UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Decomposition Methods For Resourced Constrained Project Management Problems With Uncertainty

by Alireza Etminaniesfahani

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of
Dr Hanyu Gu
Dr Leila Moslemi Naeni
Dr Amir Salehipour

University of Technology Sydney
Faculty of Science
April 2024

Certificate of Authorship / Originality

I, Alireza Etminaniesfahani, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy in the School of Mathematical and Physical Science, faculty of science at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note: Signature removed prior to publication.

Date: November 26, 2024

Abstract

The resource-constrained project scheduling problem (RCPSP) has attracted significant attention in research over recent decades. The RCPSP is NP-hard, and due to its wide range of applications (e.g., mining, manufacturing, and supply chain), researchers have attempted to propose various solution techniques for this challenging problem. This thesis considers RCPSP and its two well-known extensions, i.e., Multimode RCPSP(MRCPSP) and Stochastic RCPSP (SRCPSP). Novel hybrid metaheuristic, matheuristic, and approximate dynamic programming approaches are proposed for solving the considered optimisation problems.

In the past two decades, the literature on solving the RCPSP has witnessed a notable shift from relying solely on metaheuristic approaches to adopting hybrid methods. These hybrid methods integrate various metaheuristic strategies. Among these strategies, the Artificial Bee Colony (ABC) algorithm stands out for its strong global search ability despite suffering from slow convergence due to the lack of a powerful local search capability. This thesis proposes a novel hybrid metaheuristic, an extension of the ABC with the improved ability of local search to tackle RCPSP.

The MRCPSP involves scheduling tasks under resource and precedence constraints, with multiple execution modes available for each task. To tackle this complex problem, we designed a new matheuristic based on the idea of iteratively relaxing and solving a problem. In addition, a mathematical programming model is developed, which is a generalisation of the multi-dimensional knapsack problem. The proposed mathematical model selects task execution modes and generates initial feasible solutions. We evaluate the performance of the proposed algorithm by solving benchmark instances widely used in the literature. The results demonstrate that the proposed algorithm outperforms the state-of-the-art methods for solving the MRCPSP.

To address the problems involving stochastic task duration, an approximate dynamic programming approach is developed to solve the RCPSP with stochastic task duration (SRCPSP). In this approach, a solution from a deterministic average project is utilised to reduce the computational burden associated with the roll-out policy. The priority rules in this approach are based on the solution from the deterministic problem that is solved once at the beginning of the project. To efficiently obtain feasible solutions for an average project, we designed a novel matheuristic algorithm. The obtained solution is also used in the evaluation process. The computational results on a wide range of well known test functions show that our approach provides competitive solutions in a short computational time.

Acknowledgements

I am especially thankful to Dr. Hanyu Gu for entrusting me with the opportunity to conduct this research under his expert guidance. His unwavering support, insightful advice, and engaging scholarly discussions have shaped my research journey and enriched my understanding of Operations Research. I extend my heartfelt gratitude to Dr. Amir Salehipur and Dr. Leila Moslemi Naeni for their invaluable insights.

A special acknowledgment goes to the University of Technology Sydney for their generous support through the UTS President's Scholarship and the International Research Scholarship, which have enabled me to pursue my PhD journey at this esteemed institution. I am equally grateful to the Faculty of Science for their financial support through the HDR fund, which facilitated my participation in two significant international conferences: ICORES 2022 in Vienna, Austria, and ICORES 2024 in Rome, Italy.

I sincerely appreciate the computational resources provided by the UTS eResearch High-Performance Computer Cluster, which have significantly enhanced the efficiency of my research endeavours.

To my family, whose constant support has been my source of strength, I owe a debt of gratitude. I am profoundly grateful to my parents for their steadfast encouragement. I am equally grateful to my wife for her unwavering support during the challenging days when my dedication to my research made me less available to her. Her emotional support and encouragement enabled me to give my best to my work. It is to them that I dedicate this thesis.

Alireza Etminaniesfahani
November 26, 2024
Sydney, Australia

List of publications

Published papers:

- Publications related to Chapter 3
  - ABFIA: A hybrid algorithm based on artificial bee colony and Fibonacci indicator algorithm A Etminaniesfahani, H Gu, A Salehipour - Journal of Computational Science, 2022
  - An Efficient Combinatorial ABFIA (CABFIA) for the Resource-Constrained Project Scheduling Problem - Under review
- Publications related to Chapter 4
  - An efficient relax-and-solve method for the multi-mode resource constrained project scheduling problem A Etminaniesfahani, H Gu, LM Naeni, A Salehipour - Annals of Operations Research, 2023
- Publications related to Chapter 5
  - A Forward–Backward Relax-and-Solve Algorithm for the Resource-Constrained Project Scheduling Problem A Etminaniesfahani, H Gu, LM Naeni, A Salehipour - SN Computer Science, 2022
  - An Efficient Relax-and-Solve Algorithm for the Resource-Constrained Project Scheduling Problem. A Etminaniesfahani, H Gu, A Salehipour - ICORES, 2022
  - An Efficient Approximate Dynamic Programming Approach for Resource-Constrained Project Scheduling with Uncertain Task Duration. Etminaniesfahani, A.; Gu, H.; Naeni, L. and Salehipour, A. - ICORES, 2024
  - An Efficient Approximate Dynamic Programming Approach for Resource-Constrained Project Scheduling with Uncertain Task Duration - Under review (Book chapter);
- Other publications during the candidature
  - Gu, H., Etminaniesfahani, A., Salehipour, A. (2021). A Bayesian Optimisation Approach for Multidimensional Knapsack Problem. In: Dorronsoro, B., Amodeo, L., Pavone, M., Ruiz, P. (eds) Optimization and Learning. OLA 2021. Communications in Computer and Information Science, vol 1443.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimisation problems can be broadly categorised into two main types, i.e., real-valued optimisation and integer optimisation. Real-valued optimisation involves problems where the continuous decision variables can take real values. These problems are commonly used in engineering and economics problems [1]. Integer Optimisation, on the other hand, deals with problems where the decision variables are restricted to integer values. Integer optimisation is more complex than real-valued optimisation due to the discrete nature of the solution space. Combinatorial optimisation is a subset of integer optimisation focusing on finding the optimal arrangement or selection from a finite set of items. These problems often involve binary variables and are characterised by their combinatorial complexity, as seen in classic problems like the travelling salesman problem or the knapsack problem. The complexity of combinatorial optimisation problems typically necessitates using heuristic and metaheuristic approaches to find optimal or near-optimal solutions in a reasonable time.

The resource-constrained project scheduling problem (RCPSP) is a notably challenging combinatorial optimisation problem that captures the interest of both researchers and practitioners. Its applications range over various domains, including construction, mining operations, and manufacturing. Over the last five decades, project management and scheduling have evolved into pivotal elements in research and operational research practices. The conventional RCPSP emerges from practical scenarios where dependent tasks must be executed, utilising limited resources. This problem poses significant complexities and has become a prominent area of study.

This thesis concerns RCPSP and its two extensions,i.e., the multi-mode resource-constrained project scheduling problem (MRCPSP), which has multiple task execution modes and resource-constrained project scheduling problem with stochastic task duration (SRCPSP). The following explains why the problems considered in this thesis are important.

The RCPSP has proven to be a Non-deterministic Polynomial-time hard (Np-hard) [2] problem. An NP-hard problem is defined as a problem that is at least as difficult as the hardest problems in Non-deterministic Polynomial-time (NP). NP is a class of problems for which a proposed solution can be verified in a polynomial-time, meaning that the time required to check the solution increases at most as a polynomial function of the input size, reflecting a measure of computational complexity [3].

Considering multiple modes for executing tasks in an MRCPSP and the availability of more than two types of non-renewable resources makes this problem Np-complete. An NP-complete

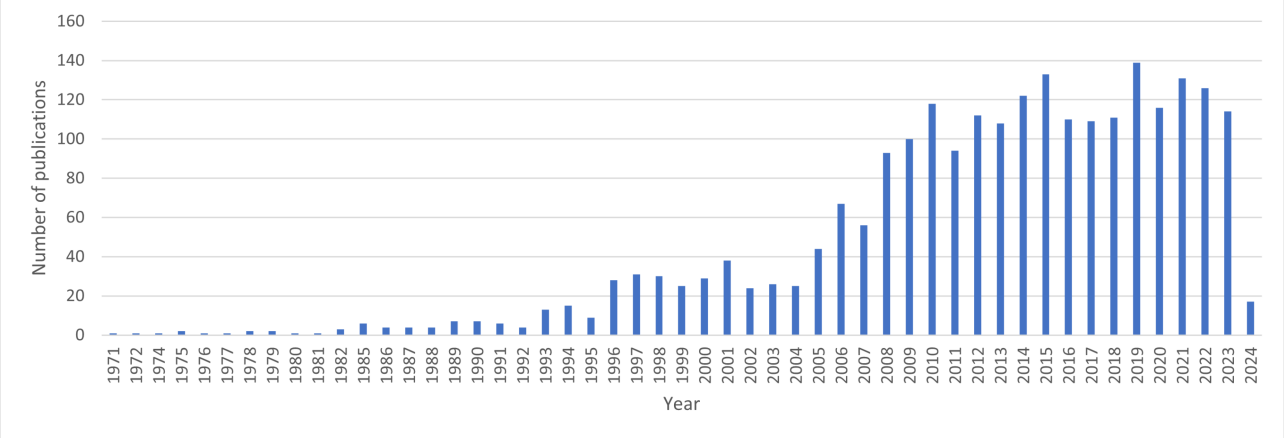

Figure 1.1: Number of publications about RCPSPs

problem is a problem that is both NP and NP-hard. Even finding a feasible solution for this problem can be very challenging [4]. Throughout the execution of a project, task implementations can encounter significant challenges arising from the variability in the time required to complete tasks due to human factors, resource availability, unforeseen difficulties, external dependencies, or fluctuations in productivity. RCPSPs with uncertain task durations are challenging problems, identified as SRCPSPs. This thesis addresses the challenges of RCPSP, MRCPSP, and SRCPSP by proposing different metaheuristic global optimisation, matheuristics, and approximate dynamic programming approaches.

The remainder of this chapter is organised as follows. Section 1.1 briefly overviews the first problem I considered, RCPSP. Then, Section 1.2 touches on the second problem, MRCPSP. In Section 1.3, I consider the third problem, SRCPSP, which focuses on uncertainties in task duration. Finally, Section 1.5 provides a summary of the main contributions of this thesis.

1.1 Resource-constrained project scheduling problem

Scheduling a large project under precedence and resource constraints has always been challenging for project managers. A common source of project delays is the lack of an efficient project scheduling technique [5]. For more than 70 years, the critical path method (CPM) has been one of the most extensively used planning and scheduling approaches. CPM assumes unlimited resources are available. Therefore, by considering precedence constraints, the CPM may provide a lower bound for the project duration.

The resource-constrained project scheduling problem (RCPSP) was introduced by Pritsker et al. in 1969 [6]. In addition to precedence constraints, the authors considered the restriction of resource availability over time. Since then, the RCPSP and its extensions have been used in a wide range of practical applications in diverse industries such as supply chain [7], mining [8] and manufacturing [9]. In recent years, the amount of research on RCPSPs has increased significantly. Figure 1.1 shows the total number of publications by searching the keyword "Resource-constrained project scheduling problem" from Scopus.

The motivations for researching the RCPSP in this thesis stem from its extensive range of real-world applications. Additionally, the RCPSP presents significant computational challenges due to its classification as an NP-hard problem, making it a complex and interesting subject of study [10]. The difficulty in finding optimal solutions within a reasonable time highlights the

need for advanced algorithms and innovative approaches.

RCPSP's solution methods are generally divided into exact algorithms and heuristics-based approaches [11]. although the optimality of the solutions in exact methods is guaranteed, it is at the expense of increasing solution time, which makes exact methods impracticable in solving large-scale problems [12]. Approaches based on heuristics can overcome the shortcomings of exact methods in computation time, although obtaining an optimal solution cannot be guaranteed [13]. There are various open-source [14] and commercial solvers such as CPLEX CP optimizer [15] that can be used for solving optimisation problems, including RCPSP, but the efficiency of those solvers typically decreases when the size of instances increases. Various heuristic-based solution techniques have been proposed for solving challenging instances over the last five decades. That, along with the inefficiency of the exact solvers, motivated us to propose methods that employ the CPLEX CP optimizer within the metaheuristics and matheuristic algorithms to effectively and efficiently solve RCPSP.

The motivation for using the Artificial Bee Colony (ABC) algorithm in this thesis is that the previous research on the ABC algorithm has indicated its effectiveness in handling high-dimensional and complex optimisation problems. ABC is a renowned metaheuristic approach widely utilised for tackling diverse optimisation problems, including the RCPSP[16]. Although acknowledged for its robust global search capabilities, the ABC algorithm suffers from slow convergence. This was the motivation behind designing an algorithm that could take advantage of the strong global search of ABC and improve its convergence speed. Chapter 3 introduces a novel extension to the ABC algorithm that utilises its global search capability while improving convergence speed. Subsequently, I formulate a new heuristic based on this enhanced algorithm and take advantage of CPLEX CP optimizer to address the RCPSP to minimise the makespan of the projects.

The research of Chapter 3 was published in the Journal of Computational Science [17] under the title "ABFIA: A hybrid algorithm based on artificial bee colony and Fibonacci indicator algorithm". The extension of the algorithm for solving the RCPSP was submitted to the peer-reviewed journal under the title "An Efficient Combinatorial ABFIA (CABFIA) for the Resource-Constrained Project Scheduling Problem".

1.2 Multi-mode resource-constrained project scheduling problem

The classic RCPSP is a problem of scheduling a set of tasks with limited availability of resources and the existence of precedence relationships among the tasks [6], [18], [19]. The aim is to determine the start time of the tasks to minimise the makespan of the project while all precedence relationships are satisfied. A generalised variant of the RCPSP in which there is more than one possible mode for executing the tasks is called the multi-mode resource-constrained project scheduling problem, or MRCPSP for short. In the MRCPSP, the duration of a task depends on the level and type of resources used in a given mode.

In general, resources can be classified as (*i*) renewable, (*ii*) non-renewable, and (*iii*) doubly restricted [20]. Renewable resources include resources employed to complete a task and will be available upon completion. Examples of renewable resources include labour and machines. Non-renewable resources, on the other hand, are not typically available once consumed for executing a task. A typical example includes raw materials. Doubly restricted resources can

be considered resource limitations in both a period and over the project horizon. The doubly restricted resources can be easily modelled as a combination of renewable and non-renewable resources.

In Chapter 4, the general MRCPSP is studied, which includes renewable and non-renewable resources and aims to minimise the makespan. According to the classification scheme presented in [21], this problem is denoted as $m, 1T|cpm, disk, mu|C_{max}$ [21].

The motivations for considering the MRCPSP in this thesis are rooted in its enhanced practicality and flexibility compared to the traditional RCPSP. The MRCPSP allows for the execution of tasks in multiple modes, each with different resource requirements and durations, offering a more realistic and flexible model that better reflects the complexities of real-world projects. This adaptability makes the MRCPSP a more applicable and robust framework for addressing various scheduling scenarios. Additionally, the MRCPSP is NP-complete, which indicates its computational challenges and the necessity for innovative algorithmic solutions.

Exact methods can only solve small-sized instances of MRCPSP in a reasonable computation time [22]. For the last few decades, several heuristic and metaheuristic methods have been proposed to improve the solving strategies of MRCPSP, and several benchmark instances have been generated to test and compare the performance of these methods [23], [24]. Nonetheless, many benchmark instances still exist for which the available methods do not deliver quality solutions [25]. The recently defined Relax-and-solve (R&S) has been successfully applied for combinatorial optimisation problems such as job-shop scheduling and aircraft landing problems [26] based on the idea of iterative relaxing and solving a problem. This motivated us to develop new relaxation methods to handle MRCPSPs.

The research in Chapter 4 was published in the Journal of Annals of Operations Research under the title "An efficient relax-and-solve method for the multi-mode resource-constrained project scheduling problem" [27].

1.3 Stochastic resource-constrained project scheduling problem

In recent decades, various heuristics, metaheuristics, and exact techniques for RCPSPs have emerged. RCPSPs typically operate under the assumption that resource demands and capacities remain constant throughout the project duration. However, this assumption is overly restrictive in dynamic project environments, e.g., in practical scenarios, the processing times or resource demand for tasks may vary over time. Traditional optimisation methods often need to be more practical when faced with uncertain real-life projects. Consequently, there is a growing demand for new approaches for these complex scenarios.

In project scheduling, managing uncertainties is challenging, even for experienced project managers [28], [29]. This has led to the development of a new research area focusing on the stochastic resource-constrained project scheduling problem (SRCPSP), which extends the RCPSP to include projects with stochastic task duration and aims to minimise the expected makespan.

The motivation for considering the SRCPSP in this thesis is driven by the increasing complexity and uncertainty in project management, which has exposed the limitations of traditional deterministic scheduling models. The SRCPSP offers a more realistic and practical approach by incorporating probabilistic elements into the scheduling process, allowing for better handling

of uncertainties such as unpredictable task durations.

Most of the approaches in solving SRCPSPs provide a sequence of all tasks at time zero without observing early task duration and are considered open-loop policies. They are static solutions that are not updated during real-time execution. However, one alternative approach is to make schedules using dynamic programming (DP) in a closed-loop policy [30].

In the closed-loop policy, instead of optimising the task sequence beforehand, a decision is made based on the available information during the execution of the project. This dynamic and flexible approach allows decision-makers to utilise new information that arises between decision points [30]. However, despite its theoretical appeal, closed-loop policies for SRCPSP have been considered computationally intractable[31]. Using priority rules to solve SRCPSP has been investigated in [32], [33], demonstrating its efficiency in addressing this problem type. Furthermore, it has been reported in [34] that a simple policy based on the solution of the so-called average project works well in most cases. The average project is a deterministic RCPSP model in which each task has the mean value as task duration. Since the deterministic RCPSP is solved only once, this approach is very efficient compared to the closed-loop approach. However, when uncertainties are significant, the performance of this approach deteriorates. This insight motivated us to design a novel, approximated dynamic programming approach. Our method incorporates priority rules and integrates information from average projects within a closed-loop policy framework, aiming to solve SRCPSP efficiently.

The research in Chapter 5 was presented in the ICORES2024 and also included in the refereed conference proceedings as "Etminaniesfahani, A.; Gu, H.; Naeni, L. and Salehipour, A. (2024). An Efficient Approximate Dynamic Programming Approach for Resource-Constrained Project Scheduling with Uncertain Task Duration. In Proceedings of the 13th International Conference on Operations Research and Enterprise Systems, ISBN 978-989-758-681-1, ISSN 2184-4372, pages 261-268". I also extended this work to be included in a book chapter. The output of Section 5.5 was presented at the 11th International Conference on Operations Research and Enterprise Systems and also included in the refereed conference proceedings as "Etminaniesfahani, A.; Gu, H. and Salehipour, A. (2022). An Efficient Relax-and-Solve Algorithm for the Resource-Constrained Project Scheduling Problem. In Proceedings of the 11th International Conference on Operations Research and Enterprise Systems, ISBN 978-989-758-548-7, ISSN 2184-4372, pages 271-277"[35]. Also, I modified the algorithm and published it in the journal of SN Computer Science [36] under the title "A Forward–Backward Relax-and-Solve Algorithm for the Resource-Constrained Project Scheduling Problem".

1.4 Objectives

The primary objective of this thesis in addressing the RCPSP is to develop a novel metaheuristic global optimisation algorithm that can solve RCPSP instances efficiently. This approach aims to overcome the limitations of existing methods by providing high-quality solutions. Another key objective is to leverage the power of commercial tools like the CPLEX CP Optimizer to generate superior results comparing state-of-the-art methods for RCPSP. By integrating this advanced optimisation tool within the proposed framework, the thesis seeks to achieve solutions that are not only optimal or near-optimal but also computationally feasible for large-scale and complex project scheduling scenarios. The thesis also aims to develop an efficient method for generating feasible solutions to RCPSP, which can be extended to solve more general problems such as MRCPSP and SRCPSP.

In addressing the MRCPSP, this thesis sets the objective of effectively handling challenging instances characterised by very tight non-renewable resources. The goal is to develop methods to manage resource constraints while ensuring that project schedules remain feasible. Additionally, the thesis aims to find an efficient relaxation method and incorporate CPLEX CP Optimizer within a heuristic framework. Another objective is to improve the best-known solutions for standard MRCPSP benchmarks.

The objectives for solving the SRCPSP in this thesis include developing a computationally tractable dynamic programming approach that can efficiently handle the stochastic nature of SRCPSP. This objective focuses on creating a method that is both practical and scalable. Furthermore, the thesis aims to identify the project characteristics and priority rules that significantly impact the solution approaches for solving SRCPSP. By understanding these factors, the research seeks to enhance the effectiveness of existing solution methods and provide insights into the critical elements that drive successful project scheduling under uncertainty. The thesis also seeks to deliver superior solutions for standard SRCPSP benchmark instances.

1.5 Contributions

Below is a summary of the contributions of this thesis.

For solving RCPSP, the main contribution of the study in Chapter 3 is to propose a novel metaheuristic global optimisation algorithm and then extend the algorithm's idea to solve the RCPSP. In particular,

- taking advantage of the fast convergence speed of FIA, which is based on the line search method and the robust search capability of the ABC.
- proposing a strategy to hybridise the ABC and FIA methods.
- providing quality solutions that can outperform the stand-alone ABC, as well as the state-of-the-art variants of the ABC and a number of available metaheuristics.
- proposing the Combinatorial ABFIA, an efficient global optimisation approach for solving combinatorial optimisation.
- proposing CFIA, a global optimisation approach based on a line search approach, for solving combinatorial optimisation.
- embedding forward-backward improvement in the CABFIA.
- utilising CPLEX CP optimizer in the ABC to improve the local search capability of the ABC.
- providing quality solutions that can outperform the stand-alone ABC and CFIA, as well as the variants of the Bee algorithms and the state-of-the-art algorithms in solving RCPSP.

For solving MRCPSP, the main contribution of this thesis in Chapter 4 is to propose a novel matheuristic for decomposing and solving the MRCPSP using a constraint programming solver. In particular,

- proposing a novel R&S matheuristic for MRCPSP;
- developing a mode selection technique to check the feasibility of MRCPSP and to generate initial solutions;

- proposing a novel technique to create relaxed problems,
- utilising the Cplex CP optimizer in a heuristic framework,
- delivering superior solutions for the standard MRCPSP benchmark instances with up to 100 tasks, nine modes, four renewable and four non-renewable resources [24], [37].
- improving the best-known solutions for six instances of MMLIB50, 32 instances of MMLIB100, and 614 instances of MMIB+.

The study's main contribution in Chapter 5 to solving SRCPSP is to develop an efficient average-project-based approximate dynamic programming (A-ADP) approach. In particular,

- reducing the computational burden of closed-loop policy using the solution obtained from the average project.
- proposing a novel R&S matheuristic to solve an average model of large instances;
- proposing the R&S enhanced by forward and backward improvement,
- employing CPLEX CP optimizer to create schedules in both forward and backward passes, whereas the CPLEX CP optimizer is usually employed to generate schedules in the forward pass,
- delivering superior solutions for the 1560 standard RCPSP benchmark instances with up to 120 tasks[37].
- identifying the project characteristic that has the most significant impact on the performance of the A-ADP approach.
- investigating the impact of minimum slack (MSLK) priority rule on the proposed A-ADP approach.
- providing competitive results to the state-of-the-art algorithms for instances with 30, 60 and 120 tasks in a short computational time.
- Additional Research Contributions

  The research in the candidature also resulted in additional research that was presented in "Optimization and Learning: 4th International Conference, OLA 2021, Catania, Italy, June 21-23, 2021, Proceedings 4", and also included in the refereed conference proceedings as "Gu, H., Etminaniesfahani, A., Salehipour, A. (2021). A Bayesian Optimisation Approach for Multidimensional Knapsack Problem. In: Dorronsoro, B., Amodeo, L., Pavone, M., Ruiz, P. (eds) Optimization and Learning. OLA 2021. Communications in Computer and Information Science, vol 1443. Springer, Cham.".

Chapter 2

Literature Review

To provide a comprehensive summary of our study, this chapter explores the research most pertinent to the optimisation problems examined in this thesis. First, this chapter briefly overviews metaheuristic algorithms for solving optimisation problems. Then the literature relevant to the research for solving RCPSP (Chapter 3 and section 5.5), MRCPSP (Chapter 4), and SRCPSP (Chapter 5), are reviewed.

2.1 Metaheuristic approaches for solving optimisation problems

In 1986, Fred Glover invented the term "metaheuristic" [38] to represent a heuristic approach that is not problem-specific. Randomness is the key feature of these algorithms to escape from the local optimal solutions and plays a crucial role in solving most nonlinear optimisation problems, which are typically NP-hard [39]. However, the randomised structure of such methods can make them unreliable in producing a consistent result in each run [40]. The "no free lunch" theorem proves that one metaheuristic cannot be claimed as the best optimisation algorithm in solving all types of problems [41]. According to this theorem, one algorithm may be best suited for a set of problems while showing poor results on another group of test problems. As a result, the research area of metaheuristics has been highly active, competitive, and challenging [42].

In this section, I first discuss various classes of metaheuristic algorithms.

2.1.1 Four classes of metaheuristics

The metaheuristic algorithms can be classified into the following groups [43]:

Physical-based methods. Examples include simulated annealing (SA) [44], gravitational search algorithm (GCA) [45] and galaxy-based search algorithm (GbSA)[46]. These methods use rules from physical phenomena to guide the algorithm towards obtaining quality solutions. These algorithms provide promising results in solving different optimisation problems. However, they are more efficient in optimising unimodal problems, where they show their strong local search capability [47]. The unimodal optimisation problems consist of objective functions with only one minimum or one maximum value.

Evolutionary algorithms. These algorithms are inspired by laws of evolution in nature, are conceptually simple, and are easy to understand. These algorithms are flexible in solving various types of optimisation problems, including nonlinear and NP-hard problems. A huge number of publications in this area indicates the attractiveness of evolutionary algorithms [48]. The evolutionary algorithms use a set of random populations to start searching the feasible area. By combining the best individuals to produce new generations, they obtain new solutions and continue the search. The genetic algorithm (GA) [49], differential evolution (DE) [50] and the evolutionary strategy (ES) [51] are among the most popular evolutionary algorithms in solving various types of optimisation problems. In comparison with the GA in solving nonlinear optimisation problems, the ES has the difficulty of attaining an optimal convergence rate if there is a large number of constraints [52]. The DE may be chosen over the GA in solving multi-objective optimisation problems because the DE explores the search space more efficiently [53].

Swarm intelligence methods. These methods, which are based on the collaboration of simple agents to find quality solutions, are commonly inspired by the colonies and folks in nature. The ant colony optimisation (ACO) [54], the particle swarm optimisation (PSO) [55] and the ABC algorithm [56], which is inspired by the foraging behavior of honey bees, are some of the most applied swarm intelligence methods. The ACO algorithm is well suited for discrete optimisation problems [57], and the PSO is used in multi-objective, dynamic optimisation, and constraint handling settings [58]. Compared with the ABC, the PSO is a faster algorithm for solving nonlinear and high-dimensional optimisation problems. The ABC, however, has a robust global search ability and is suitable for problems with many local optima [59]. Other recently proposed swarm intelligence algorithms such as Dragonfly algorithm (DA) [60] Salp swarm algorithm (SSA) [61], the ant lion optimiser (ALO) [62], the grey wolf optimiser (GWO) [42], and the whale optimisation algorithm (WOA) [63] have their own limitations.

For example, the ALO, which is inspired by the foraging behaviour of ant lion's larvae, demands long run times because of the random walk process incorporated in the algorithm [64]. The main drawback of the WOA, which is inspired by the hunting behaviour of humpback whales, is the poor exploration capability [65]. The GWO is motivated by the hunting behaviour of grey wolves in nature. The main shortcomings of the GWO algorithm are slow convergence rate and low accuracy of the algorithm [66].

Other methods. Other famous algorithms in solving optimisation problems include harmony search (HS) [67], tabu search (TS) [68], Guided Local Search (GLS) [69], and Variable Neighborhood Search (VNS)[70].

The HS is influenced by musicians' improvisational processes and offers many appealing features, including ease of implementation and a small number of parameters. The main weakness of this algorithm is its premature convergence [71]. The TS mimics a human-like approach to problem-solving by using memory structures to avoid revisiting previously explored inferior solutions[68]. Another human-based algorithm, the learner performance based behavior algorithm (LPB), is inspired by the process of accepting graduates from high schools at universities[72].

Guided Local Search (GLS), is designed to enhance the performance of local search algorithms across various combinatorial optimisation problems. The primary innovation of GLS lies in its strategic use of problem-specific and search-related information to guide local search heuristics effectively through the complex landscapes of NP-hard optimisation problems. This approach focuses on optimising the search path and distributing search efforts to maximise the exploration of potentially optimal areas within the search space [69].

Variable Neighborhood Search (VNS), introduced in 1997, is a metaheuristic algorithm that enhances the search for better solutions by systematically changing neighbourhood structures within the local search process[70]. This approach seeks to improve the solution by achieving a better objective value or satisfying other criteria. VNS is founded on three key insights: the local optimum of one neighbourhood structure may not be optimal in another, the global minimum is always a local minimum in some neighbourhoods, and local minima produced by different neighbourhood structures are often close to each other. The foundational principles and applications of VNS, particularly in addressing combinatorial and global optimisation problems, are comprehensively discussed in [73].

The recently published Fibonacci indicator algorithm (FIA) [1] is a metaheuristic algorithm inspired by the Fibonacci retracement, a characteristic that stock market traders use to predict the best moment to speculate in the market. Fluctuations in stock prices in a time interval lead to a large number of local minima and maxima. In the FIA, the objective function value is considered as the price of a stock, and the aim is to predict the local optimum values of the objective function.

The motivation for using FIA is, among the mentioned metaheuristic algorithms, ABC has a strong global search capability but suffers from weak local search. Conversely, the FIA has strong local searchability and can be combined with ABC to obtain a strong hybrid algorithm. The following details these two methods.

2.1.2 The Artificial bee colony algorithm

The Artificial bee colony algorithm (ABC) algorithm was proposed by [56] to optimise numerical problems. The operation of the ABC is based on the behaviour of honey bee swarms in nature. A bee colony consists of "employed" bees, "onlooker" bees, and "scout" bees. The onlooker and scout bees may be referred to as the "unemployed" bees. The "employed" bees evaluate food sources based on the quality and distance to the hive and share that information with the unemployed bees within the dancing areas in the hive. Around 90-95% of the unemployed bees are onlooker bees, who aim to improve the food sources. The onlooker bees watch many dances in the hive to choose the best food source and then employ themselves on that. The scout bees randomly search the environment.

There are four main phases in the ABC algorithm. In the first phase, which is also called the "initialisation" phase, the ABC creates a population of SN solutions. Each solution $x^i = (x_1^i, x_2^i, \ldots, x_D^i)$, $i = 1, 2, \ldots, SN$ is called a food source with fitness value $F(x^i)$. The j^{th} dimension of a food source x^i is generated by

$$x_j^i = l_j + rand[0,1] \cdot (u_j - l_j), \quad j = 1, \ldots, D \tag{2.1}$$

In the second phase, which is called the "employed bee" phase, new solutions are generated based on the current solutions. Let V_j^i be the j^{th} dimension of the i^{th} candidate solution. The ABC generates V_j^i from the previous solutions as follows:

$$V_j^i = x_j^i + \psi(x_j^i - x_j^k), j = 1, 2, \ldots, D, \tag{2.2}$$

where $k \neq i$ is randomly selected from $\{1, 2, \ldots, SN\}$, and ψ is a uniform random number in the range $[-1, 1]$. If the best solution is improved, the employed bees update their position, i.e., the area they search, by replacing x^i by V^i.

In the third phase of the ABC, which is called the "onlooker bee" phase, solutions are analysed and selected by the onlooker bees: A roulette wheel selection mechanism is employed to select solution x^i with probability.

$$P_i = 1 - \frac{F(x^i)}{\sum_{j=1}^{SN} F(x^j)}. \tag{2.3}$$

According to Equation (2.3), the better the fitness value of the i^{th} solution, the greater the chance of x^i to be used to generate a new solution. Next, a local search is performed around each of the selected solutions. Given the i^{th} solution is selected, the algorithm uses Equation (2.2) to perform the local search around that solution, and once a superior solution (with an improved fitness value) is obtained, the position of the solution is updated accordingly. The local search is performed for a predetermined number of trials, and if the local search fails to deliver a new solution, meaning that no updated position is recorded, the ABC algorithm starts the fourth phase.

In the fourth phase, which is called the "scout bee" phase, the solution is replaced with a new random solution generated according to Equation (2.1). The pseudo-code of the ABC [74] is summarised in Algorithm 1.

The ABC algorithm suffers from major limitations including a poor searching strategy in the onlooker bee phase and a lack of an effective strategy to avoid being trapped in low-quality solutions in the scout bee phase.

Firstly, the ABC algorithm converges very slowly, and the local and global search abilities are not balanced [74], [75]. The poor exploitation and slow convergence of the ABC roots in the execution of the scout bee phase right after the local search performed by the onlooker bees, which disrupts the convergence [76]. In the ABC algorithm, the onlooker bees aim to improve the solutions by focusing on better solutions, i.e., they select solutions according to their quality (see Equation 2.3).

Because most onlooker bees concentrate on the neighbourhood of high-quality solutions, the remainder of the solutions might not be sought, implying that the ABC may not effectively perform the search around other solutions. Consequently, the exploration capability of the ABC algorithm is adversely impacted [77], and this becomes worse when the algorithm finds a solution with a significantly superior fitness value during the onlooker bee phase. Secondly, the scout bee phase of the ABC algorithm performs a random search. A random search usually demands a large number of trials and has been documented not to be a successful strategy to escape from local optima [78], [79].

To overcome those limitations, various techniques and hybridisation methods were proposed to improve the ABC algorithm. For example, in the ABCx, the ABC algorithm was improved by employing novel search techniques in the employed and onlooker bee phases [80]. The improved ABC algorithm (IABC) uses the mutation and crossover operations of a differential evolution (DE) algorithm to enhance the ABC's exploitation capability [81], and the ABC-based fitness weighted search (ABCFWS) algorithm considers weights for fitness values of the food source and selected neighbour food sources to obtain a more balanced exploration and exploitation [82]. The global-best-solution guided ABC (GABC) by [83] aims to improve the convergence rate and exploitation ability of the ABC towards obtaining global optimum solutions. In that method, certain information of the best solution is used during the search operation. Gao and Liu proposed learning strategies in the modified ABC (MABC) algorithm [84]. They also proposed

Algorithm 1 The ABC algorithm

Input: Objective function $F(x)$, SN
Output: A feasible solution
Phase 1: Initialisation
Use Equation 2.1 to create SN initial solution (food sources); set $trial_i = 0$ for each solution.

while *the stopping criterion is not met* **do**
  Phase 2: The employed bee phase
  for $i = 1, 2, \ldots, SN$ **do**
    Generate a new candidate solution V^i using Equation (2.2);
    if $F(V^i) \leq F(x_i)$ **then** $x^i = V^i$;
    if $F(V^i) > F(x^i)$ **then** $trial_i = trial_i + 1$;
    else $trial_i = 0$;
  end
  Phase 3: The onlooker bee phase
  Calculate the probability P_i for solution x^i using Equation (2.3);
  for $i = 1, 2, \ldots, SN$ **do**
    if $rand(0, 1) \leq P_i$ **then**
      Generate a new candidate solution V^i using Equation (2.2) for the onlooker bees;
      if $F(V^i) \leq F(x^i)$ **then** $x^i = V^i$;
      if $F(V^i) > F(x^i)$ **then** $trial_i = trial_i + 1$;
      else $trial_i = 0$;
    end
  end
  Phase 4: The scout bee phase
  for $i = 1, 2, \ldots, SN$ **do**
    if $trial_i > limit$ **then**
      Replace x^i by a new randomly produced candidate solution using Equation (2.1);
    end
  end
end
Return: x;

novel search strategies to balance the original ABC algorithm's exploration and exploitation capabilities. Based on the solution update rule of DE algorithm, ABC-best has been proposed by [75]. In [59], the PSO was used in the employed and onlooker bee phases for continuous optimisation problems. The resulting hybrid algorithm showed promising results in unimodal problems compared to the stand-alone ABC and PSO algorithms. In another attempt, the PSO and ABC were used in the particle-bee algorithm [85] to solve the construction site layout optimisation problem. In the velocity-based ABC algorithm proposed for high-dimensional continuous optimisation problems, the PSO algorithm was employed to guide the search in the onlooker bee phase [86]. In another attempt, the crossover operator of the GA was applied after the employed bee phase was performed to improve the solutions before the onlooker bee phase started [87].

It is observed from the literature hybridising the ABC with existing heuristics and metaheuristics is an effective strategy for improving the performance of the ABC algorithm. To escape from being trapped in low-quality local optima, the simulated annealing (SA) was applied to the employed bee phase to enhance the exploitation of the ABC algorithm [88]. That hybrid algorithm was also shown to outperform the stand-alone ABC algorithm. In [89], a combination of the ABC with SA was introduced in which the acceptance rule allows for accepting inferior solutions with a controlled probability. The probability of accepting an inferior candidate solution is dynamically reduced during the algorithm's operation.

2.1.3 The Fibonacci indicator algorithm

The FIA is a newly published metaheuristic optimisation method [1], which is inspired by the Fibonacci retracement method, a well-known technique used by stock market traders to predict highly volatile stock prices. Predictions about the time of maximum and minimum cost of stocks are the basic motivation to develop this optimisation algorithm.

In the stock market, technical analysis support and resistance levels are fixed levels for a stock price where it is assumed that the price would begin to stop and reverse. At a support level, the stock price is likely to find support when it falls, which means that rather than breaking through this support level and falling again, the price is more likely to bounce off it. On the contrary, as the price increases, it encounters opposition at the resistance level. The Fibonacci retracement method forecasts a support level, which acts as a local minimum for the stock price, and the resistance level, which presents a local maximum for the stock price.

The above discussion follows that the FIA may predict the local minima in the line search algorithm instead of performing the expensive complete search. Assume that x and x_{best} are two different solutions, and we need to perform a line search on the line defined by these two solutions. Let a fixed number of five solutions be generated as

$$x_i = x + \tau_i(x_{\text{best}} - x), \quad i = 1, \ldots, 5, \tag{2.4}$$

where $\{\tau_i\} = \{50\%, 73.6\%, 88.2\%, 111.8\%, 150\%\}$ is the set of Fibonacci ratios suggested in [1].

Figure 2.1 illustrates how the five candidate solutions are generated using x and x_{best}. The solution with the best fitness value is selected as the result of the line search. It can be seen that this method, which is called the Fibonacci indicator (FI) method, focuses the search more around x_{best} than x.

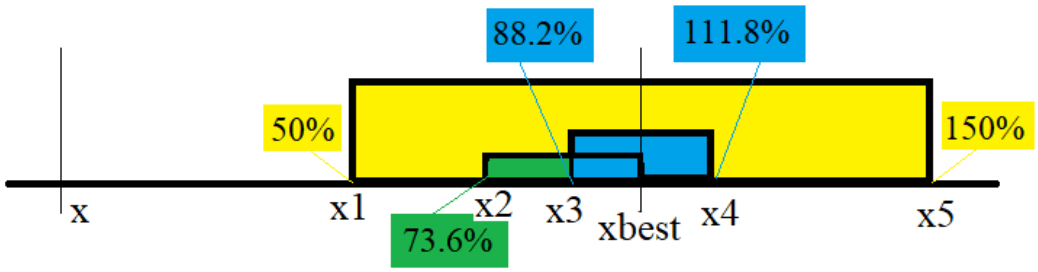

Figure 2.1: Generating solutions $(x_1, x_2, x_3, x_4, x_5)$ using the FI method.

The FIA is a population-based metaheuristic algorithm, where an initial population is comprised of N randomly generated solutions. Solutions in a population are sorted in non-decreasing order of their fitness value (I solve a minimisation problem). The solution with the best fitness value is referred to x_{best} with fitness value g_{best}.

The FIA generates a new solution by selecting two solutions, as parents, from the current population and applying a line search to parent solutions. The current best solution is consistently chosen as a parent, and the next best solution in the population that has not yet been selected is considered the second parent. If the fitness value of the new solution is superior to both parents, the current population is updated by adding the new solution to the population and removing the worst solution from the population. Whenever the current best solution is updated, all solutions in the population are treated as "never have been chosen". A new population is obtained when all the solutions in the current population have been replaced. The update of a population is referred to as Step 1.

If the population cannot be updated, i.e., at least one solution in the population cannot be removed after all solutions in the population have been attempted as parents, a new population is randomly generated.

After obtaining a new population in Step 1, the FIA performs Step 2. The FIA picks two solutions, as parents, to generate a new solution using the FI method. The best solution is always selected as one parent. With probability $1-p$, where p is a given parameter, the second parent is selected from the remaining solutions in the order of non-increasing objective function values. With probability p, the second parent x will be generated by the crossover operation defined as

$$x = (x_1^{r_1}, x_2^{r_2}, \ldots, x_D^{r_D}), \tag{2.5}$$

where r_i, $i = 1, \ldots, D$, is an integer number randomly selected between 1 and N. Step 2 is indeed designed to maintain the diversification of the population by randomly generating some solutions using (2.5) while keeping all solutions generated by line search. This mechanism ensures a good trade-off between global exploration and local exploitation of the FIA.

If the best solution is not improved after $C \in \mathbb{Z}^+$ consecutive trials of Steps 1 and 2, the search is restarted by generating a new population comprised of the current best solution x_{best} and $N-1$ randomly generated solutions.

The algorithm terminates if the number of function evaluations reaches a threshold, which is a parameter of the algorithm. The pseudo-code of the FIA is shown in Algorithm (2).

Algorithm 2 The FIA [1].

