of performance guarantees are established, building on several existing results. One contribution of this thesis is to examine the benefit of parallelism: the amount by which a performance guarantee can be improved if tasks can be processed on multiple machines simultaneously. All performance guarantees established in this thesis include a parameter which specifies the minimum degree of parallelism allowed. This thesis also establishes complete guarantees of achievability for two bounds:

- Theorem 5.2 and Theorem 5.6 concern the Brucker-Garey-Johnson Algorithm, improving on the best known result in [105]; and

- Theorem 5.14 and Theorem 5.28 concern the Garey-Johnson Algorithm, establishing a completely new bound with a comprehensive proof of achievability.

Additionally, Theorem 5.49 establishes a performance guarantee for the Zinder-Roper Algorithm, although achievability is not established in this case. Nonetheless, this theorem improves on the best known result in [120]. Results of an unprecedented nature are also established regarding the limitations of the considered families of algorithms. Theorem 5.28 establishes a "best possible" performance guarantee for a broad – indeed infinite – family of approximation algorithms, and demonstrates that the Garey-Johnson Algorithm has precisely this performance guarantee. This is a very surprising result – the Garey-Johnson Algorithm is well known, and was not invented for the purpose of having a dominant performance guarantee. It is also a very important result, since it guides future research. Any researcher who wants to improve on the performance guarantee established in Theorem 5.14 and Theorem 5.28 must search outside the considered family of algorithms. Another result for an even more general family of algorithms is established in Theorem 5.32, but it is as yet unknown if any algorithm achieves this performance guarantee.

Chapter 6 presents results of a completely unprecedented nature. It is common to establish that certain approximation algorithms provide optimal schedules under certain circumstances. This chapter introduces the notion of an algorithms domain – the class of all partial orders such that, if the precedence constraints are drawn from this class, the algorithm produces an optimal solution. Three new classes of partial orders are introduced, corresponding to the domains of the three considered approximation algorithms:

- the domain $\mathcal{D}^{\mathscr{BGJ}}$ of the Brucker-Garey-Johnson Algorithm is established in Theorem 6.6 and Theorem 6.16 as a class of partial orders defined by thirty prohibited suborders,

- the domain $\mathcal{D}^{\mathscr{ZR}}$ of the Zinder-Roper Algorithm is estabished in Theorem 6.23 and Theorem 6.75 as a class of partial orders defined by a prohibited substructure termed a "singular quartet," and

- the domain $\mathcal{D}^{\mathscr{GJ}}$ of the Garey-Johnson Algorithm is established in Theorem 6.80 and Theorem 6.105 as a class of partial orders defined by a prohibited substructure termed a "pseudo-singular quartet."

It is also established that $\mathcal{D}^{\mathscr{BGJ}} \subset \mathcal{D}^{\mathscr{GJ}} \subset \mathcal{D}^{\mathscr{ZR}}$. Partial orders of these classes can be recognized in polynomial time. These results guide future researchers, since they demonstrate that no improvements on the solvability results established in this

thesis are possible. An area for future research is to establish the domains of other approximation algorithms. It is also desirable to characterise these classes of partial orders in terms of their asymptotic growth rates – such characterisations will provide insight into the relative quality of the corresponding approximation algorithms. In Chapter 5 it was established that the Garey-Johnson algorithm dominates a broad family of algorithms in terms of (a specific kind of) performance guarantee. In this chapter it is established, in Theorem 6.75, that the Zinder-Roper Algorithm dominates a broad family of algorithms in terms of its domain, meaning its domain is a superclass of the domain of every algorithm in the considered family. This is another completely unprecedented result. It guides future research by establishing that, if any additional solvable precedence constraints are desired, then an algorithm outside the considered family must be used. All of the results in this chapter consider the domain for non-malleable tasks (tasks which cannot be processed in parallel on multiple machines) and the maximum lateness. Consideration of other task models and other objective functions requires new analysis, and will likely lead to new classes of partial orders.

Having established limitations on what can be achieved within the considered family of approximation algorithms, Chapter 7 looks outside this family. In particular, this chapter defines a non-polynomial algorithm for obtaining an optimal schedule. Combined with a branch-and-bound approach, possibly using some of the approximation algorithms described in this thesis, this could be an effective approach for small- to mid-sized problems, but this is an area for future research. This approach is similar to that used in [35] to solve interval ordered tasks. Since interval orders are outside the domain of the Zinder-Roper Algorithm, this demonstrates that it is necessary to search beyond the family $\mathfrak{F}$ to find algorithms for all easily solvable subproblems of $P|prec, pmtn|L_{max}$ and $P|prec, pmtn, u_j|L_{max}$.