Input: Objective function $F(x)$, Parameters N, p, C;

Output: Feasible solutions.
Set $I = 0$;
Randomly generate a population of size N;
while *stopping criterion is not met* **do**
  while $I < C$ **do**
    Search phase:
    Perform Step 1: Improve the population focusing on exploitation;
    Perform Step 2: Improve the population with a balance of exploration and exploitation;
    $I = I + 1$;
    if *the current best solution was updated* **then** $I = 0$;
  end
  Set $I = 0$, and generate a population of size N, including the current best solution and $N - 1$ random solutions;
end
Return: x;

2.2 Resource-constrained project scheduling problem

Scheduling projects under precedence and resource constraints can be challenging for project managers. The lack of an efficient project scheduling technique [5] can be a source of project delays. The critical path method (CPM) has been one of the most extensively used planning and scheduling approaches for many decades. This approach assumes unlimited resources are available and can provide a lower bound for the project duration.

The resource-constrained project scheduling problem (RCPSP) was introduced by Pritsker et al. in 1969 [6]. In addition to precedence constraints, the authors considered the restriction of resource availability over time. Since then, the RCPSP and its extensions have been used in a wide range of practical applications in diverse industries such as supply chain [7], mining [8] and job-shop scheduling [9].

2.2.1 Problem definition

The RCPSP includes a set of n task. Task i has a certain non-negative duration presented by d_i. Precedence constraints also exist between tasks that can be modelled as an activity-on-node network $G = (V, A)$. Every single task relates to a node in the node-set $V = \{1, 2, \ldots, n + 1\}$. Arc (i, j) in the arc set A represents the precedence constraint that task i has to complete before task j can start. Dummy nodes 0 and $n + 1$ are added to the set of nodes to represent the project's start and finish times, respectively. The duration of dummy nodes (tasks), i.e., d_0 and d_{n+1} are 0. Because time cannot reverse, the graph G is always acyclic. A fixed set of renewable resources represented by RR is available. Each resource $k \in RR$ has a non-negative and fixed capacity R_k at each time in planning horizon T. Each task i needs a non-negative amount of r_{ik} of each resource $k \in RR$. Tasks cannot be interrupted once started (non-preemptive).

Figure 2.2, shows a typical example of RCPSP. This example has seven tasks (0 and 8 are dummy start and end tasks, respectively), while the maximum resource availability for any

particular time period is seven units. Each task's processing time and resource requirements are also provided under each task node.

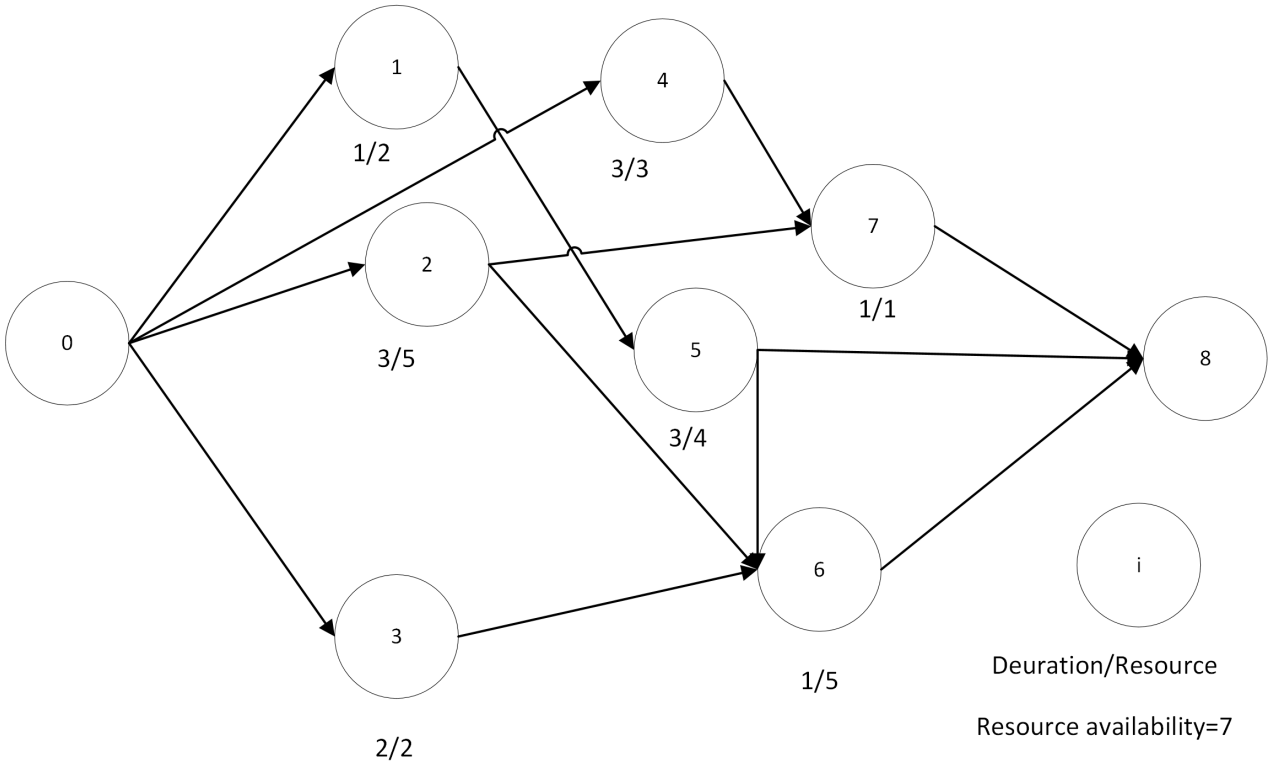

Figure 2.2: RCPSP example

The mathematical formulation of the RCPSP can be expressed as follows, where S_i presents the starting time of the task i. The first dummy task starts at $S_0 = 0$, and the finishing time of the project (makespan) is the finishing time of the last dummy task. Because the duration of dummy tasks equals zero, the project makespan is then equal to S_{n+1}.

$$\min S_{n+1} \tag{2.6}$$

subject to

$$S_j \geq S_i + d_i, \quad \forall (i,j) \in A, \tag{2.7}$$

$$\sum_{i \in \tau(t)} r_{ik} \leq R_k, \forall k \in RR, \forall t \in \{0, \ldots, T-1\}, \tag{2.8}$$

$$\tau(t) = \{i \in V \mid S_i \leq t < S_i + d_i\}, \tag{2.9}$$

$$S_i \in \{0, 1, \ldots, T - d_i\}, \quad \forall i \in V, \tag{2.10}$$

where T is the given upper bound on the project duration.

2.2.2 Solution approaches for resource-constrained project scheduling problem

The RCPSP's vast range of applications, as well as its computational challenges, has attracted the attention of many scholars [90]. The RCPSP's solution methods are generally divided into exact algorithms and heuristics-based approaches [11]. While the optimality of the solutions in exact methods is guaranteed, it is at the expense of increasing solution time, which makes exact methods impracticable in solving large-scale problems [12]. Approaches based on heuristics can overcome the shortcomings of exact methods in computation time, although obtaining an optimal solution cannot be guaranteed [13].

Heuristic approaches in solving RCPSP can be classified into simple priority rule-based (Pr-based) heuristics [91], [92], Metaheuristic, and Matheuristic algorithms. Schedule generation schemes (SGS) are the foundation of most heuristic approaches designed for the RCPSP [93]. Typically, an SGS starts from scratch and progressively develops a feasible schedule by extending a partial schedule. A partial schedule is a scheduling structure where only a subset of project tasks has been scheduled. There are two variants of SGS, i.e., the parallel SGS that performs time incrementation and the serial SGS that performs activity incrementation. The parallel SGS utilises time incrementation. Each iteration g has a scheduling time t_g and a corresponding set of eligible tasks. The eligible set contains all the unscheduled tasks that are eligible for scheduling, given that all their predecessor tasks have been scheduled. Tasks are chosen from this set and initiated at t_g until no further eligible tasks remain. Subsequently, the scheduling scheme steps to the next scheduled time, determined by the earliest completion time of the tasks in progress [93]. The serial SGS, on the other hand, operates over n stages. In every stage, a non-dummy task is chosen from the eligible set and is scheduled at the earliest time that meets both precedence and resource constraints [93].

Forward-backward scheduling is a well-known algorithm that has successfully been combined with other metaheuristics. Forward-backward scheduling techniques utilise an SGS to iteratively schedule the project, switching between forward and backward scheduling modes. In the forward pass, a feasible schedule is generated, and each task starts as early as possible. In the backward pass, tasks are started at the latest time so that all constraints are satisfied and the target finish time of the project is met. Iterative scheduling in the forward and backward passes can improve the results[94]. This procedure terminates when there is no improvement in two successive solutions obtained by forward and backward passes. This iterative process terminates once two consecutive solutions, derived from alternating forward and backward iterations, showcase no improvements. As pointed out by [95], this approach can be hybridised with heuristic algorithms denoted as the forward-backward improvement approach (FBI). Notably, many of the best strategies for resolving RCPSP incorporate this technique [96].

A priority rule (Pr) is a method that assigns a value to each task within the set of eligible tasks in the SGS. It defines whether the tasks with the lowest or the highest assigned value should be chosen. Pr-based heuristics are defined as single-pass and multi-pass methods [93]. The single-pass methods generate a single schedule, while multi-pass methods generate more than one schedule. In multi-pass methods, Prs can be combined with SGS. One of the most commonly used multi-pass methods is the forward-backward scheduling approach [94]. For a review of diverse Prs in solving RCPSPs, see [97].

Metaheuristics are heuristic methods that are not tailored to a specific problem. Metaheuristics include classical metaheuristics (i.e., Simulated Annealing [98], Tabu Seach[99], Genetic Algo-

rithms[100], ABC [56]) and Non-standard metaheuristics. It is referred to [93], [95], [96] for the details of the classification of metaheuristic algorithms. The genetic algorithm (GA) [49] is one of the most used metaheuristic methods to solve the RCPSP [101]. Nonetheless, several studies used FBI with GA [101]–[111]. [112] improved the evolutionary algorithm by FBI in EA(FBI). [113] developed combined FBI and the ant colony optimisation (ACO) [57] to solve RCPSP. Bee algorithms leverage the intelligent foraging behaviours of honey bees to guide the search process. Different models of Bee algorithms [114], such as ABC [56] and Bee swarm optimisation (BSO) [115] have been proposed in the literature. The performance of three algorithms of bee algorithm (BA) [116], ABC [56] and bee swarm optimisation (BSO) [117] was investigated in [16] for solving an RCPSP. They also implemented the variants of the algorithms in which the algorithms were combined with the FBI: BSO-FBI, ABC-FBI, and BA-FBI. The results show that incorporating FBI leads to quality schedules. [118] used FBI with a scatter search metaheuristic. In each iteration, the algorithm reverses the project network. A hybrid FBI with Lagrangian relaxation (LR-FBI) was proposed by [119].

In recent years, there has been a significant increase in the development of new heuristic techniques for the RCPSP. Instead of relying on a single metaheuristic, these new approaches combine different strategies, components, and optimisation techniques and are often called hybrid algorithms. Generally, hybrid algorithms can be classified into two types.

The first type combines metaheuristics with other operations research techniques like constraint programming or tree-based search methods [36], [120]. The second type of hybrid method combines various concepts from different metaheuristics. For instance, it might involve using local search procedures alongside population-based methods [121]. For tackling RCPSP utilising this type of hybrid method, two types of hybrid techniques, namely, Integrative hybrid and Collaborative hybrid, are introduced [122].

The Integrative approach is represented as M1(M2), with M1 as the primary metaheuristic and M2 as the secondary. This approach is a common way to combine metaheuristics by hybridising population-based and local search metaheuristics. Within the Integrative approach, a local search algorithm or a population-based (secondary) metaheuristic is integrated into the primary metaheuristic to enhance the exploration of solutions[16], [123].

The collaborative approach includes parallel and sequential hybrids. The parallel hybrids simultaneously operate two distinct algorithms working in parallel. Each metaheuristic searches within the solution space and shares information with the other to collectively seek an optimal solution denoted as (M1, M2) [124]. On the other hand, sequential hybrids execute the collaborative metaheuristics sequentially, which may involve iterative processes. In this setup, one metaheuristic operates after the other, each using the previous output as its input, denoted as M1+M2 [125].

One successful approach for solving scheduling problems is the matheuristic approach [26], [126]–[129]. Matheuristics are optimisation algorithms that employ mathematical programming in a heuristic framework [130]. The constraint programming (CP) has successfully been used to solve scheduling problems [131]–[133] including the RCPSP [134]–[137]. The relax-and-solve method (R&S) is a recently developed matheuristic for solving large combinatorial optimisation problems and has delivered high-quality solutions for various job-shop scheduling and aircraft landing problems [26], [126]–[129]. This approach improves a given feasible solution by repeatedly relaxing the task order constraints (the execution order) in the current solution and then rescheduling the newly relaxed problem using a local search including a mixed-integer programming (MIP) solver. Although the relaxed problem still involves all tasks and is thus

large, the solution time is significantly reduced in practice. In [126], for nearly 57% of a set of 72 benchmark instances, ranging from 10 to 20 tasks and up to 200 operations, new best solutions were obtained for a variant of the job-shop scheduling problem. In [26], the R&S is employed to solve the aircraft landing problem aiming at minimising the total deviation of aircraft landing time from target arrival times. The relax phase in [26] destructs a sequence of aircraft landings, and the solve phase reschedules the aircraft landings. Utilising R&S, all the best solutions for up to 500 aircraft can be obtained within one minute, which shows the superiority of this algorithm's performance compared with the state-of-the-art algorithms.

The literature has related approaches for solving mixed-integer programming models, including relax-and-fix and fix-and-optimise. The relax-and-fix method involves dividing the binary variables through a rolling time window into two groups: The fixed variables and the optimised variables. The integrality constraint for variables outside of the rolling window is relaxed [138]. On the other hand, the fix-and-optimise method deals with two groups of variables: the fixed and optimised. A key advantage of the fix-and-optimise method is that the solutions obtained are always feasible because the method does not relax integrality constraints [139], [140].

A new algorithm based on R&S was proposed to solve the RCPSP in [35], [141]. The proposed R&S for solving the RCPSP first generates a feasible schedule (a solution). It then improves the solution iteratively in two phases of relax and solve. In the relax phase, the order of some tasks in the solution is relaxed. However, one of the biggest challenges in applying R&S to the RCPSP is the lack of a clear task order or sequence definition in a feasible solution since several tasks can run simultaneously. In [35], [141], time windows are adopted to determine which tasks will undergo rescheduling. The CP constraints, such as "start-at-end" were utilised to preserve the relative task order for tasks outside the window.

In the solve phase, the relaxed problem is solved using CPLEX CP optimizer [15] to create a task execution schedule [141]. The reason for using CP instead of a typical MIP solver is that CP is faster and can obtain high quality solutions for scheduling problems [132], [133], including the RCPSP [134]–[137].

2.3 Multi-mode resource-constrained project scheduling problem

MRCPSP is an extension of RCPSP. In this problem, the duration of each task is related to the level and type of resources allocated to it. The representation of the MRCPSP is closely similar to the RCPSP, with the difference in the availability of mode options. In the following, the MRCPSP is briefly defined, and then the solution approaches for the MRCPSP are reviewed.

2.3.1 Problem definition

The MRCPSP is comprised of a set J of tasks. Each task $i \in J$ must be completed in one mode denoted as m_i. The mode m_i is chosen out of a set of $|M_i|$ different execution modes for task i, i.e., $m_i \in M_i = \{1, \ldots, |M_i|\}$. The non-negative duration of executing task i in mode m_i is denoted by d_{i,m_i}.

Each task must be completed in a unique mode, meaning that when the task starts, its execution mode cannot be changed. Also, each task must be completed without interruption. The MRCPSP can be modelled by using a directed acyclic graph in an activity-on-node network

$G = (V, A)$ [142], where V is the set of nodes and A is the set of arcs. A node corresponds to a single task (for a project with n tasks). To show the project's start time and finish time, the dummy nodes 0 and $n+1$ are added to the set of nodes resulting in $V = \{0, 1, \ldots, n+1\}$. The duration of dummy tasks equals 0, i.e., $d_0 = d_{n+1} = 0$.

Precedence relationships between tasks are represented by the arc set A. Each arc $(i, j) \in A, i, j \in J, i \neq j$ stands for a precedence constraint that ensures task j cannot be started before task i is finished. The availability of RR renewable resources within the planning horizon T is presented by $R^r_{k_r}, k_r = 1, \ldots, RR$. The total amount of each non-renewable resource NR that can be consumed during the course of a project is presented by $R^n_{k_n}, k_n = 1, \ldots, NR$. Each task i that is performed in mode m_i demands a non-negative amount of r^r_{i,m_i,k_r} renewable resource and r^n_{i,m_i,k_n} non-renewable resource.

Let S_i denote the start time of task i and $S_0 = 0$ represent the start time of the project. The makespan of the project is the completion time of the last dummy task, and because the duration of dummy tasks is equal to zero, S_{n+1} is, therefore, the project makespan. The objective of the MRCPSP is to minimise the project completion time or the makespan. Figure 2.3, shows a typical example of MRCPSP. In this example, four tasks with two dummy start and end tasks. This example shows each task can be performed in different modes, and for each mode, there is a fixed value of required resources and duration.

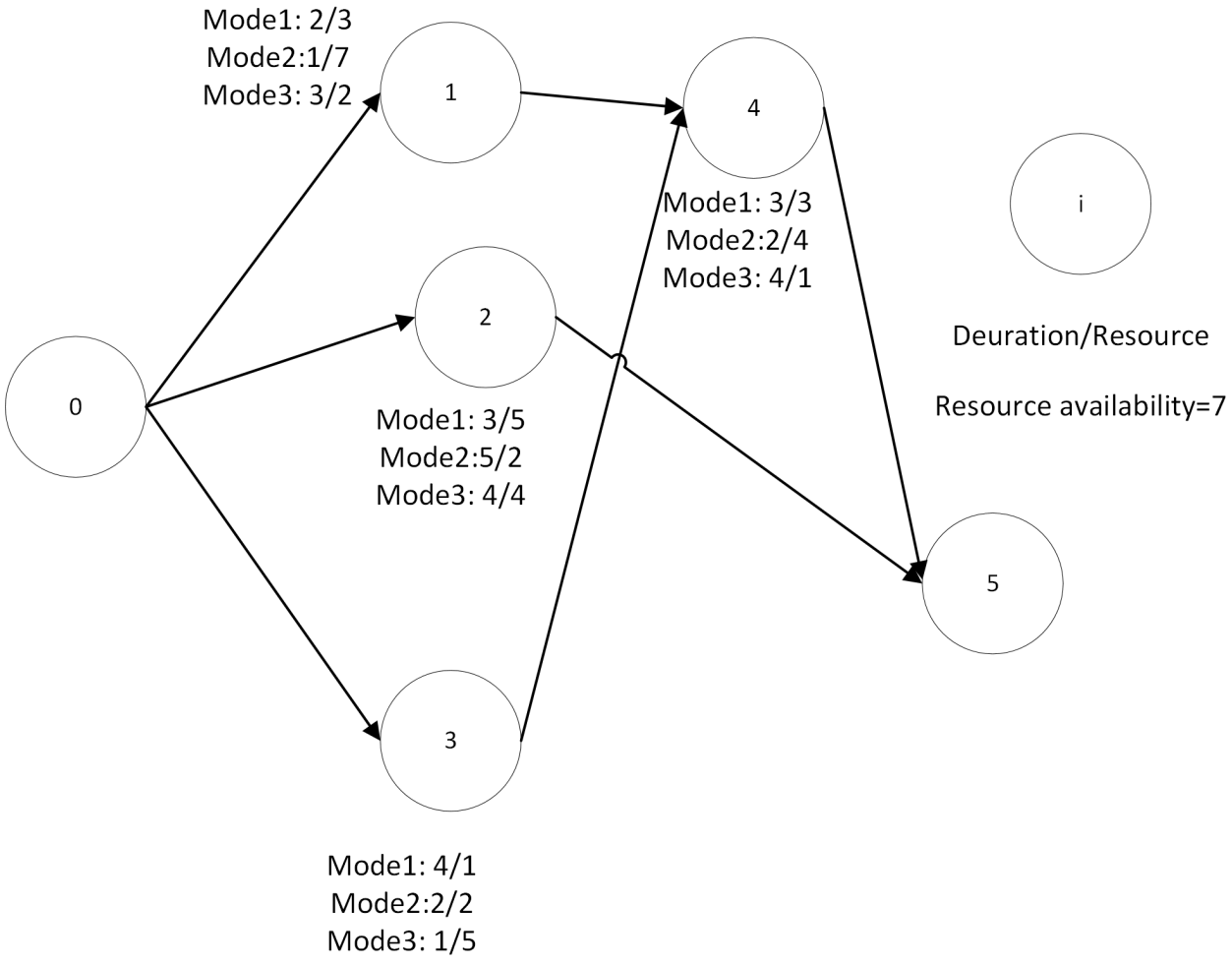

Figure 2.3: MRCPSP example

The MRCPSP can be formulated as follows [143].

$$\min S_{n+1} \tag{2.11}$$

subject to

$$S_j \geq S_i + d_{i,m_i}, \quad \forall i, j \in A, i \neq j, \tag{2.12}$$

$$\sum_{i=0}^{n+1} r^n_{i,m_i,k_n} \leq R^n_{k_n}, \quad \forall k_n \in \{1, \ldots, NR\}, \tag{2.13}$$

$$\sum_{i \in \tau(t)} r^r_{i,m_i,k_r} \leq R^r_{k_r}, \forall k_r \in \{1, \ldots, RR\}, \quad \forall t \in \{0, \ldots, T-1\}, \tag{2.14}$$

$$m_i \in M_i, \quad \forall i \in V, \tag{2.15}$$

$$S_i \in \{0, 1, \ldots, T - d_{i,m_i}\}, \forall i \in V \tag{2.16}$$

where the planning time horizon T, is a given upper bound on the project makespan.

The objective function (2.11) minimises the makespan of the project. Constraint (2.12) imposes the precedence constraints with the zero time lag. Constraints (2.13) and (2.14) ensure that non-renewable and renewable resources are met. In (2.14), $\tau(t) = \{i \in V \mid S_i \leq t < S_i + d_i\}$ is the set of tasks that are active at time t. Constraints (2.15) ensure that one mode is selected for each task. Finally, constraint (2.16) takes a non-negative integer start time for each task.

In this section, I first review solution approaches for MRCPSP. Then, I review the recently developed relax-and-solve method.

2.3.2 Solution approaches for multi-mode resource-constrained project scheduling problem

Solution methods for solving the MRCPSP can be classified into two main approaches, namely, the exact methods and the heuristic and metaheuristic algorithms.

The exact methods for solving the MRCPSP mainly include the CP-based and branch-and-bound (B&B) algorithms. To the best of our knowledge, the first exact solution method for solving the MRCPSP was proposed in [144], which solved an example of a problem with three tasks and five identical resource units of one type. The CP has been employed to solve larger instances, with up to 120 tasks for the instances of RCPSP and with up to 100 tasks for the instances of MRCPSP; see for examples [134]–[137], [145], [146].

The B&B methods were attempted in [147]–[149]. In [149] instances with 12 tasks were optimally solved, and in [147], [148] instances with up to 20 tasks were tackled. In another attempt, a branch-and-cut algorithm was proposed and results were reported for the instances with 20 and 30 tasks [150]. An example of applying a B&B to solve MRCPSP with non-preemptive activity splitting is presented in [22]. It should be noted that the exact optimisation methods were shown to be unable to solve instances with more than 20 tasks and three modes for each

task in a reasonable amount of time [148]. In general, solving MRCPSP by using exact methods is very time-consuming and may not be a preferred approach for large instances associated with real-world projects [151]. The efficiency of solvers also deteriorates when the dimensions of the instances increase. For example, according to our experiments (to be detailed in Section 4.3.4), I observed that the CPLEX CP optimizer could not produce feasible solutions for the tested instances of MRCPSP with 50 tasks, three modes and two renewable and two non-renewable resources within 1,200 seconds of CPU time. Therefore, heuristic and metaheuristic methods have been considered in most studies.

A few efficient heuristic methods were proposed to use mode selection and priority rules [20], [152], [153]. These heuristics are easy to implement and can solve very large instances. However, solution qualities can be poor for hard instances. Therefore, metaheuristics were also utilised to solve MRCPSP, including simulated annealing [154]–[157], tabu search [158], path-relinking [159], [160], and variable neighbourhood search [25], [161].

Most metaheuristic solution approaches for MRCPSP are based on the mechanism of evolutionary processes. Among them, the genetic algorithm (GA), which is developed by Holland [49], is the most commonly used approach; see [162]–[164] for some examples. Different variants of GA were also developed to solve MRCPSP [165]. A bi-population GA is proposed in [143], which embeds the forward-backward improvement local search by employing two different populations for the forward and the backward passes. In [166], a novel local search is embedded in GA, where the local search focuses on maximising the use of non-renewable resources as a means to minimise the makespan. The study suggests that prioritising tasks with a lower float is more efficient in reducing the makespan.

Hybridisation of GA with other algorithms to solve MRCPSP has also been investigated in [167], [168]. In [167],a mode reduction procedure is used to remove inefficient and non-executable modes. The study also improved the method originally proposed by [169] for RCPSP and solved multi-mode instances with up to 100 tasks.

In [168], the bi-population GA (BPGA) proposed in [143] was improved by being combined with an estimation of distribution algorithm (EDA) [170], which is responsible for the mode assignment. In EDA, the search process is guided by a probabilistic model of promising results.

Another line of approximate algorithms to solve the MRCPSP was developed based on swarm intelligence metaheuristic methods such as particle swarm optimisation (PSO) [55] and ant colony optimisation (ACO) [54].

A few examples of applying the PSO for solving the MRCPSP include [171]–[173]. In [171], a new methodology to solve MRCPSP based on PSO is proposed, which represents potential solutions for the MRCPSP in terms of mode and priority combination for tasks. A feasible solution is then transformed into a schedule using a serial generation scheme.

In [172], a combinatorial PSO was proposed which uses local search to optimise the schedule after a mode is assigned to each task in PSO. The PSO was hybridised with the differential evolution (DE) algorithm in [173]. It has two levels: The first level schedules a sequence produced by the PSO algorithm, whereas the second level uses the DE to decide the execution mode of tasks.

In the ACO proposed in [174], two levels of selection are performed. The first level assigns modes to tasks and the second level is employed to obtain the best set of modes and the execution sequence of tasks as well.

A hyper-heuristic method was developed by [175], which is a hybrid of hyper-heuristic methods, Monte-Carlo tree search, memetic algorithms, and novel neighbourhood moves. In [175], the searching process is performed in two different phases, i.e., the construct phase and the improve phase. In the construct phase, an initial solution is generated using a Monte-Carlo Tree Search method. The improvement phase is a combination of a set of heuristics controlled by a memetic algorithm which is an extension of hyper-heuristic components proposed in [176], [177].

2.4 Stochastic resource-constrained project scheduling problem

The RCPSP is a well-known problem in operations research with wide-ranging practical applications, and as it is classified as an NP-hard problem [178], many researchers have devoted significant effort to developing efficient algorithms to solve this problem. While the numerous practical applications of RCPSP, it has limitations in addressing real-world problems that are inherently uncertain, such as inaccurate estimations, new or dropped tasks, and unforeseen conditions. Consequently, a more robust approach is necessary to ensure the success of project implementation [28], [29]. This has led to the development of a new research area focusing on the stochastic resource-constrained project scheduling problem (SRCPSP), which extends the RCPSP to include projects with stochastic task duration and aims to minimise the expected makespan.

2.4.1 Problem definition

The objective of an SRCPSP is to manage a project with a stable resolution within an unpredictable environment and reduce the total project duration. In this study, I adopt the Activity on Node (AoN) approach to establish the fundamental model of an RCPSP as shown in Figure 2.2. Each task's duration (d) is regarded as uncertain and denoted by $\tilde{d}$. The mathematical model to satisfy precedence and resource constraints is as follows.

In this model, S_i presents the starting time of the task i. The first dummy task starts at $S_0 = 0$, and the makespan of each scenario s is the finishing time of the last dummy task in scenario s. Because the duration of dummy tasks equals zero, the makespan of the scenario s equals S_{n+1}, s.

The expected makespan $E[S_{n+1}]$, is calculated by averaging S_{n+1} over $|S|$ scenarios where $\tilde{d_{i,s}}$ indicates the predicted duration of the task i in the scenario s, which is randomly generated using a Probability Distribution (PD) with d_i, the duration of the task i. The nonanticipativity constraint, as defined in [179] for the SRCPSP, ensures that task sequencing decisions are made solely based on existing information, implying that the duration of a task is not known upon its completion.

$$\min E[S_{n+1}] = \frac{\sum_{s=1}^{|S|} S_{n+1,s}}{|S|}. \tag{2.17}$$

subject to

$$S_{j,s} \geq S_{i,s} + \tilde{d_{i,s}}, \quad \forall (i,j) \in A_s, \forall s, \tag{2.18}$$

$$\sum_{i \in \tau(t)_s} r_{ik,s} \leq R_{k,s}, \forall k \in RR, \forall t \in \{0, \ldots, T_s - 1\}, \forall s, \tag{2.19}$$

$$\tau(t)_s = \{i \in V_s \mid S_{i,s} \leq t < S_{i,s} + \tilde{d_i}, s\}, \forall s, \tag{2.20}$$

$$S_{i,s} \in \{0, 1, \ldots, T_s - \tilde{d_{i,s}}\}, \quad \forall i \in V_s, \forall s, \tag{2.21}$$

, where the problem satisfies the nonanticipativity constraint, and T_s, is the given upper bound on the scenario s.

2.4.2 Solution approaches for stochastic resource-constrained project scheduling problem

The well-known PERT analysis [180] is a classical method used in project management to deal with uncertainty in task duration. PERT estimates the expected project makespan and its variation by assuming a probability distribution of task duration. The PERT method improved in [181], [182], but this research does not explicitly consider resource constraints, as it assumes unlimited resources are available for project execution.

The project scheduling literature mainly focuses on generating a feasible schedule that satisfies precedence and resource constraints and optimises scheduling objectives, most commonly the project duration. This baseline schedule, also called a predictive or preschedule, serves several important functions. The baseline allocates resources to tasks to optimise performance and also serves as a basis for planning external tasks, such as material procurement [28]. Many real-life scheduling problems, such as course scheduling, sports timetabling, and railway and airline scheduling, can be modelled as variations of resource-constrained project scheduling problems. Executing tasks according to the preschedule is mandatory in these environments and is imposed by the customer, and although technically possible, tasks are not started before their scheduled starting time[28].

The baseline schedule is also employed in robust project scheduling. Robust project scheduling is a widely used approach in project management to deal with exogenous disruptions affecting task duration and resource availability. This approach aims to create a stable project schedule that can function within the limitations of available resources. One way to achieve this is to proactively create a robust schedule that minimises the deviation from the baseline schedule by including time buffers in the schedule [183]–[185]. Another approach is to reactively revise and optimise the schedule during project execution based on the newly available information to minimise deviation from the original baseline schedule [186], [187]. Some researchers have developed proactive-reactive procedures that combine both approaches [188], [189].

In recent years, research has increasingly focused on sim-opt algorithms, which integrate simulation and optimisation techniques [190]. Various metaheuristic approaches were designed in this framework, such as [191] that used resource-based policy in their metaheuristic algorithm, and a new pre-processing procedure, developed using a two-phase GA [192]. An evolutionary approach was proposed in [193] to reduce the number of simulation runs. The research in [194] investigated how time-varying weather conditions affect SRCPSP and proposed an improved Estimation of Distribution Algorithm (EDA) with a ranking and selection method using common random numbers and indicated that the enhanced EDA could produce a shorter expected makespan and a faster convergence to the optimal solution compared to the original EDA.

All these approaches provide a sequence of all tasks at time zero without observing early task duration and are considered open-loop policies. They are static solutions that are not updated during real-time execution.

Priority rule (PR) heuristics are essential for solving RCPSP and SRCPSP. Their significance stems from several reasons: Firstly, PRs are speedy and require less computational time in comparison to other heuristics. Secondly, many sophisticated heuristics and exact procedures integrate PRs as a component of their algorithm, such as swarm optimisation techniques, which commonly use PRs to establish the initial solutions. Thirdly, PRs are user-friendly and intuitive, making them a popular choice for commercial project scheduling software due to their simplicity. In practical applications, a project manager can easily apply PRs to schedule a project quickly, even without the assistance of a computer. The efficiency of many PRs in solving RCPSP has been extensively researched [32], [33]. However, little attention has been given to PR heuristics for solving the SRCPSP. In [190], 17 priority rule heuristics on SRCPSPs were examined. It is common to neglect a project's stochastic nature and employ a deterministic model to provide a solution [195]. In [190], Out of the 17 priority rules, 12 were obtained from the literature that deals with solving RCPSP in a deterministic setting. The remaining five rules were devised using stochastic data related to the SRCPSP. Their results showed that The conclusions drawn from RCPSP cannot be directly applied to SRCPSP. In [195], They attempted to address a significant inquiry regarding using heuristics. Specifically, they sought to determine the conditions under which it is appropriate to use the deterministic RCPSP and when it becomes essential to transition to the stochastic model.

Another approach to solving RCPSPs under uncertainty is using policy-type decision-making. In this approach, scheduling decisions are assumed to be executed sequentially without a prior baseline schedule. In the deterministic RCPSP, it is common for heuristics to select an SGS first and then use the SGS to transform all solutions into activity start times[196]. The policy-type decision serves a similar function to the SGS in the RCPSP, and a policy-type decision is required to specify which task(s) to start at each decision point [197], [198]. A set of policy classes, including resource-based policy class and activity-based policy class, were defined in [190]. In the resource-based policy class, the tasks are ranked according to a predefined priority rule and started in the order specified by the priority. Although priority-based policies are simple to implement and execute quickly, there are cases where the policy is not monotone: decreasing the duration of a task adversely affects the makespan and results in the longer completion time of the project, which is called Graham anomalies [199]. The other shortcoming of this policy is that due to the greedy usage of resources, there may be no priority-based policy that can generate an optimal schedule in some instances. Therefore, other approaches may be necessary [200]. In the activity-based policy class, extra side constraints are added to ensure the policy is always monotone. To this aim, for a given scenario d, a priority list L is the ordering list of all tasks that start as early as possible, and if i and j are two tasks and $i <_L j$, then the side constraint is added for starting time of task i and j, $S_i(d) \leq S_j(d)$. In [190], they claimed that on average, in a lot of experiments, the resource-based policy outperforms activity-based policy, and among the several policy classes they examined, they used the resource-based policy to solve the SRCPSP.

An alternative solution approach to solve SRCPSP, where scheduling decisions are made sequentially using dynamic programming, is a closed-loop policy. Instead of optimising the task sequence beforehand, this method aims to find the best decision rule or policy for selecting tasks to start at each decision point based on the available information. This adaptive and dynamic approach allows decision-makers to utilise the new information that arises between

decision points [30] and consequently is more flexible. However, despite its theoretical appeal, closed-loop policies for SRCPSP have been considered computationally intractable [31], and only a few papers have attempted to offer closed-loop policies. In [201], a simplified job-shop version of SRCPSP is modelled. Each project has a fixed sequence of tasks, and a DP algorithm with a limited heuristic state space is introduced for a project with 17 tasks. In [202], an exact closed-loop policy is introduced for the problems when the task duration has exponential distribution. Their method can be used to optimally solve problems with up to 60 tasks. In [31], a closed-loop approximate dynamic programming approach was introduced to solve instances with uncertain task duration and up to 120 tasks. Dealing with SRCPSPs with uncertain resource availability, [203] proposed an approximate dynamic programming. Their approach is computationally tractable for problems up to 120 tasks.

2.5 Summary of methods, strengths, and limitations

This chapter reviewed the literature on three optimisation problems, RCPSP, MRCPSP, and SRCPSP. The RCPSP's solution methods are generally divided into exact algorithms and heuristics-based approaches [11]. Exact methods are impracticable in solving large-scale problems [12]. As a result, the majority of literature in this field focuses on using heuristic-based approaches, which are better suited for handling the complexities and computational challenges associated with such problems.

In tackling RCPSPs and MRCPSPs, various metaheuristic algorithms and hybrid methods have been explored, each offering unique strengths and weaknesses. The SGS provides a simple baseline for solving RCPSP, though its limited exploration of the solution space and potential inefficiency in complex scenarios highlight the need for more sophisticated approaches. GA, known for its strong capability to explore vast solution spaces, is adaptable and effective in avoiding local optima but requires careful parameter tuning and can be computationally intensive. Similarly, the Bees Algorithm and its extensions effectively explore global optima but suffer from weak local search capability. TS, with its efficient memory structures, avoids cycles and revisits, making it a powerful tool when properly configured, though it also faces challenges such as parameter tuning and memory intensity.

Hybrid methods, which combine the strengths of multiple algorithms, represent a robust approach to RCPSP. By taking advantage of different optimisation strategies, these methods can achieve higher solution quality and adaptability to specific problem characteristics. However, the complexity of implementation and the challenge of extensive parameter tuning across multiple algorithms can be significant drawbacks. Despite these challenges, hybrid methods often yield superior results, making them suitable to solve RCPSP. An example of a hybrid metaheuristic algorithm is our recently published work [17], which proposed hybridising the ABC and FIA. The ABC algorithm is renowned for its strong global search ability among Swarm intelligence methods. However, the lack of a powerful local search capability results in slow convergence [17]. The FIA has been shown to be an efficient and effective algorithm in solving a wide range of optimisation problems and superior convergence speed and solution quality performance over an extensive range of optimisation problems [1].

A matheuristic approach combines mathematical programming techniques with heuristic methods. It uses the strengths of both exact and heuristic methods to explore solution spaces efficiently, often incorporating problem-specific insights to guide the search process. The recently proposed matheuristic R&S was designed for job-shop scheduling and aircraft landing problems.

This algorithm is based on generating a feasible schedule (a solution) and then improving the solution by iteratively relaxing the problem and solving the relaxed problem using a local search, including a mixed-integer programming solver. In solving MRCPSP, the mode reduction [22], [164], [165], [167] and mode selection [20], [152] approaches are two preprocessing methods for solving MRCPSP. In [204], a knapsack-like approach was employed for the mode selection. In [35], [141], I proposed new matheuristic algorithms based on R&S for solving RCPSP and MRCPSP. Generating a feasible solution for the MRCPSPs can be very challenging, so in the proposed R&S, a mathematical program is designed for preprocessing and generating initial solutions.

In the context of solving the SRCPSP, a successful approach for addressing external disruptions in project scheduling, known as robust project scheduling, aims to create stable schedules despite task duration and resource availability uncertainties. In contrast to deterministic instances, schedules with fixed task start times can become infeasible due to unpredictable task durations. Robust scheduling requires establishing a baseline schedule upfront and allows limited flexibility for revisions during project execution. Another approach involves policy-based decision-making, where scheduling decisions are made sequentially without a predefined baseline schedule, offering an alternative method for handling uncertainty in solving SRCPSP. A theoretically attractive approach is employing closed-loop policies. The closed-loop policies dynamically make decisions as the project progresses. This adaptive strategy allows decision-makers to adjust task selection based on real-time information, enhancing flexibility. The computational complexity of closed-loop policies is the main drawback of this approach. A closed-loop approximate dynamic programming approach proposed in [31] was introduced to reduce the complexity of the closed-loop policy and solve instances with uncertain task duration and up to 120 tasks. In [195], the investigation explores the conditions under which the deterministic RCPSP is suitable and when transitioning to the stochastic model becomes essential.

The heuristics based on PR are essential for solving RCPSP and SRCPSP. The efficiency of these heuristics for solving RCPSP has been researched [32], [33]. However, little attention has been given to the applications for the SRCPSP. In [205], I proposed a new computationally tractable approximate dynamic programming approach based on using PRs and taking advantage of data from a deterministic problem in solving the stochastic problem and dealing with up to 120 tasks.

Chapter 3

A Metaheuristic Global Optimisation Approach for Resource-Constrained Project Scheduling

3.1 Introduction

Researchers have recently tried to develop efficient methods to achieve optimal or near-optimal solutions for the resource-constrained project scheduling problem (RCPSP). Global optimisation has successfully addressed continuous and nonlinear optimisation problems across various domains. Extending this approach to RCPSP poses a promising avenue for enhancing project management efficiency. By employing the principles of global optimisation, strategies can be developed to effectively handle the complexities of the RCPSP, including task dependencies and limited resource availability.

A nonlinear optimisation problem can be formulated as [206], [207]:

$$z = \min_x \{f(x) : l \leq x \leq u\}, \tag{3.1}$$

where $x = (x_1, x_2, \ldots, x_D) \in \mathbb{R}^D$ is a D-dimensional vector of decision variables, $f(x)$ is a nonlinear objective function, $l = (l_1, l_2, \ldots, l_D)$ and $u = (u_1, u_2, \ldots, u_D)$ are the lower and upper limits for decision variables, respectively.

Due to the computational challenges of solving nonlinear optimisation problems, various non-exact methods, such as metaheuristic algorithms, have been developed to obtain "good" solutions instead of finding the global optimal solutions. Many metaheuristic algorithms are relatively easy to implement and do not require the problem's gradient information. Furthermore, by employing stochastic operators [206], [208], the metaheuristic algorithms will have the capability of escaping from the local optima, leading therefore to obtaining solutions that are reasonably close to the global optima. Those advantages have led metaheuristics to be a common tool for solving not only engineering optimisation problems but also problems in areas such as finance and science [74], [209]–[211]. The line search algorithm [212] is well-known for its high convergence speed. In the line search method, first, a descent direction along which the objective function is optimised (e.g., decreased for a minimisation problem) is identified. Then, the step size is calculated, either exactly or approximately, to determine how far the candidate solution can move in the descent direction [213]. Combining the line search with

other algorithms has been shown to produce promising results [214]–[216]. Nonetheless, the line search method has certain limitations. For example, using an exact method in the line search to identify the step size restricts the method's usability on continuously differentiable functions, i.e., on smooth functions. Further, computing the exact step size is computationally expensive, and as such, the line search usually needs to be terminated early to reduce the cost of evaluating the objective function.

The remainder of this chapter is organised as follows. Section 3.2 details the motivation of this chapter; in Section 3.3, the hybridised algorithm is proposed, which is named ABFIA, and discusses the components of the hybrid ABFIA algorithm. In Section 3.4, we extend the ABFIA and define a new metaheuristic, CABFIA, to solve RCPSP. We perform comprehensive computational experiments to assess the performance of the ABFIA and CABFIA in Section 3.5 and Section 3.6, respectively. The chapter concludes in Section 3.7.

3.2 Motivation

This chapter is motivated by the potential of metaheuristic algorithms to enhance the searchability and efficiency of solving complex problems. The following details the specific motivations driving the research presented in this chapter. Among the metaheuristic optimisation methods widely used for global optimisation, the artificial bee colony (ABC) algorithm and its variants have been the most successful in solving various nonlinear optimisation problems [56]. The ABC algorithm has a powerful exploration ability, is easy to implement, and benefits from a small number of parameters. A powerful exploration capability is critical for the non-exact algorithms because the lack of a robust global search scheme in an algorithm increases the possibility of getting trapped in low quality local optima [217]–[220]. Compared to other metaheuristics, e.g., particle swarm optimisation (PSO), the ABC algorithm has a slower convergence rate [221]. One strategy to improve the slow convergence rate of the ABC algorithm without losing its strong global search capability is through hybridising the ABC with local search algorithms such as the line search method [222].

The newly published Fibonacci indicator algorithm (FIA) [1] proposes a new line search scheme based on the Fibonacci percentages. The FIA does not suffer from the shortcomings of the original line search method. Because the FIA applies different methods to avoid getting trapped in low quality local optima, it may obtain superior results. The results in [1] show an excellent performance for the FIA, along with a fast convergence speed for FIA in optimising a broad range of objective functions, which are not restricted to smooth functions. Nonetheless, as discussed by [1], the FIA has certain shortcomings. For example, the FIA may not be suitable for solving high-dimensional multimodal functions, which demand a robust exploration scheme. Also, the FIA may not deliver quality solutions for functions with multiple local optima and where a balanced exploration and exploitation strategy is needed.

Compared to the ABC and its variants, the FIA converges very slowly in solving high-dimensional multimodal problems. The reason could be that after a restart, the population is randomly generated without taking advantage of the information from previous iterations other than the current best solution. On the contrary, when the ABC restarts the search in the scout bee phase, only one solution is randomly generated while the others are retained. Another possible reason is that the best solution is consistently selected as one parent in the FIA. However, the onlooker bees randomly select the candidate solutions for the local search in the ABC algorithm, leading to a more robust global convergence.

Previous studies have demonstrated the efficacy of the ABC algorithm in addressing the RCPSP [16]. The ABC algorithm's strong exploration ability and the FIA's fastness motivate us to hybridise the FIA with the ABC algorithm. We expect that hybridisation can enhance the performance of FIA to solve challenging problems, such as high-dimensional multimodal problems. At the same time, the FIA can guide the ABC to attain a faster convergence.

This chapter proposes an approach for solving RCPSP. The idea of this algorithm is first used to tackle nonlinear problems, demonstrating the method's efficiency in handling complex, nonlinear scenarios. These problems are unconstrained and easier than RCPSPs to handle. The method is then extended to efficiently address RCPSP, which involves a comprehensive set of constraints. Although neither ABC nor FIA are robust algorithms on their own, ABC possesses strong global search capabilities, while FIA offers fast convergence. Combining these two algorithms could result in a robust algorithm. Tackling RCPSP using the idea of the ABFIA has challenges, including effectively handling precedence and resource constraints and incorporating discrete decision variables into the ABFIA framework, efficiently exploring the complex and discrete solution space, tackling challenges associated with local optima, and ensuring computational efficiency.

3.3 Hybridising the Artificial bee colony algorithm with the Fibonacci indicator algorithm: The ABFIA

We propose hybridising the ABC and FIA by integrating the FIA with the ABC in the exploitation process, i.e., during the onlooker bee and scout bee phases, as an effective strategy to escape from local optima. We call this algorithm the ABFIA. The pseudo-code of the ABFIA is provided in Algorithm 3. Figure 3.1 illustrates the scheme to hybridise the ABC with FIA. The details of each component of ABFIA are explained in the following sections.

3.3.1 Initialisation

The initialisation of the ABFIA is the same as that of the original ABC algorithm. Accordingly, the ABFIA generates SN random solutions in the initialisation phase using Equation (2.1).

3.3.2 The employed bee phase

The employed bee phase of the ABFIA is the same as in the original ABC algorithm. In the employed bee phase, each employed bee is sent to one solution (food source). If an employed bee obtains a better solution in the neighbourhood of the current solution, the current solution is updated according to Equation (2.2).

3.3.3 The onlooker bee phase

The onlooker bee phase consists of two different strategies. In the first strategy, for a given parameter $P_{Onlooker}$ percent of iterations, the onlooker bee phase is the same as in the original ABC algorithm. However, in the second strategy, for $1 - P_{Onlooker}$ percent of iterations, the onlooker bees use the FIA to search the feasible space.In the first strategy, the neighbourhood of each of the selected solutions by the roulette wheel mechanism (see Equation 2.3) is searched (see Equation (2.2)). The food source is updated when a solution with better fitness is obtained within its neighbourhood. Otherwise, the trial counter is increased by 1.

Algorithm 3 The ABFIA.

Input: Objective function $F(x)$, Parameters $SN, limit_1, limit_2, N, p, C$.
Output: A feasible solution.

Phase 1: Initialisation
Generate random solutions x^i, $i = 1, 2, \ldots, SN$ by using Equation (2.1);
Set $trial_i = 0$, $i = 1, 2, \ldots, SN$;
while *stopping criterion for the ABFIA is not met* **do**
  Phase 2: The employed bee phase (from the ABC algorithm)
  Search around all existing solutions and update *trial* accordingly;
  Phase 3: The onlooker bee phase
  if $rand < P_{Onlooker}$ **then**
    First strategy: The onlooker bee phase (from the ABC algorithm):
    Search around solutions selected by the roulette wheel mechanism and update *trial* accordingly;
  end
  else
    Second strategy:
    for $i = 1, 2, \ldots, SN$ **do**
      Generate a population $\mathcal{T}$ of size N including x^i and $(N-1)$ solutions randomly selected from remaining solutions;
      Generate a new candidate solution V^i using FIA with $\mathcal{T}$ as the initial population;
      if $F(V^i) \leq F(x^i)$ **then** $x^i = V^i$;
      if $F(V^i) > F(x^i)$ **then** $trial_i$ = $trial_i$ + Number of function evaluations within FIA;
      else $trial_i = 0$;
    end
  end
  Phase 4: The scout bee phase
  for $i = 1, 2, \ldots, SN$ **do**
    if $trial_i > limit$ **then**
      if $rand < P_{Scout}$ **then**
        x^i=FIA(Objective function $F(x)$,N, p, C);
        $trial_i = 0$;
      end
      else
        The scout bee phase (from the ABC algorithm):
        Replace x^i by a new randomly produced candidate solution using Equation (2.1);
      end
    end
  end
end
Return: x;

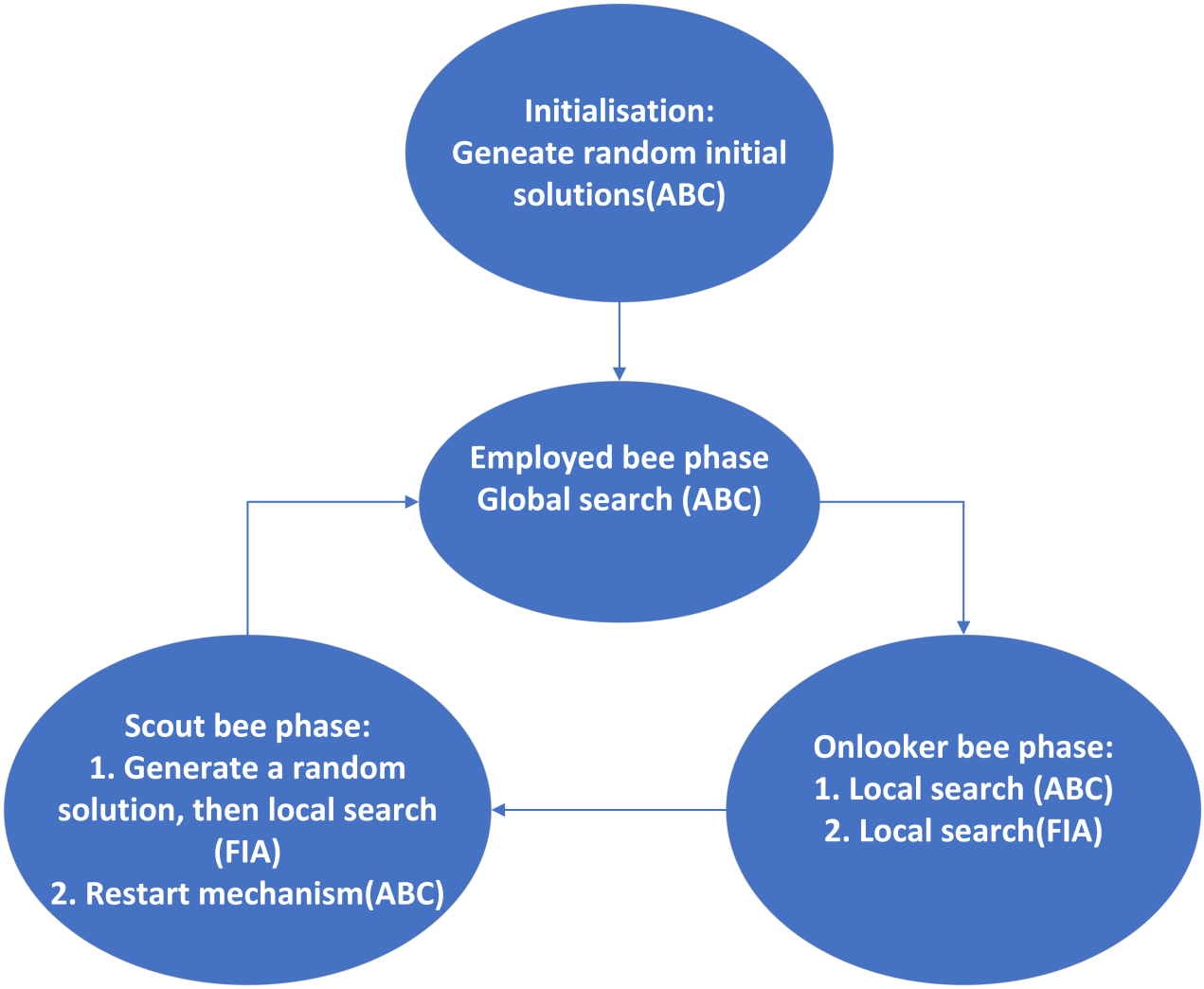

Figure 3.1: The conceptual diagram of the ABFIA hybrid algorithm.

In the second strategy, the solution will be attempted by the FIA. Here, an initial population of the FIA is comprised of the target solution and $N-1$ solutions, which are randomly selected from the remaining $SN-1$ solutions. The ABC suffers from getting trapped in a local optimum due to the roulette wheel mechanism, which favours solutions with low fitness values. The ABFIA overcomes this shortcoming by randomly selecting $N-1$ solutions and the FIA's search methods. The methods of selecting the worst of the remaining solutions as a parent and performing a crossover between solutions to generate a parent improve the diversification of the solutions. The improved diversification is not at the expense of solution quality because strong local search capability is embedded within the FIA.

3.3.4 The scout bee phase

The main goal of the scout bee phase in the ABFIA is to escape from local optima. In this phase, each solution with more than *limit* trials, for P_{Scout} percent of iterations, is replaced by a solution found by the FIA. The FIA is initialised with a random population to maintain the diversity of the current population.
The scout bee phase in the ABC is a restart strategy that replaces the current solutions with randomly generated solutions. Therefore, the scout bee phase is not an effective mechanism

for escaping from local optima. That leads to slow convergence of the ABC [78]. The ABFIA employs the original scout bee phase for $1 - P_{Scout}$ percent of iterations and for other iterations employs the original FIA in the scout bee phase to find a better solution than randomly generating a solution, which leads to improving the convergence rate of the ABFIA against the ABC. On the other hand, the FIA in the scout bee phase employs far fewer iterations than the original FIA [1].

3.4 Combinatorial ABFIA

The ABC algorithm has previously been used to solve RCPSP [16]. The ABFIA extends the ABC algorithm to solve nonlinear optimisation problems [17]. This extension of the ABC takes advantage of strong global search strategies in the ABC[56] and the efficient local search of the FIA [1]. This motivated us to extend the ABFIA and design the combinatorial ABFIA (CABFIA) to solve RCPSP.

This section discusses the ideas for dealing with combinatorial optimisation problems. The FIA combined with the ABC, was primarily developed for continuous optimisation problems. However, in solving the RCPSP, traditional FIA formulations face challenges due to the availability of constraints and the discrete nature of decision variables. Designing the combinatorial FIA (CFIA) is motivated by the strength of the FIA in local search, and we aim to use its local search ability for discrete solution spaces.

3.4.1 Solving resource-constrained project scheduling with the Combinatorial ABFIA

In this section, we explore different aspects of CABFIA, which helps us to use the powerful search methods of ABFIA to solve the RCPSPs efficiently. We discuss how CABFIA represents solutions, deals with constraints, and local searches for solutions using the proposed CFIA and taking advantage of the Cplex Cp optimizer.

3.4.1.1 Representation of solutions

The CABFIA algorithm adopts the starting time of each task to represent its solutions. To generate schedules, each task's start time is considered the task's priority, i.e., the task started earlier has a higher priority to start. The priority list of tasks is denoted by Pr_i^j. This list comprises priority values in $[l_j, u_j]$, where j belongs to the set $\{0, 1, \ldots, n+1\}$. The l_j represents the lower bound, and u_j is the upper bound of the starting time for the task j. For each task j, we determine the earliest possible start time, denoted as l_j, by considering the earliest precedence feasible start time. The latest start time, denoted as u_j, is obtained by a backward recursion [223] from an upper bound of the project's finish time T [93].

3.4.1.2 Combinatorial FIA for local search

The FIA [1], inspired by the Fibonacci retracement, a well-known technique that stock market traders use to predict maximum and minimum stock prices. FIA performs a line search on the line defined by two available solutions using a method known as the Fibonacci Indicator (FI) as follows.

Let a fixed number of five solutions be generated between two available x and x_{best} as

$$x_i = x + \tau_i(x_{\text{best}} - x), \quad i = 1, \ldots, 5, \tag{3.2}$$

where $\{\tau_i\} = \{50\%, 73.6\%, 88.2\%, 111.8\%, 150\%\}$ is the set of Fibonacci ratios suggested in [1].

The FIA is a component of the ABFIA [17], originally designed for solving unconstrained optimisation problems. However, to have a feasible solution for the RCPSP, all problem resource and precedence constraints should be satisfied. To bridge this gap in CABFIA and generate a feasible schedule for the RCPSP, we employ a schedule generation scheme (SGS) that guarantees the feasibility of each generated solution.

Schedule generation schemes (SGS) are the foundation of most heuristic approaches designed for the RCPSP [93]. Typically, an SGS starts from scratch and progressively develops a feasible schedule by extending a partial schedule. A partial schedule is a scheduling structure where only a subset of project tasks has been scheduled. According to [95], the most effective heuristics are based on the serial SGS. In this approach, a non-dummy task is chosen from the eligible set in every stage and scheduled at the earliest time that meets both precedence and resource constraints [93]. In the CABFIA, we adopt serial SGS with a priority list of tasks to select one or more tasks from the eligible set[93] in each stage of making a feasible schedule.

This section defines a combinatorial FIA (CFIA) to be employed in the CABFIA. The algorithm initiates with an initial population of N randomly generated solutions denoted as x_i where $i \in 0, 1, \ldots, n+1$. Each solution is represented by the list of task start times. To obtain x_i^j, representing the starting time of each task j in solution i, first Pr_i^j, a set of randomly selected priority values within the range $[l_j, u_j]$ is generated. Here, l_j and u_j denote the earliest and latest start times for task j, and $j \in 0, 1, \ldots, n+1$. The Pr_i cannot be considered as the starting time of tasks because it can easily assign the starting time to some tasks that violate the problem constraints, so the Pr_i needs to be mapped to a feasible schedule using the serial SGS. The fitness value of each solution (schedule) is the makespan of the project, x_i^{n+1}. These solutions are subsequently organised in ascending order according to their fitness values. The solution with the best fitness value is x_{best}, with its associated fitness value g_{best}.

The CFIA generates a new solution by choosing two parents from the current population and performs a line search on these solutions. The current best solution is selected as the first parent. Subsequently, the second parent is chosen from the remaining solutions based on their ranking, with preference given to the next highest-ranking solution not yet selected.

If the fitness value of this newly generated solution surpasses that of both parents, the current population is modified. The new solution is added, and the worst solution is removed. Every time the current best solution is refreshed, all solutions in the population are treated as "never have been chosen". A new population is obtained once every solution in the current population is substituted, and this population renewal process is referred to as *Step 1.* If the population fails to be updated, meaning that when all solutions have been tried as parents, and not even one solution can be removed, a new population is generated randomly.

After obtaining a new population in Step 1, the CFIA proceeds to *Step 2.* In this step, CFIA selects two solutions as parents to generate a new solution using the FI method. The best solution is consistently chosen to be one of the parents. The second parent, represented as x_{sp}, is selected from the remaining solutions based on descending objective function values with a probability of $1 - p$ (where p is a predefined parameter). Alternatively, with a probability of p, the second parent is generated using a crossover operation defined as:

$$Pr_{sp} = (x_0^{r_0}, x_1^{r_1}, \ldots, x_{n+1}^{r_{n+1}}), \tag{3.3}$$

where r_i, $i = 0, \ldots, n+1$, is a randomly selected integer number between 1 and N. The obtained Pr_{sp} is then mapped to a feasible scheduled x_{sp} using serial SGS.

Step 2 is designed to uphold the diversification of the population. It achieves this by randomly producing solutions using Equation (3.3) while keeping all the solutions derived from the line search. This methodology provides a balanced trade-off between the global exploration and local exploitation capabilities of the CFIA.

The search is restarted if the best solution remains unchanged after $C \in \mathbb{Z}^+$ consecutive iterations of Steps 1 and 2. This is accomplished by generating a new population consisting of the current best solution x_{best} and $N-1$ solutions generated randomly. The maximum number of function evaluations is the stopping criterion of this algorithm. Algorithm 4 shows the pseudo-code of the CFIA.

Algorithm 4 The CFIA.

Input: An RCPSP instance, Parameters N, p, C;

Output: A feasible schedule.
Set $I = 0$;
Find the l_j and u_j, the lower bound and upper bound of the starting time of each task j, where $j \in 0, 1, \ldots, n+1$;
Generate a set of N random priority lists for the tasks of the instance;
Map priority lists to feasible schedules using serial SGS;
while *stopping criterion is not met* **do**
  while $I < C$ **do**
    Search phase:
    Perform Step 1: Improve the population focusing on exploitation;
    Perform Step 2: Improve the population with a balance of exploration and exploitation;
    $I = I + 1$;
    if *the current best solution was updated* **then** $I = 0$;
  end
  Set $I = 0$, and generate a set of N priority lists for the tasks of the instance, including the current best solution and $N-1$ random solutions;
  Map the priority lists to feasible schedules using serial SGS;
end
Return: The best obtained schedule (the solution);

3.4.1.3 Cplex Cp optimizer for local search

The CPLEX CP optimizer [15] is a constraint programming based tool that can be used for solving optimisation problems, including RCPSP. Previous research shows that in the case of complex problems, tight available resources, and large problem sizes, the efficiency of this solver typically decreases [27], [36]. One main reason for that weakness is the CPLEX CP optimizer's difficulty in generating a quality initial solution[27].On the other hand, the solver has demonstrated its effectiveness as a potent local search tool. It can be integrated into a

heuristic to explore the neighbourhood of promising solutions, initiating the process with those solutions.

In the CABFIA, we utilise the CPLEX CP optimizer in the local search phase of the algorithm. When the number of tries of the CABFIA for improving a candidate solution reaches the predetermined maximum number of generated schedules, the available solution is used as a warm start for the optimiser.

3.4.2 Implementation of the Combinatorial ABFIA

This section introduces the CABFIA for solving an RCPSP by integrating the CFIA with ABC during the exploitation and exploration. This algorithm also takes advantage of the FBI by generating schedules through forward and backward passes in the search process, as well as the Cplex Cp optimizer, to enhance the local search. Algorithm 5 is the pseudo-code of the CABFIA. This algorithm iteratively generates priority lists, which are then decoded into feasible schedules through serial SGS, and aims to minimise the makespan of an RCPSP. The following sections explain the details of each component of the CABFIA.

3.4.2.1 Initialisation and the employed bee phase

The CABFIA generates a set of SN potential solutions in the initialisation process. To obtain each potential solution, first, a randomly determined priority list Pr_j where $j \in \{0, 1, \ldots, n+1\}$ is generated, then using a serial SGS, x_j the starting time of each task j is calculated. Each population i is a feasible schedule represented by the starting time of each task, i.e., x_j^i, where $j \in \{0, 1, \ldots, n+1\}$. During the initialisation phase, for each $i \in \{1, 2, \ldots, SN\}$, CABFIA assigns the dummy task $j = 0$ the priority Pr_0^i=0. Subsequently, the priority for the remaining tasks $j \in \{1, 2, \ldots, n+1\}$ is computed using Equation (3.4) to randomly select a value within $[l_j, u_j]$, where l_j and u_j are the earliest start time and the latest start time of the task j.

$$Pr_j^i = l_j + \text{rand}\{0, 1\} \cdot (u_j - l_j), \quad j = 1, \ldots, D \tag{3.4}$$

In the CABFIA, the employed bee phase is the same as the original ABFIA algorithm. During this phase, each employed bee targets a specific solution, referred to as a food source. New priority lists are generated using the available solutions. When V_j^i represents the j^{th} dimension of the i^{th} candidate priority list.

$$V_j^i = Pr_j^i + \psi(Pr_j^i - Pr_j^k), j = 1, 2, \ldots, D, \tag{3.5}$$

Given that $k \neq i$ is chosen randomly from the set$\{1, 2, \ldots, SN\}$, and ψ is a uniformly distributed random number between $[-1, 1]$. Each solution is a feasible schedule represented by the starting time of each task j. If an employed bee identifies a priority list V^i that results in a superior solution within the neighbourhood of the solution i, the existing priority list Pr^i first gets substituted by V^i based on the Equation (3.5). Then, the serial SGS generates a feasible solution i using Pr^i denoted as x_j^i where $j \in \{0, 1, \ldots, n+1\}$.

3.4.2.2 The onlooker bee phase

The onlooker bee phase consists of two distinct strategies. The first strategy is based on the idea of local search in the ABFIA, and for all the iterations in this strategy, the algorithm utilises

Algorithm 5 The CABFIA.

Input: An RCPSP instance, Parameters $SN, limit, N, p, C, P_{Onlooker}, P_{Scout}$.
Output: A feasible schedule.

Phase 1: Initialisation
Find the l_j and u_j, of each task j, where $j \in 0, 1, \ldots, n+1$;
Generate random priority lists Pr^i, $i = 1, 2, \ldots, SN$ using (Equation (3.4));
Map Pr^i to feasible schedules x^i, $i = 1, 2, \ldots, SN$ using Serial SGS Set $trial_i = 0$, $i = 1, 2, \ldots, SN$;
while *the maximum number of generated schedules in the ABFIA is not met* **do**
  Phase 2: The employed bee phase Search around all existing solutions and then update *trial* accordingly
  Map the obtained priority lists to a feasible schedule using serial SGS;
  Phase 3: The onlooker bee phase
  if $rand < P_{Onlooker}$ **then**
    First strategy: The onlooker bee phase: Search around solutions selected by the roulette wheel mechanism and update *trial* accordingly;
    Map the obtained priority lists in this phase to feasible schedules using serial SGS in the backward pass;
  end
  else
    for $i = 1, 2, \ldots, SN$ **do**
      Generate a population $\mathcal{T}$ of size N including x^i and $(N-1)$ solutions randomly selected from remaining solutions;
      Generate a new priority list V^i using $CFIA_{backward}$ with $\mathcal{T}$ as the initial population;
      Map the obtained priority lists in this phase to feasible schedules using serial SGS in the backward pass;
      if *comparing* x^i, V^i *results in a schedule with lower makespan* **then** $x^i = V^i$ and $trial_i$=0;
      else
        $trial_i = trial_i$ + Number of generated schedules within $CFIA_{backward}$;
      end
    end
  end
  Second strategy:
  for $i = 1, 2, \ldots, SN$ **do**
    if $trial_i > limit_{Onlooker}$ **then** Apply the Cplex CP Optimizer to solve the problem, use initial solution x^i, with a time limit of t_0.
    if *Cplex Cp optimizer found a solution* Sol_{cp} *with a lower makespan for the problem* **then** x^i=Sol_{cp}; $trial_i = 0$;
    else $trial_i = trial_i + \lfloor t_0/5 \rfloor$;
  end
  Phase 4: The scout bee phase
  for $i = 1, 2, \ldots, SN$ **do**
    if $trial_i > limit$ **then**
      if $rand < P_{Scout}$ **then**
        x^i=CFIA(The RCPSP instance,N, p, C);
        $trial_i = 0$;
      end
      else
        The scout bee phase:
        Generate a new list using (Equation (3.4)), then map it to a feasible schedule using serial SGS;
        Replace x^i with the obtained feasible schedule;
      end
    end
  end
end
Return: The best obtained schedule (the solution);

serial SGS in the backward pass. In this strategy, for a given $\lfloor P_{Onlooker} \times iterations \rfloor$ where $P_{Onlooker}$ is a real number in [0,1], a roulette wheel mechanism, Equation (3.6), is employed to select the solution x^i that its neighbourhood should be searched.

$$P_i = 1 - \frac{F(x^i)}{\sum_{j=1}^{SN} F(x^j)}. \tag{3.6}$$

As per Equation (3.6), a better fitness value of the i^{th} solution increases the likelihood of selecting x^i for generating a subsequent solution. After this, a local search is used around each chosen solution. When the i^{th} solution is selected, Equation (3.5) generates a new priority list in the neighbourhood of the x^i. If the priority list provides a better solution than x^i, then x^i is replaced by the generated solution. Otherwise, the trial counter increments by 1. During the remaining iterations in the first strategy, onlooker bees employ the $CFIA_{backward}$ to explore the feasible space. We denote the CFIA when serial SGS is in the backward pass as $CFIA_{backward}$. The initial population for the $CFIA_{backward}$ includes the target solution and $N-1$ randomly selected solutions from the remaining $SN-1$ solutions.

The onlooker bee phase plays a crucial role in the exploitation phase of the algorithm and explores the neighbourhood of solutions selected through the roulette wheel mechanism. $CFIA_{backward}$ contributes a robust local search capability, enhancing the algorithm's exploitation potential in the onlooker bee phase of CABFIA.

In the second strategy, we employ the Cplex CP optimizer to enhance the neighbourhood search of the solutions chosen by the first strategy in the onlooker bees, conducting trials exceeding the threshold of $limit_{Onlooker}$. The solutions chosen by the first strategy are the initial solutions for the Cplex CP optimizer, with the time limit set to t_0. The Cplex CP Optimizer does not employ serial SGS, and obtaining the exact number of generated schedules is impossible. However, given that the reported minimum time for generating a feasible solution for each instance is five seconds[27], [36], we consider $\lfloor t_0/5 \rfloor$ as the number of generated schedules when utilising the Cplex CP Optimizer.

3.4.2.3 The scout bee phase

The scout bee phase of the CABFIA is designed to perform a balanced local and global search and includes a restart mechanism to escape from getting trapped in a local optimum. In the CABFIA, each solution with more than *limit* trials, for P_{Scout} percentage of iterations, is replaced by a solution found by the CFIA. To maintain the diversification of the solutions as well as take advantage of a local search, the CFIA is initialised with the random population(randomly generated priority lists) and far fewer iterations than the original CFIA [1]. In this stage, the priority lists in the CFIA are mapped to feasible schedules using serial SGS. Employing CFIA in P_{Scout} percentage of iterations improves the convergence speed of the CABFIA. The restart mechanism is employed in the remaining $1 - P_{Scout}$ percentage of iterations. The generated priority lists using Equation (3.4) are mapped to feasible schedules using serial SGS.

3.5 Computational experiments for the ABFIA

This section provides the computational experiments on ABFIA and CABFIA. To assess the performance of the proposed ABFIA, we compare the performance of the ABFIA over several

benchmark functions, and that of ABC and certain variants of the ABC, such as the GABC [83], two modified ABC algorithms, namely the ABC-Best 1 and ABC-Best 2 [75], the MABC [84], the ABCx [80], and the ABCFWS [82], and with FIA, the grey wolf optimiser (GWO) [42], the whale optimisation algorithm (WOA) [63], the hybrid of GWO with WOA (WOAGWO) [224], the dragonfly algorithm (DA) [60], the salp swarm algorithm (SSA) [61], and the learner performance based behavior algorithm (LPB) [72]. We use Matlab R2020a version 9.8.0.1323502 under Windows 10 operating system, and all experiments are conducted on a personal machine with an Intel(R) Core™ i7 clocked at 2 GHz, and 6 GB of memory. All tested functions are of type minimisation.

Next, we explain the benchmark functions, followed by tuning the parameters of algorithms in Section 3.5.2 and the convergence analysis of the ABC, FIA and ABFIA in Section 3.5.3. In Sections 3.5.4 to 3.5.7, we detail outcomes of the computational experiments.

3.5.1 Benchmark problems for ABFIA

It is important to benchmark the performance of heuristic and metaheuristic optimisation algorithms on problems with different characteristics. The main characteristics that shape the problem's landscape are modality, dimensionality, separability, and basins.

The peaks in the landscape, which represent global or local minimum areas (for a minimisation problem), are defined by modality. The unimodal functions with one global minimum are suitable to test the exploitation and local search abilities of an optimisation algorithm. However, multimodal functions contain many local minimum areas and are commonly used to test the exploration ability of the algorithms [225]. Both unimodal and multimodal functions are often rotated to add complexity in the landscape [226]. Table 3.1 presents 20 well-known basic benchmark functions including 8 unimodal and 12 multimodal functions that we investigated in the present study.

The other main characteristic of benchmark functions is separability. Optimisation problems in which each variable is independent of other variables are called separable. Separable benchmark functions are relatively easy to solve. The optimum value for each variable can be found by solving an independent optimisation process. Non-separable functions, e.g., Griewangk, Ackley, and Alpine functions, include variables related to each other, and as such, the variables cannot be optimised independently.

The other characteristic that shapes the problem's landscape is basins. A basin is a sharp falling area surrounding a wide region. Optimisation algorithms are typically attracted to such regions, and it is challenging for those algorithms to escape from there. In multimodal problems, e.g., in the Ackley function, which contains a narrow, deep basin in the middle [226], finding the basin of global minima and avoiding the basin of local minima is usually challenging for metaheuristic algorithms.

Increasing dimensionality of the instances exponentially enlarges the landscape, which leads to an increase in local optimum areas. Low-dimensional instances are relatively easy to solve, and as such highly dimensional instances are suitable for evaluating the performance of optimisation algorithms. Our experiments consider various functions representing the those characteristics, including 30-, 60- and 200-dimensional benchmark functions. As an additional evaluation on the ABFIA, a set of 10 modern CEC2019 benchmark test functions, which are designed to be used in the 2019 annual optimization contest, is used [227]. Table 3.2 shows these test functions, which are known as "The 100-Digit Challenge" and include multimodal, unimodal,

Table 3.1: The basic numeric functions

Test Function	Name	Search Range	C	Function
F1	Sphere	[-100,100]	US	$f(x_1 \cdots x_n) = \sum_{i=1}^{n} x_i^2$
F2	Elliptic	[-100,100]	UN	$f(x_1 \cdots x_n) = \sum_{i=1}^{n} (10^6)^{(i-1)/(n-1)} x_i^2$
F3	Sumsquars	[-10,10]	US	$f(x_1 \cdots x_n) = \sum_{i=1}^{n} i x_i^2$
F4	SumPower	[-10,10]	MS	$f(x_1 \cdots x_n) = \sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$
F5	schwefel2.22	[-10,10]	UN	$f(x_1, ..., x_n) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$
F6	Quartic	[-1.28,1.28]	US	$f(x_0 \cdots x_n) = \sum_{i=0}^{n} i x_i^4 + \text{random}[0, 1)$
F7	Rosenbrock	[-10,10]	UN	$f(\mathbf{x}) = f(x_1, ..., x_n) = \sum_{i=1}^{n-1} [b(x_{i+1} - x_i^2)^2 + (a - x_i)^2]$
F8	Rastrigin	[-5.12,5.12]	MS	$f(x_1 \cdots x_n) = 10d + \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i))$
F9	Griewank	[-600,600]	MN	$f(x_1 \cdots x_n) = 1 + \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$
F10	schwefel2.26	[-500,500]	MN	$f(\mathbf{x}) = f(x_1, x_2, ..., x_n) = 418.9829n - \sum_{i=1}^{n} x_i sin(\sqrt{\lvert x_i \rvert})$
F11	Ackley	[-32,32]	MN	$f(x_1 \cdots x_n) = -20exp(-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}) -$ $exp(\frac{1}{n} \sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$
F12	Alpine	[-10,10]	MS	$f(x_1 \cdots x_n) = \sum_{i=1}^{n} x_i sin(x_i) + 0.1 x_i$
F13	Schaffer	[-100,100]	MN	$f(x_1 \cdots x_n) = 0.5 + \frac{sin^2 \sqrt{\sum_{i=1}^{n} x_i^2} - 0.5}{(1 + 0.001(\sum_{i=1}^{n} x_i^2))^2}$
F14	Himmelblau	[-5,5]	MS	$f(x_1 \cdots x_n) = \frac{1}{n} \sum_{i=1}^{n} (x_i^4 - 16x_i^2 + 5x_i)$
F15	Shifted Rastrigin	[-5.12,5.12]	MS	$f(x_1 \cdots x_n) = 10d + \sum_{i=1}^{n} (z_i^2 - 10cos(2\pi z_i)), z = x - o$
F16	Shifted Griewank	[-600,600]	MN	$f(x_1 \cdots x_n) = 1 + \frac{1}{4000} \sum_{i=1}^{n} z_i^2 - \prod_{i=1}^{n} cos(\frac{z_i}{\sqrt{i}}), z = x - o$
F17	Shifted Ackely	[-32,32]	MN	$f(x_1 \cdots x_n) = -20exp(-0.2\sqrt{\frac{1}{n} \sum_{i=1}^{n} z_i^2}) -$ $exp(\frac{1}{n} \sum_{i=1}^{n} cos(2\pi z_i)) + 20 + e, z = x - o$
F18	Shifted Alpine	[-10,10]	MN	$f(x_1 \cdots x_n) = \sum_{i=1}^{n} z_i sin(z_i) + 0.1 z_i, z = x - o$
F19	Discus	[-5.12,5.12]	US	$f(x_1 \cdots x_n) = 10^6 x_1^2 + \sum_{i=2}^{n} x_i^2$ $exp(\frac{1}{n} \sum_{i=1}^{n} cos(2\pi z_i)) + 20 + e, z = x - o$
F20	Schwefel2.20	[-10,10]	US	$f(x_1, ..., x_n) = \sum_{i=1}^{n} \lvert x_i \rvert$

expanded multimodal, and hybrid composition functions. In the table, functions F21 to F23 have different dimensions, and functions F24 to F30 are shifted and rotated. We refer the interested reader to [227] for details.

3.5.2 Parameter settings

There are three groups of parameters in the hybrid ABFIA algorithm, i.e., parameters for the FIA component, parameters for the ABC component, and parameters due to the hybridisation. In this section, we discuss setting the value of those parameters.

3.5.2.1 Parameters setting for FIA

There are three parameters in this group, i.e., C (the search restart control), N (the population size) and p (the probability for a crossover) in Algorithm 4. In the present study, the values of all parameters for the FIA are as suggested by [1]. More precisely, we set $p = 0.25$. As for the stopping criterion, we note that setting a large value for the stopping criterion of the FIA within the ABFIA means that the FIA component is preferred heavily (compared to the ABC component). Following some preliminary experiments, for each iteration, we set the stopping criterion for the FIA within ABFIA equal to 12 function evaluations and $C = 5$, which is less than half of the stopping criterion.

Table 3.2: The 100-Digit Challenge Test Functions

Test Function	Name	C	D	Search Range
F21	Storn's Chebyshev Polynomial Fitting Problem	MN	9	[-8192,8192]
F22	Inverse Hilbert Matrix Problem	MN	16	[-16384,16384]
F23	Lennard-Jones Minimum Energy Cluster Problem	MN	18	[-4,4]
F24	Shifted and Rotated Rastrigin's Function	MN	10	[-100,100]
F25	Shifted and Rotated Griewank's Function	MN	10	[-100,100]
F26	Shifted and Rotated Weierstrass Function	MN	10	[-100,100]
F27	Shifted and Rotated Schwefel's Function	MN	10	[-100,100]
F28	Shifted and Rotated Expanded Schaffer's F6 Function	MN	10	[-100,100]
F29	Shifted and Rotated Happy Cat Function	MN	10	[-100,100]
F30	Shifted and Rotated Ackley Function	MN	10	[-100,100]

3.5.2.2 Parameters setting for ABC

The best outcome in the ABC algorithm is obtained when the number of solutions is equal to the number of employed and onlooker bees[228]. In all experiments conducted in the present chapter, the same number of employed and onlooker bees are used, both of which are equal to the number of solutions (denoted as SN) and are set to 100. The parameter $limit$, which determines the maximum number of trials for a solution to be abandoned, is set according to sixty percent of the problem dimensions multiplied by the number of employed bees, i.e., $0.6 \times SN \times D$.

3.5.2.3 Parameters setting for ABFIA

The maximum number of iterations, run time, and the maximum number of function evaluations (FEs) are commonly used as stopping criteria for heuristic algorithms.

To have a fair comparison when reporting the results of ABFIA, the stopping criterion changed according to the study that we report their results. In experiments 1 and 2, the maximum number of function evaluations is considered as the stopping criterion, and the maximum number of iterations is used in experiment 3.

The parameters $P_{Onlooker}$ and P_{Scout} in the ABFIA are set to balance between the first and second strategies of the onlooker bee phase and restart procedure in the scout bee phase. Parameter $P_{Onlooker}$ in the ABFIA is the probability of using the original onlooker bee phase from ABC and increasing this parameter up to 1, decreases the probability of using FIA within this phase. Parameter P_{Scout} determines the probability of using FIA in the scout bee phase. To tune parameters $P_{Onlooker}$ and P_{Scout} we use F20, which is a unimodal function, and F9 and F12, which are two multimodal benchmark functions. For different values of $P_{Onlooker}$ between zero and one, we report the average of the results of solving those three test functions for each 30-, 60-, and 200-dimension instances. The maximum number of function evaluations is set to $1000 \times D$. Table 3.3 shows the impact of $P_{Onlooker}$ on the results while $P_{Scout} = 0$ which means the scout bee phase is the same as original ABC.

The parameter P_{Scout} is then tuned while using the best value for the $P_{Onlooker}$. the average of three independent runs is considered per instance, meaning that we run the ABFIA three times, and we report the average results over those three runs. According to Table 3.3, we set $P_{Onlooker} = 0.8$ for all instances and solve the same test functions for different values of P_{Scout}

Table 3.3: The impact of parameter $P_{Onlooker}$ on different benchmark functions. $P_{Scout} = 0$.

$P_{Onlooker}$	30D	60D	200D	Average
0	1.02E+01	7.45E+01	7.87E+06	2.62E+06
0.10	2.22E-03	2.78E-01	1.22E+00	5.01E-01
0.20	1.20E-04	2.85E-01	1.76E-01	1.54E-01
0.30	7.12E-07	9.29E-04	4.83E-02	1.64E-02
0.40	1.09E-07	1.00E-04	1.40E-02	4.69E-03
0.50	4.09E-08	4.65E-05	2.03E-03	6.93E-04
0.60	1.29E-08	7.08E-06	9.30E-04	3.12E-04
0.70	**3.33E-09**	2.24E-05	3.52E-04	1.25E-04
0.80	3.27E-09	**4.20E-06**	**4.00E-05**	**1.47E-05**
0.90	4.60E-09	9.89E-06	2.27E-04	7.89E-05
1.00	7.79E-09	3.52E-05	2.95E-04	1.10E-04

between zero and one. Table 3.4 shows that for 30-dimensional problems $P_{Scout} = 0.1$ provides the best results and for 60- and 200-dimensional test functions $P_{Scout} = 0.2$ leads to the best value. Based on these experiments, in this research we set $P_{Onlooker} = 0.8$ and $P_{Scout} = 0.2$ for all of the experiments.

Table 3.4: The impact of parameter P_{Scout} on different benchmark functions. $P_{Onlooker} = 0.8$.

P_{Scout}	30D	60D	200D	Average
0	1.60E-12	1.53E-09	4.84E-09	2.12E-09
0.1	**5.06E-13**	1.05E-08	4.71E-08	1.92E-08
0.2	6.34E-13	**7.61E-11**	**4.80E-10**	**1.86E-10**
0.3	6.64E-13	8.24E-10	2.20E-08	7.59E-09
0.4	1.91E-13	9.31E-10	7.39E-09	2.77E-09
0.5	9.72E-13	1.77E-09	1.07E-08	4.16E-09
0.6	2.88E-13	4.06E-09	4.98E-09	3.01E-09
0.7	3.50E-13	3.14E-09	2.20E-08	8.39E-09
0.8	3.22E-13	5.76E-09	4.66E-09	3.47E-09
0.9	1.82E-13	3.73E-10	1.36E-08	4.67E-09
1	2.62E-13	3.37E-09	7.89E-09	3.75E-09

3.5.3 Convergence comparison

In order to assess the convergence behavior of the ABFIA, we compare the ABFIA and the ABC and FIA on the 100-dimensional Griewangk and 10-dimensional Discus functions. The details of setting the parameters are explained in section 3.5.2.

Figure 3.2 demonstrates that the ABC algorithm converges slowly in comparison to the FIA and ABFIA on both functions. In solving the Discus function, the FIA converges faster than other algorithms. However, the ABFIA obtains better solutions. In the Griewangk function, ABFIA converges faster than other algorithms and delivers solutions closer to the global optima. As

shown in Figure 3.2, the FIA converges faster than the ABC but gets trapped in poorer local optima.