Finally, in Chapter 8 a beginning was made on the project of producing equivalent results for the problem with integer preemptions, denoted $P|prec, int - pmtn|L_{max}$. Probably the most important contribution of this chapter is Algorithm IP, which is able to perform much the same function for integer preemptions that Algorithm P is for arbitrary preemptions. The schedules produced by this algorithm were characterised and analysed, and from this analysis arose several performance guarantees. The study of this algorithm is in its infancy at the moment, and much work remains to be done. The effect of parallelism on the schedules produced by Algorithm IP (or rather its equivalent for malleable tasks) appears to be rather complex and warrants some study. Similarly, it is desirable to characterise the domains of the Brucker-Garey-Johnson, Garey-Johnson and Zinder-Roper Algorithms when used in conjunction with Algorithm IP, and to try to reproduce the dominance

results of Chapter 5 and Chapter 6.

In conclusion, this thesis has produced substantial results, in some cases results that are completely unprecedented in nature. It has contributed significantly to the current state of the art, and it can be anticipated that it will prompt further study into the new areas of research that it has pioneered.

Future Research

Chapter 3 focused on the relationship between computational complexity of certain scheduling problems and the number (in asymptotic terms) of partial orders to which their precedence constraints belong, as summarised in Table 3.2. One area of future research is to complete Table 3.2, both by

- providing complexity results for the problem $P|series-parallel, pmtn|C_{max}$ and $P|intervalorder, pmtn|C_{max}$, and by
- establishing more precise asymptotic growth rates for the classes of unlabelled interval orders and posets with width at most three.

Table 3.2 could be expanded by considering additional classes of partial orders from which precedence constraints are drawn and additional objective functions. Further results in this area may provide great insight into what, precisely, makes a problem difficult to solve.

Chapter 4 considers a variety of approximation algorithms, but there are more that are worthy of consideration: see [46] for an example; there may also be more families of approximation algorithms that merit further study. These constitute areas for further research. Additionally, although this thesis considers worst-case performance (Chapter 5) and circumstances of optimal performance (Chapter 6), average performance is not considered. Since average performance is likely to be an important consideration in practice, it is desirable that this be investigated, either by theoretical means or by simulation.

The performance guarantees established in Chapter 5 introduce the parameter u_{min} of minimum parallelism. This examination of the effect of parallelism on certain algorithms can be taken further, in several ways.

- Previously established performance guarantees can be re-examined with a view to establishing how they are affected by parallelism. This can be applied to approximation algorithms outside the framework of identical parallel machines considered in this thesis.

- The effect of parallelism on the optimal cases of an approximation algorithm (see Chapter 6) can also be examined.

- The effect of parallelism on the computational complexity of certain scheduling problems (see Chapter 3) can be investigated.

Additionally, some of the performance guarantees established in Chapter 5 have not been proven tight, and this is an area for further research – perhaps they can be substantially improved. Once this has been done, it may be possible to establish the existence of a dominant algorithm for difference bounds, similar to the way Theorem 5.28 established that the Garey-Johnson Algorithm is dominant in terms of difference bounds. Furthermore, no particularly good bound on the difference between the Zinder-Roper Algorithm's maximum lateness and the optimal maximum lateness has been found to date. It is worth investigating whether this is because it is simply a difficult result to establish, or because the Zinder-Roper Algorithm has relatively bad worst-case performance.

In Chapter 6 the solvable cases for three approximation algorithms were reviewed in a unprecedented detail. As can be seen from the length of this chapter results of this kind require a great deal of work to establish. It would be very informative to examine other approximation algorithms, or other scheduling problems, in the same manner – this would likely lead to the description of new classes of partial orders. In particular, examining the domain of the Brucker-Garey-Johnson Algorithm, Garey-Johnson Algorithm and Zinder-Roper Algorithm for the problems $P|prec, pmtn|C_{max}$, $P|prec, pmtn, u_j|C_{max}$ and $P|prec, pmtn, u_j|L_{max}$. Additionally, it is to be noted that the Garey-Johnson Algorithm has a strictly smaller domain than the dominant Zinder-Roper Algorithm, but it has not yet been proven whether or not the Zinder-Roper Algorithm is dominant in the sense of performance guarantee, as was the Garey-Johnson Algorithm. An interesting area of future research is to discover an algorithm from the family $\mathfrak{F} \cap \mathfrak{S}$ that is dominant in both senses, if such an algorithm exists.