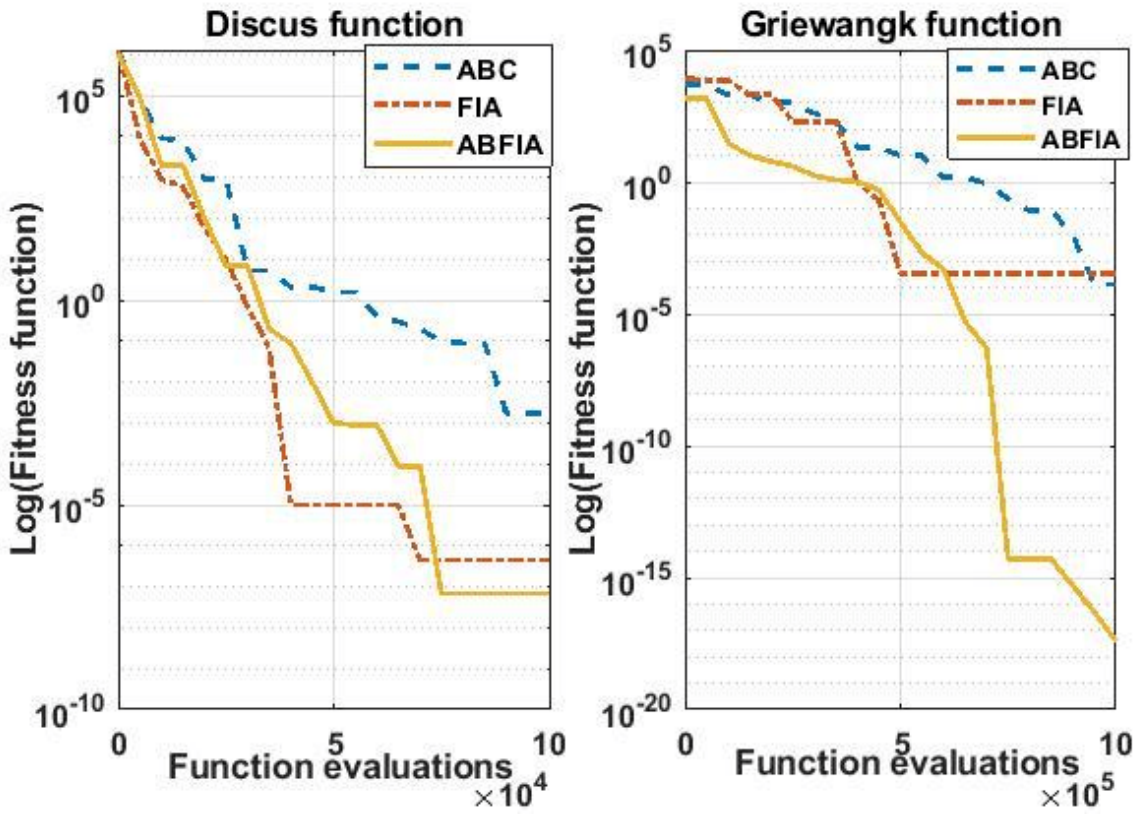

Figure 3.2: Convergence curves for the ABC, FIA and ABFIA.

Next, we detail the outcomes of the computational experiments.

3.5.4 Experiment 1: Unimodal benchmark test functions

Table 3.5: Average results of thirty runs for 30-dimensional unimodal functions

Test Function		ABFIA	FIA	ABC	GABC	ABCBest1	ABCBest2	MABC	ABCX	ABCFWS
	Ave	**0.00E+00**	5.47E-28	5.10E-16	4.31E-15	3.11E-47	5.96E-35	9.43E-32	**0.00E+00**	**0.00E+00**
F1	Std	**0.00E+00**	6.17E-28	7.41E-16	7.12E-17	3.44E-47	3.61E-35	6.67E-32	**0.00E+00**	**0.00E+00**
	Rank	**1**	7	9	8	4	5	6	**1**	**1**
	Ave	9.81E-32	2.11E-17	1.18E-15	3.62E-16	5.35E-24	1.70E-28	3.66E-28	**0.00E+00**	**0.00E+00**
F2	Std	1.02E-32	7.17E-17	3.92E-16	7.88E-17	4.91E-24	2.35E-28	5.96E-28	**0.00E+00**	**0.00E+00**
	Rank	3	7	9	8	6	4	5	**1**	**1**
	Ave	7.52E-39	2.15E-16	8.17E-16	4.55E-16	6.50E-48	5.55E-36	2.10E-32	**0.00E+00**	**0.00E+00**
F3	Std	8.41E-39	3.24E-16	7.41E-17	7.00E-17	6.04E-48	3.36E-36	1.56E-32	**0.00E+00**	**0.00E+00**
	Rank	4	7	9	8	3	5	6	**1**	**1**
	Ave	**0.00E+00**	1.41E-2	1.09E-15	1.35E-15	2.10E-25	1.36E-18	2.40E-17	1.84E-05	1.85E-03
F5	Std	**0.00E+00**	3.51E-2	2.04E-16	1.36E-16	9.08E-26	4.27E-19	9.02E-18	6.62E-06	2.53E-04
	Rank	**1**	9	5	6	2	3	4	7	8
	Ave	**0.00E+00**	3.93E-17	3.15E-16	1.21E-16	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F6	Std	**0.00E+00**	4.28E-16	5.42E-17	3.99E-17	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	**1**	7	9	8	**1**	**1**	**1**	**1**	**1**
	Ave	**3.12E-11**	4.13E-06	2.41E-02	3.21E-01	1.49E+01	5.45E+00	6.11E-01	3.08E+01	2.89E+00
F7	Std	**2.55E-11**	4.01E-06	3.59E-02	8.21E-01	2.87E+01	8.40E+00	4.55E-01	2.12E+01	2.02E+00
	Rank	**1**	2	3	4	8	7	5	9	6
	Average Rank	1.83E+00	6.50E+00	7.33E+00	7.00E+00	4.00E+00	4.17E+00	4.50E+00	3.50E+00	3.00E+00
	Sup	4	0	0	0	1	1	1	4	4

Table 3.6: Average results of thirty runs for 60-dimensional unimodal functions

Test Function		ABFIA	FIA	ABC	GABC	ABCBest1	ABCBest2	MABC	ABCx	ABCFWS
	Ave	**0.00E+00**	3.43E-03	1.11E-15	1.06E-15	3.92E-44	4.82E-33	6.03E-29	**0.00E+00**	**0.00E+00**
F1	Std	**0.00E+00**	5.42E-03	1.09E-16	1.21E-16	2.64E-44	2.59E-33	4.31E-29	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	8	7	4	5	6	**1**	**1**
	Ave	**0.00E+00**	2.91E-16	1.11E-15	8.97E-16	1.70E-41	5.86E-27	3.51E-25	**0.00E+00**	**0.00E+00**
F2	Std	**0.00E+00**	3.41E-16	1.32E-16	9.29E-17	9.16E-42	1.13E-26	2.72E-25	**0.00E+00**	**0.00E+00**
	Rank	**1**	7	9	8	4	5	6	**1**	**1**
	Ave	**0.00E+00**	8.18E-12	2.73E-15	1.04E-15	2.06E-44	9.10E-34	1.39E-29	**0.00E+00**	**0.00E+00**
F3	Std	**0.00E+00**	7.74E-12	3.19E-16	1.27E-16	1.83E-44	3.87E-34	8.84E-30	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	8	7	4	5	6	**1**	**1**
	Ave	**3.87E-57**	5.41E-01	1.14E-15	2.96E-15	8.48E-24	1.58E-17	6.96E-16	5.71E-02	1.62E-01
F5	Std	**2.23E-57**	6.43E-01	2.39E-16	1.85E-16	2.31E-24	3.32E-18	1.20E-16	1.08E-02	2.14E-02
	Rank	**1**	9	5	6	2	3	4	7	8
	Ave	**0.00E+00**	7.53E-09	4.28E-16	3.73E-16	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F6	Std	**0.00E+00**	2.44E-09	5.41E-17	6.67E-17	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	8	7	**1**	**1**	**1**	**1**	**1**
	Ave	**9.43E-17**	5.94E-02	8.41E-02	3.30E+00	5.04E+01	5.10E+01	1.51E+00	7.17E+01	4.34E+01
F7	Std	**3.28E-17**	6.28E-02	3.28E-01	1.28E+01	5.46E+01	3.77E+01	1.34E+00	2.54E+01	1.32E+01
	Rank	**1**	2	3	5	7	8	4	9	6
	Average Rank	1.00E+00	7.50E+00	6.83E+00	6.67E+00	3.67E+00	4.50E+00	4.50E+00	3.33E+00	3.00E+00
	Sup	6	0	0	0	1	1	1	4	4

In this section, the performance of ABFIA is evaluated on various unimodal 30- and 60-dimensional test functions.

As shown in Tables 3.5 and 3.6 we compare the outcomes of ABFIA, and those of FIA, ABC, and basic ABC variants which taken from [80] and [82]. In all algorithms, the stopping criterion is set to $5000 \times D$ number of function evaluations, and the objective function values of less than $1E-60$ are set to zero. For each function, the first and second rows of the tables present the mean and standard deviation of the best values of thirty independent runs. The third row shows the rank of each algorithm in this experiment. The last row of each table (Sup) shows the number of tests each given algorithm was the best. The algorithms are ranked according to the average objective function values for each test function. For 30-dimensional functions of F1, F5, F6, and F7, the ABFIA outperforms all other algorithms. The ABCx and ABCFWS in F2 and F3 functions, and ABC-Best 1 in F3 provide better solutions than ABFIA.

For solving 60-dimensional instances, the ABFIA outperforms all other algorithms, and ABCFWS and ABCx are ranked second and third, respectively (see Table 3.6). The results clearly show that the ABFIA outperforms the stand-alone ABC and FIA in all unimodal test functions, as well as the tested algorithms. ABFIA, ABCX, and ABCFWS were the best in four tests of 30-dimensional problems, and in solving 60-dimensional instances ABFIA was the best in all tests.

To show the scalability of our algorithm, we also increased the dimension of the test functions up to 200. Table 3.9 includes the results of 200-dimensional unimodal test functions and shows the ability of the ABFIA in dealing with high-dimensional problems. We note that we use the maximum number of function evaluations as the stopping criterion for a fair comparison and not the computation time.

3.5.5 Experiment 2: Multimodal benchmark test functions

In the second experiment, the performance of ABFIA is investigated over a set of 30-, 60- and 200- dimensional multimodal test functions.

Tables 3.7 and 3.8 represent the results of 30- and 60-dimensional instances, respectively. The maximum number of function evaluations in all runs is set to $5000 \times D$. Results are averaged

Table 3.7: Average results of thirty runs for 30-dimensional multimodal functions

Test Function		ABFIA	FIA	ABC	GABC	ABCBest1	ABCBest2	MABC	ABCX	ABCFWS
	Ave	**0.00E+00**	3.43E-07	5.71E-16	1.64E-17	**0.00E+00**	3.00E-46	**0.00E+00**	**0.00E+00**	**0.00E+00**
F4	Std	**0.00E+00**	2.11E-07	2.25E-17	8.07E-18	**0.00E+00**	1.07E-45	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	8	7	**1**	6	**1**	**1**	**1**
	Ave	**0.00E+00**	2.41E-03	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F8	Std	**0.00E+00**	8.28E-03	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	Ave	2.47E-21	3.38E-19	6.93E-11	3.70E-17	**0.00E+00**	1.81E-08	**0.00E+00**	**0.00E+00**	**0.00E+00**
F9	Std	5.22E-22	4.43E-20	2.18E-11	5.32E-17	**0.00E+00**	6.29E-08	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	5	6	8	7	**1**	9	**1**	**1**	**1**
	Ave	**2.18E-16**	5.96E-05	2.23E-12	9.42E-02	1.33E-12	1.76E-12	1.21E-13	5.23E-09	7.07E-06
F10	Std	**3.41E-16**	4.73E-05	1.52E-12	5.16E-01	8.18E-13	3.32E-13	4.53E-13	2.86E-08	2.38E-05
	Rank	**1**	8	3	9	4	5	2	6	7
	Ave	**9.63E-25**	5.41E-12	6.67E-14	3.20E-14	3.01E-24	3.07E-14	4.13E-14	1.04E-14	2.38E-14
F11	Std	**8.73E-25**	1.18E-12	8.33E-15	3.36E-15	2.91E-15	3.43E-15	2.17E-15	3.08E-15	3.38E-15
	Rank	**1**	9	7	6	2	5	8	3	4
	Ave	9.32E-26	8.44E-10	8.18E-10	3.41E-09	4.74E-16	9.47E-16	1.58E-16	**9.68E-58**	**1.57E-32**
F12	Std	2.28E-27	7.43E-10	6.13E-10	1.13E-08	1.80E-15	2.46E-15	2.48E-16	**1.52E-57**	**5.57E-48**
	Rank	3	8	7	9	5	6	4	**1**	2
	Ave	**8.36E-04**	7.11E-03	6.18E-01	2.66E-01	2.39E-01	2.81E-01	2.95E-01	8.36E-02	1.81E-01
F13	Std	**2.28E-05**	4.51E-04	7.24E-02	4.39E-02	6.13E-02	3.92E-02	3.17E-02	2.23E-02	3.72E-02
	Rank	**1**	2	9	6	5	7	8	3	4
	Ave	**-7.83E+01**	-7.67E+01	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**
F14	Std	**2.71E-13**	9.02E-02	**3.29E-10**	**2.94E-14**	**6.68E-12**	**4.86E-09**	**2.06E-07**	**-1.07E-01**	**8.25E-14**
	Rank	**1**	9	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	Ave	**0.00E+00**	1.96E-03	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F15	Std	**0.00E+00**	1.19E-03	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	Ave	9.88E-21	2.18E-9	5.32E-11	3.33E-17	8.81E-16	1.46E-07	**0.00E+00**	**0.00E+00**	1.54E-04
F16	Std	3.17E-20	5.25E-9	6.99E-10	5.17E-17	3.38E-15	7.78E-07	**0.00E+00**	**0.00E+00**	2.57E-04
	Rank	3	7	6	4	5	8	**1**	**1**	**9**
	Ave	**8.41E-16**	9.39E-12	3.67E-14	3.20E-14	2.89E-14	3.01E-14	4.92E-14	1.39E-14	1.61E-14
F17	Std	**7.33E-16**	3.47E-12	1.74E-14	2.80E-15	2.59E-15	3.70E-15	5.31E-15	2.16E-15	2.23E-15
	Rank	**1**	9	7	6	4	5	8	2	3
	Ave	**3.44E-19**	8.18E-08	1.23E-09	6.65E-08	1.50E-16	1.33E-13	1.38E-16	1.09E-16	8.30E-06
F18	Std	**3.33E-20**	7.41E-08	7.65E-09	2.39E-07	2.48E-16	4.89E-13	8.11E-17	7.76E-17	5.84E-06
	Rank	**1**	8	6	7	4	5	3	2	9
	Average Rank	1.67E+00	7.75E+00	5.33E+00	5.33E+00	2.83E+00	4.92E+00	3.25E+00	1.92E+00	3.58E+00
	Sup	9	0	3	3	5	3	6	7	5

over thirty independent runs, and values less than $1E-60$ are set to zero. In the tables, for each test function the first and second rows show the average and standard deviation of the best results, and the third row represents the rank of each algorithm. According to Table 3.7, and Table 3.8 except for functions F9, F12, and F16, the ABFIA outperforms all other algorithms in both 30- and 60-dimensional test functions. According to the average rank of the algorithms, and also according to the number of tests that each algorithm finds the bests results, the ABFIA shows the best performance, which proves the advantage of hybridising the ABC with FIA. The results of tests on 200-dimensional benchmark functions are also provided in Table 3.9, which further show the ability of our proposed algorithm in solving high dimensional multimodal problems.

3.5.6 Experiment 3: CEC2019

In experiment 3, the ABFIA is examined on CEC2019 test functions. These benchmark functions allow investigating the exploration and exploitation abilities of the algorithms.

Table 3.10 reports computational results of DA, WOA, LPB, WOAGWO, GWO, and also the average of the best results for thirty runs of ABFIA and the stand-alone FIA for each benchmark function. To have a fair comparison with the results of other algorithms, the maximum number of iterations in each run is set to 500. As the table shows, the ABFIA produces superior solutions in 8 out of 10 tested functions, demonstrating the ABFIA's ability to balance exploration and exploitation. Comparing the results of the ABFIA with ABC and FIA shows the impact of

Table 3.8: Average results of thirty runs for 60-dimensional multimodal functions

Test Function		ABFIA	FIA	ABC	GABC	ABCBest1	ABCBest2	MABC	ABCx	ABCFWS
	Ave	**0.00E+00**	1.28E-01	6.18E-17	2.85E-17	0.00E+00	7.53E-39	3.00E-59	0.00E+00	0.00E+00
F4	Std	**0.00E+00**	7.13E-02	2.71E-18	1.01E-17	0.00E+00	3.95E-38	3.87E-59	0.00E+00	0.00E+00
	Rank	**1**	9	8	7	**1**	6	5	**1**	**1**
	Ave	**0.00E+00**	3.11E-02	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F8	Std	**0.00E+00**	5.14E-02	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	Ave	1.35E-19	8.41E-11	2.11E-16	2.47E-04	**0.00E+00**	3.96E-09	**0.00E+00**	**0.00E+00**	**0.00E+00**
F9	Std	5.27E-19	6.17E-11	1.34E-16	1.35E-03	**0.00E+00**	2.04E-08	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	5	7	6	9	**1**	8	**1**	**1**	**1**
	Ave	**7.81E-14**	3.16E-04	8.77E-01	3.97E+01	3.99E-11	3.95E+00	3.56E-11	7.90E+00	4.72E-05
F10	Std	**3.19E-14**	2.98E-04	3.61E+00	6.47E+01	3.64E-12	2.16E+01	2.18E-12	3.00E+01	7.73E-05
	Rank	**1**	5	6	9	3	7	2	8	4
	Ave	**5.32E-21**	7.17E-11	9.91E-14	7.31E-14	6.93E-14	7.47E-14	1.37E-13	2.74E-14	5.44E-14
F11	Std	**3.92E-21**	7.11E-11	8.23E-14	5.57E-15	5.00E-15	4.12E-15	1.24E-14	3.86E-15	3.86E-15
	Rank	**1**	9	7	5	4	6	8	2	3
	Ave	8.17E-23	7.21E-09	3.64E-07	7.34E-07	5.29E-16	2.23E-11	8.20E-16	2.04E-17	**1.83E-26**
F12	Std	6.77E-23	3.49E-09	4.51E-08	1.70E-06	1.25E-15	3.77E-11	4.69E-16	1.11E-16	**2.04E-26**
	Rank	2	7	8	9	4	6	5	3	**1**
	Ave	**9.45E-03**	3.27E-02	5.11E-01	4.62E-01	4.61E-01	4.68E-01	4.84E-01	2.87E-01	3.93E-01
F13	Std	**4.02E-04**	4.13E-02	1.11E-03	1.79E-02	1.15E-02	9.17E-03	3.62E-03	2.67E-02	6.73E-02
	Rank	**1**	2	8	6	5	7	9	3	4
	Ave	**-7.83E+01**	-7.51E+01	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**	**-7.83E+01**
F14	Std	**5.14E-10**	3.08E-01	**2.44E-09**	**4.89E-14**	**3.71E-11**	**1.76E-08**	**2.40E-07**	**8.85E-02**	**2.11E-13**
	Rank	**1**	9	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	Ave	**0.00E+00**	5.31E-02	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F15	Std	**0.00E+00**	2.17E-02	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	**1**	9	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	Ave	6.08E-19	2.11E-10	3.26E-16	6.66E-17	**0.00E+00**	1.44E-08	**0.00E+00**	**0.00E+00**	**0.00E+00**
F16	Std	4.18E-19	3.67E-11	7.21E-17	1.08E-16	**0.00E+00**	7.17E-08	**0.00E+00**	**0.00E+00**	**0.00E+00**
	Rank	5	8	7	6	**1**	9	**1**	**1**	**1**
	Ave	**1.11E-14**	6.17E-11	8.47E-14	7.54E-14	6.90E-14	7.39E-14	2.00E-13	3.45E-14	3.88E-14
F17	Std	**5.98E-14**	9.43E-11	7.23E-15	5.00E-15	4.82E-15	3.54E-15	3.07E-14	3.97E-15	2.27E-15
	Rank	**1**	9	7	6	4	5	8	2	3
	Ave	**2.45E-17**	3.53E-08	2.27E-07	1.24E-05	1.80E-16	2.53E-10	9.71E-16	3.80E-16	4.59E-05
F18	Std	**5.14E-17**	5.23E-08	3.42E-07	5.65E-05	1.17E-16	1.17E-09	5.70E-16	3.09E-16	2.78E-05
	Rank	**1**	6	7	8	2	5	4	3	9
	Average Rank	1.75E+00	7.42E+00	5.58E+00	5.67E+00	2.33E+00	5.17E+00	3.83E+00	2.25E+00	2.50E+00
	Sup	9	0	3	3	6	3	5	6	7

hybridisation of these two algorithms, and according to the average rank of the algorithms, ABFIA outperforms all other methods.

3.5.7 Statistical analysis

We conduct non-parametric statistical test, including Friedman test [229] and then Holm's post-hoc test[230] as suggested in [231], to show the efficiency of the ABFIA in comparison to other methods tested in the present chapter in solving the challenging CEC2019 test functions (see Section 3.5.6).

We employ the Friedman statistical test with the null hypothesis (H_0) of no significant differences among the compared algorithms, and the alternative hypothesis (H_1) of existence of a significant difference among the algorithms' performance. We choose a typical significant level of 5%, i.e., $\alpha = 0.05$.

The Friedman test is a multiple comparison test that aims to detect significant differences between the results of two or more algorithms. The average rank of algorithms and p-value obtained from the Friedman test are shown in Table 3.11.

The p-value of the Friedman statistical test is equal to $3.00E-06$, which suggests rejecting the null hypothesis, showing therefore significant differences among the tested algorithms. The Friedman statistical test, however, can only conclude significant differences over multiple comparisons and cannot establish superiority of a particular algorithm. Therefore, we use Holm's post-hoc test method to compare ABFIA and other algorithms. Table 3.11 shows that in all

Table 3.9: Average results of thirty runs for 200-dimensional functions

Test Function	Unimodal functions		Test Function	Multimodal functions	
F1	Ave	2.55E-43	F4	Ave	7.53E-55
	Std	8.77E-43		Std	8.93E-55
F2	Ave	1.98E-41	F8	Ave	0.00E+00
	Std	2.01E-41		Std	0.00E+00
F3	Ave	4.74E-23	F9	Ave	2.46E-18
	Std	5.96E-23		Std	5.43E-18
F5	Ave	4.03E-34	F10	Ave	3.44E-08
	Std	6.86E-34		Std	7.18E-08
F6	Ave	3.23E-15	F11	Ave	6.01E-15
	Std	6.63E-15		Std	9.35E-15
F7	Ave	5.51E-09	F12	Ave	3.12E-18
	Std	7.15E-09		Std	4.98E-18
			F13	Ave	9.11E-03
				Std	4.17E-03
			F14	Ave	-7.83E+01
				Std	9.21E-09
			F15	Ave	4.63E-12
				Std	7.19E-12
			F16	Ave	1.15E-24
				Std	5.69E-24
			F17	Ave	2.39E-07
				Std	5.46E-07
			F18	Ave	9.98E-15
				Std	3.21E-15

experiments except for the LPB algorithm, the null hypothesis is rejected (p-value < 0.005). Therefore, hybridising ABC and FIA is superior than the stand-alone ABC and FIA. Also, the results show that the ABFIA is significantly superior to all tested algorithms except the LPB. For the LPB algorithm, the test did not reject the null hypothesis.

3.6 Computational experiments for the Combinatorial ABFIA

This section provides computational results of the ABFIA on benchmark sets of instances from the well-known PSPLIB [37]. We consider benchmark sets J30, J60 and J120, which include 30, 60, and 120 tasks, respectively. Benchmark sets J30 and J60 have 480 instances, and J120 contains 600. The complexity of these instances depends on three parameters: network complexity (NC), the resource factor (RF), and resource strength (RS) [90]. This chapter

Table 3.10: Average results of thirty runs for CEC2019 functions

Test Function		ABFIA	ABC	FIA	DA	WOA	SSA	LPB	WOAGWO	GWO
	Ave	6.93E+04	7.39E+07	1.22E+08	5.43E+10	4.11E+10	6.05E+10	7.49E+09	**4.76E+04**	2.13E+08
F21	Std	7.28E+03	2.44E+07	1.15E+08	6.69E+10	5.42E+10	4.75E+10	8.14E+09	**5.19E+03**	3.07E+08
	Rank	2	3	4	8	7	9	6	**1**	5
	Ave	**1.73E+01**	1.85E+01	1.73E+01	7.80E+01	1.73E+01	1.83E+01	1.76E+01	1.83E+01	1.83E+01
F22	Std	**8.24E-15**	3.26E-03	2.18E-03	8.78E+01	4.50E-03	5.00E-04	3.19E-01	4.72E-04	3.04E-04
	Rank	**1**	5	1	9	1	5	4	5	5
	Ave	1.27E+01	1.37E+01	1.37E+01	1.37E+01	1.37E+01	1.37E+01	**1.27E+01**	1.37E+01	1.37E+01
F23	Std	1.18E-15	9.37E-07	4.28E-08	7.00E-04	0.00E+00	3.00E-04	**0.00E+00**	1.83E-05	1.92E+00
	Rank	1	3	3	3	3	3	**1**	3	3
	Ave	**2.09E+01**	3.31E+02	4.38E+02	3.44E+02	3.95E+02	4.17E+01	7.79E+01	2.54E+02	3.01E+02
F24	Std	**1.11E+01**	4.33E+02	1.88E+02	4.14E+02	2.49E+02	2.22E+01	2.99E+01	5.39E+02	6.87E+02
	Rank	**1**	6	9	7	8	2	3	4	5
	Ave	**1.08E+00**	1.15E+00	2.01E+00	2.56E+00	2.73E+00	2.21E+00	1.19E+00	2.43E+00	2.43E+00
F25	Std	**4.31E-02**	3.41E-01	6.21E-02	3.25E-01	2.92E-01	1.06E-01	1.09E-01	2.62E-01	2.52E-01
	Rank	**1**	2	4	8	9	5	3	6	7
	Ave	5.22E+00	1.11E+01	2.53E+01	9.90E+00	1.07E+01	6.08E+00	**3.74E+00**	1.14E+01	1.19E+01
F26	Std	2.18E+00	4.77E+00	5.62E+00	1.64E+00	1.03E+00	1.49E+00	**8.23E-01**	1.64E+00	7.31E-01
	Rank	3	6	9	4	5	2	**1**	7	8
	Ave	**4.05E+01**	2.21E+02	3.57E+02	5.79E+02	4.91E+02	4.10E+02	1.45E+02	5.88E+02	5.35E+02
F27	Std	**1.21E+02**	1.09E+02	1.73E+02	3.29E+02	1.95E+02	2.91E+02	1.78E+02	3.49E+02	2.92E+02
	Rank	**1**	3	4	8	6	5	2	9	7
	Ave	**4.37E+00**	5.57E+00	4.93E+00	6.87E+00	6.91E+00	6.37E+00	4.89E+00	5.59E+00	5.40E+00
F28	Std	**5.55E-01**	1.64E+00	3.68E+00	5.02E-01	4.27E-01	5.86E-01	6.79E-01	1.02E+00	9.94E-01
	Rank	**1**	7	3	8	9	6	2	5	4
	Ave	**2.36E+00**	3.25E+00	7.01E+00	6.05E+00	5.94E+00	3.67E+00	2.89E+00	5.67E+00	1.47E+01
F29	Std	**1.11E-01**	2.12E+00	6.12E-01	2.87E+00	1.66E+00	2.36E-01	2.31E-01	8.81E-01	5.00E+01
	Rank	**1**	3	8	7	6	4	2	5	9
	Ave	**1.63E+01**	2.12E+01	2.14E+01	2.13E+01	2.13E+01	2.10E+01	2.00E+01	2.16E+01	2.15E+01
F30	Std	**9.28E+00**	6.52E-02	5.22E-02	1.72E-01	1.11E-01	7.80E-02	2.33E-03	9.22E-02	6.85E-02
	Rank	**1**	4	7	5	6	3	2	9	8
	Sup	8	0	1	0	1	0	2	1	0

compares the CABFIA with CFIA and some typical Bee algorithms in the literature. The quality of solutions is assessed using $\%Dev_L$, which represents the average percentage deviation between the values obtained by an algorithm and the lower bound values C_L.

For algorithm A, $\%Dev_L$ is defined as:

$$\%Dev_L = \frac{\sum_{p=1}^{P} \frac{C_{A,p}-C_{L,p}}{C_{L,p}} \times 100\%}{P}, \tag{3.7}$$

where P is the number of instances in a benchmark set, and $C_{A,p}$ is the makespan obtained by the algorithm A for the instance p. For the J30 benchmark, where the optimal solutions for all instances are known, $C_{L,p}$ denotes the optimal value for instance p. However, for the J60 and J120 benchmarks, $C_{L,p}$ represents the critical path length for the instance p.

We coded the algorithm using MATLAB R2020b. We conducted all the experiments on a machine with a CPU Intel Xeon Gold 6238R 2.2GHz. The following discusses the parameter setting and the detailed computational results.

3.6.1 Parameters setting for Combinatorial ABFIA

This section sets the parameters for the CFIA as an independent heuristic algorithm for solving the RCPSP, CFIA as the CABFIA's component, and the remaining parameters used in the CABFIA.

Table 3.11: Results of the statistical test while comparing ABFIA and other optimisation algorithms on CEC2019 functions

Algorithm	Average rank	Holm p-value (vs. ABFIA)
ABFIA	1.30E+00	-
LPB	2.70E+00	0.1224985
ABC	4.20E+00	0.0035477
SSA	4.60E+00	0.0012617
FIA	5.30E+00	0.0001049
WOAGWO	5.60E+00	3.66E-05
WOA	6.00E+00	7.50E-06
GWO	6.20E+00	3.52E-06
DA	6.80E+00	2.41E-07
Friedman test p-value	3.00E-06	-

In the CFIA, there are three parameters: C (the search restart control), N (the population size), and p (the probability for a crossover), as referenced in Algorithm 4. In this study, parameter values for the CFIA are adopted as suggested by [1] i.e., $N = 10$, $p = 0.25$, and $C = 120$. This algorithm's stopping criterion is the maximum number of generated schedules.

When incorporating CFIA as a component within CABFIA, it is essential to note that a large stopping criterion value for CFIA indicates a strong preference for the CFIA component in the CABFIA. In line with [17], we set the stopping criterion for the CFIA within CABFIA to be $RunCFIA_{limit}$ function evaluations. We aim to leverage CFIA to enhance the exploitation capability of CABFIA, so we eliminate the restart strategy in CFIA by ensuring that C exceeds the maximum number of generated schedules. The other parameters remain consistent with CFIA when independently solving RCPSPs.

To investigate the impact of each parameter $RunCFIA_{limit}$, SN, $limit$, $limit_{Onlooker}$, $P_{Onlooker}$, and P_{Scout}, on the performance of the algorithm, we set t_0=5sec which is enough time for the Cplex Cp optimizer to generate a feasible solution, and $limit_{Onlooker}$=$limit$ to ensure we used the best solution obtained by the onlooker bees as the initial solution for the Cplex Cp optimizer. We generated a subset of 48 instances $J60_{sub} \in \{$j60-1-1, j60-2-1, ..., j60-48-1$\}$, that includes one instance for each combination of NC, RF, and RS. For $RunCFIA_{limit} \in \{5, 10, 15, 20\}$, $SN \in \{10, 25, 50, 75, 100\}$, $P_{Onlooker} \in \{0.2, 0.4, 0.6, 0.8\}$, $P_{Scout} \in \{0.2, 0.4, 0.6, 0.8\}$, and for $limit$=$l \times SN \times D$ where $l \in \{0.4, 0.5, 0.6, 0.7\}$ and D=60 we solved all the instances in $J60_{sub}$. The average $\%Dev_L$ of 256 samples for each population size in Tables 3.12 to 3.16 shows the impact of each parameter on the performance of the CABFIA for the combinations of $RunCFIA_{limit}$, $limit$, $P_{Onlooker}$, and P_{Scout} when SN=10, 25, 50, 75, 100.

The averages of $\%Dev_L$ of all the experiments for SN=10, 25, 50, 75, 100 are 12.91, 13.50, 13.74, 13.90, and 14.00, respectively. These results demonstrate the adverse impact of increasing SN on the algorithm's performance, attributed to the fixed value of the stopping criterion used in our experiments. As SN increases, the algorithm's local search diminishes

To better investigate the impact of each single parameter on the algorithm's efficiency, we compared the average of $\%Dev_L$ of all the experiment results associated with each parameter when SN=10 in Figures 3.3 to 3.6. As is shown in Figure 3.3, increasing l to the values larger than 0.5 adversely impacts the algorithm's overall performance. Also Figures 3.4 to 3.6 show

Table 3.12: $\%Dev_L$ for SN=10 across all $J60_{sub}$

	l	0.40				0.50			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	12.83	12.83	12.95	12.97	12.96	12.98	12.86	13.02
0.2	0.4	12.88	12.82	13.06	12.90	12.94	12.73	12.92	12.80
0.2	0.6	12.76	12.80	12.69	12.71	12.69	12.87	12.96	12.86
0.2	0.8	12.74	12.83	12.99	12.99	12.61	12.79	12.80	12.77
0.4	0.2	12.69	12.96	12.81	12.90	12.96	12.93	12.92	13.02
0.4	0.4	12.78	12.71	12.86	12.78	12.85	12.94	13.06	12.98
0.4	0.6	12.93	12.81	13.12	13.08	12.90	12.89	12.71	12.79
0.4	0.8	12.75	12.89	12.71	12.87	12.88	12.83	13.06	12.83
0.6	0.2	13.10	12.80	13.00	12.98	12.87	12.97	13.00	12.94
0.6	0.4	12.91	12.86	12.80	12.95	12.71	12.90	12.86	12.88
0.6	0.6	12.79	13.16	12.83	12.69	12.91	12.86	13.04	12.92
0.6	0.8	12.87	12.92	12.77	12.84	**12.59**	12.86	12.75	12.91
0.8	0.2	13.02	13.16	13.05	13.05	12.98	12.89	13.04	12.67
0.8	0.4	13.12	13.18	12.91	12.82	13.18	13.03	13.08	12.95
0.8	0.6	12.90	12.94	13.11	13.25	13.06	12.91	12.70	12.89
0.8	0.8	12.93	12.96	12.98	13.04	13.08	12.87	13.09	13.09

Table 3.12 (continued)

	l	0.6				0.7			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	12.97	13.05	12.95	12.81	12.83	12.92	12.98	12.95
0.2	0.4	12.70	12.94	12.87	13.09	12.65	13.01	12.91	12.98
0.2	0.6	12.83	12.75	12.92	13.13	13.02	12.72	12.84	12.80
0.2	0.8	12.79	13.03	12.89	12.87	12.66	12.76	12.77	12.59
0.4	0.2	12.76	13.04	12.51	12.88	12.95	12.93	12.91	12.96
0.4	0.4	12.89	12.84	12.97	12.76	12.84	12.81	12.81	12.78
0.4	0.6	12.90	12.67	12.94	13.02	12.79	12.92	12.97	12.78
0.4	0.8	12.95	12.60	12.93	12.93	12.80	13.04	12.95	12.94
0.6	0.2	13.04	12.90	12.87	13.08	12.96	12.92	13.04	13.06
0.6	0.4	13.01	13.12	13.01	12.86	13.05	12.70	12.80	12.82
0.6	0.6	12.79	12.80	13.02	13.05	12.88	12.96	13.09	12.87
0.6	0.8	12.87	13.18	12.65	12.92	12.79	12.77	12.82	**12.50**
0.8	0.2	12.84	13.06	13.05	13.20	12.84	13.05	13.02	12.95
0.8	0.4	13.24	12.91	12.94	12.96	13.06	13.05	13.15	13.31
0.8	0.6	13.14	13.20	12.89	13.07	13.01	13.18	13.11	13.10
0.8	0.8	12.91	13.04	13.00	13.02	13.23	12.97	13.18	12.92

Table 3.13: $\%Dev_L$ for SN=25 across all $J60_{sub}$

	l	0.4				0.5			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	13.49	13.45	13.65	13.64	13.30	13.59	13.28	13.50
0.2	0.4	13.44	13.47	13.56	13.62	13.16	13.67	13.45	13.53
0.2	0.6	13.44	13.22	13.49	13.33	13.31	13.30	13.23	13.51
0.2	0.8	13.44	13.44	13.30	13.66	**13.10**	13.43	13.39	13.38
0.4	0.2	13.36	13.50	13.70	13.45	13.30	13.36	13.14	13.51
0.4	0.4	13.35	13.56	13.56	13.34	13.45	13.47	13.39	13.59
0.4	0.6	13.38	13.48	13.53	13.46	13.51	13.51	13.29	13.36
0.4	0.8	13.47	13.41	13.46	13.36	13.52	13.54	13.36	13.39
0.6	0.2	13.51	13.44	13.58	13.71	13.61	13.76	13.26	13.74
0.6	0.4	13.77	13.64	13.50	13.38	13.44	13.44	13.40	13.40
0.6	0.6	13.80	13.44	13.46	13.43	13.57	13.44	13.49	13.68
0.6	0.8	13.38	13.41	13.43	13.56	13.69	13.62	13.38	13.39
0.8	0.2	13.71	13.78	13.57	13.51	13.37	13.68	13.72	13.44
0.8	0.4	13.70	13.57	13.67	13.65	13.64	13.66	13.42	13.82
0.8	0.6	13.68	13.59	13.68	13.55	13.69	13.66	13.85	13.53
0.8	0.8	13.80	13.52	13.67	13.66	13.61	13.51	13.77	13.71

Table 3.13 (continued)

	l	0.6				0.7			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	13.47	13.26	13.49	13.52	13.35	13.35	13.27	13.46
0.2	0.4	13.56	13.56	13.33	13.41	13.37	13.28	13.58	13.20
0.2	0.6	13.51	13.39	13.56	13.38	13.24	13.46	13.49	13.53
0.2	0.8	13.30	13.43	13.40	13.61	**13.13**	13.44	13.47	13.44
0.4	0.2	13.55	13.66	13.36	13.62	13.49	13.52	13.44	13.49
0.4	0.4	13.28	13.16	13.47	13.37	13.47	13.36	13.62	13.28
0.4	0.6	13.50	13.55	13.61	13.60	13.54	13.54	13.46	13.27
0.4	0.8	13.55	13.46	13.44	13.30	13.34	13.48	13.50	13.27
0.6	0.2	13.29	13.37	13.42	13.44	13.64	13.46	13.29	13.47
0.6	0.4	13.59	13.40	13.43	13.52	13.38	13.30	13.58	13.49
0.6	0.6	13.50	13.21	13.29	13.49	13.45	13.27	13.69	13.63
0.6	0.8	13.40	13.53	13.55	13.55	13.70	13.69	13.27	13.50
0.8	0.2	13.70	13.69	13.55	13.64	13.71	13.53	13.51	13.68
0.8	0.4	13.74	13.57	13.43	13.68	13.74	13.64	13.61	13.75
0.8	0.6	13.58	13.43	13.50	13.63	13.39	13.47	13.76	13.46
0.8	0.8	13.71	13.36	13.81	13.61	13.75	13.54	13.54	13.65

Table 3.14: $\%Dev_L$ for SN=50 across all $J60_{sub}$

	l	0.4				0.5			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	13.66	13.82	13.67	13.69	13.57	13.86	13.84	13.84
0.2	0.4	13.67	13.86	13.61	13.67	13.63	13.77	13.70	13.81
0.2	0.6	13.61	13.73	13.80	13.67	13.44	13.92	13.47	13.65
0.2	0.8	13.77	13.63	13.69	13.89	13.65	13.76	13.60	13.73
0.4	0.2	13.45	13.69	13.86	13.49	13.51	13.58	13.72	13.81
0.4	0.4	13.92	13.48	13.74	13.72	13.79	13.66	13.63	13.58
0.4	0.6	13.85	13.58	13.73	13.67	13.91	13.65	13.44	14.00
0.4	0.8	13.51	13.71	13.54	13.70	13.77	13.67	13.94	13.77
0.6	0.2	13.63	13.69	13.67	13.79	13.67	13.68	13.71	13.67
0.6	0.4	13.51	13.72	13.82	13.75	13.68	13.86	13.64	13.78
0.6	0.6	13.75	**13.37**	13.53	13.93	13.73	13.68	13.80	13.76
0.6	0.8	13.95	13.94	13.72	13.88	13.84	13.80	13.70	13.98
0.8	0.2	13.86	13.60	13.64	14.02	13.98	13.64	14.03	13.89
0.8	0.4	14.14	14.06	13.89	13.72	13.79	13.74	13.86	13.87
0.8	0.6	13.63	13.76	13.58	13.85	14.07	14.06	13.77	13.80
0.8	0.8	13.53	13.87	13.79	13.95	13.86	13.73	13.84	13.75

Table 3.14 (continued)

	l	0.6				0.7			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	13.83	13.69	13.37	13.60	13.79	13.75	13.81	13.91
0.2	0.4	13.70	13.61	13.58	13.65	13.70	13.70	13.55	13.88
0.2	0.6	13.76	13.74	13.71	13.73	13.51	13.52	13.52	13.84
0.2	0.8	13.68	13.79	13.67	13.59	13.69	13.66	13.62	13.61
0.4	0.2	13.69	13.87	13.81	13.64	13.59	13.80	13.58	13.74
0.4	0.4	13.78	13.71	13.59	13.78	**13.23**	13.69	13.71	13.51
0.4	0.6	13.54	13.62	13.64	13.67	13.84	13.59	13.59	13.79
0.4	0.8	13.69	13.53	13.73	13.78	13.59	13.71	13.63	13.58
0.6	0.2	13.71	13.61	13.90	13.50	13.58	13.82	13.66	13.70
0.6	0.4	13.84	13.76	13.84	13.71	13.87	13.74	13.62	13.71
0.6	0.6	13.94	13.55	13.78	13.52	13.67	13.55	13.81	13.76
0.6	0.8	13.72	13.98	13.68	13.56	13.60	13.67	13.63	13.29
0.8	0.2	13.85	13.92	13.94	14.08	13.82	13.83	13.99	13.90
0.8	0.4	13.97	13.97	13.86	13.75	13.73	13.72	13.98	13.91
0.8	0.6	13.93	13.97	13.94	13.91	13.82	13.95	13.72	13.71
0.8	0.8	13.82	13.75	13.84	13.76	13.92	13.97	14.02	14.21

Table 3.15: $\%Dev_L$ for SN=75 across all $J60_{sub}$

	l	0.4				0.5			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	14.03	13.62	13.79	14.01	13.85	13.99	13.89	13.88
0.2	0.4	13.74	13.89	13.69	13.90	13.85	13.83	13.95	13.95
0.2	0.6	13.85	13.98	14.01	13.87	13.91	13.78	13.78	13.85
0.2	0.8	13.68	13.87	13.77	13.85	13.87	13.57	13.81	13.74
0.4	0.2	13.81	13.83	13.92	**13.50**	13.78	13.77	13.87	13.97
0.4	0.4	13.86	13.81	14.02	13.70	13.91	13.68	13.95	13.77
0.4	0.6	13.84	13.81	13.72	13.58	13.82	13.81	13.85	13.70
0.4	0.8	13.87	13.92	14.07	13.84	13.93	13.79	13.75	13.96
0.6	0.2	13.90	14.01	13.71	13.70	14.02	13.86	13.90	13.93
0.6	0.4	13.87	13.94	14.02	14.14	13.80	13.81	13.82	13.96
0.6	0.6	13.98	14.25	14.23	13.82	13.89	13.95	13.92	13.92
0.6	0.8	14.04	13.92	13.77	14.16	13.85	14.00	13.84	13.92
0.8	0.2	14.14	13.86	14.09	14.14	13.89	14.01	14.01	14.00
0.8	0.4	13.91	14.00	13.97	13.98	14.03	14.29	14.08	14.09
0.8	0.6	13.99	13.98	13.95	14.13	13.83	14.21	14.16	14.02
0.8	0.8	14.02	14.07	14.02	13.83	13.90	14.12	14.10	14.04

Table 3.15 (continued)

	l	0.6				0.7			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	13.85	13.95	13.88	14.04	13.89	13.98	14.00	13.82
0.2	0.4	13.81	13.91	13.77	13.84	13.62	13.88	13.84	13.80
0.2	0.6	13.81	13.82	13.91	13.79	13.90	13.87	13.94	13.75
0.2	0.8	13.79	13.91	14.11	13.91	13.65	14.00	13.93	13.71
0.4	0.2	13.57	13.90	13.85	13.88	13.73	13.89	13.75	13.63
0.4	0.4	14.02	13.93	13.79	13.72	13.95	14.01	13.75	13.87
0.4	0.6	13.86	13.79	13.82	13.62	13.80	13.73	13.84	13.88
0.4	0.8	13.96	14.01	14.01	13.96	13.75	13.81	13.94	13.96
0.6	0.2	13.91	14.05	13.83	13.80	13.96	13.85	13.72	13.86
0.6	0.4	13.99	13.95	14.00	13.97	14.13	13.96	13.81	13.95
0.6	0.6	13.89	13.85	13.90	13.92	14.07	13.88	14.08	13.76
0.6	0.8	13.92	14.05	13.95	13.87	13.88	13.70	13.70	13.64
0.8	0.2	13.83	14.04	13.99	13.85	13.90	13.83	14.20	13.91
0.8	0.4	**13.54**	14.01	13.82	13.77	13.98	14.01	14.12	14.14
0.8	0.6	14.10	14.04	13.68	13.82	13.90	14.15	14.14	14.04
0.8	0.8	14.09	14.18	14.02	13.98	13.96	14.01	13.95	14.04

Table 3.16: $\%Dev_L$ for SN=100 across all $J60_{sub}$

	l	0.4				0.5			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	13.93	13.91	13.81	14.05	13.92	13.99	13.85	13.97
0.2	0.4	14.08	14.06	13.80	13.93	14.17	14.00	14.03	13.76
0.2	0.6	13.85	13.93	13.94	14.08	14.02	13.93	14.05	14.13
0.2	0.8	14.19	**13.66**	14.07	14.06	13.88	13.86	13.86	13.93
0.4	0.2	14.08	14.15	13.88	14.11	14.10	13.70	13.69	14.04
0.4	0.4	14.13	14.10	13.73	13.96	14.25	13.95	13.93	13.96
0.4	0.6	13.78	13.74	13.74	13.64	14.12	13.93	13.77	13.78
0.4	0.8	13.89	13.96	14.06	13.97	14.01	13.93	14.03	14.02
0.6	0.2	13.84	13.99	14.08	14.33	14.08	14.04	14.01	14.13
0.6	0.4	14.29	13.82	13.83	14.02	14.09	13.96	13.97	14.04
0.6	0.6	13.89	13.70	13.94	14.04	14.10	14.16	14.08	13.92
0.6	0.8	13.91	13.95	13.85	13.75	14.05	13.91	13.99	13.89
0.8	0.2	14.04	14.09	14.14	14.04	13.91	14.52	14.35	13.95
0.8	0.4	14.15	14.23	14.18	14.04	14.34	14.10	14.03	14.13
0.8	0.6	14.10	14.28	14.19	14.12	14.16	14.31	14.16	14.09
0.8	0.8	14.27	14.20	14.26	14.25	14.15	14.16	14.17	14.12

Table 3.16 (continued)

	l	0.6				0.7			
	$RunCFIA_{limit}$	5	10	15	20	5	10	15	20
$P_{Onlooker}$	Pscout								
0.2	0.2	13.98	14.10	13.72	14.04	13.82	13.94	13.86	14.14
0.2	0.4	14.11	13.91	13.91	13.94	13.69	13.61	13.98	14.35
0.2	0.6	13.82	13.76	**13.56**	13.96	13.89	13.89	13.77	14.04
0.2	0.8	13.89	13.80	14.03	14.11	13.92	13.86	13.93	13.91
0.4	0.2	13.81	14.03	13.88	13.85	13.76	13.68	13.71	13.97
0.4	0.4	13.94	13.98	14.06	14.05	13.99	13.94	14.00	14.10
0.4	0.6	13.98	13.91	13.93	14.00	13.88	13.87	13.92	14.09
0.4	0.8	13.91	13.90	13.91	14.03	14.00	13.97	13.96	14.08
0.6	0.2	14.00	14.09	14.01	14.03	13.94	13.91	14.02	14.03
0.6	0.4	13.93	13.87	13.94	13.99	13.98	13.96	14.02	14.08
0.6	0.6	13.99	13.91	13.98	13.99	14.01	13.98	14.02	14.01
0.6	0.8	13.94	13.98	14.02	14.03	14.04	14.05	14.03	14.07
0.8	0.2	14.08	14.01	14.04	14.00	13.97	14.02	14.08	14.04
0.8	0.4	13.95	13.97	14.03	14.01	14.05	14.04	14.08	14.07
0.8	0.6	13.98	13.94	13.99	14.02	14.03	14.05	14.07	14.06
0.8	0.8	14.05	14.03	14.06	14.04	14.08	14.09	14.09	14.08

increasing P_{Scout} and reducing $RunCFIA_{limit}$, and $P_{Onlooker}$ increase the overall performance of the algorithm.

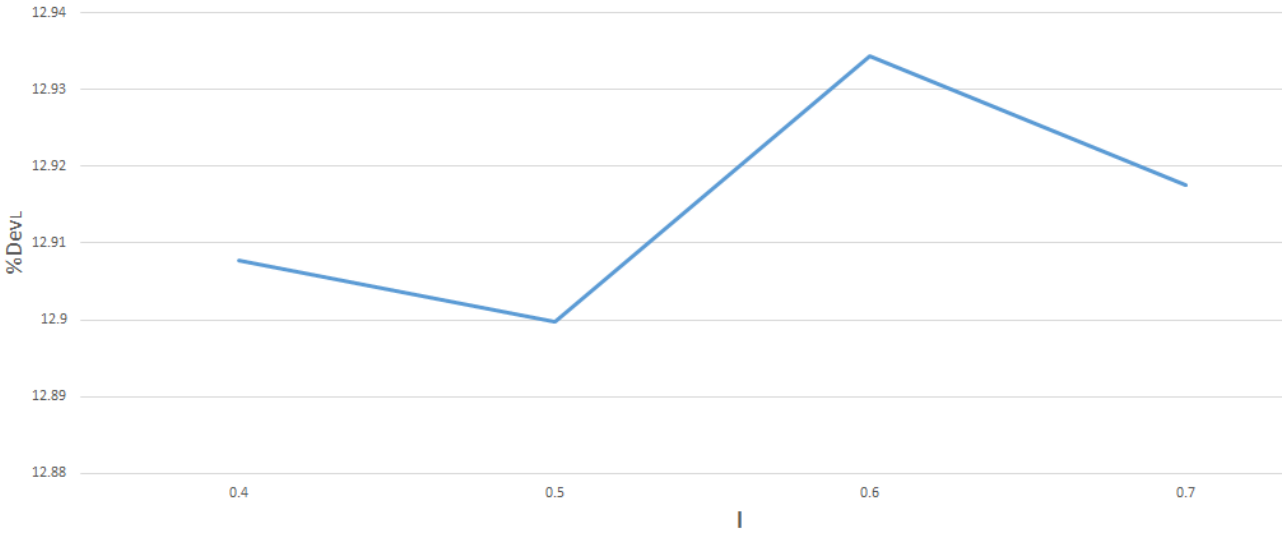

Figure 3.3: The impact of parameter l on the algorithm's overall performance

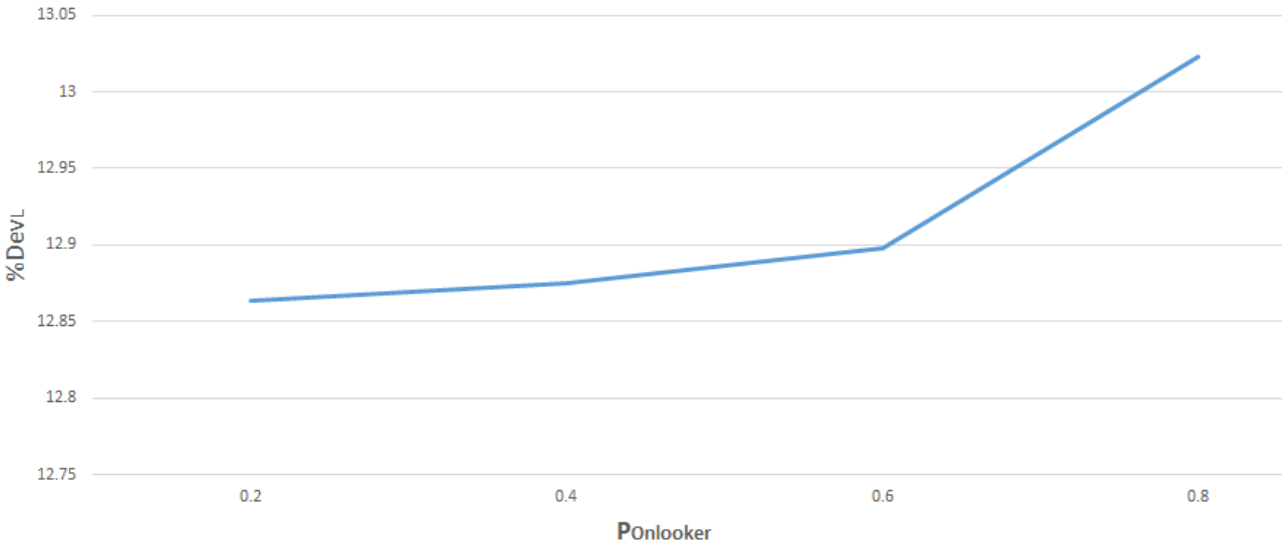

Figure 3.4: The impact of parameter $POnlooker$ on the algorithm's overall performance

According to Tables 3.12 to 3.16, $RunCFIA_{limit}$=20, SN=10, l=0.7, $P_{Onlooker}$ = 0.6, and P_{Scout}=0.8, are the best parameters resulted in $\%Dev_L$=12.50 for $J60_{sub}$. We use the same parameter values for solving all the instances. We also set the parameter $t0$=5sec, which, according to our preliminary experiments, is sufficient time to generate a feasible solution for all the instances[27]. We performed five runs for each instance of J30, J60, and J120 and presented the mean results in Table 3.17 to show the performance of the CABFIA.

3.6.2 Comparison with Bee algorithms

In this section, we compare the results obtained from CABFIA with CFIA, Bee algorithms, and their extensions. To assess the performance of the proposed CABFIA in comparison to existing Bee algorithms, we present $\%Dev_L$ of CFIA and CABFIA using Equation (3.7). In all

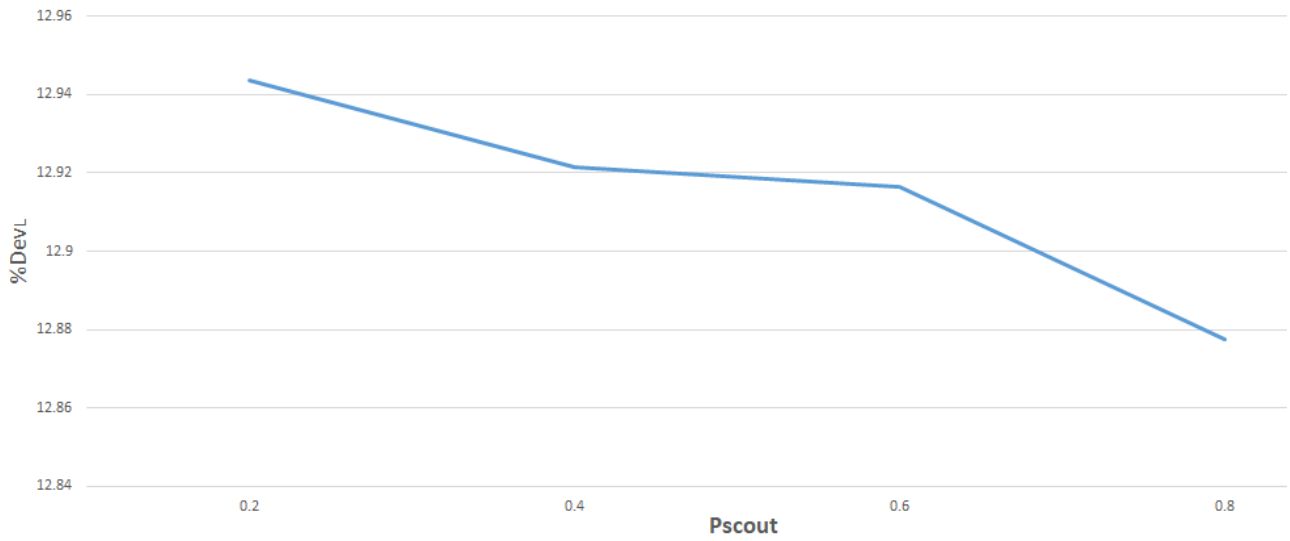

Figure 3.5: The impact of parameter *Pscout* on the algorithm's overall performance

Table 3.17: Performance of CABFIA for PSPLIB datasets

Algorithm	Schedule	Problem	J30	J60	J120
CABFIA	1000	$\%Dev_L$	0.00	10.77	31.95
		Computational time(Seconds)	12.36	53.53	289.98
	5000	$\%Dev_L$	0.00	10.69	31.58
		Computational time(Seconds)	316.25	890.79	1330.44
	50,000	$\%Dev_L$	0.00	10.62	31.48
		Computational time(Seconds)	1171.36	4020.54	19934.66

the experiments, we used the parameters suggested in Section 3.6.1 and the stopping criterion of 1000, 5000, and 50,000 generated schedules.

As indicated in Tables 3.18 to 3.20, CABFIA consistently outperforms all extensions of the Bee algorithms in solving for J30 and J60 across all experiments. This superiority is evident irrespective of whether the stopping criterion is set at 1000, 5000, or 50,000 generated schedules.

3.6.3 Comparison with existing algorithms

The comparative study in this section investigates the CABFIA's performance against the existing heuristics and metaheuristics algorithms i.e., R&S [36], [120], GA based algorithms[103]–[111], [118], PSO based algorithms [235]–[238], and other metaheuristic algorithms [90], [112], [239]–[242] in solving PSPLIB datasets.

The results for solving J30 instances are presented in table 3.21. As the table indicates, CABFIA consistently outperforms all algorithms, delivering superior results when the stopping criterion is set to 1000, 5000, and 50,000 generated schedules.

When addressing J60 instances, Table 3.22 highlights CABFIA as the top performer when the stopping criterion for all algorithms is set to 1000 generated schedules. However, with the stopping condition increased to 5000 schedules, Modified FA surpasses CABFIA. CABFIA achieves a $\%Dev_L$ of 10.62% with 50,000 generated schedules, while the best result is attained by R&S at 10.53%.

Table 3.18: Comparison of $\%Dev_L$ of the CABFIA, CFIA, and Bee algorithms in solving J30 instances

Algorithm	**Refernce**	**SGS**	**1000**	**5000**	**50,000**
ABC	[16]	Serial	0.98	0.57	0.20
BSO	[16]	Serial	0.65	0.36	0.17
BA	[16]	Serial	0.63	0.33	0.16
ABC-FBI	[16]	Serial	0.47	0.28	0.09
BSO-FBI	[16]	Serial	0.45	0.22	0.07
BA-FBI	[16]	Serial	0.42	0.19	0.04
BCOLS	[232]	Serial	0.34	0.17	–
BCO	[233]	Serial and Parallel	0.35	0.12	0.04
BCOGA	[125]	Serial and Parallel	0.25	0.15	0.02
BCOLS	[121]	Serial	0.21	0.05	–
BCOPSO	[234]	Serial	0.24	0.13	–
CFIA	This study	Serial	1.75	1.05	0.53
CABFIA	This study	Serial	**0.00**	**0.00**	**0.00**

Table 3.19: Comparison of $\%Dev_L$ of the CABFIA, CFIA, and Bee algorithms in solving J60 instances

Algorithm	**Refernce**	**SGS**	**1000**	**5000**	**50,000**
ABC	[16]	Serial	14.57	13.12	12.53
BSO	[16]	Serial	13.67	12.70	12.45
BA	[16]	Serial	13.35	12.83	12.41
ABC-FBI	[16]	Serial	12.61	12.24	11.23
BSO-FBI	[16]	Serial	12.58	12.29	11.21
BA-FBI	[16]	Serial	12.55	12.04	11.16
BCOLS	[232]	Serial	12.35	11.96	–
BCO	[233]	Serial and Parallel	12.75	11.48	11.18
BCOGA	[125]	Serial and Parallel	11.80	11.38	10.90
BCOLS	[121]	Serial	11.74	11.16	–
BCOPSO	[234]	Serial	12.14	11.90	–
CFIA	This study	Serial	16.02	14.82	14.65
CABFIA	This study	Serial	**10.77**	**10.69**	**10.62**

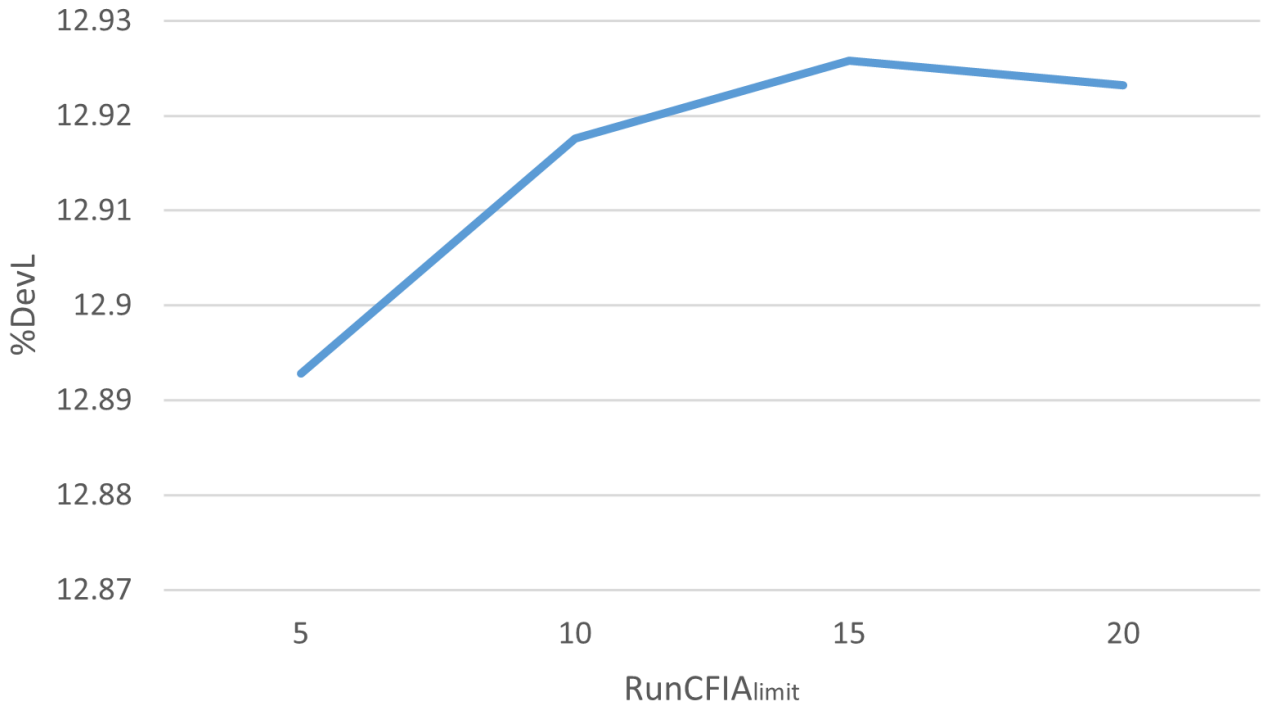

Figure 3.6: The impact of parameter $RunCFIA_{limit}$ on the algorithm's overall performance

Turning to J120 instances, Table 3.23 indicates that CABFIA outperforms the other algorithms for 1000 generated schedules. When the stopping criterion is 5000 generated schedules, CABFIA is ranked second. In the case of 50,000 generated schedules, GA(SA(FBI)) stands out, achieving the best results with a $\%Dev_L$ of 30.66%, while CABFIA records 31.48%.

3.7 Summary

In this chapter, a hybrid optimisation algorithm called ABFIA was proposed to solve nonlinear optimisation problems. This algorithm combines the FIA and ABC algorithms and uses the powerful exploration capability of ABC and the fast convergence ability of FIA. The proposed ABFIA was tested on various benchmark functions with up to 200 dimensions. According to the outcomes of the comprehensive experimental tests, the hybridisation of ABC and FIA ensures an excellent trade-off between exploration and exploitation of the ABC, improves the global exploration of the FIA, and reinforces the local exploitation ability of the ABC. Our results demonstrate that the ABFIA is a leading method for solving the tested functions.

ABFIA was designed to solve nonlinear optimisation problems. The idea of the ABFIA was employed in a new heuristic CABFIA for solving RCPSP. We investigated the algorithm's performances on datasets with 30, 60, and 120 tasks from PSPLIB. Comparing the results of the proposed version of the CABFIA with ABC and CFIA shows this hybridisation successfully improves both ABC and CFIA algorithms. The CABFIA provides competitive results against the compared existing algorithms and can successfully be used in solving an RCPSP.

Table 3.20: Comparison of $\%Dev_L$ of the CABFIA, CFIA, and Bee algorithms in solving J120 instances

Algorithm	**Refernce**	**SGS**	**1000**	**5000**	**50,000**
ABC	[16]	Serial	43.24	39.87	37.36
BSO	[16]	Serial	41.18	37.86	35.70
BA	[16]	Serial	40.38	38.12	36.12
ABC-FBI	[16]	Serial	37.85	36.82	35.02
BSO-FBI	[16]	Serial	37.84	36.51	34.86
BA-FBI	[16]	Serial	37.72	36.76	34.55
BCOLS	[232]	Serial	36.84	35.79	–
BCO	[233]	Serial and Parallel	36.29	34.18	33.69
BCOGA	[125]	Serial and Parallel	35.86	34.37	33.14
BCOLS	[121]	Serial	36.40	33.72	31.49
BCOPSO	[234]	Serial	36.15	35.28	–
CFIA	This study	Serial	45.64	43.62	43.54
CABFIA	This study	Serial	**31.95**	**31.58**	**31.48**

Table 3.21: Comparison of $\%Dev_L$ of the existing metaheuristics and CABFIA in solving J30 instances

Algorithm	**Schedule**		
	1000	**5000**	**50,000**
CABFI(This study)	**0.00**	**0.00**	**0.00**
R&S (FBI) [120]	-	-	0
R&S [36]	-	-	0
GA(SA(FBI))[103]	0.21	0.07	0.01
GA(FBI)[110]	0.17	0.06	0.02
Sequential(SS(FBI))[118]	0.10	0.02	0.00
GA(FBI)[109]	0.27	0.06	0.02
GA(FBI)[107]	0.14	0.04	0.00
GA(FBI)[108]	0.14	0.04	0.00
PSO(LS) [235]	0.44	0.13	0.01
GA(FBI) [106]	0.25	0.06	0.03
ALNS(FBI)[239]	0.18	0.07	0.02
SS(EM + FBI)[240]	0.27	0.11	0.01
GA(FBI)[104]	0.16	0.04	0.01
PSO(FBI)[236]	0.22	0.05	0.02
EA(FBI)[112]	0.24	0.09	0.03
GA(FBI)[111]	0.32	0.02	0.01
PSO(FBI)[237]	0.30	0.10	0.01
GA(FBI)[105]	-	0.08	0.03
DBGA [102]	0.15	0.04	0.02
COA [241]	0.04	0.00	0.00
Specialist(PSO(LS)[238]	0.26	0.04	0.01
MA [90]	-	0.00	0.00
Modified FA [242]	0.42	0.27	-

Table 3.22: Comparison of $\%Dev_L$ of the existing metaheuristics and CABFIA in solving J60 instances

Algorithm	**Schedule**		
	1000	**5000**	**50,000**
CABFI(This study)	**10.77**	10.69	10.62
R&S (FBI) [120]	-	-	**10.53**
R&S [36]	-	-	10.54
GA(SA(FBI))[103]	11.73	11.14	10.63
GA(FBI)[110]	11.45	11.00	10.69
Sequential(SS(FBI))[118]	11.38	10.93	10.58
GA(FBI)[109]	11.56	11.10	10.73
GA(FBI)[107]	11.55	10.96	10.57
GA(FBI)[108]	11.33	10.94	10.56
PSO(LS) [235]	11.74	10.94	10.62
GA(FBI) [106]	11.89	11.19	10.84
ALNS(FBI)[239]	11.58	11.12	10.73
SS(EM + FBI)[240]	11.73	11.10	10.71
GA(FBI)[104]	11.43	10.96	10.81
PSO(FBI)[236]	11.86	11.19	10.85
EA(FBI)[112]	11.90	11.31	10.91
GA(FBI)[111]	–	11.56	10.57
PSO(FBI)[237]	12.02	11.33	10.79
GA(FBI)[105]	–	11.68	10.66
DBGA [102]	11.45	10.95	10.68
COA [241]	11.13	10.77	10.58
Specialist(PSO(LS)[238]	11.74	11.13	10.68
MA [90]	-	10.72	10.55
Modified FA [242]	11.56	**10.29**	-

Table 3.23: Comparison of $\%Dev_L$ of the existing metaheuristics and CABFIA in solving J120 instances

Algorithm	**Schedule**		
	1000	**5000**	**50,000**
CABFI(This study)	**31.95**	31.58	31.48
R&S (FBI) [120]	-	-	31.07
R&S [36]	-	-	31.09
GA(SA(FBI))[103]	34.95	32.75	**30.66**
GA(FBI)[110]	34.29	32.34	30.75
Sequential(SS(FBI))[118]	34.01	32.52	31.16
GA(FBI)[109]	34.07	32.54	31.24
GA(FBI)[107]	35.18	33.11	31.28
GA(FBI)[108]	34.02	32.89	31.30
PSO(LS) [235]	35.77	33.46	31.43
GA(FBI) [106]	36.53	33.91	31.49
ALNS(FBI)[239]	34.35	32.91	31.54
SS(EM + FBI)[240]	35.22	33.10	31.57
GA(FBI)[104]	33.71	32.57	31.65
PSO(FBI)[236]	35.60	33.78	32.40
EA(FBI)[112]	35.83	34.08	32.52
GA(FBI)[111]	–	35.94	32.76
PSO(FBI)[237]	36.77	35.16	32.89
GA(FBI)[105]	–	37.61	33.82
DBGA [102]	34.19	32.34	30.82
COA [241]	34.04	32.90	31.22
Specialist(PSO(LS)[238]	35.20	32.59	31.23
MA [90]	-	32.76	31.12
Modified FA [242]	32.18	**29.59**	-

Chapter 4

A Matheuristic Approach for Multi-Mode Resource-Constrained Project Scheduling

4.1 Introduction

The multi-mode resource-constrained project scheduling problem (MRCPSP) is an NP-hard optimisation problem involving scheduling tasks under resource and precedence constraints, while there are several modes for executing each task. MRCPSP presents a significantly higher complexity level than the RCPSP, which is NP-hard. This chapter introduces a novel matheuristic for solving the MRCPSP. The approach involves generating a feasible solution for the MRCPSP, then iteratively relaxing and solving the relaxed problem. This innovative method effectively handles MRCPSPs under tight resource constraints, which are particularly challenging for other existing approaches.

The mathematical notation used in this chapter is provided in Table 4.1

The following summarises this approach.

1. The constrained programming approach is employed as it outperforms the mixed-integer programming approach in solving RCPSP and its extensions[243].
2. Generating a feasible solution for MRCPSP can be very challenging, so this thesis conducts a preprocessing valuable for addressing complexities in MRCPSPs.
3. A mathematical programming model, which is the generalisation of the multi-dimensional knapsack problem, is developed. That model conducts the mode selection process to generate an initial feasible solution.
4. An iterative process of relaxing the execution order of tasks and solving the relaxed problem is employed to solve the MRCPSP.
5. The performance of the proposed algorithm is evaluated by solving benchmark instances widely used in the literature.

The rest of this chapter is organised as follows. Section 4.2, explains the proposed R&S method for solving the MRCPSP. The computational experiments and results are presented in Sec-

tion 4.3. Finally, Section 4.4, concludes this chapter.

4.2 The Relax-and-solve method for multi-mode resource-constrained project scheduling problem

4.2.1 The algorithmic framework

This chapter proposes an R&S-based approach for the multi-mode RCPSP. I first incorporate a mode selection method to relax the original problem, which will be detailed in Section 4.2.2. The underlying idea behind mode selection includes optimising a multi-dimensional 0-1 knapsack mathematical program for selecting the modes. Solving that program results in assigning one mode to every task. As such, the relaxed problem is indeed a single-mode RCPSP. Given the mode for every task, I employ the CPLEX CP optimizer to generate a feasible initial solution for the original multi-mode RCPSP.

I note that this is the only step of the algorithm in which the number of modes is reduced, and because there are no changes to the precedence and resource constraints of the original multi-mode RCPSP, the obtained feasible solution by the CPLEX CP optimizer for the relaxed single-mode RCPSP is feasible for the original multi-mode RCPSP.

Upon obtaining a feasible solution, the relax operation takes place (see Section 4.2.3). During the relax operation, I employ a rolling time window mechanism in order to define the relaxed and non-relaxed sequences. The relaxed problem is then solved using the CPLEX CP optimizer solver (the solve phase; see Section 4.2.4 for details). The obtained solution in each iteration is feasible for the original problem. As explained in Section 4.2.3, the algorithm relaxes the problem by adding extra constraints to forbid re-ordering some selected tasks. As the constraints of the original MRCPSP are still present in the relaxed problem, the solution for the relaxed problem is always feasible for the original MRCPSP. However, the feasible space is much reduced in the relaxed problem due to the added "start-at-end" constraints (as the result of the relax phase). Also, the solution from the previous iteration is feasible for the current relaxed problem, which can be used as the warm-start to speed up the CPLEX CP optimizer.

The general R&S algorithm for the MRCPSP is summarised in Algorithm 6. All the mathematical notations used in this chapter are provided in Table 4.1.

4.2.2 Prepossessing and generating initial schedules

In this section, I explain how I obtain feasible initial schedules for multi-mode RCPSP. Note that solving the MRCPSP involves selecting an execution mode for each task, in addition to meeting the precedence and resource requirements. The execution mode for each task determines the duration of the task and the amount of required resources.

In the MRCPSP literature, two main approaches were proposed for dealing with the execution modes for tasks choosing the execution modes for tasks, namely the mode reduction and the mode selection. The mode reduction approach was introduced in [149] to reduce the search space and the associated search cost. Their approach was followed in [22], [164], [165], [167]. In [149], redundant non-renewable resources, inefficient modes, and non-executable modes were formally defined, and it was suggested to remove those from input data. In their definition, redundant non-renewable resources are the resources that can be excluded without an effect on

Table 4.1: The mathematical notation used in this chapter.

Problem settings	
Sets and indices	
J	Set of all the tasks $J = \{1, \ldots, n\}$ indexed by i
M_i	Set of execution modes for task i indexed by m_i
V	Set of nodes $V = \{0, \ldots, n+1\}$ indexed by i
A	Set of arcs to represent precedence relation between tasks
$\tau(t)$	Set of tasks that are active at time t
k_r	Index for the renewable resources
k_n	Index for the non-renewable resources
Parameters	
NR	Number of non-renewable resources
RR	Number of renewable resources
r^r_{i,m_i,k_r}	Required renewable resources k_r to operate task i in mode m_i
r^n_{i,m_i,k_n}	Required non-renewable resources k_n to operate task i in mode m_i
d_{i,m_i}	Duration of task i executing in mode m_i
$R^n_{k_n}$	The total amount of non-renewable resource k_n
$R^r_{k_r}$	The availability of renewable resource k_r at each time
T	Planning time horizon
Variables	
m_i	Mode selected for task i
S_i	The start time of task i
$C_{\max}$	The makespan of the project
R&S settings	
Parameters	
$overlap$	The overlap between different relaxed problems
L	The length of each time window in R&S
$Iternum$	The total number of iterations in solving an instance
$Ct_{\text{init-cplex}}$	The time limit of the CPLEX CP optimizer for obtaining an initial solution
Ct_{cplex}	The time limit of the CPLEX CP optimizer for solving each relaxed problem
$Ct_{\text{stop-R\&S}}$	The time limit of R&S (our stopping criterion)
$Ct_{R\&S}$	The average of the total computation time R&S spent for a solving a set of instances
Variables	
st	The starting time of a time window in R&S

Algorithm 6 The R&S for MRCPSP.

Input: An MRCPSP instance.
Output: A feasible schedule, or determining the infeasiblity of the instance.
Generate an initial feasible solution x^0 (see Algorithm 7);
Set time window, starting time $st = 0$ and iteration counter $k = 0$;
while *the stopping condition is not met* **do**
    if $st > C_{\max} - L$ **then**
        $st = 0$;
    end
    Determine the tasks outside the time window (group 1) and the tasks of within the time window (group 2) based on solution x^k (see Sections 4.2.3.1);
    $k = k + 1$;
    Generate a relaxed problem (see Section 4.2.3);
    Solve the relaxed problem using the CPLEX CP optimizer (Section 4.2.4) with solution x^k;
    $st = st + L - overlap$;
end
return *The best obtained schedule (the solution);*

Algorithm 7 Generating an initial solution for the MRCPSP.

Input: An MRCPSP instance.
Output: A feasible schedule.
Select an executing mode for each task (see Section 4.2.2);
Generate an RCPSP by removing the deselected modes from the original MRCPSP problem, i.e., generating a single mode problem;
Solve the generated RCPSP using the CPLEX CP optimizer;
return *The generated feasible schedule by the CPLEX CP optimizer (the solution);*

the set of feasible solutions. Based on their definition of inefficient modes, if there is an optimal solution for an MRCPSP then there is a solution in which no task is executed in an inefficient mode. Moreover, no task can be executed in a non-executable mode, and in solving a problem, those modes do not need to be considered.

In addition to mode reduction, the execution mode for each task can also be selected to generate a feasible solution. This is the main idea behind the mode selection approaches. The mode selection method was introduced in [20], [152] and involves three different rules for selecting modes: the shortest feasible mode (SFM), the least resource proportion (LRP), and the least criticality ratio or least critical resource (LCR). As the name suggests, the SFM selects the modes resulting in the shortest task completion time. Usually, these modes have the highest resource demands; therefore, the SFM rule can adversely affect the makespan of the project.

The LRP assumes all resources are renewable. This rule selects the modes such that the value of $max_{k_r} r^r_{i,m_i,k_r}/R^r_{k_r}$ is minimised, where $\forall k_r \in \{1, \ldots, RR\}$, and r^r_{i,m_i,k_r} is the amount of the required resource k_r to perform each task i in mode m_i, and $R^r_{k_r}$ is the total amount of resource k_r.

In the LCR, the modes, overall resource types, that lead to the least criticality ratios are selected. The criticality ratio for a given resource is the ratio of the peak of the resource utilisation to the maximum available amount of that resource. The LCR method attempts to

schedule most tasks in parallel.

The idea of employing knapsack-like approaches to solve the mode selection problem in the MRCPSP was discussed in [204], in which the NP-hardness of the feasibility problem of the MRCPSP was also established. That study introduced a local search technique that starts by obtaining a feasible solution and then performs a single-neighbourhood search within the set of feasible modes.

In our proposed algorithm, I employ a mathematical program to select the execution modes. The mathematical program is as follows.

$$GMKP: \quad Minimise \quad \sum_{i \in J} \sum_{m_i=1}^{|M_i|} c_{i,m_i} x_{i,m_i} \tag{4.1}$$

Subject to:

$$\sum_{m_i=1}^{|M_i|} x_{i,m_i} = 1, \quad \forall i \in \{0, 1, 2, \ldots, n+1\}, \tag{4.2}$$

$$\sum_{i=1}^{n} \sum_{m_i=1}^{|M_i|} \mathrm{r}^n_{i,m_i,k_n} \cdot x_{i,m_i} \leq R^n_{k_n}, \quad \forall k_n \in \{1, 2, \ldots, NR\}, \tag{4.3}$$

$$x_{i,m_i} \in \{0, 1\}, i \in J \tag{4.4}$$

where the binary variable $x_{i,m_i} = 1$ if task i is operated in mode m_i, and $x_{i,m_i} = 0$ otherwise, and parameter c_{i,m_i} is the cost of executing task i in mode m_i.

I denote this model as GMKP since it is a generalisation of the famous multi-dimensional knapsack problem. The objective of GMKP is to select the executing mode of each task to minimise the total cost of executing all tasks. The constraints in Equation (4.2) ensure each task can be performed in exactly one mode, and the constraints in Equation (4.3) guarantee the total required non-renewable resources for the selected modes do not exceed the available non-renewable resources.

Since the multi-dimensional knapsack problem [244] is NP-hard, GMKP is also NP-hard in general. However, it is practically easy to solve and takes no more than a few seconds for the largest benchmark instances in MRCPSP. It is easy to show that GMKP is feasible if and only if MRCPSP is feasible.

The solution of GMKP gives a feasible mode assignment. However, c_{i,m_i} should be selected carefully so that the solution can be used as an initial solution for the R&S algorithm to produce a good solution for the MRCPSP. For this purpose, five different objective functions are defined for GMKP to give different mode assignments. The proposed objective functions are inspired by various criteria: (1) SFM, (2) minimising the total usage of renewable and non-renewable resources, (3) maximising the total usage of renewable and non-renewable resources,

(4) balancing the selection between the shortest duration modes and the modes requiring the least total resources, and (5) balancing the selection between the shortest duration modes and the modes requiring the maximum total resources. By testing these objective functions, I aim to identify the one that yields the best algorithm performance in the computational experiments of this chapter.

1) Objective function 1:

$$c_{i,m_i} = d_{i,m_i} \tag{4.5}$$

This function is motivated by the SFM rule, which selects the shortest feasible modes for each task [20]. When enough resources are available, SFM can result in the shortest makespan. It should be noted that SFM may not be able to generate a feasible assignment of modes in the case of tight non-renewable resources, while GMPK can always give a feasible solution when MRCPSP is feasible.

2) Objective function 2:

$$c_{i,m_i} = \sum_{k_n=1}^{NR} r^n_{i,m_i,k_n} + \sum_{k_r=1}^{RR} r^r_{i,m_i,k_r} \tag{4.6}$$

The second objective function is to select the modes that need the lowest amount of the sum of renewable and non-renewable resources. Due to the lower amount of total required resources, more tasks may be performed in parallel.

3) Objective function 3:

$$c_{i,m_i} = -(\sum_{k_n=1}^{NR} r^n_{i,m_i,k_n} + \sum_{k_r=1}^{RR} r^r_{i,m_i,k_r}) \tag{4.7}$$

When resources are abundant, I can choose the modes with the greatest resource demand, which are typically associated with the shortest modes. For that reason, I consider maximising the sum of renewable and non-renewable resources in the objective function in Equation (4.7).

4) Objective function 4:

$$c_{i,m_i} = (\sum_{k_n=1}^{NR} r^n_{i,m_i,k_n} + \sum_{k_r=1}^{RR} r^r_{i,m_i,k_r}) \cdot d_{i,m_i} \tag{4.8}$$

This objective function aims to balance the selection of the shortest duration modes and the lowest required total resource modes simultaneously.

5) Objective function 5:

$$c_{i,m_i} = -\frac{\sum_{k_n=1}^{NR} r^n_{i,m_i,k_n} + \sum_{k_r=1}^{RR} r^r_{i,m_i,k_r}}{\max(d_{i,m_i}, \epsilon)} \tag{4.9}$$

where a small $\epsilon > 0$ is introduced for the case that d_{i,m_i} can be 0. This objective function aims to balance the selection of the shortest duration modes and the maximum required total resource modes.

This objective function aims to select the shortest duration and maximum required total resource modes. When resources are unlimited, this objective function is equivalent to objective function 1. However, with limited resources, this objective function changes the relative weights in the linear objective function and favours the modes with shorter duration and greater total resource demands.

In Section 4.3.4, I evaluate the performance of the proposed objective functions in generating initial solutions. I select the objective function that leads to the lowest makespan after improving the initial solution using the CPLEX CP optimizer. The results suggest using the balanced objective function 4 (Equation (4.8)) in all experiments.

4.2.3 The relax phase

I utilise a rolling time window to generate a relaxed problem. Given a time window with the length of $L > 0$ and a feasible schedule for the MRCPSP, the tasks inside the time window or overlapping with the time window are free to optimise in the relaxed problem, while those outside the time window have limited flexibility in terms of changing modes and starting time. The longer the time window length, the more tasks the time window typically holds, and as such more challenging for the CPLEX CP optimizer to solve.