The methods for optimally solving certain NP-hard scheduling problems described in Chapter 7 require substantial further investigation. It was the goal of this chapter to establish a framework for the optimal solution of modestly sized scheduling problems, but no work has been done on how to use this in a practical setting. Substantial investigation into lower bounds on an optimal objective function are required before a branch-and-bound approach becomes feasible. However, once this has been accomplished the resulting optimal algorithm can be used in computational experiments to examine how often (and how much) certain approximation algorithms differ from an optimal algorithm.

Finally, Chapter 8 begins applying some of the preceding analysis to the case of integer preemptions. As can be seen, the additional constraint of integer preemptions substantially complicates the design and analysis of algorithms for the problem. It is worth examining how the rest of the results in this thesis can be adapted to the case of integer preemptions. In particular, it would be useful to know whether

- integer preemptions change the computational complexity of any of the scheduling problems examined in Chapter 3;
- algorithms and performance guarantees for integer preemptions can be generalised to malleable tasks, as in Chapter 4 and Chapter 5;
- the domains established in Chapter 6 are the same for problems with integer preemptions; and
- the algorithm described in Chapter 7 can be adapted to optimally solve scheduling problems with integer preemptions.

A great deal of research is still required to put integer preemptions on as sound a theoretical footing as more traditional preemption that can be applied at arbitrary points in time.

In conclusion, the research presented in this thesis establishes a basis for much future work. Indeed, some of the results are intended to guide future researchers looking for more powerful approximation algorithms. The thesis also introduces several new concepts that can be used in analysis of approximation algorithms for other scheduling problems not considered here.

Bibliography

[1] Abdel-Wahab, H. M., & Kameda, T. (1978). Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. *Operations Research*, 26(1), 141–158.

[2] Abdul-Razaq, T. S., Potts, C. N., & Van Wassenhove, L. N. (1990). A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26(2), 235–253.

[3] Ali, H. H., & El-Rewini, H. (1993). The time complexity of scheduling interval orders with communication is polynomial. *Parallel Processing Letters*, 3(01), 53–58.

[4] Baker, K. R. (1974). *Introduction to sequencing and scheduling.* John Wiley & Sons.

[5] Baker, K. R., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1983). Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, 31(2), 381–386.

[6] Baptiste, P. (2000). Preemptive scheduling of identical machines. *UTC Research Report 2000/314*.

[7] Baptiste, P., Le Pape, C., & Peridy, L. (1998). Global constraints for partial CSPs: A case-study of resource and due date constraints. In *International Conference on Principles and Practice of Constraint Programming*, 87–101. Springer Berlin Heidelberg.

[8] Baruah, S., Bonifaci, V., DAngelo, G., Li, H., Marchetti-Spaccamela, A., Van Der Ster, S., & Stougie, L. (2012). The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *2012 24th Euromicro Conference on Real-Time Systems*, 145–154. IEEE.

[9] Benabid, A., & Hanen, C. (2010). Minimizing lateness for precedence graphs with constant delays on dedicated pipelined processors. *Electronic Notes in Discrete Mathematics*, 36, 791–798.

[10] Benabid, A., & Hanen, C. (2010). Performance of GareyJohnson algorithm for pipelined typed tasks systems. *International Transactions in Operational Research*, 17(6), 797–808.

[11] Benabid, A., & Hanen, C. (2010). Performance of Zinder-Roper algorithm for unitary RCPSP with constant precedence latencies. *ROADEF 2010*.

[12] Blazewicz, J. (1976). Scheduling dependent tasks with different arrival times to meet deadlines. In *Proceedings of the International Workshop organized by the Commission of the European Communities on Modelling and Performance Evaluation of Computer Systems*, 57–65. North-Holland Publishing Co..

[13] Blazewicz, J., & Kobler, D. (2002). Review of properties of different precedence graphs for scheduling problems. *European Journal of Operational Research*, 142(3), 435–443.