At the start of the algorithm, the start of the rolling time window is set to $st = 0$. After each iteration, the start of the time window is updated by moving forward by $\Delta > 0$.

The parameter $overlap \geq 0$ is the difference between the length of the time window and the amount of its moving forward in each iteration, i.e., $overlap = L - \Delta$. This parameter controls the ability of R&S to move tasks across the time windows in consecutive iterations. In the case of $overlap = 0$, a task will always stay in the same time window as in the first iteration, which can easily lead to a poor local optimum.

The total number of iterations required to solve an instance, denoted as $Iternum$, is equal to the total number of relaxed problems. The parameter $Iternum$ can be calculated as $1 + \lceil (C_{max} - L)/\Delta \rceil$ where C_{max} is the makespan of the initial solution of the instance.

The length of the time window L and $overlap$ are two important parameters in the algorithm and should be set by the user. In Section 4.3.5, I will explore the impact of parameters L and $overlap$ on the performance of the algorithm.

4.2.3.1 Formulation of the relaxed problem

The rolling time window splits tasks into two sets, i.e., $J = J_1 \cup J_2$. The tasks that are beyond the time window's bounds belong to J_1, and the remaining tasks are included in J_2. To forbid any changes in the relative execution order of tasks in J_1, the algorithm adds a set of "start-at-end" constraints that glue two tasks in J_1 when the start time of one task equals the finish time of the other task in the given feasible solution. Let $A_1 = \{(i,j) \mid i \in J_1 \vee j \in J_1, S_i + d_{i,m_i} = S_j$ in the *given feasible solution*$\}$, which includes all these "start-at-end" constraints.

Using the same notation as in the definition of the MRCPSP in Section 2.3.1, the relaxed problem can be modelled as follows.

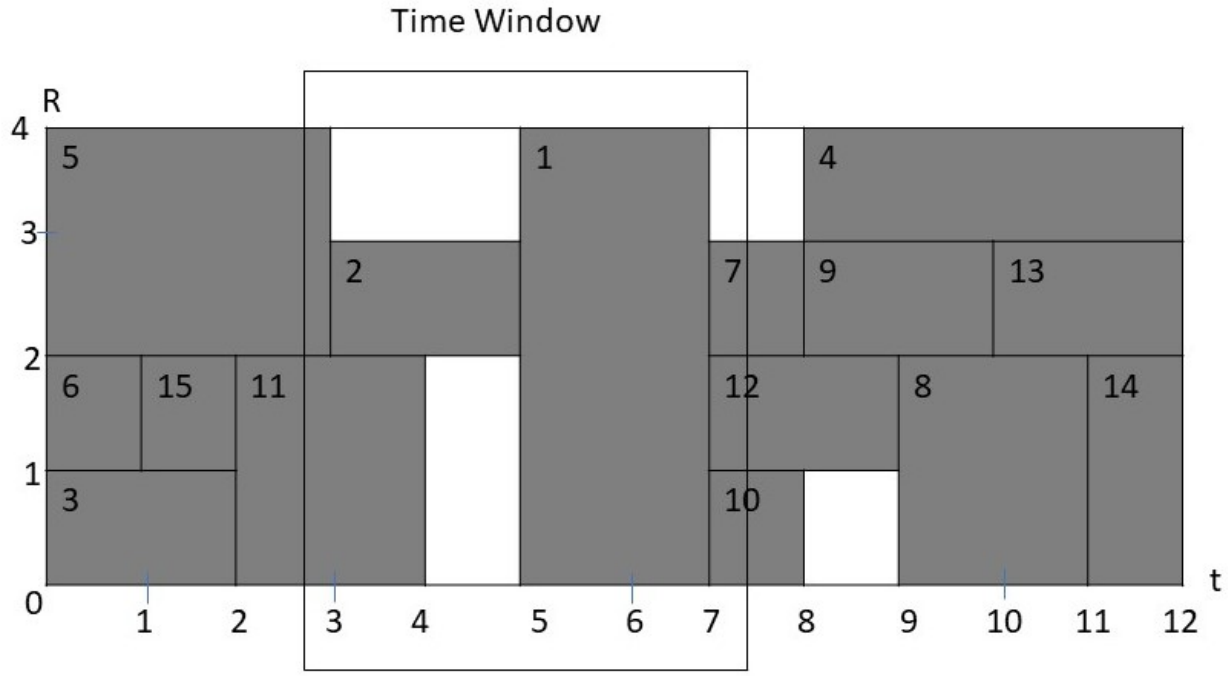

Figure 4.1: An example demonstrating how the time window is defined for MRCPSP.

$$Minimise \quad S_{n+1} \tag{4.10}$$

subject to

Equations (2.12) to (2.16),

$$S_j = S_i + d_{i,m_i}, \quad \forall (i,j) \in A_1. \tag{4.11}$$

4.2.3.2 An illustrative example of the relaxed problem

The following example illustrates how a relaxed problem is created.

In Figure 4.1, J_1 includes the tasks that finish before the start of the time window or start after the finish of the time window, i.e. $J_1 = \{3, 6, 15\} \cup \{4, 9, 8, 13, 14\}$.

The remaining tasks belong to $J_2 = \{5, 11, 2, 1, 7, 12, 10\}$. It should be noted that the start of the time window may not be an integer.

I note that "start-at-end" constraints are added for task pairs $(6, 15)$, $(9, 13)$ and $(8, 14)$, which creates three "super-tasks" in J_1. The R&S also adds "start-at-end" constraints between the tasks in J_1 and J_2. More precisely,

constraints are added for the pair of tasks $(3, 11)$ and $(15, 11)$ and also for the pairs $(7, 4)$, $(7, 9)$ and $(12, 8)$. Now tasks 3, 6, 15 and 11 are glued together as a super task. Tasks 4, 7, 9 and 13 are also form a super task. Finally, tasks 12, 8, 14 are glued together as a super task. This significantly reduces the problem size. Accordingly, the arc set includes $A1 = \{(6, 15), (9, 13), (8, 14), (3, 11), (15, 11), (7, 4), (7, 9), (12, 8)\}$.

Table 4.2: Dimensions of the various tested instances.

Instance	#Problems	#Tasks	#Modes	#RR	#NR
J10	537	10	3	2	2
J12	548	12	3	2	2
J14	552	14	3	2	2
J16	551	16	3	2	2
J18	553	18	3	2	2
J20	555	20	3	2	2
J30	552	30	3	2	2
MMLIB50	540	50	3	2	2
MMLIB100	540	100	3	2	2
MMLIB+	3240	(50,100)	(3,6,9)	(2,4)	(2,4)

4.2.4 The solve phase

When a relaxed problem is obtained, the algorithm starts the solve phase. The relaxed problem is easier to solve because the added "start-at-end" constraints significantly speed up the CP-based optimizer inference. Conceptually, fewer tasks are allowed to be reordered. When the solve phase terminates, a feasible solution is obtained for the relaxed problem, which is always feasible for the original problem, as discussed earlier. That also implies that the makespan for the relaxed problem is valid for the original problem.

In the proposed R&S algorithm, I use a time-limit for the CPLEX CP optimizer in order to generate an initial feasible solution ($Ct_{\text{init-cplex}}$) and to solve the relaxed problem (Ct_{cplex}). Therefore, the algorithm moves to the next iteration as soon as the computation time reaches the limit set by the user or the proven optimal solution for the relaxed problem is obtained. Even with a small time limit the solve phase is guaranteed to produce a feasible solution no worse than the solution delivered in the previous iteration because the solution from the previous iteration is used as a warm-start.

4.3 Computational experiments

In this section, the performance of the proposed R&S is evaluated by solving standard benchmarks of MRCPSP, taken from PSPLIB and MMLIB [24]. Overall, I solve 8,168 instances, details of which are shown in Table 4.2. In this table, for each set of instances, the number of non-dummy tasks (#Tasks), mode per each task (#Modes), renewable resources (#RR), and non-renewable resources (#NR) are listed. All the experiments were conducted on a machine with CPU Intel Xeon Gold 6238R 2.2GHz. I developed the proposed R&S algorithm in the programming language Python version 3.6.5, and I used CPLEX CP optimizer version 12.10.0.0 [15]. Unless otherwise stated, I used the default value for all parameters of the CPLEX CP Optimizer.

4.3.1 Parameters setting

Unless otherwise stated, I use objective function 4 to generate the initial solution based on the results in Section 4.3.4. I set $L = 0.5C_{max}$, and $overlap = 0.75L$ so that the $Iternum = 5$. The performance of R&S for different values of L and $overlap$ is discussed in Section 4.3.5.

Table 4.3: Time limits in seconds for R&S.

Instance	$Ct_{\text{init-cplex}}$	Ct_{cplex}	$Ct_{\text{stop-R\&S}}$
J10	5	5	30
J12	5	5	30
J14	5	5	30
J16	5	5	30
J18	5	5	30
J20	5	5	30
J30	10	10	60
MMLIB50	10	10	60
MMLIB100	30	30	180
MMLIB+	20	20	120

To conduct a fair comparison of various solution methods, the results of all algorithms should be reported by using the same stopping criterion. In [13], it is argued that using the number of generated schedules as a stopping criterion has two benefits: (1) it works on any platform, and (2) future studies can build upon the benchmark results using the same criterion. However, it also has limitations such as (1) computation time for one schedule may vary among metaheuristics, and (2) this method is not suitable for all heuristic strategies [24]. According to [13], one schedule is equivalent to assigning a start time to each activity as performed by a schedule generation scheme (SGS). Counting the number of schedules based on this definition means that a new schedule must be counted each time there is a change in modes during a local search process. However, CPLEX CP optimizer does not use SGS for generating schedules. Consequently, the number of generated schedules cannot be used to compare R&S with other methods.

I use the total computation time denoted as $Ct_{stop-R\&S}$, as a stopping criterion for comparing R&S with other algorithms. In the available results published after 2014 only five studies of [25], [151], [166], [245], [246] report computation times. The metaheuristic presented in [245] reports computation times that are far longer than other papers in the literature. [166], [246] did not report computation time for all sets of instances, whereas [25], [151] detail computation time for the results corresponding to the 1,000 and 5,000 maximum generated schedules and compare comprehensively with other methods. Therefore, I set the parameters $Ct_{init-cplex}$ and Ct_{cplex} to have the total computation time comparable with studies of [25], [151]. Because these two papers use a faster computing platform than ours, I believe this is fair to compare the results of the R&S and the results reported in these two papers under the maximum number of generated schedules.

The time limit in R&S can be calculated as $Ct_{stop-R\&S} = Ct_{init-cplex} + Iternum \times Ct_{cplex}$. I set $Ct_{init-cplex}$ and Ct_{cplex} according to Table 4.3 so that the average running time of R&S is about the same as the reported times in the two studies of [25], [151]. More precisely, In [25], [151] the maximum number of generated schedules is set to 1,000 for MMLIB50 and MMLIB100 instances and to 5,000 for the remaining instances, i.e., for MMLIB+, J10, J12, J14, J16, J18, J20, and J30. The average computation times for R&S and for the methods discussed in [25], [151] are provided in Figure 4.2. It can be seen that the computation times of R&S lie between the computation times reported in [151] and [25]. It should be noted that studies of [25], [151] used an Intel Core i7 CPU clocked at 3.4 GHz, while I use an Intel Xeon Gold 6238R clocked at 2.2GHz.

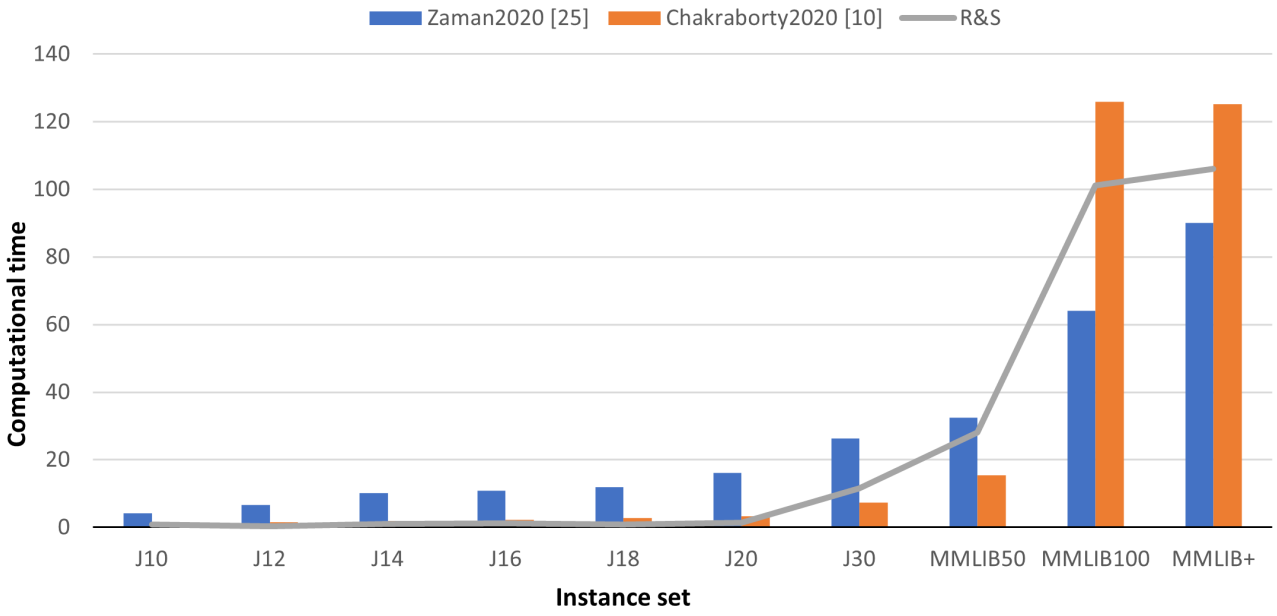

Figure 4.2: The average computation time in seconds for various sets of instances for the method of R&S and the methods presented in [25], [151].

4.3.2 Test results for small instances

In this section, the performance of the proposed R&S is evaluated on benchmark sets from PSPLIB. The test sets are J10, J12, J14, J16, J18, J20. The optimal value for each instance is known, and I compare the results obtained with those of state-of-the-art algorithms.

The quality of solutions is evaluated by comparing $\%Dev_{opt}$, the average percentage of deviations between the obtained values by an algorithm and the known optimal values ($C_{opt,p}$). For algorithm A, $\%Dev_{opt}$ is defined as:

$$\%Dev_{opt} = \frac{\sum_{p=1}^{P} \frac{C_{A,p} - C_{opt,p}}{C_{opt,p}} \times 100\%}{P}, \tag{4.12}$$

where P is the number of instances in a benchmark set, $C_{A,p}$ is the makespan obtained by the algorithm A for the instance p, and $C_{opt,p}$ is the optimal value for the instance p.

The obtained results for different benchmark sets for different algorithms are provided in Table 4.4. Except for the local branching algorithm (see [245]), the rest of the results are taken from [151]. The second column shows the names of the algorithms. I sorted the algorithms with respect to the values of parameter $\%Dev_{opt}$ for the J20. The boldface in each benchmark clarifies the best result achieved by the corresponding algorithm.

From Table 4.4, our proposed R&S obtained the best results for all J10, J12, J14, J16, J18 and J20 benchmark sets.

4.3.3 Test results for hard instances

In this section, instances from both PSPLIB and MMLIB [24] are solved to evaluate the performance of our R&S on complex MRCPSP. I solved the instances with two renewable resources, two non-renewable resources, and three different execution modes for 30 tasks in J30 from

Table 4.4: $\%Dev_{opt}$ for J10 to J20 benchmarks.

Id	Algorithm	Name	J10	J12	J14	J16	J18	J20
1	Cooperative Coevolutionary	WLZO	0.28	0.79	1.18	2.75	NA	NA
2	Simulated Annealing	JSA	1.16	1.73	2.60	4.07	5.52	6.74
3	Artificial Bee Colony	CLPE	1.06	0.41	3.43	3.36	2.12	5.78
4	Ant Colony	CHA	0.32	NA	NA	NA	NA	2.05
5	GA	AGA	0.24	0.73	1	1.12	1.43	1.91
6	PSO and DE	ZLZH	0.00	NA	NA	NA	NA	1.82
7	Reinforcement Learning	TCGLS	0.33	0.52	0.93	1.08	1.32	1.69
8	Modified Shuffled Frog-leaping	RSS	0.18	0.65	0.89	0.95	1.21	1.64
9	Multi-Agent based Learning	SLC	0.05	0.21	0.46	0.82	1.21	1.62
10	Ant Colony	EFEA	0.14	0.24	0.77	0.91	1.30	1.62
11	Estimation of distribution	JRR	0.28	0.41	0.54	0.75	0.92	1.55
12	GA	LCSFLA	0.10	0.21	0.46	0.58	0.94	1.40
13	Cooperative discrete PSO	LCEDA	0.12	0.14	0.43	0.59	0.90	1.28
14	Shuffled Frog-leaping	LZA	0.09	0.13	0.4	0.57	1.02	1.1
15	Discrete PSO	SEEDA	0.09	0.12	0.36	0.42	0.85	1.09
16	Controlling Search	LHGA	0.06	0.17	0.32	0.44	0.63	0.87
17	Artificial Immune System	MAN	0.05	0.09	NA	0.22	0.18	0.80
18	GA	CWC	0.01	NA	NA	NA	NA	0.71
19	Local Search	VPVAIS	0.02	0.07	0.2	0.39	0.52	0.70
20	GA	VPVGA	0.01	0.09	0.22	0.32	0.42	0.57
21	Local Branching	Local Branching[245]	0.00	0.00	0.00	0.09	0.1	0.13
22	Modified Variable Neighborhood	MVNSH	0.15	0.62	0.81	0.15	0.89	0.27
23	PR	Albert18[160]	0.01	0.00	0.05	0.03	0.09	0.06
24	Evolutionary Hybrid Algorithm	H-EA[151]	0.00	0.00	0.08	0.04	0.09	0.06
25	**Relax- and-Solve Algorithm**	**The proposed R&S**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**

Table 4.5: $\%Dev_{cpl}$ for J30, MMLIB50, MMLIB100 and MMLIB+.

Id	Algorithm	Name	J30	MMLIB50	MMLB100	MMLIB+
1	GA	COEL11[247]	14.44	NA	NA	NA
2	GA	OZDA99	27.38	NA	NA	NA
3	Ant Colony Optimisation	CHIA08	15.18	NA	NA	NA
4	GA	MORE97	24.99	NA	NA	NA
5	GA	ALCA03	21.83	56.06	61.08	177.55
6	Multi-agent learning	WAUT11	13.91	NA	NA	145.78
7	PSO	JARB08	18.14	49.98	NA	NA
8	PR	Albert18 [160]	12.85	NA	NA	86.84
9	Modified Variable Neighborhood	MVNSH	13.63	42.5	73.2	**48.00**
10	GA	TSEN09	17.06	65.17	72.95	183.02
11	PSO	ZHAN06	18.63	49.25	57.42	NA
12	GA	ELLO10	16.16	43.84	56.21	130.06
13	Simulated Annealing	JOZE01	18.64	49.06	53.97	121.09
14	Estimation of Distribution	WANG12	15.55	43.17	52.94	144.84
15	DE	DAMA09	15.43	46.19	52.31	126.69
16	Mirror-based GA	MGA	19.55	45	52	137
17	Scatter Search	RANJ09	16.21	38.49	45.16	131.45
18	GA	HART01	15.96	35.4	39.96	132.01
19	GA	VANP10	13.75	34.07	37.58	NA
20	Local Branching	Local Branching [245]	13.17	25.60	32.41	NA
21	Scatter Search	VANP11	13.66	28.17	29.77	101.45
22	Evolutionary Hybrid Algorithm	H-EA [151]	12.55	26.53	27.78	74.48
23	**Relax- and-Solve Algorithm**	**The proposed R&S**	**12.35**	**23.08**	**22.94**	86.73

PSPLIB, for 50 and 100 tasks in MMLIB50 and MMLIB100 from MMLIB, respectively. I also solved the MMLIB+, which is the most complex set of instances. Instances in MMLIB+ have up to 100 tasks, up to 9 execution modes, and 2 to 4 renewable and non-renewable resources. The optimal solutions for all the instances in this section are still unknown.

To show the performance of an algorithm, $\%Dev_{cpl}$, the average percentage of deviations between the obtained values by the algorithm and the critical path lower bound ($C_{cpl,p}$), is used. For an algorithm A, $\%Dev_{cpl}$ is calculated as:

$$\%Dev_{cpl} = \frac{\sum_{p=1}^{P} \frac{C_{A,p} - C_{cpl,p}}{C_{cpl,p}} \times 100\%}{P}, \tag{4.13}$$

where $C_{cpl,p}$ is the critical path lower bound for each instance p, $C_{A,p}$ is the makespan obtained by algorithm A for the instance p, and P is the number of instances in a benchmark set.

Table 4.5 presents the results of different algorithms. Except for the algorithm local branching (see [245]), all results are directly taken from [151]. The names of the algorithms are shown in the second column, and the best results for J30, MMLIB50, MMLIB100 and MMLIB+ are highlighted in boldface. Our proposed R&S delivers a feasible solution for all the instances. It is also shown in Table 4.5 that R&S is the best algorithm dealing with J30, MMLIB50 and MMLIB100. For solving MMLIB+, the algorithm MVNSH [25] and the algorithm H-EA [151] are better than R&S.

All algorithms are compared through statistical tests discussed in Section 4.3.6, which clearly shows that R&S is the best algorithm overall, comparing all the algorithms that are capable of obtaining at least a feasible solution for every instance.

I further compare our results for MMLIB50, MMLIB100 and MMLIB+ and other algorithms employing a maximum number of 50,000 generated schedules [24], [25]. The results reported in [245] for a time limit of 3,600 seconds are also included. As shown in Table 4.6, with the exception of MMLIB+, the solutions achieved by R&S using $Ct_{stop-R\&S}$ as the stopping criterion (see Table 4.3) outperforms all other algorithms.

I also utilised the "solutionsUpdate" software of [248] to compare with the best-known solutions reported in the literature. With $Ct_{stop-R\&S} = 4,800$ seconds, notably, R&S updated six upper bounds of MMLIB50, 32 upper bounds of MMLIB100, and 614 upper bounds of MMLIB+. Additionally, within the MMLIB+ dataset, our proposed R&S discovered optimal solutions for four open instances by delivering new improved upper bounds.

4.3.4 Mode selection by solving GMKP

I evaluate the five different objective functions for mode selection (see Section 4.2.2) when generating the initial feasible solution and the impact on the overall performance of CPLEX CP optimizer.

For that reason, I solved the 540 instances of MRCPSP from MMLIB50 by CPLEX CP optimizer. CPLEX CP optimizer has difficulty in finding feasible solution for many of the instances. Figure 4.3 shows the number of instances of MMLIB50 that a feasible solution is obtained within different time limits. CPLEX CP optimizer delivers a feasible solution for 521 instances in 15 seconds. However, after 1,200 seconds, there are still a few instances that the CPLEX CP optimizer cannot obtain a feasible solution. However, assuming that the modes selected by

Table 4.6: $\%Dev_{cpl}$ for MMLIB50, MMLIB100 and MMLIB+ for 50,000 generated schedule.

Id	Algorithm	Name	MMLIB50	MMLB100	MMLIB+
1	PSO	JARB08	32.02	40.23	137.99
2	PR	Albert18 [160]	23.52	24.01	83.66
3	Modified Variable Neighborhood	MVNSH[25]	32.00	60.00	**36.00**
4	GA	TSEN09	29.44	37.04	142.14
5	PSO	ZHAN06	30.23	35.35	NA
6	GA	ELLO10	26.92	30.02	106.35
7	Simulated Annealing	JOZE01	27.81	30.27	103.19
8	Estimation of Distribution	WANG12	28.76	30.45	110.43
9	DE	DAMA09	26.27	28.92	103.75
10	Scatter Search	RANJ09	28.55	34.17	131.07
11	GA	HART01	26.81	29.04	111.45
12	GA	VANP10	24.93	25.63	NA
13	Local Branching	Local Branching [245]	25.60	32.41	NA
14	Scatter Search	VANP11	23.79	24.02	92.76
15	**Relax- and-Solve Algorithm**	**The proposed R&S**	**23.08**	**22.94**	86.73

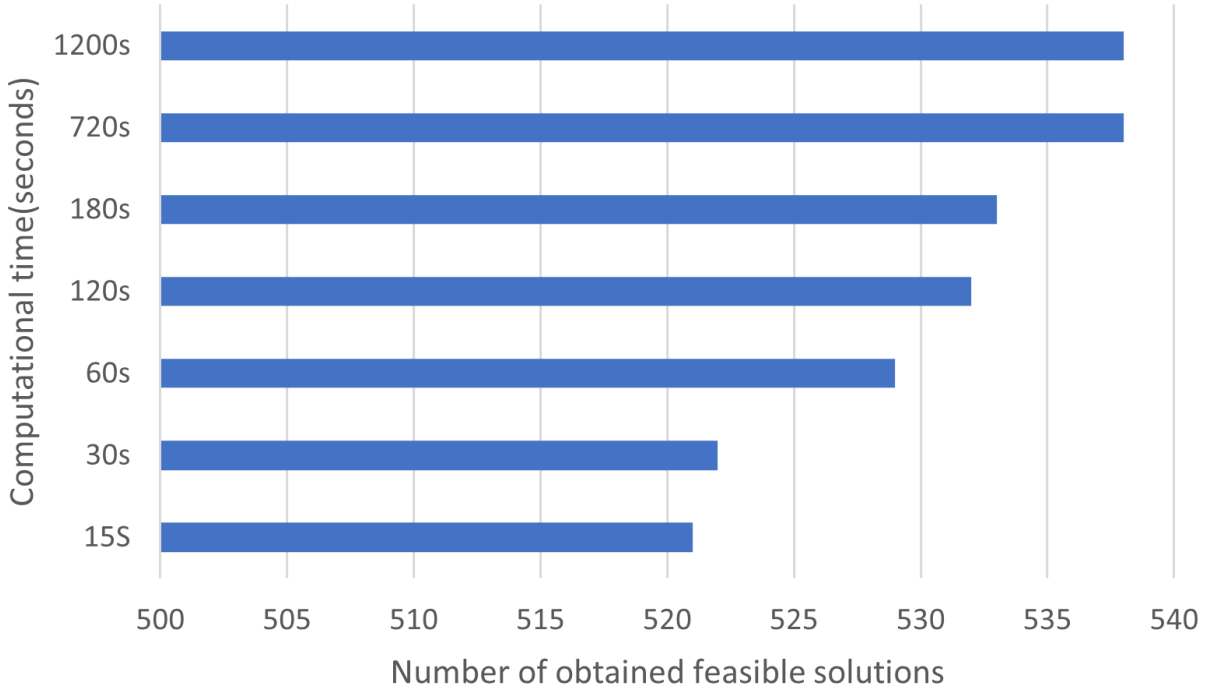

Figure 4.3: CPLEX CP optimizer in solving 540 instances from MMLIB50

solving GMKP are fixed, CPLEX CP optimizer can find feasible solutions for all the instances within 4 seconds, irrespective of which objective function is used in the GMKP.

To evaluate the impacts of the five objective functions on the overall performance of CPLEX CP optimizer, I solve the single-mode RCPSP created from the GMKP solution to generate a feasible solution to the MRCPSP. This solution is used to warm-start CPLEX CP optimizer to solve the original MRCPSP. For that, I set $Ct_{init-cplex} = 5$ seconds and $Ct_{cplex} = 30$ seconds.

Table 4.7 summarises the results of criterion $\%Dev_{cpl}$ on solving the set MMLIB50 when using different objective functions for selecting modes. The column "Initial solution ($\%Dev_{cpl}$)" is the average deviation from the critical path lower bound in percentage for the initial solution which is obtained by solving a single mode RCPSP, and the column "CP with warm-start ($\%Dev_{cpl}$)" is the average deviation from the critical path lower bound in percentage for CPLEX CP optimizer which is obtained by solving the original MRCPSP with the warm-start. Table 4.8 shows the average computation time in seconds for obtaining an initial solution and for the CPLEX CP optimizer with a warm start.

The objective function 1 favours the selection of the shortest modes and provides the best results in generating initial solutions. The objective function 2 minimises total resource consumption, while the objective function 3 maximises total resource consumption. Objective function 3 gives better initial and overall solution quality than objective function 2 but requires a little bit more computation time. Objective function 5 favours the selection of the modes with the maximum required resources and minimum execution time since, normally, the modes with the shorter execution time require more resources. The performance of objective function 5 is dominated by objective function 4 in both solution quality and time. Objective function 4 favours the selection of the modes with the shortest execution time and lowest required renewable and non-renewable resources. It dominates all the other objectives in terms of solution time and

Table 4.7: Impact of mode selection on $\%Dev_{cpl}$ in MMLIB50

	Initial solution($\%Dev_{cpl}$)	CP with warm start($\%Dev_{cpl}$)
Objective 1	**40.84**	23.38
Objective 2	124.37	23.85
Objective 3	67.72	23.52
Objective 4	49.84	**23.24**
Objective 5	56.17	23.43

Table 4.8: Impact of mode selection on average computational time (seconds) in MMLIB50

	Initial solution(seconds)	CP with warm start(seconds)
Objective 1	**2.09**	**18.75**
Objective 2	2.86	19.68
Objective 3	3.67	20.28
Objective 4	2.69	19.13
Objective 5	3.07	19.48

quality except for objective function 1. Objective function 4 gives the best overall solution quality and requires a slightly longer computation time than objective function 1.

4.3.5 Impacts of parameters L and *overlap* on the performance of Relax-and-solve

In this section, I first explore the influence of parameter L on the algorithm's performance. I then investigate the impact of the parameter *overlap* given the value of parameter L is fixed to the best value obtained earlier.

To investigate the impact of parameter L on the performance of R&S, I solve all the MMLIB50 instances by setting $Ct_{stop-R\&S} = 60$ seconds and the parameter $overlap = 0.5L$. For different values of $L \in \{0.2C_{\max}, 0.3C_{\max}, 0.4C_{\max}, 0.5C_{\max}, 0.7C_{\max}\}$ the corresponding results for $\%Dev_{\text{cpl}}$ and the average computation time spent by R&S ($Ct_{R\&S}$) are presented in Figures 4.4 and 4.5.

Figure 4.4 illustrates that as L increases from $0.2C_{max}$ to $0.5C_{max}$, the $\%Dev_{cpl}$ decreases from 25.96% to 23.56%. With further increments in L the quality of the solutions exhibits gradual improvement at the cost of increased computation time. The relationship between computation time and the parameter L is demonstrated in Figure 4.5, which highlights an increase as the value of L is increased. Based on this experiment, I set the parameter $L = 0.5C_{max}$ for our computations.

To investigate the impact of the parameter *overlap* on the performance of R&S, I set $L = 0.5C_{max}$ and $Ct_{stop-R\&S} = 60$ seconds and solve all the MMLIB50 instances for different values of parameter *overlap*, where $overlap \in \{0.5L, 0.66L, 0.75L, 0.8L\}$. The $\%Dev_{cpl}$ of the results is demonstrated in Figure 4.6, and $Ct_{R\&S}$ is shown in Figure 4.7.

As depicted in Figures 4.6 and 4.7, increasing *overlap* from $0.5L$ to $0.75L$ leads to improved results. However, excessive increase in *overlap* negatively impacts the quality of solutions.

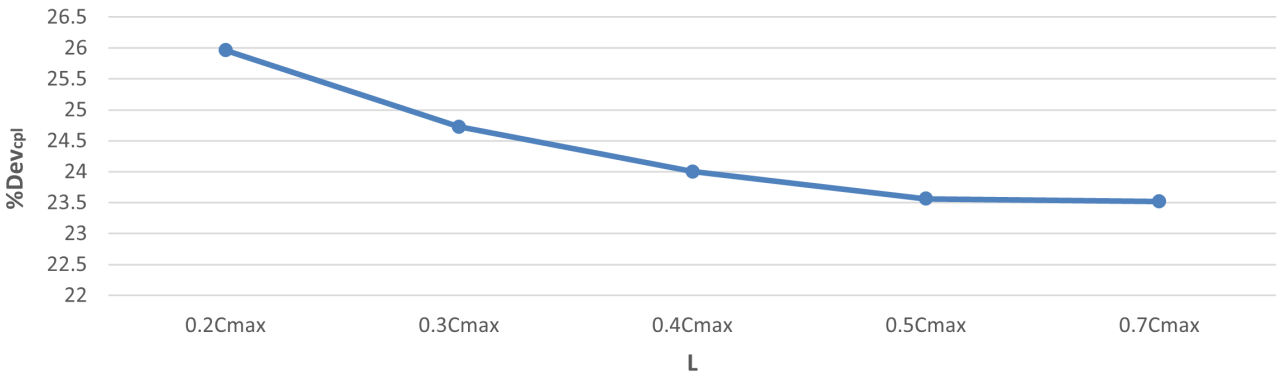

Figure 4.4: The impact of parameter L on $\%Dev_{cpl}$.

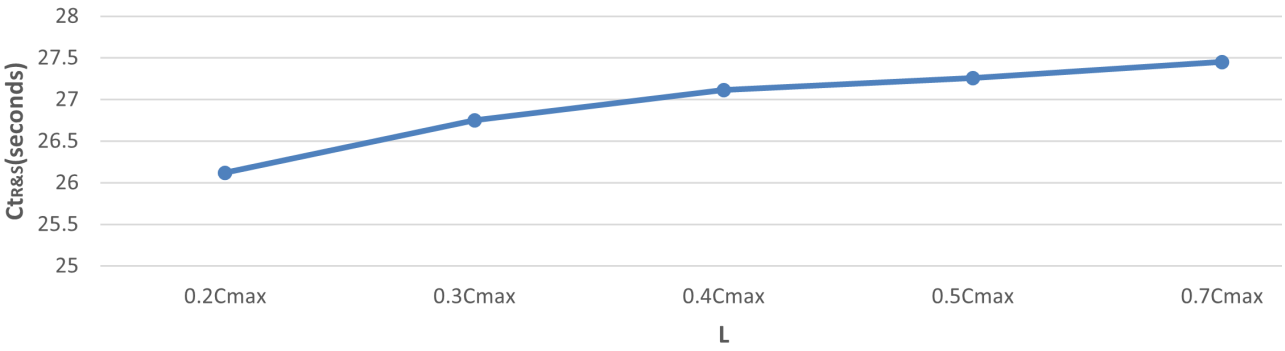

Figure 4.5: The impact of parameter L on $Ct_{\mathrm{R\&S}}$.

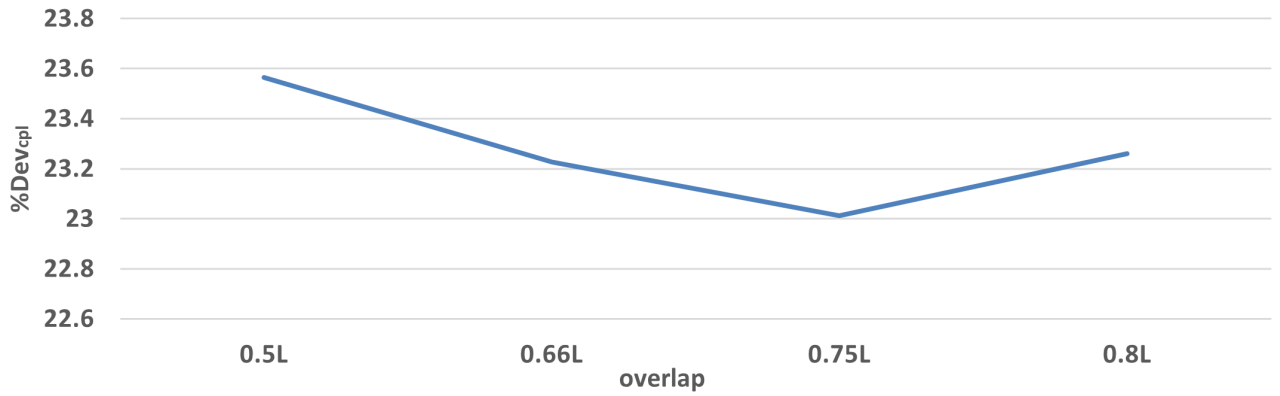

Figure 4.6: The impact of parameter *overlap* on $\%Dev_{cpl}$.

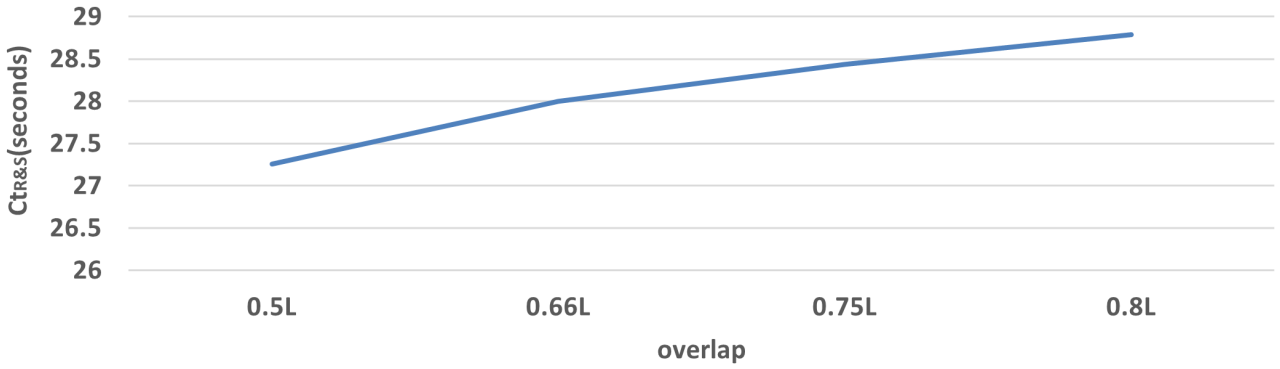

Figure 4.7: The impact of parameter *overlap* on $Ct_{\mathrm{R\&S}}$.

4.3.6 Statistical tests

Finally, I employ two statistical tests of Kruskal-Wallis test and Friedman rank in order to evaluate the performance of the algorithms as in [151].

The Kruskal-Wallis test is a non-parametric approach to the one-way analysis of variance test (ANOVA). It is used to determine if there is a significant difference between the median of more than two independent groups. I perform the Kruskal-Wallis test to determine the rank of different algorithms using the benchmark instances in Table 4.2. I assume a default null hypothesis H_0 representing no significant difference between different methods with a 95% confidence level. The p_values computed for the results in Tables 4.4 and 4.5 are 1.35E-09 and 0.95 respectively, which means the Kruskal-Wallis test fails to reject H_0 for Table 4.5, and therefore I cannot claim there is significant difference among the compared algorithms.

I attempted another non-parametric statistical test, the Friedman rank test, as suggested in [151] to compare all tested algorithms that can find a feasible solution for all the instances. The Friedman test aims to detect significant differences between more than one algorithm through a multiple comparison test. Table 4.9 shows the the average rank of the algorithms and p-values obtained from the Friedman test. In this method, the null hypothesis H_0 of no significant differences among the compared algorithms is rejected by the p-value being less than the significance level of 0.05. Obtaining p-value $= 4.11E-21$ over comparing algorithms on J10-J20 and p-value $= 1.18E-05$ for comparing algorithms on large scale instances, the null hypothesis H_0 is rejected, meaning that the test suggests the significant differences between the results obtained by different algorithms. In Table 4.9, boldface values represent the least mean rank among all considered algorithms. It can be seen that our R&S algorithm is ranked first in both the small and large instances.

4.4 Summary

This chapter presented a novel R&S matheuristic for solving MRCPSP. In this algorithm, an integer program-based mode selection method is first solved to identify the infeasibility of an instance due to the limited availability of non-renewable resources or to select a mode for each task using one of the five objective functions. Based on the selected modes, a reduced RCPSP instance is solved by a CPLEX CP optimizer to obtain a feasible solution for MRCPSP. Finally, this feasible solution is improved in an iterative relax and solve processes.

The results from a comprehensive computational study suggest that CPLEX CP optimizer can easily obtain a feasible solution for all the instances when warm-started by the model selection method, and the proposed R&S algorithm can generate superior results to the algorithms available in the literature.

Table 4.9: Results of the statistical test while comparing R&S and other methods.

Tests on J10-J20		Tests on Large-scale instances	
Algorithms	Average Rank	Algorithms	Average Rank
Simulated Annealing(JOZE01)	19.84	Estimation of Distribution(JRR)	8
Artificial Bee Colony (CLPE)	18.42	Scatter Search(VANP11)	6.75
GA(AGA)	17.84	Simulated Annealing(JOZE01)	8.75
Reinforcement Learning(TCGLS)	17	GA(ELLO10)	8.25
Modified Shuffled Frog-leaping(RSS)	15.92	GA(HART01)	6
Ant Colony(EFEA)	14.25	DE(DAMA09)	7
Estimation of distribution(JRR)	13.58	Scatter Search(VANP10)	3.5
GA(LCSFLA)	11.5	Mirror-based GA(MGA)	9
Shuffled Frog-leaping(LZA)	9.92	Modified Variable Neighborhood (MVNSH)	5.25
Discrete PSO(SEEDA)	8.25	Evolutionary Hybrid Algorithm(H-EA)	2
Cooperative discrete PSO(LCEDA)	10.67	**The proposed R&S**	**1.5**
Multi-Agent based Learning(SLC)	12.17	**Friedman test p-value**	**1.18E-05**
Controlling Search(LHGA)	8.17	Kruskal–Wallis test p-value	0.95
Local Search(VPVAIS)	6		
GA(CWC)	5.58		
Local Branching[245]	3		
PR(Albert18)	2.83		
Modified Variable Neighborhood (MVNSH)	10.83		
Evolutionary Hybrid Algorithm(H-EA)[151]	2.75		
The proposed R&S	**1.5**		
Friedman test p-value	**4.11E-21**		
Kruskal–Wallis test p-value	1.35E-09		

Chapter 5

An Approximate Dynamic Programming Approach for Resource-Constrained Project Scheduling with Uncertain Task Duration

5.1 Introduction

In past decades, significant attention has been devoted to scheduling projects under resource constraints. In 1969, Pritsker et al. introduced the Resource-Constrained Project Scheduling Problem (RCPSP) [249], which involves determining the optimal scheduling of tasks subject to precedence constraints and resource limitations over time.