[14] Braschi, B., & Trystram, D. (1994). A new insight into the Coffman-Graham algorithm. *SIAM Journal on Computing*, 23(3), 662–669.

[15] Brightwell, G. (1993). Models of random partial orders. *Surveys in Combinatorics*, 53–83.

[16] Brightwell, G., & Goodall, S. (1996). The number of partial orders of fixed width. *Order*, 13(4), 315–337.

[17] Brightwell, G., Grable, D. A., & Prömel, H. J. (1999). Forbidden induced partial orders. *Discrete Mathematics*, 201(1), 53–80.

[18] Brightwell, G., Prömel, H. J., & Steger, A. (1996). The average number of linear extensions of a partial order. *Journal of Combinatorial Theory*, Series A, 73(2), 193–206.

[19] Brinkmann, G., & McKay, B. D. (2002). Posets on up to 16 points. *Order*, 19(2), 147–179.

[20] Brucker, P. (2007). *Scheduling Algorithms*. Berlin: Springer.

[21] Brucker, P., Garey, M. R., & Johnson, D. S. (1977). Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Mathematics of Operations Research*, 2(3), 275–284.

[22] Brucker, P., & Knust, S. (2009). Complexity results for scheduling problems, http://www.informatik.uni-osnabrueck.de/knust/class/.

[23] Buttazzo, G. C., Bertogna, M., & Yao, G. (2013). Limited preemptive scheduling for real-time systems: a survey. *IEEE Transactions on Industrial Informatics*, 9(1), 3–15.

[24] Carroll, T. E., & Grosu, D. (2010). Incentive compatible online scheduling of malleable parallel jobs with individual deadlines. In *2010 39th International Conference on Parallel Processing*, 516–524. IEEE.

[25] Chardon, M., & Moukrim, A. (2005). The Coffman-Graham Algorithm Optimally Solves UET Task Systems with Overinterval Orders. *SIAM Journal on Discrete Mathematics*, 19(1), 109–121.

[26] Chen, C. Y., & Chu, C. P. (2013). A 3.42-approximation algorithm for scheduling malleable tasks under precedence constraints. *IEEE Transactions on Parallel and Distributed Systems*, 24(8), 1479–1488.

[27] Chen, N. F. (1975). An analysis of scheduling algorithms in multiprocessing computing systems. *Technical Report UIUCDCS-R-75-724*, Department of Computer Science, University of Illinois at Urbana-Champaign.

[28] Chen, N. F., & Liu, C. L. (1975). On a class of scheduling algorithms for multiprocessors computing systems. *Parallel Processing*, 1–16.

[29] Chen, S., Tong, L., & He, T. (2011). Optimal deadline scheduling with commitment. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference*, 111–118. IEEE.

[30] Cheung, M., & Shmoys, D. B. (2011). A primal-dual approximation algorithm for min-sum single-machine scheduling problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 135–146. Springer Berlin Heidelberg.

[31] Clausen, J. (1999). Branch and bound algorithms – principles and examples. *Department of Computer Science, University of Copenhagen*, 1–30.

[32] Coffman Jr, A. P. E., & Graham, R. L. (1972). Optimal scheduling for two-processor systems. *Acta Informatica*, 1(3), 200–213.

[33] Dauzère-Pérès, S., & Sevaux, M. (2003). Using Lagrangean relaxation to minimize the weighted number of late jobs on a single machine. *Naval Research Logistics (NRL)*, 50(3), 273–288.

[34] Dauzère-Pérès, S., & Sevaux, M. (2004). An exact method to minimize the number of tardy jobs in single machine scheduling. *Journal of Scheduling*, 7(6), 405–420.

[35] Djellab, K. (1999). Scheduling preemptive jobs with precedence constraints on parallel machines. *European Journal of Operational Research*, 117(2), 355–367.

[36] Do, V. H., Hanen, C., & Zinder, Y. (2003). Scheduling UET tasks and preemptive tasks on parallel processors under the restriction of precedence constraints. *Proceedings of the 6th workshop on Models and Algorithms for Planning and Scheduling Problems*.

[37] Du, J., & Leung, J. Y-T. (1989). Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics*, 2(4), 473–487.

[38] Du, J., & Leung, J. Y-T. (1990). Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3), 483–495.

[39] Du, J., Leung, J. Y-T., & Young, G. H. (1990). Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75(3), 347–355.

[40] Du, J., Leung, J. Y-T., & Young, G. H. (1991). Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92(2), 219–236.

[41] El-Zahar, M. H. (1989). Enumeration of ordered sets. In *Algorithms and Order* 327–352.

[42] Fan, L., Zhang, F., Wang, G., & Liu, Z. (2012). An effective approximation algorithm for the malleable parallel task scheduling problem. *Journal of Parallel and Distributed Computing*, 72(5), 693–704.

[43] Finta, L., Liu, Z., Mills, I., & Bampis, E. (1996). Scheduling UET-UCT series-parallel graphs on two processors. *Theoretical Computer Science*, 162(2), 323–340.

[44] Garey, M. R., Graham, R. L., & Johnson, D. S. (1978). Performance guarantees for scheduling algorithms. *Operations Research*, 26(1), 3–21.

[45] Garey, M. R., & Johnson, D. S. (1976). Scheduling tasks with nonuniform deadlines on two processors. *Journal of the ACM*, 23(3), 461–467.

[46] Garey, M. R., & Johnson, D. S. (1977). Two-processor scheduling with start-times and deadlines. *SIAM Journal on Computing*, 6(3), 416–426.

[47] Garey, M. R., & Johnson, D. S. (1978). "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)*, 25(3), 499–508.

[48] Gary, M. R., & Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-completeness.

[49] Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2), 416–429.

[50] Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5, 287–326.

[51] Günther, E., König, F. G., & Megow, N. (2014). Scheduling and packing malleable and parallel tasks with precedence constraints of bounded width. *Journal of Combinatorial Optimization*, 27(1), 164–181.

[52] Guo, S., & Kang, L. (2010). Online scheduling of malleable parallel jobs with setup times on two identical machines. *European Journal of Operational Research*, 206(3), 555–561.

[53] Hall, L. A., Schulz, A. S., Shmoys, D. B., & Wein, J. (1997). Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3), 513–544.

[54] Hanen, C., & Zinder, Y. (2009). The worst-case analysis of the Garey-Johnson algorithm. *Journal of Scheduling*, 12(4), 389–400.

[55] Hanen, C., & Zinder, Y. (2005). The worst case analysis of the Garey-Johnson algorithm for preemptive tasks on m processors. *Multidisciplinary International Conference on Scheduling: Theory and Applications*, 2, 453–470.

[56] Havill, J. T. (2010). Online malleable job scheduling for $m \leq 3$. *Information Processing Letters*, 111(1), 31–35.

[57] Höhn, W., & Jacobs, T. (2015). On the performance of Smiths rule in single-machine scheduling with nonlinear cost. *ACM Transactions on Algorithms (TALG)*, 11(4), 482–493.

[58] Hoogeveen, J. A., Lenstra, J. K., & Veltman, B. (1996). Preemptive scheduling in a two-stage multiprocessor flow shop is NP-Hard. *European Journal of Operational Research*, 89(1), 172–175.

[59] Horn, W. A. (1973). Technical noteminimizing average flow time with parallel machines. *Operations Research*, 21(3), 846–847.

[60] Hu, T. C. (1961). Parallel sequencing and assembly line problems. *Operations Research*, 9(6), 841–848.

[61] Jansen, K., & Zhang, H. (2012). Scheduling malleable tasks with precedence constraints. *Journal of Computer and System Sciences*, 78(1), 245–259.

[62] Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1), 61–68.

[63] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Sixteenth Annual ACM Symposium on Theory of Computing*, 302–311.

[64] Karp, R. M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations* 85–103.

[65] Khachiyan, L. G. (1980). Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1), 53–72.

[66] Kleitman, D. J., & Rothschild, B. L. (1975). Asymptotic enumeration of partial orders on a finite set. *Transactions of the American Mathematical Society*, 205, 205–220.

[67] Kravchenko, S. A., & Werner, F. (2011). Parallel machine problems with equal processing times: a survey. *Journal of Scheduling*, 14(5), 435–444.

[68] Lam, S., & Sethi, R. (1977). Worst case analysis of two scheduling algorithms. *SIAM Journal on Computing*, 6(3), 518–536.

[69] Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 497–520.

[70] Lawler, E. L. (1978). Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2, 75–90.

[71] Lawler, E. L. (1982). Preemptive scheduling of precedence-constrained jobs on parallel machines. In *Deterministic and Stochastic Scheduling*, 101–123. Springer Netherlands.