The RCPSP with uncertainties has been widely studied. However, a research gap remains, as most existing approaches focus on open-loop task scheduling, with only a few studies exploring dynamic and adaptive closed-loop policies, which are often considered computationally time-consuming. In this chapter, I propose an average project-based approximate dynamic programming (A-ADP) approach that is computationally tractable for solving the RCPSP with stochastic task duration (SRCPSP).

The motivation in this study stems from the findings reported in [34], where it is stated that a simple policy based on the solution of the so-called average project works well in most cases. The average project is a deterministic RCPSP model in which each task has the mean value as its duration. Since the deterministic RCPSP is solved only once, this approach is highly efficient compared to the closed-loop approach. Numerous exact and heuristic methods exist for solving the average project. Instead of using the approach proposed in Chapter 3, this chapter proposes a novel heuristic that can provide better results and is based on the R&S algorithm for solving RCPSP and strengthened by the forward-backward improvement. I use this approach to design a policy for solving an SRCPSP. However, when uncertainties are significant, the performance of this policy deteriorates. This insight is the motivation to develop a novel, approximated dynamic programming approach. This method incorporates priority rules and integrates information from average projects within a closed-loop policy framework, aiming to

solve SRCPSP efficiently.

The remainder of this chapter is organised as follows. In Section 5.2, the Markov decision process model (MDP) [250] is defined and used for the SRCPSP. Section 5.3 explains our proposed A-ADP approach for solving SRCPSP. This approach uses different priority rules and a roll-out policy to make decisions. The solution from a deterministic average project is utilised to reduce the computational burden associated with the roll-out policy. An illustrative algorithm example is provided in Section 5.4. For solving large scale instances, a matheuristic approach is presented in Section 5.5. The computation experiments and results are provided in Section 5.7. Finally, in Section 5.8, I conclude this chapter.

5.2 Markov decision process model formulation of stochastic resource-constrained project scheduling problem

This section introduces the SRCPSP's assumptions and formulations. Then, its MDP formulation is presented, which will serve as the basis for developing A-ADP algorithms in the upcoming sections.

5.2.1 Assumptions

Consider a project network represented by an activity-on-node (AON) diagram denoted as $G(V, E)$, where the set of tasks in the project is $V = \{0, 1, \ldots, n, n+1\}$. Here, tasks 0 and $n+1$ represent the beginning and end of the project, respectively. The AON network is assumed to be acyclic; once a task is started, it cannot be interrupted. The set of edges E represents the precedence relationships among the tasks, indicating that a task j cannot begin before task i has been completed if $(i, j) \in E$. The project requires utilising a set of resources $K = \{1, 2, \ldots, m\}$ for its execution. Each resource $k \in K$ has a limited capacity of R_k units available during each period. The execution of a task j requires r_{jk} units of resource k. The duration of a task j, denoted by a random variable D_j, follows a probability distribution (PD) known to the decision-maker. The duration of a task can only be observed once the task has been completed. The objective is to determine a feasible policy regarding the precedence and resource constraints that minimises the project's expected completion time (makespan).

5.2.2 Stochastic resource-constrained project scheduling problem formulation

I considered the MDP model for the SRCPSP as suggested in [31]. This model includes stages, states, decisions, transition processes and cost functions.

A decision stage is defined as a point in time when the project is started, or when a task is finished. I use t_i to represent the time associated with the ith decision stage, where i ranges from 1 to the number of decision stages, denoted by L. The state $\mathcal{S}_i$ contains all the necessary information to decide at each stage i. It includes the completed tasks, active tasks, the duration of the tasks that have been executed, the observed duration of active tasks up to the current stage, and the start times of all completed and active tasks. The decision x_i is a set of tasks that can start at stage i and satisfy both precedence and resource constraints.

The other component of the SRCPSP model is the transition function from the current stage i to the next stage $i+1$, which is denoted as $S^M(\cdot)$.

$$\mathcal{S}_{i+1} = S^M\left(\mathcal{S}_i, \boldsymbol{x}_i, D_{j:j\in A_i}\right), \tag{5.1}$$

Where A_i is the set of active tasks at stage i. Given the decision taken at stage i, $\boldsymbol{x}_i$, and the set of observed duration of active tasks $D_{j:j\in A_i}$, the transition function $S^M(.)$ maps the current state $\mathcal{S}_i$ to the next state $\mathcal{S}_{i+1}$. Equation (5.1) shows that the next state depends only on the current state, the decision made, and the realised duration of active tasks, not on the decision history. This is known as the Markov property.

At stage $i \in \{1, ..., L-1\}$, for the time point t_i, the cost of transition from S_i to S_{i+1} is denoted by $g(S_i, x_i, S_{i+1})$, so the cost function at stage i given state S_i, can be written as:

$$J_i\left(\mathcal{S}_i\right) = \mathbb{E}\left\{\sum_{j=i}^{L-1} g\left(\mathcal{S}_j, \boldsymbol{x}_j, \mathcal{S}_{j+1}\right)\right\} \tag{5.2}$$

The goal is to minimise the expected project makespan by selecting the optimal policy π among a set of policy Π.

$$V_i\left(\mathcal{S}_i\right) = \min_{\pi\in\Pi} J_i(S_i) \tag{5.3}$$

Let the cost-to-go function, or the objective function from stage i and state S_i when using the optimal policy π is:

$$V_i\left(\mathcal{S}_i\right) = \mathbb{E}\left\{\sum_{j=i}^{L-1} g\left(\mathcal{S}_j, \boldsymbol{x}_j^{\pi}, \mathcal{S}_{j+1}\right)\right\} \tag{5.4}$$

The well-known recursion Bellman function[251] computes the optimal policy which is the optimal set of tasks started at each stage i as below:

$$\boldsymbol{x}_i^{\pi} = \arg\min_{\boldsymbol{x}\subset\mathcal{X}(\mathcal{S}_i)} \mathbb{E}\{g\left(\mathcal{S}_i, x_i, \mathcal{S}_{i+1}\right) + V_{i+1}\left(S_{i+1}\right))\} \tag{5.5}$$

where $\mathcal{X}\left(\mathcal{S}_i\right)$ is the set of eligible tasks.

5.3 Approximate dynamic programming approach for stochastic resource-constrained project scheduling problem

A successful approximation paradigm for solving MDPs is the roll-out policy for combinatorial optimisation [252]. At each stage, a roll-out policy replaces the exact cost-to-go function $V_i(S_i)$ with a heuristic H to approximate it, denoted as $V_i^H(S_i)$, which is then used to decide in the current state. In this policy at each state S_i, a decision $x_i^{P_H}$ is obtained using the heuristic H as

$$\boldsymbol{x}_i^{P_H} = \arg\min_{\boldsymbol{x}\subset\mathcal{X}(\mathcal{S}_i)} \mathbb{E}\left\{g\left(\mathcal{S}_i, x_j, \mathcal{S}_{i+1}\right) + V_{i+1}^H(S_{i+1})\right\} \tag{5.6}$$

What motivated us to use the roll-out policy is that it has been shown to be effectively applied in a wide range of NP-hard combinatorial optimisation problems, including SRCPSPs [31], [203], to improve a closed-loop solution sequentially. Such a rollout policy is a strategy that evaluates the expected cost of each decision at each stage using the Monte Carlo (MC) simulation. MC simulation, which has been effectively employed in various algorithms for the SRCPSP, involves using probabilistic techniques to model project uncertainties [195]. This simulation method generates multiple random scenarios based on their probability distributions, such as task durations and resource availabilities.

In [31], [203] a lookup table is used to reduce the computational cost of their approach. The lookup table stores the evaluation of state-decision pairs learned from randomly generated MC samples. Their approach was efficient because the training process to create the lookup table occurs only once per instance. After that, the algorithm consults the table at each stage. According to the law of large numbers, as the number of occurrences of a state-decision pair in the lookup table increases, the estimate converges to its true value. In cases where both the state and solution spaces have low dimensions, every state-decision pair can be adequately evaluated through a large number of MC samples. However, it is often impractical to visit every state-decision pair for high-dimensional state and solution spaces due to the computational burden. In such cases, a rollout procedure is necessary to address the issue of missing state-decision pairs in the lookup table. The efficacy of a closed-loop policy heavily relies on the capability to solve the sub-problem in each iteration to obtain the cost of each decision. Previous research, such as [32], [33], has explored the use of priority rules in addressing SRCPSP, demonstrating its effectiveness in this context. Studies such as [34] have indicated that a straightforward policy based on the average project solution performs well in many scenarios. This motivated us to develop our novel approximated dynamic programming approach based on integrating priority rules and employing the information from average projects within a closed-loop policy framework to tackle SRCPSP efficiently. This motivation led us to not only use effective priority rules to tackle the scheduling sub-problem but also evaluate the cost of each decision efficiently.

Algorithm 8 The Average project-based roll-out policy for SRCPSP.

Input: current time t, state S, eligible tasks $\mathcal{X}$ ($\mathcal{S}$)
Output: Selected task;
if *t=0* **then**
  Obtain a priority list π_L ;
end
Use MC to generate $|\Omega|$ scenarios;
Generate the shortlisted eligible tasks $\widehat{\mathcal{X}}$
for $j \in \widehat{\mathcal{X}}$ **do**
  for ω*=1:*$|\Omega|$ **do**
    $C(\Omega(\omega))$=SSGS(π_L,$\Omega(\omega)$, S, j, t);
  end
  $Cost(j) = \frac{\sum_{\omega=1}^{|\Omega|} C(\Omega(\omega)}{|\Omega|}$;
end
Evaluate and select a task $k \in \widehat{\mathcal{X}}$;
return *Selected task k*;

The minimum slack priority rule [190] is a scheduling heuristic that prioritises tasks based on their slack. Slack represents the time by which a task's completion can be delayed without

affecting the project's overall duration. In the minimum slack priority rule, tasks with the least slack are given higher priority for scheduling. The rationale behind this rule is to prioritise tasks with less flexibility in their start times, ensuring that critical tasks are scheduled as early as possible to minimise the risk of project delays.

Algorithm 8 represents our proposed policy for the A-ADP approach. In this policy, the priority list of the tasks (π_L) is generated before starting the project (at time $t = 0$). All the tasks are ranked according to their starting time in the average project and the value of their slack. To obtain the π_L, I solve the average project to find the starting time of tasks. The slack value of each task i is the difference between the earliest start time and the latest task start time i in the average project. The tasks scheduled earlier in the average project, with lower slack value, are assigned higher priorities for selection within the policy. I call this priority rule as $PR_{T_{det}-MSLK}$.

In this policy, the algorithm goes through all eligible tasks at each decision stage that do not violate the problem's constraints. Then, the algorithm generates a shortlist of eligible tasks, aiming to simplify the decision-making process within the roll-out policy. This shortlist comprises only a few eligible tasks, selected based on their highest priorities according to π_L.

To approximately evaluate the expected cost-to-go function starting from S_i, a set of $|\Omega|$ scenarios is generated using the MC simulation. The approximated value for the cost-to-go is obtained by calculating the average makespan across all $|\Omega|$ scenarios. The makespan for each scenario is obtained by a schedule generation scheme (SGS). SGS generates a feasible schedule from scratch by incrementally extending a partial schedule. A partial schedule refers to a schedule where only a subset of tasks have been scheduled. The serial schedule generation scheme (SSGS) is an SGS that schedules tasks as soon as possible from a given list one by one. This scheme considers both the precedence relationships and resource constraints of the project [93]. In this research, I use a different version of SSGS to generate a schedule. In this version, the tasks already scheduled at the current time t_i are not overtaken, meaning they are considered as already scheduled, and the algorithm only schedules the remaining tasks according to π_L. Additionally, some scenarios may not be consistent with the active tasks, where the duration so far is longer than the duration in the scenario. In this scheme, when a task i is taken at time t_i, the task j selected afterwards cannot be started before t_i.

For evaluation in this policy, I use an approach that I call EP_{T-SLK}. The EP_{T-SLK} is conducted to identify the task associated with the minimum cost among the shortlisted eligible tasks. This evaluation employs a dual-ranking mechanism. Firstly, it ranks the tasks based on their respective slack values derived from the average project. Additionally, it utilises the approximated expected cost-to-go function, which estimates the cost associated with completing each task and progressing through the remaining project stages. Combining these two ranking criteria, the algorithm assigns a priority to each task within the shortlist. The task with the highest average rank, indicating both minimal slack and low expected cost-to-go, is designated as the optimal choice for execution.

5.4 An illustrative example of the algorithm

The following example shows how the algorithm A-ADP makes decisions during the execution of a project.

The Figure 5.1 illustrates a project with ten tasks, two dummy tasks for the start and end of the

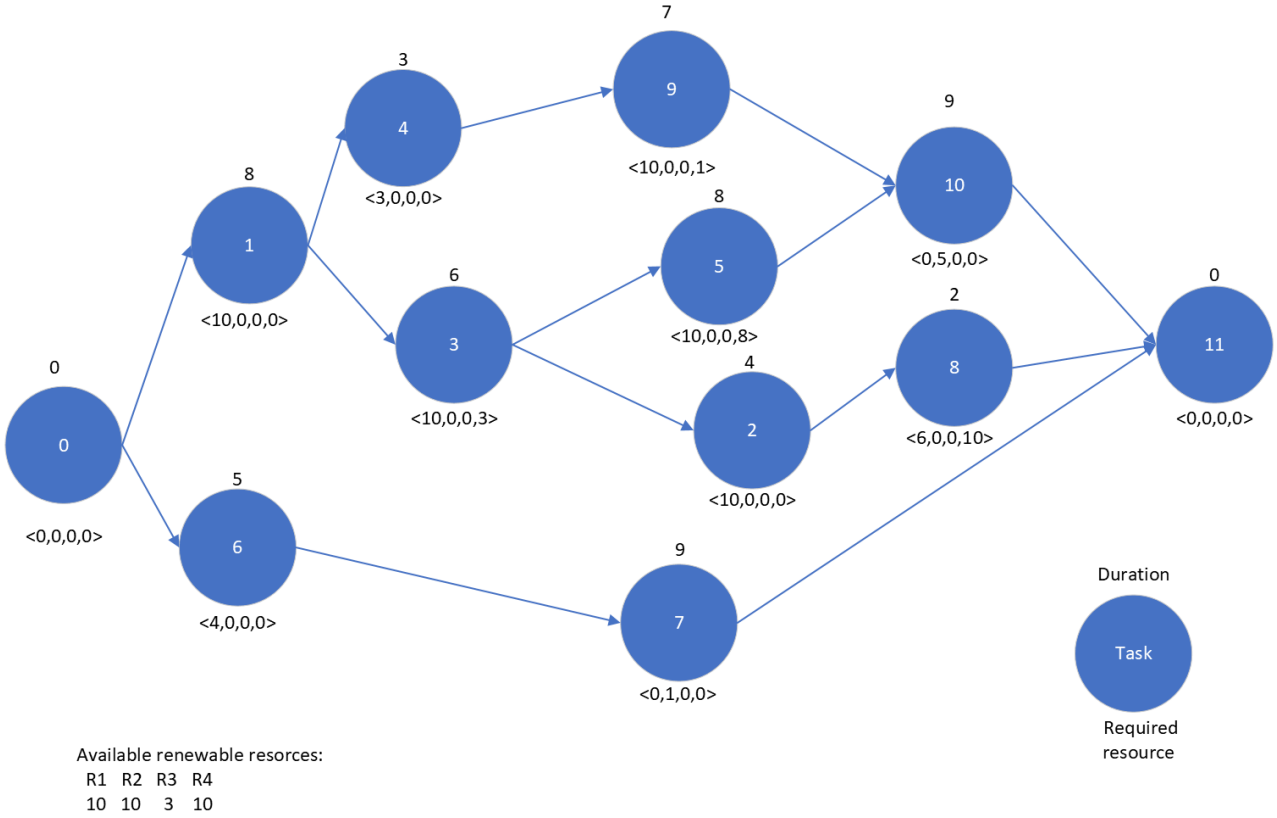

Figure 5.1: An illustrative example demonstrating the decision-making using A-ADP.

project, and four renewable resources. The Figure 5.2 shows how I assess our policy in solving a set of S scenarios of a given problem instance. The depicted flowchart in Figure 5.2 illustrates the generation of diverse scenarios using a given instance. Throughout the project execution, the duration of not completed tasks is unknown to the policy, and decisions regarding task initiation occur either at time zero or immediately upon completion of a preceding task. The policy autonomously determines the tasks to initiate based on dynamically evolving project conditions.

Table 5.1: Given scenario for the example

Task	0	1	2	3	4	5	6	7	8	9	10	11
Duration	0	7	5	7	3	6	7	12	1	7	11	0

In Table 5.1, the given scenario of the problem is depicted, where the duration for each task i is generated using a uniform distribution, $d_i \sim \text{Uniform}(0.8d_i, 1.5d_i)$. For simplicity in this example, I used a simple priority rule to obtain a priority list π_L at time t=0 and also a simple evaluation to select a task. In this priority rule, I prioritise the tasks started earlier in solving the average project, which I call $PR_{T_{det}}$. The priority list of tasks in the Table 5.2 was obtained by solving the average project in Figure 5.1, using the Cplex CP optimizer. I also set $|\Omega|$=3 and evaluate the cost of each eligible task j at each stage in the policy based on the expected makespan when selecting the task j.

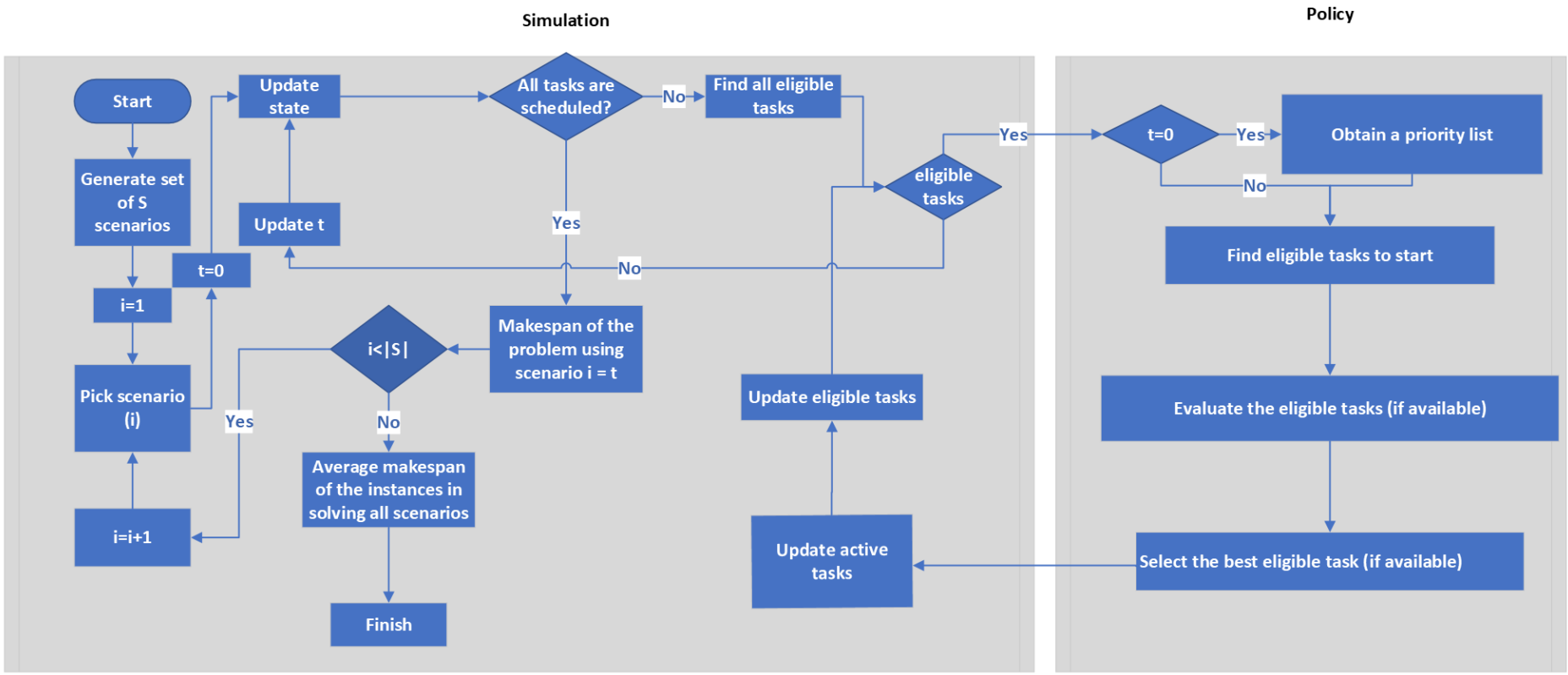

Figure 5.2: Simulation process for generating scenarios and decision making in policy

Table 5.2: Priority list obtained by Cplex CP optimizer

Task	0	1	2	3	4	5	6	7	8	9	10	11
Priority	0	0	34	8	22	14	22	27	38	27	34	43

Table 5.3: Solution of the given scenario

Stage	Time	Eligible tasks	Completed tasks	Active tasks	Selected tasks
1	0	1,6	0	1	1
2	7	3,4,6	1	3	3
3	14	2,5,4,6	3	2	2
4	19	4,5,6,8	2	4,6	4,6
5	22	5,8,9	4	8,6	8
6	23	**	8	6	**
7	26	5,7,9	6	5,7	5,7
8	32	9	5	9,7	9
9	38	**	7	9	**
10	39	10	9	10	10
11	50	11	10	11	11

As depicted in Figure 5.2, a priority list should be obtained at stage 1, at time t=0. As shown in Table 5.3 at this stage, tasks 1 and 6 are eligible to start. The expected makespan when selecting task 1 is 44.8, and when selecting task 6 is 46.1, so the task 1 is selected. At $t = 7$, task 1 is completed, initiating stage 2. Eligible tasks for stage 2 include tasks 3, 4, and 6, with task 3 selected by the policy. At $t = 14$, task 2 is selected, and upon completion at $t = 19$, tasks 4, 5, 6, and 8 become eligible. At this stage, tasks 4 and 6 start simultaneously. Upon completing task 4 in stage 5, task 8 is selected. When task 8 is completed at stage 6, no eligible tasks remain. Upon completing task 6 at stage 7, sufficient resources become available, allowing tasks 5 and 7 to start concurrently. Task 5's completion enables the initiation of task 9. At stage 9, no eligible tasks remain. Task 9 is completed at stage 10, and the project is completed at stage 11.

5.5 A matheuristic approach for solving the deterministic average project

In the A-ADP, the average project should be solved in the first stage. The average project is an RCPSP that can be challenging in large instances. The RCPSP is a special case of the MRCPS, featuring just one execution mode for each task. In Chapter 4, I introduced the Relax and Solve (R&S) algorithm, which showed promising results in addressing MRCPSP. Inspired by its success, I adapted its principles to solve RCPSP. The forward-backward improvement technique is widely used regarding the heuristics for solving the RCPSP [122]. This heuristic involves iterating through the schedule and examining forward and backward passes to identify potential improvements. By utilising this approach, the overall quality of the schedule can be gradually enhanced. This motivated us to take advantage of generating solutions using both forward and backward passes when solving RCPSP.

This section proposes a novel matheuristic approach for solving the RCPSP. This approach is called forward-backward relax-and-solve (R&S). It is based on the idea of iteratively relaxing and solving an RCPSP and taking advantage of forward-backward improvement. The following summarises this approach.

1. The R&S method used in this chapter first generates a feasible schedule (a solution).
2. The feasible solution is improved in an iterative process.
3. The iterative process includes:
   (a) The relaxing phase that relaxes the execution order of a few tasks at a time to generate a relaxed problem.
   (b) The solving phase solves the whole problem with a solver, which leads to developing a schedule for executing the tasks.
   (c) The CPLEX CP optimizer is employed in both forward and backward passes to create schedules.

Relaxing the problem this way helps to reduce the solution time because smaller problems are solved at every iteration. The main challenging part of employing such a relaxation method in solving RCPSP includes defining the set of tasks in a schedule to be relaxed. To handle that issue, I use a rolling time window to distinguish between tasks that are inside the time window (permitted to be reordered) and the tasks that are located outside the time window. To relax the problem, when the completion time of one task and the starting time of another task are the same, the algorithm adds start-at-end constraints between them. In this way, the algorithm prevents the change in the relative order of tasks outside the time window.

5.5.1 Forward-backward relax-and-solve method

This section proposes an efficient forward-backward R&S matheuristic algorithm for the RCPSP. The main idea of this method is to reduce the computational time by reducing the complexity of the problem at every iteration. I aim to relax the problem in a way that all tasks are involved in the solution while taking advantage of the forward-backward method to improve the search-ability of the algorithm. In the proposed forward-backward R&S algorithm an initial feasible solution is generated and gradually improved like most heuristics. For a feasible schedule a rolling time window is defined for the "relax" phase, in which all tasks outside the window are "glued" with each other with respect to the order in the current solution.

Only tasks inside the time window can be reordered. In the "solve" phase, a feasible schedule is obtained by solving the relaxed problem. As I do not remove the precedence and resource constraints the obtained solution is always feasible for the problem.

CPLEX CP optimizer is used to generate a solution in the forward pass considering problem's resource and precedence constraints. In addition to the forward pass, I generate a solution in a reversed direction i.e., in a backward pass. To this aim, the precedence constraints should be reversed, which means each arc (i, j) should be changed to arc (j, i). By doing so tasks are scheduled in a reversed order starting from the last task.

The algorithm starts with the initial solution generated in the backward pass. In each iteration, the algorithm relaxes certain tasks in the problem, which results in a relaxed problem, and then solves the relaxed problem in the forward pass. At the end of each iteration, the algorithm

removes all added start-at-end constraints and solves the problem in the backward pass for a limited time to avoid being trapped in a local optimum.

The general forward-backward R&S algorithm is summarised in Algorithm 9.

Algorithm 9 The forward-backward R&S algorithm.

Input: A RCPSP instance.
Output: A feasible schedule.
Generate an initial feasible solution using CPLEX CP optimizer on the backward pass formulation of the instance (see Section 5.5.2).
Set time window starting time $st = 0$
while *the stopping condition is not met* **do**
  if $st > C_{\max} - L$ **then**
    $st = 0$ ▷ $C_{\max}$ is the makespan and L is the time window length
  end
  Determine the tasks of group 1 and the tasks of group 2 (see Sections 5.5.3.2, and 5.5.3.3);
  Generate a relaxed problem (see Section 5.5.3);
  Solve the relaxed problem in the forward pass by using the CPLEX CP optimizer (Section 5.5.4.1);
  $st = st + L - overlap$; ▷ *overlap* is a parameter
  Solve the original problem in the backward pass with CPLEX CP optimizer (Section 5.5.4.2);
end
return *the best obtained schedule (the solution);*

In what follows, I discuss: generating an initial solution for the problem; generating and solving a relaxed problem in each iteration; and setting the stopping criterion for the algorithm.

5.5.2 Initial solution

[253] discussed the effectiveness of the CP in generating initial solutions for a crew scheduling problem. CPLEX CP optimizer, which is a commercial solver for CP, can generate quality solutions for challenging combinatorial optimization problems [243]. I use CPLEX CP optimizer in the backward pass to generate an initial solution.

To solve RCPSP, most approaches prefer to operate on the representation of the solutions and then employ decoding methods to transform those representations into a schedule. Common representation methods for RCPSP were proposed in [254]. In the present chapter, I operate on schedules rather than representations. Therefore, the solution for an instance is proposed through starting time for each task.

5.5.3 Generating the relaxed problem

Figure 5.3 illustrates an example of a feasible solution that will be relaxed in solving an RCPSP using the forward-backward R&S method. A rolling time window is employed to specify which part of a problem is relaxed.

In the figure, the time window divides the tasks into group 1 and group 2. In what follows I explain how the rolling time window works and how the tasks of each group are treated to generate a relaxed problem.

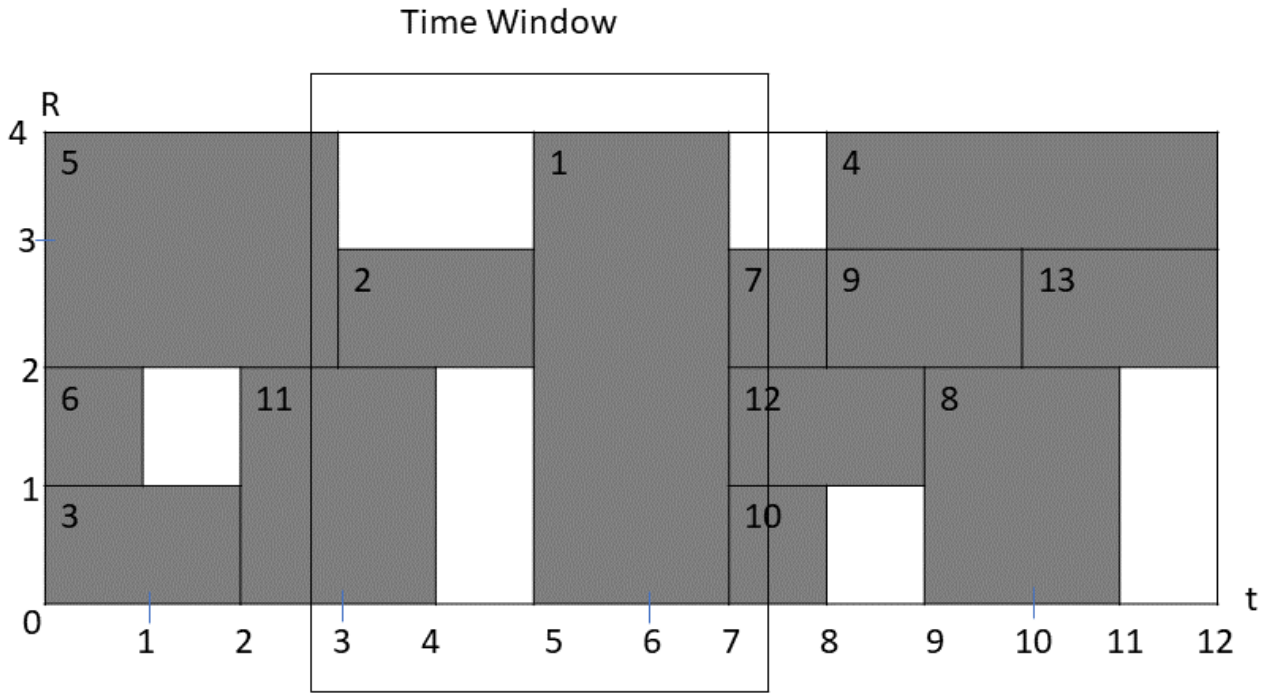

Figure 5.3: An example demonstrating how time window is defined for RCPSP.

5.5.3.1 The rolling time window

A time window is utilised to relax a schedule. I assume there is a rolling frame on the time horizon of an existing schedule. I provide an example in Figure 5.3 to show how a rolling time window specifies which part of the problem should be relaxed. Generally, increasing the length of the time window increases the number of tasks that are free to be reordered. The time window's starting time is set as $t = 0$, and after each iteration it moves forward at a specific rate. As soon as the time window reaches the end of the time horizon, it is set to $t = 0$. While moving the time window forward, a critical parameter preventing the algorithm from being trapped in local optima is the *overlap*, which is the amount of overlap between time windows in successive iterations. If there is no overlap, then the tasks are restricted to be inside a time window; hence, those tasks cannot move to other time windows. I suggest defining the length of the time window and *overlap* between time windows proportion to the makespan of the current feasible schedule of the problem that is subject to relaxing. Later in section 5.6, I propose to set the value of parameter *overlap* between 0 and 1.

I propose to calculate the length of the time window L as a function of the number of generated relaxed problems i.e., N, and also *overlap*, and makespan i.e.,$C_{\max}$, in particular $L = (\frac{C_{\max}}{N}) \times (1 + overlap)$.

5.5.3.2 Tasks of group 1 ($G1$)

The tasks that finish before the start of the time window, or start after the finish of the time window, i.e., those tasks that are entirely beyond the time window belong to group 1. For example, in Figure 5.3, the set of tasks of group 1 includes $G1 = \{3, 6, 4, 9, 8, 13\}$. The algorithm does not allow the tasks of group 1 to change their relative execution order. That is the main idea behind relaxation, as I define and use in the present chapter.

That can be achieved by adding "start-at-end" constraints between the tasks, which states that the start time of one task is equal to the finish time of another task. Examples include $(3, 11)$ or $(12, 8)$ in Figure 5.3. In this example, tasks $7, 4, 9, 13$ are also fixed together.

A point that should be noted here is that one task starts as soon as its resource constraints and precedence constraints are satisfied. So, each task starts at $t = 0$ or starts at the finish time of another task. In such a case, as for task 6 in the example, the algorithm does not need to fix the task with other tasks

5.5.3.3 Tasks of group 2 ($G2$)

The tasks of group 2 or $G2$ are those tasks that are completely or partially inside the time window. For example, $G2 = \{5, 11, 2, 1, 7, 12, 10\}$ in Figure 5.3 shows tasks of group 2. The tasks of this group can be reordered subject to satisfying all problem's constraints if that leads to an improved schedule

5.5.4 Solving Phase

This phase includes two steps: 1) solving each relaxed problem and 2) solving the original problem. In both steps, the forward-backward R&S algorithm uses the CPLEX CP optimizer.

5.5.4.1 Solving the relaxed problem

The relaxed problem includes a fewer tasks to be reordered compared to the original problem, which results in the relaxed problem to be relatively easier to solve. The obtained solution to the relaxed problem is always feasible because the tasks of group 1 cannot be reordered, however, the starting time of each set of fixed tasks is flexible, so the makespan of the relaxed problem is always the makespan of the original problem. I solve each relaxed problem in the forward pass.

5.5.4.2 Solving the original problem

The algorithm solves the general problem immediately after solving each relaxed problem. Solving the original problem even for a few seconds in backward pass improves the global exploration of the algorithm. Therefore, it is beneficial to search the space in both directions.

5.5.5 Stopping criterion

The maximum number of iterations should be determined before solving a problem. In each iteration, a time limit is set for the CPLEX CP optimizer to solve each forward or backward pass. Once the CPLEX CP optimizer obtains the optimal solution, or if the computation time reaches the limit, the algorithm goes to the next iteration. The algorithm terminates when it reaches the maximum number of iterations.

5.6 Computational experiment for solving the deterministic average project

In this section, I provide computational results of the proposed forward-backward R&S on instances from the well-known PSPLIB [37]. I consider instance sets J30, J60 and J120, which

Table 5.4: comparison of LCG, FDS, SMT, CPLEX, and forward-backward R&S .

	J30		J60		J120	
Method	Δ_{LB}	t	Δ_{LB}	t	Δ_{LB}	t
LCG	0	-	2.17	-	9.76	-
FDS	0	0.93s	1.91	67.44s	7.02	322.52s
SMT	0	0.22s	1.88	61.90s	9.55	320.50s
CPLEX CP optimizer	0	4.71s	1.11	52.83s	4.69	350.40s
R&S	0	5.12s	1.06	46.75s	4.63	235.65s
Forward-backward R&S	0	7.17s	1.06	66.21s	4.61	200.11s

include 30, 60, and 120 tasks, respectively. Instance sets J30 and J60 include 480 instances and the set J120 includes 600 instances. I coded the algorithm by using the Python programming language version 3.6.5, and I used CPLEX CP optimizer version 12.10.0.0 [15].

Unless otherwise stated, I use default value for CP parameters. I conduct all the experiments on a machine with CPU Intel Xeon Gold 6238R 2.2GHz.

Next, I discuss the parameter setting followed by the detailed computational results.

5.6.1 Parameters setting

The initial solution generated in a backward pass is obtained by setting the time limit to 30 seconds. The stopping time for solving each relaxed problem is set to 60 seconds. The allocated time for solving the original problem in the backward pass (at the end of each iteration) is set to 30 seconds. Those lead to average computational times of less than 150, 300 and 600 seconds, for instance sets J30, J60, and J120, respectively. I set $overlap = 0.33$ and the maximum number of relaxed problems $N = 0.1 \times n$.

Each relaxed problem should be solved in one iteration, meaning that the maximum number of iterations is equal to 3, 6, and 12 for J30, J60, and J120, respectively.

5.6.2 Results

In this section, I compare our results and those obtained by CPLEX CP optimizer and three exact methods, including failure-directed search (FDS)[255], lazy clause generation (LCG) [256], satisfiability modulo theories (SMT) by [257], that shown competitive results in solving RCPSP.

To compare the results delivered by the proposed method and the exact methods, I present a summary of the results in Table 5.4. In this table, t is the computation time in seconds and the average deviation from the best-known lower bound for each instance is calculated as:

$$\Delta_{LB} = \frac{\sum_{p=1}^{p=P} \frac{C_{max,forward-backwardR\&S,p} - C_{max,LB,p}}{C_{max,LB,p}} \times 100\%}{P}, \tag{5.7}$$

where P is the number of instances in each set, $C_{max,forward-backwardR\&S,p}$ is the makespan obtained by Algorithm 9, and $C_{max,LB,p}$ is the best known lower bound on the makespan for each instance.

Table 5.5: Comparison of the state-of-the-art metaheuristics and forward-backward R&S

Algorithm	**J30**	**J60**	**J120**
Forward-backward R&S (Proposed in this chapter)	**0**	**10.53**	31.07
R&S (Proposed in this chapter)	0	10.54	31.09
CABFIA (Proposed in Chapter 3)	0	10.62	31.48
GA(SA(FBI))[103]	0.01	10.63	**30.66**
GA(FBI)[110]	0.02	10.68	30.82
Sequential(SS(FBI))[118]	0	10.58	31.16
GA(FBI)[109]	0.02	10.73	31.24
GA(FBI)[107]	0	10.57	31.28
GA(FBI)[108]	0	10.71	31.3
PSO(LS) [235]	0.05	10.62	31.43
GA(FBI) [106]	0.03	10.84	31.49
ALNS(FBI)[239]	0.02	10.73	31.54
SS(EM + FBI)[240]	0.01	10.71	31.57
GA(FBI)[104]	0.01	10.81	31.65
PSO(FBI)[236]	0.02	10.85	32.4
EA(FBI)[112]	0.03	10.91	32.52
GA(FBI)[111]	–	10.57	32.76
PSO(FBI)[237]	0.01	10.79	32.89
GA(FBI)[105]	–	10.66	33.82
BCO(FBI)[258]	0.04	11.16	34.55
DBGA [102]	0.02	10.68	30.69
GA-FBI [259]	0.0	10.56	32.76
COA [241]	0.0	10.58	31.22
PSO-SS[238]	0.01	10.68	31.23
MA [90]	0	10.55	31.12

For each method, the computation time and the average deviation from the best known lower bound are provided in Table 5.4. Results show that employing the forward-backward method in R&S improves the performance of R&S, leading to superior results, in terms of Δ_{LB} criterion, to other methods for solving J60 and J120 instances. All of the tested methods obtain optimal solutions for J30. Comparing the result of the CPLEX CP optimizer, as a stand-alone method, and both R&S and forward-backward R&S for J60 and J120 illustrates the efficiency of the forward-backward R&S in solving the tested instances. For almost all instances of J30, the CPLEX CP optimizer finds the optimal solution in a short time.

In Table 5.5, we compare the results of forward-backward R&S with CABFIA that I proposed in Chapter 3, and the state-of-the-art metaheuristics as in [90], including memetic algorithm (MA), consolidated optimization algorithm (COA) [241], PSO-based hyper-heuristic algorithm (PSO-HH) [238] and genetic algorithm using forward-backward improvement (GAFBI) [259], and those algorithms that combine the forward-backward method with GA [101], [102], [104]–[111], PSO [235]–[237], ACO [113], ABC [258], the Lagrangian relaxation [119] and scatter search [118].

In the literature, it is common to set the number of generated schedules as the stopping criterion. However, it does not apply to our algorithm because it used the CPLEX CP optimizer. Therefore, I compare the best results of the tested algorithms for 50,000 generated schedules,

which time-wise is very close to the computational times of our methods. The average deviation from the critical path method (CPM), i.e., Δ_{CPM} is calculated by using Equation (5.7), where the lower bound of the makespan ($C_{max,LB,p}$) is obtained by CPM.

Table 5.5 for the instances with 30 tasks shows the average deviation of the solution of each algorithm from the optimal solutions. In this set of instances, forward-backward R&S can find the optimal solution for all the instances. The results for the other instances in this table are the average deviation of the solution for each algorithm from the CPM. In solving instances with 60 tasks, our method provides the lowest average deviation from the CPM among all the algorithms, so it is the best method for this set of instances. Dealing with the instances with 120 tasks, DBGA [102], GA(SA(FBI))[103], and GA(FBI)[110] reported better solution than our algorithm, and our method is better than the remaining state-of-the-arts

5.7 Computational experiments for solving stochastic resource-constrained project scheduling problem

The computational experiments provided in this chapter are to investigate the efficiency of our proposed A-ADP approach. I investigate the project characteristic that has the most significant impact on the algorithm's performance and identify when it is important for the algorithm to work with the stochastic RCPSP. I also investigate the efficiency of the proposed A-ADP when using different priority rules obtained from the average project. The algorithm is tested on J30 (instances with 30 tasks), J60 (instances with 60 tasks), and J120 (instances with 120 tasks). The PSPLIB dataset, available at http://www.om-db.wi.tum.de/psplib/, is created using the ProGen project instance generator, which is characterised by network complexity NC, resource factor RF, and resource strength RS [260].

I followed the assumptions in [195] for the stochastic task duration. Table 5.6 shows the employed PDs in this research. The deterministic duration for task j is denoted by d_j, and the PDs include Uniform distributions, Beta distributions and exponential distribution (EXP).

Uniform distributions have a constant probability, with slight variance in U1 and intermediate variances in U2. The exponential distribution (EXP) also maintains a constant failure rate but exhibits a larger standard deviation compared to the uniform distribution.

Beta distributions, characterised by slight variance in B1 and intermediate variances in B2, are well-known probability distributions in scheduling under uncertainty. These distributions are represented using two shape parameters (α and β) to generate random values.