[72] Lawler, E. L. (1990). A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26(1), 125–133.

[73] Lawler, E. L., & Labetoulle, J. (1978). On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM*, 25(4), 612–619.

[74] Lawler, E. L., & Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations research*, 14(4), 699–719.

[75] Lazarev, A. A., Salnikov, A. M., Baranov, A. V., Gafarov, E., & Werner, F. (2010). Estimation of Absolute Error for Minimization Maximum Lateness. *Avtomatika i Telemekhanika*, (10), 80–89.

[76] Lenstra, J. K., & Rinnooy Kan, A. H. G. (1978). Complexity of scheduling under precedence constraints. *Operations Research*, 26(1), 22–35.

[77] Lenstra, J. K., Rinnooy Kan, A. H. G., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, 343–362.

[78] Leung, J. Y. (Ed.) (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press.

[79] Lipton, R. J., & Regan, K. W. (2012). Branch and Bound – Why Does it Work? *Gödel's Lost Letter and P=NP*, https://rjlipton.wordpress.com/2012/12/19/branch-and-bound-why-does-it-work/

[80] Makarychev, K., & Panigrahi, D. (2014). Precedence-constrained scheduling of malleable jobs with preemption. In *International Colloquium on Automata, Languages, and Programming*, 823–834. Springer Berlin Heidelberg.

[81] McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, 6(1), 1–12.

[82] Möhring, R. H. (1989). Computationally tractable classes of ordered sets. In *Algorithms and Order*. Springer Netherlands.

[83] Möhring, R. H., & Schäffter, M. W. (1999). Scheduling seriesparallel orders subject to 0/1-communication delays. *Parallel Computing*, 25(1), 23–40.

[84] Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1), 102–109.

[85] Moukrim, A. (1999). Optimal scheduling on parallel machines for a new order class. *Operations Research Letters*, 24(1), 91–95.

[86] Moukrim, A., & Quilliot, A. (2005). Optimal preemptive scheduling on a fixed number of identical parallel machines. *Operations Research Letters*, 33(2), 143–150.

[87] Odlyzko, A. M. (1995). Asymptotic enumeration methods. In *Handbook of Combinatorics*, 2, 1063–1229.

[88] Papadimitriou, C. H., & Yannakakis, M. (1979). Scheduling interval-ordered tasks. *SIAM Journal on Computing*, 8(3), 405–409.

[89] Pinedo, M. L. (2012). *Scheduling: Theory, Algorithms, and Systems*. Springer.

[90] Potts, C. N., & Van Wassenhove, L. N. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2), 363–377.

[91] Potts, C. N., & Van Wassenhove, L. N. (1983). An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *European Journal of Operational Research*, 12(4), 379–387.

[92] Prömel, H. J. (1987). Counting unlabeled structures. *Journal of Combinatorial Theory, Series A*, 44(1), 83–93.

[93] Pricopi, M., & Mitra, T. (2014). Task scheduling on adaptive multi-core. *IEEE Transactions on Computers*, 63(10), 2590–2603.

[94] Prot, D., & Bellenguez-Morineau, O. (2015). Impact of precedence constraints on complexity of scheduling problems: a survey. *arXiv preprint*.

[95] Ramachandra, G., & Elmaghraby, S. E. (2006). Sequencing precedence-related jobs on two machines to minimize the weighted completion time. *International Journal of Production Economics*, 100(1), 44–58.

[96] Recalde, D., Rutten, C., Schuurman, P., & Vredeveld, T. (2010). Local search performance guarantees for restricted related parallel machine scheduling. In *Latin American Symposium on Theoretical Informatics*, 108–119. Springer Berlin Heidelberg.

[97] Sadykov, R. (2012). A dominant class of schedules for malleable jobs in the problem to minimize the total weighted completion time. *Computers & Operations Research*, 39(6), 1265–1270.

[98] Sanders, P., & Speck, J. (2012). Energy efficient frequency scaling and scheduling for malleable tasks. In *European Conference on Parallel Processing*, 167–178. Springer Berlin Heidelberg.

[99] Sanders, P., & Speck, J. (2011). Efficient parallel scheduling of malleable tasks. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, 1156–1166. IEEE.

[100] Sauer, N. W., & Stone, M. G. (1989). Preemptive scheduling of interval orders is polynomial. *Order*, 5(4), 345–348.

[101] Sethi, R. (1977). On the complexity of mean flow time scheduling. *Mathematics of Operations Research*, 2(4), 320–330.

[102] Sidney, J. B. (1979). The two-machine maximum flow time problem with series parallel precedence relations. *Operations Research*, 27(4), 782–791.

[103] Singh, G. (2001). Performance of critical path type algorithms for scheduling on parallel processors. *Operations Research Letters*, 29(1), 17–30.

[104] Singh, G. (2005). Scheduling UET-UCT outforests to minimize maximum lateness. *European Journal of Operational Research*, 165(2), 468–478.

[105] Singh, G., & Zinder, Y. (2000). Worst-case performance of critical path type algorithms. *International Transactions in Operational Research*, 7(4–5), 383–399.

[106] Singh, G., & Zinder, Y. (2000). Worst-case performance of two critical path type algorithms. *Asia Pacific Journal of Operational Research*, 17(1), 101–122.

[107] Stanley, R. P. (1974). Enumeration of posets generated by disjoint unions and ordinal sums. *Proceedings of the American Mathematical Society*, 45(2), 295–299.

[108] T'kindt, V., Della Croce, F., & Esswein, C. (2004). Revisiting branch and bound search strategies for machine scheduling problems. *Journal of Scheduling*, 7(6), 429–440.

[109] Ullman, J. D. (1975). NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3), 384–393.

[110] Ullman, J. D. (1976). Complexity of sequencing problems. In *Computer and Job-Shop Scheduling Theory*. John Wiley & Sons.

[111] Valdes, J., Tarjan, R. E., & Lawler, E. L. (1979). The recognition of series parallel digraphs. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, 1–12. ACM.

[112] Verriet, J. (2000). Scheduling tree-like task systems with non-uniform deadlines subject to unit-length communication delays. *Discrete Applied Mathematics*, 101(1), 269–289.

[113] Walker, S., & Zinder, Y. (2015). Scheduling with precedence constraints in the form of an interval order. *ASOR Recent Advances 2015*.

[114] Walker, S., & Zinder, Y. (2013). The solvable cases of a scheduling algorithm. *Algorithms and Computation*, 229–239.

[115] Wang, J. B., Ng, C. T., & Cheng, T. E. (2008). Single-machine scheduling with deteriorating jobs under a seriesparallel graph constraint. *Computers & Operations Research*, 35(8), 2684–2693.

[116] Zinder, Y., Walker, S., & Hanen, C. (2013). The worst-case performance of one class of scheduling algorithms for flexible multiprocessor tasks. In *The 11th Workshop on Models and Algorithms for Planning and Scheduling Problems*.

[117] Zinder, Y. (2007). The strength of priority algorithms. *Proceedings, MISTA*, 531–537.

[118] Zinder, Y., & Roper, D. (1995). A minimax combinatorial optimisation problem on an acyclic directed graph: polynomial-time algorithms and complexity. In *Proceedings of the AC Aitken Centenary Conference, Dunedin*, 391–400.

[119] Zinder, Y., & Roper, D. (1998). An iterative algorithm for scheduling unit-time tasks with precedence constraints to minimise the maximum lateness. *Annals of Operations Research*, 81, 321–343.

[120] Zinder, Y., & Singh, G. (2005). Preemptive scheduling on parallel processors with due dates. *Asia-Pacific Journal of Operational Research*, 22(04), 445–462.

[121] Zinder, Y., Su, B., Singh, G., & Sorli, R. (2010). Scheduling UET-UCT tasks: branch-and-bound search in the priority space. *Optimization and Engineering*, 11(4), 627–646.

[122] Zinder, Y., & Walker, S. (2010). Scheduling flexible multiprocessor tasks: performance guarantees and polynomially solvable cases. In *8th International Conference on Optimization: Techniques and Applications* 336–337.

[123] Zinder, Y., & Walker, S. (2015). Algorithms for scheduling with integer preemptions on parallel machines to minimize the maximum lateness. *Discrete Applied Mathematics*, 196, 28–53.

[124] Zinder, Y., & Walker, S. (2009). Scheduling flexible multiprocessor tasks on parallel machines. In *The 9th Workshop on Models and Algorithms for Planning and Scheduling Problems* 23–25.