Similar to [31], to determine the performance of each experiment, a set S of 50 scenarios are generated for an instance r, $r \in \{1, \ldots, R\}$ using MC and a known PD provided in Table 5.6. I solve the problem by Algorithm 8 for each scenario $i \in S$ with the $Makespan(i)$. The expected makespan for the instance r is calculated as $E_r = \frac{\sum_{i=1}^{|S|} Makespan(i)}{|S|}$, where $|S| = 50$. The parameter Gap is defined to compare the results in different experiments and is the percentage of the average deviation of the expected makespan from the deterministic critical path length(CPL) of instances, i.e., $Gap = \frac{1}{R}\sum_{r=1}^{R} \frac{E_r - CPL_r}{CPL_r}$.

Our algorithm involves two main computational components. In solving small instances with 30 and 60 tasks, I allocate a time limit of 5 seconds for solving the average projects with 30 tasks and 10 seconds for average projects with 60 tasks which is enough to obtain a feasible solution

using Cplex CP optimizer[35], [120]. In solving instances with 120 tasks, I used the forward-backward R&S algorithm for solving the average project in the A-ADP. I set $overlap = 0.33$ and the maximum number of relaxed problems $N = 0.1 \times n$ and the computational time for the Cplex CP optimizer to obtain the initial solution, solve each relaxed problem and solve the original problem in backward pass to 10 seconds.

The maximum number of schedules considered for the remaining stages of the algorithm is determined as follows. The maximum number of schedules considered for a problem with n tasks when the number of tasks in the shortlisted eligible tasks is L_{sle}, and employing $|\Omega|$ scenarios for evaluating the tasks in the policy, is $n \times |\Omega| \times L_{sle}$.

5.7.1 A special case: when the shortlist has length 1

This section investigates the performance of the policy when $L_{sle} = 1$. In this special case, the policy does not need to evaluate the shortlisted eligible tasks because this task is the sole eligible task selected to be in the shortlisted eligible tasks. In this experiment, I solved all 480 instances of J30. The case $L_{sle} = 1$ is considered to assess the impact of using the results of the average project on the A-ADP approach's performance. To this aim within Algorithm 8, π_L is determined solely based on the starting time of each task in the average project, disregarding the impact of task slack values denoted as the priority rule $PR_{T_{det}}$. Additionally, task evaluation is conducted using $EP_T = \arg\min_j(\text{Cost}(j))$ for selecting each task j in the policy, considering only the expected makespan.

Table 5.6: Probability distributions of task duration.

PD	Limits		Mean	Variance
	lb	ub	μ	σ
U1	$dj - \sqrt{dj}$	$dj + \sqrt{dj}$	-	$\frac{dj}{3}$
U2	0	$2dj$	-	$\frac{dj^2}{3}$
Exp	-	-	dj	dj^2
B1	$\frac{dj}{2}$	$2dj$	-	$\frac{dj}{3}$
B2	$\frac{dj}{2}$	$2dj$	-	$\frac{dj^2}{3}$

I compare the results obtained when setting L_{sle} and $|\Omega|$ to three. I denote the achieved gap as $Gap_{3T_{det}}$ under these conditions. This is compared with the case where L_{sle} is set to one, resulting in a gap denoted as $Gap_{1T_{det}}$. In the latter case, the policy selects the eligible task with the highest priority without approximating the cost-to-go function. Consequently, decision-making solely relies on the deterministic solution derived from the average project.

The availability of resources in a problem is determined by $RS \in \{0.2, 0.5, 0.7, 1\}$. In Figure 5.4, the difference between $Gap_{3T_{det}}$ and $Gap_{1T_{det}}$ for each value of RS is provided for each PD. The results obtained from this experiment show that in the problems with $RS = 0.2$, the difference between the two policies is the maximum for all PDs. By increasing RS, which means increasing the availability of the resources, the difference between the results obtained by the policies decreases. For U1 where the variance of the probability distribution is relatively low, when $RS = 1$, the policy based on the average project performs better than A-ADP when there are three tasks in the shortlisted eligible tasks. I did not find a meaningful relation between the performance of the A-ADP with characteristics NC and RF.

Table 5.7 compares the results of the state-of-the-art algorithms in solving SRCPSPs, with

Table 5.7: Comparison of results obtained by A-ADP and state-of-the-art algorithms for J30 [193].

Algorithm	Gap				
	U1	U2	EXP	B1	B2
PPGA[192]	19.87	30.67	45.56	19.93	30.76
A-HBA[31]	16.63	42.37	45.13	12.60	16.63
LFT [190]	21.60	30.89	46.47	21.59	30.87
SLFT [190]	21.60	30.83	46.32	21.60	30.76
DH [190]	21.36	31.18	46.86	21.36	31.21
S-COA [193]	**1.56**	**8.67**	**16.66**	**1.29**	**7.72**
A-ADP	7.72	12.94	29.26	4.15	11.37

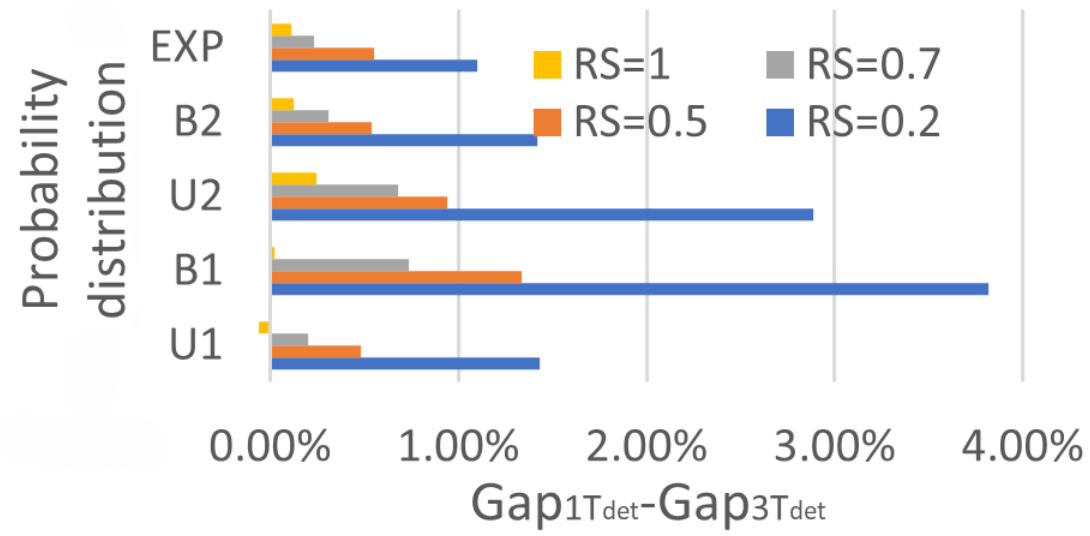

Figure 5.4: The impact of *RS* on the performance of the policy when considering uncertainty.

$Gap_{3T_{det}}$ obtained by our A-ADP [31], [190], [192], [193]. The stopping criterion for schedule generation is commonly defined as 5000 in the literature. I set the parameter $|\Omega|$ to 3, indicating the number of scenarios in the policy. Given that the length of the shortlisted eligible tasks is limited to 3, the policy's maximum number of generated schedules at each stage is 9. Consequently, for a problem comprising 30 tasks, the maximum number of schedules considered is 270.

Table 5.7 illustrates that more variance in task duration results in a higher *Gap* for the same PD. Comparing other algorithms, except for the S-COA, our policy outperforms the other algorithms, including [31], the other ADP algorithm [31].

5.7.2 The impact of the priority rules and the number of scenarios in policy

This section investigates the impact of the proposed priority rule in generating the priority list at t=0 and the number of scenarios, $|\Omega|$ in this policy. For this purpose, I employed subsets of datasets J30 and J60 from PSPLIB denoted as $J_{48}30$ and $J_{48}60$. J30 and J60 comprise 48 different problem characteristic combinations, with each problem having ten distinct instances.

To comprehensively assess the algorithm across all these combinations, I chose the first instance from each problem,i.e., $J_{48}30 = \{j301-1, j302-1, \ldots, j3048-1\}$ and $J_{48}60 = \{j601-1, j602-1, \ldots, j6048-1\}$.

I compared the results obtained with the $PR_{T_{det}-MSLK}$ to those with $PR_{T_{det}}$. The results of solving $J_{48}30$ and $J_{48}60$ are provided in Table 5.8, and Table 5.9. The $Gap_{1T_{det}-MSLK}$, is the *Gap* in which the shortlisted eligible tasks in the policy have just one task, and the priority rule is $PR_{T_{det}-MSLK}$.

When comparing $Gap_{1T_{det}}$ and $Gap_{1T_{det}-MSLK}$ for all PDs in order to solve $J_{48}30$ and $J_{48}60$, Table 5.8 and Table 5.9 clearly demonstrate that using the priority rule $PR_{T_{det}-MSLK}$, which takes into account the minimum slack value of tasks, yields superior results when compared to cases where slack value is not considered in the priority rule.

Table 5.8: Results of $J_{48}30$ when L_{sle}=1.

PD	$Gap_{1T_{det}}$	$Gap_{1T_{det}-MSLK}$
B1	3.91	**2.33**
B2	10.82	**9.62**
EXP	28.70	**27.79**
U1	7.40	**5.53**
U2	12.63	**10.98**

Table 5.9: Results of $J_{48}60$ when L_{sle}=1.

PD	$Gap_{1T_{det}}$	$Gap_{1T_{det}-MSLK}$
B1	15.78	**13.75**
B2	22.94	**21.02**
EXP	42.65	**40.84**
U1	18.87	**16.66**
U2	24.24	**22.15**

Additionally, I examined two approaches for evaluating each task in the eligible shortlisted tasks. One was the evaluation method proposed in section 5.3 denoted as EP_{T-SLK}, while the other involved assessing tasks by disregarding their slack values, EP_T. Furthermore, I explored how the number of scenarios, represented as $|\Omega|$, impacts the algorithm's efficiency. Our investigation included a comparison of results when $|\Omega|$ was set to 3, with the results obtained when $|\Omega|$ was equal to 1, and the average project was the single scenario in the policy. This analysis shows how altering the number of scenarios impacts the algorithm's performance.

Table 5.10 represents the results of $J_{48}30$. Comparing the results for different PDs, there is no meaningful difference when $|\Omega|$ is one or three, and in terms of computational efficiency, employing the task duration of an average project to calculate the cost-to-go function when $|\Omega|$=1 reduces the computations by up to a third compared to the other approach of finding the expected makespan by averaging the makespan of scenarios. Table 5.11 displays the results for the $J_{48}60$ experiment, which leads us to the same conclusion. Consequently, I use the average project with $|\Omega|$ set to one.

When I compare the different evaluation methods in both tables, regardless of the PDs and whether I am looking at $Gap_{3T_{det}}$ or $Gap_{3T_{det}-MSLK}$, consistently EP_{T-SLK} outperforms EP_T.

Moreover, if I specifically consider the EP_{T-SLK} approach, it is clear that the values of $Gap_{3T_{det}-MSLK}$ consistently surpass those of $Gap_{3T_{det}}$.

Table 5.10: Results of $J_{48}30$ for different $|\Omega|$, priority rule and evaluation approaches when L_{sle}=3.

PD	$\|\Omega\|$=1				$\|\Omega\|$=3			
	$Gap_{3T_{det}}$		$Gap_{3T_{det}-MSLK}$		$Gap_{3T_{det}}$		$Gap_{3T_{det}-MSLK}$	
	EP_{T-SLK}	EP_T	EP_{T-SLK}	EP_T	EP_{T-SLK}	EP_T	EP_{T-SLK}	EP_T
B1	2.22	3.48	**1.87**	2.96	2.32	3.31	2.00	3.01
B2	9.68	11.01	9.49	10.32	9.76	10.89	**9.44**	10.51
EXP	27.93	28.89	**27.42**	28.87	27.82	28.69	27.98	28.05
U1	5.60	7.15	**5.45**	6.58	5.80	7.02	5.52	6.65
U2	10.98	12.62	10.94	12.15	10.96	12.36	**10.92**	11.76

Table 5.11: Results of $J_{48}60$ for different $|\Omega|$, priority rule and evaluation approaches when L_{sle}=3.

PD	$\|\Omega\|$=1				$\|\Omega\|$=3			
	$Gap_{3T_{det}}$		$Gap_{3T_{det}-MSLK}$		$Gap_{3T_{det}}$		$Gap_{3T_{det}-MSLK}$	
	EP_{T-SLK}	EP_T	EP_{T-SLK}	EP_T	EP_{T-SLK}	EP_T	EP_{T-SLK}	EP_T
B1	14.40	15.77	13.55	14.65	14.49	15.62	**13.50**	14.28
B2	21.28	23.40	**20.91**	21.68	21.61	23.41	20.98	21.74
EXP	41.18	43.72	**40.78**	42.41	41.36	43.16	40.84	42.33
U1	16.94	19.16	16.67	18.07	19.14	17.46	**16.47**	17.92
U2	23.18	24.71	**21.82**	23.55	23.08	24.13	22.55	22.67

5.7.3 Comparison with the state-of-the-art algorithms on small instances

Based on the parameters discussed in Section 5.7.2, this section compares our obtained $Gap_{3T_{det}-MSLK}$, when $|\Omega|$=1, and using EP_{T-SLK} with the state-of-the-art algorithms. I solved all J30 and J60 sets of instances. The results of our proposed algorithm are compared with other algorithms for different PDs, i.e., U1, U2, EXP, B1, and B2.

As L_{sle}=3 and $|\Omega|$=1, the maximum number of generated schedules in solving J30 is 90, and in solving J60 is 180. Table 5.12 and table 5.13, compare the results of our proposed A-ADP algorithm with the state-of-the-art algorithm. The stopping criterion of the other algorithms is 5000 generated schedules. As is shown in Table 5.12 in solving J30 instances except for A-COA, our algorithm outperforms other state-of-the-art algorithms. Table 5.13 presents the results for J60 instances. When solving J60 with the U1 probability, A-HBA and S-COA perform better than our A-ADP. However, S-COA is the best-performing algorithm for the remaining probability distributions, and A-ADP is the second-best compared to the state-of-the-art algorithms. To show the efficiency of A-ADP in comparison to S-COA, the CPU times of these algorithms are presented in Table 5.15. As is highlighted, A-ADP demonstrates significantly shorter computational times than S-COA when solving both J30 and J60 problems.

Table 5.12: Comparison of $Gap_{3T_{det}-MSLK}$ and Gap in the state-of-the-art algorithms for J30 [193].

Algorithm	Gap				
	U1	U2	EXP	B1	B2
PPGA	19.87	30.67	45.56	19.93	30.76
A-HBA	16.63	42.37	45.13	12.60	16.63
LFT	21.60	30.89	46.47	21.59	30.87
SLFT	21.60	30.83	46.32	21.60	30.76
DH	21.36	31.18	46.86	21.36	31.21
S-COA	**1.56**	**8.67**	**16.66**	**1.29**	**7.72**
A-ADP	5.66	11.16	27.95	1.81	9.61

Table 5.13: Comparison of $Gap_{3T_{det}-MSLK}$ and Gap in the state-of-the-art algorithms for J60 [193].

Algorithm	Gap				
	U1	U2	EXP	B1	B2
PPGA	18.91	29.08	45.74	18.98	29.17
A-HBA	14.14	28.57	45.36	18.31	28.77
LFT	19.94	28.49	44.97	19.95	28.63
SLFT	19.89	28.42	44.94	19.90	28.55
S-COA	**12.60**	**19.31**	**28.70**	**12.62**	**18.54**
A-ADP	18.09	23.67	43.17	14.52	22.21

5.7.4 Comparison with the state-of-the-art algorithms on large instances

This section compares our obtained $Gap_{3T_{det}-MSLK}$, when $|\Omega|$=1, and using EP_{T-SLK} with the state-of-the-art algorithms. The proposed forward-backward R&S algorithm is used to solve the deterministic average project in solving large instances. I considered U1, U2, EXP, B1, and B2 PDs and compared our proposed algorithm with others.

Table 5.14: Comparison of $Gap_{3T_{det}-MSLK}$ and Gap in the state-of-the-art algorithms for J120 [193].

Algorithm	Gap				
	U1	U2	EXP	B1	B2
AB-GA	51.49	78.65	120.22	-	-
AB-GR	46.84	72.58	114.42	47.17	75.97
RB-EDA	47.29	59.54	72.50	47.65	58.29
GP-H	46.71	55.95	71.71	46.87	55.95
PPGA	48.86	59.91	76.03	49.01	58.82
EDA	47.29	56.54	72.50	47.65	58.29
ADP-HBA	42.11	71.94	74.90	38.93	46.58
S-COA	**34.57**	**37.15**	**39.58**	**34.53**	**35.40**
A-ADP	42.09	50.99	68.78	45.76	49.91

Results in the Table 5.14 indicate that A-ADP outperforms other algorithms except for S-COA

and ADP-HBA (when PD is B1 and B2). When comparing the computational time of A-ADP with S-COA, as shown in Table 5.15, it becomes apparent that although S-COA yields better results, A-ADP is significantly faster.

Table 5.15: Comparison of the computational time of A-ADP and S-COA.

Prob.	Alg.	CPU Time (seconds)				
		U1	U2	EXP	B1	B2
J30	S-COA	58.72	69.09	70.32	64.42	67.24
	A-ADP	**4.66**	**4.63**	**4.47**	**4.87**	**4.63**
J60	S-COA	181.21	185.45	193.87	183.7	190.88
	A-ADP	**36.93**	**36.92**	**35.39**	**38.19**	**39.26**
J120	S-COA	1325.03	1325.05	1362.66	1190.88	1188.97
	A-ADP	**615.80**	**617.29**	**621.86**	**667.46**	**663.92**

5.8 Summary

This chapter proposed a novel A-ADP approach for solving SRCPSP. To solve the deterministic RCPSP in this approach, I presented a matheuristic. In this algorithm for solving the RCPSP, a solution is presented by a schedule that indicates the starting time of each task and then relaxed by removing precedence constraints for a set of tasks that are specified by a rolling time window. The relaxed problems are then solved using the CPLEX CP optimizer in forward and backward passes. An important aspect of our method's efficiency lies in relaxing an instance without decreasing the number of tasks. In our method, certain tasks are not allowed for their execution order to be altered. In this way, the obtained solution is always feasible. A set of 1560 instances of PSPLIB with 30, 60 and 120 tasks were used to test the proposed algorithm. Instances with 30 tasks can be optimally solved by the CPLEX CP optimizer quickly. However, for larger instances, i.e., instances with 60 and 120 tasks, our algorithm provides superior solutions in a shorter time to the stand alone CPLEX CP optimizer. The results indicate that arming the R&S with forward and backward passes leads to better results than the R&S with only forward passes. Generating a solution in both passes prevents the algorithm from being trapped in the local optima and reduces the dependency of the algorithm on initial solutions.

In the A-ADP approach, results obtained from solving the deterministic problem are utilised for decision-making in solving the SRCPSP. For the average project instances in the J120 dataset, I employed the forward-backward R&S method. However, for smaller instances, I opted for the Cplex CP optimizer. Overall, the A-ADP approach yields competitive results in solving SRCPSP instances with 30, 60, and 120 tasks. Investigating different problem characteristics also indicates that the larger shortlisted eligible tasks are required for the problems with the lower availability of resources. In making and evaluating the shortlisted eligible tasks, our experiments show the importance of prioritising tasks with lower slack values and tasks started earlier in the average project.

Chapter 6

Conclusions and Future Work

This thesis introduced and discussed several algorithms for solving three Np-hard scheduling problems. For each of the discussed problems, I summarised the contributions to the existing research and offered suggestions for future research.

- The first problem I tackled in this thesis is RCPSP. In Chapter 3 ABFIA, a hybrid metaheuristic optimisation algorithm is proposed. ABFIA is an extension of the Artificial bee colony algorithm that has been improved by reinforcing its local search capability. The computational results on a wide range of benchmark functions show the algorithm's strong exploration and fast convergence. The CABFIA is proposed based on the idea of the ABFIA for solving RCPSP. The computational results show the efficiency of the CABFIA for solving RCPSP. The reason for the efficiency of the CABFIA is that, besides the strong exploration and fast convergence of the ABFIA, the search space is reduced in the initialisation phase. Solutions are generated in both forward and backward passes, and the Cplex Cp optimizer is warm-started with good solutions from the onlooker bee phase. For J30 instances, CABFIA consistently delivered optimal solutions, regardless of whether we stopped at 1,000, 5,000, or 50,000 generated schedules. However, when solving larger instances, such as J60 and J120, the best-known algorithms outperformed CABFIA by margins of 0.1% and 0.8%, respectively. This indicates a need for further improvements in CABFIA to remain competitive as the best algorithm for larger instances. The main components of the CABFIA are FIA as a strong local search algorithm and also the CPLEX CP solver inside the algorithms. In Section 5.5, a matheuristic approach, forward-backward R&S, is proposed. In this method, the CPLEX CP solver is utilised to solve the relaxed problems in both forward and backward passes. The results of the tests on 1560 instances from the standard library PSPLIB with up to 120 tasks show the capability of this method to obtain good quality solutions. More specifically, forward-backward R&S finds the optimal solutions for all J30 instances. For J60 instances, it improves upon the previously best-known results by 0.1%, establishing itself as the top-performing algorithm. However, for J120, there is a discrepancy of 0.4% compared to the best-performing algorithms, indicating room for improvement in handling larger instances. The main component of this approach in solving RCPSP is using the idea of making super tasks and iteratively relaxing the problem without removing constraints so the solutions obtained for each iteration are feasible for the original problem.

- The second problem I considered in this thesis is MRCPSP. Even finding a feasible solution for an MRCPSP can be very challenging in the shortage of nonrenewable resources.

In Chapter 4, GMKP, an integer program-based mode selection method, was designed and employed to select the execution mode for each task. Findings from an extensive computational investigation indicate that the suggested R&S algorithm can produce outcomes that outperform existing algorithms documented in the literature. The 'SolutionsUpdate' software [248] was used to check the feasibility of solutions and keep the best-known solutions up to date. The R&S method updated six upper bounds for set MMLIB50, 32 upper bounds for set MMLIB100, and 614 upper bounds for set MMLIB+. In MMLIB+, updating the upper bounds led to reporting the optimal solution for four open instances. The reasons for the efficiency of the R&S include efficient mode selection by definition of the objective function for GMKP, making relaxed problems so that the solution from the relaxed problem is always feasible for the original problem, using a constrained programming approach, and efficiently using Cplex Cp optimizer in our heuristic.

- The last problem I considered in this thesis is SRCPSP. An approximate dynamic programming approach is defined in Chapter 5. This approach considers the solution from solving the deterministic problem to determine each task's priority. Investigating different problems' characteristics indicates that with the greater availability of nonrenewable resources, results from deterministic average projects are more reliable for solving the problem under uncertainty. Also, it is important to give higher priority to the tasks that have a lower slack value. The computational results in solving SRCPSP with 30, 60, and 120 tasks show that the A-ADP approach provides competitive results quickly compared to the existing approaches. More specifically, in solving different instances, although another algorithm provides better results than A-ADP, the computational time of A-ADP is significantly less—almost 90% less for J30 instances, 80% less for J60, and 55% less for J120 compared to that algorithm. The key advantages of the proposed A-ADP are solving the deterministic problem once for each instance, employing effective priority rules, and including the slack value of tasks in evaluation.

When addressing real-world problems that do not align directly with the objectives, constraints, or environments studied in this thesis, the following general guidelines can facilitate method selection and adaptation:

- Begin by thoroughly evaluating the specific requirements and constraints of the project at hand. Consider the project's complexity, resource availability, and uncertainties. This initial assessment will guide the choice of solution methods and determine whether standard or more innovative approaches are required.

- In tackling large-scale problems, the concept of 'super tasks' can be beneficial. This approach helps reduce the problem's complexity without removing constraints and provides feasible solutions for the original problem.

- For problems with tight constraints, I recommend adopting the strategy outlined in Chapter 4. This involves focusing initially on the most challenging aspects of the problem to generate feasible solutions. Subsequently, these solutions can be refined using heuristic methods, which are less computationally intensive and can quickly produce improved solutions.

- When dealing with uncertainty, the availability of resources becomes a pivotal factor. If resources are plentiful, it may be reasonable to model the problem deterministically and apply solutions from such models directly. However, it is important to focus more on the uncertainty aspect in scenarios where resources are limited. In such cases, evaluating

different scenarios becomes essential to understanding how resource constraints might impact project outcomes and developing robust strategies under various conditions.

The potential future research directions for the work presented in this thesis are as follows.

The future works related to Chapter 3 are,

- Enhanced Parameter Tuning through Machine Learning: Investigate leveraging machine learning methods, particularly reinforcement learning, to enhance the parameter tuning process of ABFIA and CABFIA algorithms. This adaptive approach could enable the algorithms to autonomously adjust parameters based on problem characteristics and dynamics of the solution space, improving their robustness and adaptability across various optimisation tasks.
- Balancing Exploration and Exploitation: Achieving an optimal balance between exploration (searching a wide range of solutions) and exploitation (refining promising solutions) is crucial. Research could focus on methods to navigate this balance more effectively, considering the complex and dynamic nature of optimisation problems.
- Integrating FIA with Other Metaheuristics: Explore combining the FIA with other metaheuristics, such as PSO algorithm, to assess the added benefits of FIA. PSO's strong exploration capabilities could be complemented by FIA's mechanisms, particularly in environments where exploitation is challenging. This could address issues in PSO related to convergence in complex, multimodal landscapes.

The future works related to Section 5.5 and chapter 4 are,

- Enhancing R&S Efficiency with Preprocessing Techniques: Explore the use of preprocessing techniques that account for problem-specific attributes to enhance the efficiency of the R&S algorithm for solving RCPSP and MRCPSP. Different schemes for relaxing the execution order of tasks could be investigated, particularly for projects where RCPSPs exhibit varying characteristics.
- Dynamic Adjustment of Time Windows: Investigate dynamically adjusting the duration of time windows in the R&S algorithm based on task dependencies rather than setting a fixed length. This adjustment could improve the algorithm's flexibility and responsiveness to changes in task scheduling.
- New Rules for Task Reordering: Develop new rules for relaxing the execution order of tasks. Preliminary results suggest that tasks with a higher amount of slack are crucial for reordering, which could lead to significant improvements in scheduling efficiency.
- Adaptation of R&S to Other Optimisation Problems: Examine the potential of the R&S algorithm's inherent structure for adaptation to solve a broader range of optimisation problems beyond RCPSP and MRCPSP, enhancing its applicability across different domains.
- Integration with Other Solution Methods: Explore the integration of the R&S algorithm with available heuristics, metaheuristics, and exact solvers during the solution phase to potentially boost performance and achieve more robust solutions.

The future works related to Chapter 5 are,

- Integration of Machine Learning Approaches: Explore the potential of incorporating machine learning techniques to automatically select policies for solving SRCPSP. This in-

tegration could significantly enhance the flexibility and adaptability of the approach, enabling it to handle a diverse range of problems without manual intervention efficiently.

- Development of Priority Rules and Decision-Making Strategies: Investigate different priority rules and design more efficient decision-making strategies to improve the approach's effectiveness. Optimising these strategies could lead to better performance and more optimal solutions in practical applications.
- Extension to Uncertain Resource Environments: Extend the current approach to tackle project scheduling problems characterized by various uncertainties, such as fluctuating resource availability. This could involve developing methods that are robust to changes and capable of adapting to unexpected variations in project conditions.
- Addressing Multi-Mode Instances: Explore the extension of the approach to address multi-mode instances of project scheduling, which involve multiple possible modes for task execution. This research could provide more realistic and practical scheduling solutions considering different execution scenarios and constraints.

Bibliography

[1] A. Etminaniesfahani, A. Ghanbarzadeh, and Z. Marashi, "Fibonacci indicator algorithm: A novel tool for complex optimization problems," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 1–9, 2018, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2018.04.012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197618300915.

[2] J. Blazewicz, J. Lenstra, and A. Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983, ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(83)90012-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0166218X83900124.

[3] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness* (Mathematical Sciences Series). Freeman, 1979.

[4] N. Shukla, A. Choudhary, P. Prakash, K. Fernandes, and M. Tiwari, "Algorithm portfolios for logistics optimization considering stochastic demands and mobility allowance," *International Journal of Production Economics*, vol. 141, no. 1, pp. 146–166, 2013, Metaheuristics for manufacturing scheduling and logistics problems, ISSN: 0925-5273.

[5] W. Herroelen and R. Leus, "Identification and illumination of popular misconceptions about project scheduling and time buffering in a resource-constrained environment," *Journal of the Operational Research Society*, vol. 56, no. 1, pp. 102–109, 2005.

[6] A. Pritsker, L. J. Waiters, and P. Wolfe, "Multiproject scheduling with limited resources: A zero-one programming approach," *Management Science*, vol. 16, pp. 93–108, 1969.

[7] J. Liu and M. Lu, "Optimization on supply-constrained module assembly process," cited By 2, 2017, pp. 813–820. DOI: 10.24928/2017/0104.

[8] C. Alford, M. Brazil, and D. H. Lee, "Optimisation in underground mining," in *Handbook Of Operations Research In Natural Resources*. Boston, MA: SpringerUS, 2007, pp. 561–577, ISBN: 978-0-387-71815-6. DOI: 10.1007/978-0-387-71815-6_30. [Online]. Available: https://doi.org/10.1007/978-0-387-71815-6_30.

[9] E. Demeulemeester and W. Herroelen, "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem," *Management Science*, vol. 38, no. 12, pp. 1803–1818, 1992.

[10] J. Blazewicz, J. Lenstra, and A. Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983, ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(83)90012-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0166218X83900124.

[11] W. Herroelen, B. De Reyck, and E. Demeulemeester, "Resource-constrained project scheduling: A survey of recent developments," *Computers & Operations Research*, vol. 25, no. 4, pp. 279–302, 1998, ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-

0548(97)00055-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054897000555.

[12] H. Chen, G. Ding, S. Qin, and J. Zhang, "A hyper-heuristic based ensemble genetic programming approach for stochastic resource constrained project scheduling problem," *Expert Systems with Applications*, vol. 167, p. 114174, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.114174. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420309118.

[13] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *European Journal of Operational Research*, vol. 174, no. 1, pp. 23–37, 2006, ISSN: 0377-2217.

[14] Laurent Perron and Vincent Furnon, *Or-tools version 7.2*, Google, 2019. [Online]. Available: https://developers.google.com/optimization/.

[15] I. I. CPLEX, *Version 12.8.0*, Armonk, New York, U.S., 2017.

[16] K. Ziarati, R. Akbari, and V. Zeighami, "On the performance of bee algorithms for resource-constrained project scheduling problem," *Applied Soft Computing*, vol. 11, no. 4, pp. 3720–3733, 2011, ISSN: 1568-4946.

[17] A. Etminaniesfahani, H. Gu, and A. Salehipour, "Abfia: A hybrid algorithm based on artificial bee colony and fibonacci indicator algorithm," *Journal of Computational Science*, vol. 61, p. 101651, 2022, ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2022.101651. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877750322000679.

[18] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, pp. 1–14, Nov. 2010. DOI: 10.1016/j.ejor.2009.11.005.

[19] S. Hartmann and D. Briskorn, "An updated survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 297, no. 1, pp. 1–14, 2022, ISSN: 0377-2217.

[20] F. F. Boctor, "Heuristics for scheduling projects with resource restrictions and several resource-duration modes," *International Journal of Production Research*, vol. 31, no. 11, pp. 2547–2558, 1993. DOI: 10.1080/00207549308956882.

[21] W. Herroelen, E. Demeulemeester, and B. Reyck, "A classification scheme for project scheduling problems," *Katholieke Universiteit Leuven, Open Access publications from Katholieke Universiteit Leuven*, pp. 1–26, Jan. 1997.

[22] J. Cheng, J. Fowler, K. Kempf, and S. Mason, "Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting," *Computers & Operations Research*, vol. 53, pp. 275–287, 2015, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2014.04.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054814001099.

[23] R. Kolisch and A. Sprecher, "Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program," *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(96)00170-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221796001701.

[24] V. Van Peteghem and M. Vanhoucke, "An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances," *European Journal of Operational Research*, vol. 235, no. 1, pp. 62–72, 2014, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2013.10.012.

[25] R. K. Chakrabortty, A. Abbasi, and M. J. Ryan, "Multi-mode resource-constrained project scheduling using modified variable neighborhood search heuristic," *International Transactions in Operational Research*, vol. 27, no. 1, pp. 138–167, 2020. DOI: https://doi.org/10.1111/itor.12644.

[26] M. M. Ahmadian and A. Salehipour, "Heuristics for flights arrival scheduling at airports," *International Transactions in Operational Research*, vol. 29, no. 4, pp. 2316–2345, 2022. DOI: https://doi.org/10.1111/itor.12901.

[27] A. Etminaniesfahani, H. Gu, L. M. Naeni, and A. Salehipour, "An efficient relax-and-solve method for the multi-mode resource constrained project scheduling problem," *Annals of Operations Research*, 2023.

[28] W. Herroelen and R. Leus, "Project scheduling under uncertainty: Survey and research potentials," *European Journal of Operational Research*, vol. 165, no. 2, pp. 289–306, 2005, Project Management and Scheduling, ISSN: 0377-2217.

[29] Q. Yuan and Y. Polychronakis, "The development of a robust resource constrained project scheduling framework," *International Journal of Project Organisation and Management*, vol. 4, pp. 339–367, 2012.

[30] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. II*, 3rd. Athena Scientific, 2007, ISBN: 1886529302.

[31] H. Li and N. K. Womer, "Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming," *European Journal of Operational Research*, vol. 246, no. 1, pp. 20–33, 2015, ISSN: 0377-2217.

[32] R. Kolisch, "Efficient priority rules for the resource-constrained project scheduling problem," *Journal of Operations Management*, vol. 14, no. 3, pp. 179–192, 1996, ISSN: 0272-6963.

[33] T. Browning and A. Yassine, "Resource-constrained multi-project scheduling: Priority rule performance revisited," *International Journal of Production Economics*, vol. 126, pp. 212–228, Aug. 2010.

[34] F. Stork, "Branch-and-bound algorithms for stochastic resource-constrained project scheduling," *Technical rep*, pp. 702–2000, 2000.

[35] A. Etminaniesfahani., H. Gu., and A. Salehipour., "An efficient relax-and-solve algorithm for the resource-constrained project scheduling problem," in *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - ICORES,*, INSTICC, SciTePress, 2022, pp. 271–277.

[36] A. Etminaniesfahani, H. Gu, L. Naeni, and A. Salehipour, "A forward–backward relax-and-solve algorithm for the resource-constrained project scheduling problem," *SN Computer Science*, vol. 4, pp. 104–114, Dec. 2022.

[37] R. Kolisch and A. Sprecher, "Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program," *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(96)00170-1.

[38] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986, Applications of Integer Programming, ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(86)90048-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0305054886900481.

[39] M. Bellare and P. Rogaway, "The complexity of approximating a nonlinear program," *Mathematical Programming*, vol. 69, pp. 429–441, Jul. 1995. DOI: 10.1007/BF01585569. [Online]. Available: https://doi.org/10.1007/BF01585569.

[40] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of Statistical Physics*, vol. 34, no. 5, pp. 975–986, 1984, Applications of Integer Programming, ISSN: 1572-9613. DOI: 10.1007/BF01009452. [Online]. Available: https://doi.org/10.1007/BF01009452.

[41] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[42] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014, ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2013.12.007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965997813001853.

[43] M. Gendreau and J.-Y. Potvin, Eds., *Handbook of Metaheuristics* (International Series in Operations Research and Management Science 978-1-4419-1665-5), 2nd ed. Springer, Mar. 2013. DOI: 10.1007/978-1-4419-1665-5.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983, ISSN: 0036-8075. DOI: 10.1126/science.220.4598.671.

[45] B. Webster and P. Bernhard, "A local search optimization algorithm based on natural principles of gravitation," in *IKE*, 2003, pp. 671–680.

[46] H. Shah-Hosseini, "Principal components analysis by the galaxy-based search algorithm: A novel metaheuristic for continuous optimisation," *International Journal of Computational Science and Engineering*, vol. 6, pp. 132–140, Jul. 2011. DOI: 10.1504/IJCSE.2011.041221.

[47] B. Vahidi and A. Foroughi Nematolahi, "Physical and physic-chemical based optimization methods: A review," *Journal of Soft Computing in Civil Engineering*, vol. 3, no. 4, pp. 12–27, 2019, ISSN: 2588-2872. DOI: $10.22115/scce.2020.214959.1161$.

[48] B. Alatas and U. Can, "Physics based metaheuristic optimization algorithms for global optimization," *American Journal of Information Science and Computer Engineering*, pp. 1–14, Jan. 2015.

[49] J. H. Holland, "Genetic algorithms," *Scientific American*, Jul. 1992.

[50] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 23, Jan. 1995.

[51] I. Rechenberg, "Evolutionsstrategien," in *Simulationsmethoden in der Medizin und Biologie*, B. Schneider and U. Ranft, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 83–114, ISBN: 978-3-642-81283-5.

[52] D. Vasiljevic, "Comparison of optimization algorithms," in *Classical and Evolutionary Algorithms in the Optimization of Optical Systems*. Boston, MA: Springer US, 2002, pp. 83–88, ISBN: 978 − 1 − 4615 − 1051 − 2. DOI: $10.1007/978-1-4615-1051-2_5$. [Online]. Available: $https://doi.org/10.1007/978-1-4615-1051-2_5$.

[53] T. Tušar and B. Filipič, "Differential evolution versus genetic algorithms in multiobjective optimization," in *Evolutionary Multi-Criterion Optimization*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 257–271, ISBN: 978-3-540-70928-2.

[54] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical Computer Science*, vol. 344, no. 2, pp. 243–278, 2005, ISSN: 0304-3975. DOI: $https://doi.org/10.1016/j.tcs.2005.05.020$. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397505003798.

[55] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39–43.

[56] D. Karaboga, "An idea based on honey bee swarm for numerical optimization, technical report - tr06," *Technical Report, Erciyes University*, Jan. 2005.

[57] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization." artificial life 5, 137-172," *Artificial Life*, vol. 5, pp. 137–172, Apr. 1999. DOI: 10.1162/106454699568728.

[58] V.Selvi and R.Umarani, "Comparative analysis of ant colony and particle swarm optimization techniques," *International Journal of Computer Applications*, vol. 5, pp. 1–6, Aug. 2010. DOI: 10.5120/908-1286.

[59] M. El-Abd, "A hybrid abc-spso algorithm for continuous function optimization," in *2011 IEEE Symposium on Swarm Intelligence*, Apr. 2011, pp. 1–6. DOI: 10.1109/SIS.2011.5952576.

[60] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Computing and Applications*, vol. 27, pp. 1053–1073, 2015.

[61] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, and S. M. Mirjalili, "Salp swarm algorithm: A bio-inspired optimizer for engineering design problems," *Advances in Engineering Software*, vol. 114, pp. 163–191, 2017, ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2017.07.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965997816307736.

[62] S. Mirjalili, "The ant lion optimizer," *Advances in Engineering Software*, vol. 83, pp. 80–98, 2015, ISSN: 0965-9978.

[63] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016, ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2016.01.008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965997816300163.

[64] K. C. Kilic Haydar. Yuzgec Ugur., "A novel improved antlion optimizer algorithm and its comparative performance," *Neural Computing and Applications*, vol. 32, pp. 3803–3824, 2020.

[65] S. Mostafa Bozorgi and S. Yazdani, "Iwoa: An improved whale optimization algorithm for optimization problems," *Journal of Computational Design and Engineering*, vol. 6, no. 3, pp. 243–259, 2019, ISSN: 2288-4300. DOI: https://doi.org/10.1016/j.jcde.2019.02.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2288430018301994.

[66] L. S.-X. Wang Jie-Sheng., "An Improved Grey Wolf Optimizer Based on Differential Evolution and Elimination Mechanism," *Scientific Reports*, vol. 9, pp. 71–81, 2019.

[67] Z. W. Geem, J. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *Simulation*, vol. 76, pp. 60–68, 2001.

[68] F. Glover and M. Laguna, *Tabu Search*. USA: Kluwer Academic Publishers, 1997, ISBN: 079239965X.

[69] C. Voudouris and E. Tsang, "Guided local search and its application to the travelling salesman problem," *European Journal of Operational Research*, vol. 113, no. 2, pp. 469–499, 1999.

[70] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[71] P. F. Pai, W. Sun, and X. Chang, "Comparative analysis of ant colony and particle swarm optimization techniques," *Journal of Electrical and Computer Engineering*, vol. 2015, pp. 712–753, Feb. 2015.

[72] C. M. Rahman and T. A. Rashid, "A new evolutionary algorithm: Learner performance based behavior algorithm," *Egyptian Informatics Journal*, vol. 22, no. 2, pp. 213–223, 2021, ISSN: 1110-8665. DOI: https://doi.org/10.1016/j.eij.2020.08.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110866520301419.

[73] "Variable neighborhood search: The power of change and simplicity," *Computers & Operations Research*, vol. 155, p. 106 221, 2023, ISSN: 0305-0548.

[74] Z. Li, W. Wang, Y. Yan, and Z. Li, "Ps–abc: A hybrid algorithm based on particle swarm and artificial bee colony for high-dimensional optimization problems," *Expert Systems with Applications*, vol. 42, no. 22, pp. 8881–8895, 2015, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2015.07.043. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417415005035.

[75] W. Gao, S. Liu, and L. Huang, "A global best artificial bee colony algorithm for global optimization," *Journal of Computational and Applied Mathematics*, vol. 236, no. 11, pp. 2741–2753, 2012, ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2012.01.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042712000246.

[76] K. Hussain, M. N. Mohd Salleh, S. Cheng, Y. Shi, and R. Naseem, "Artificial bee colony algorithm: A component-wise analysis using diversity measurement," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 7, pp. 794–808, 2020, ISSN: 1319-1578.

[77] S. Imamura, T. Kaihara, N. Fujii, D. Kokuryo, and A. Kitamura, "Characteristic analysis of artificial bee colony algorithm with network-structure," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 21, no. 3, pp. 496–506, 2017.

[78] D. Yazdani and M. R. Meybodi, "A novel artificial bee colony algorithm for global optimization," in *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, 2014, pp. 443–448. DOI: 10.1109/ICCKE.2014.6993393.

[79] S. Anuar, A. Selamat, and R. Sallehuddin, "A modified scout bee for artificial bee colony algorithm and its performance on optimization problems," *Journal of King Saud University - Computer and Information Sciences*, vol. 28, no. 4, pp. 395–406, 2016, ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2016.03.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1319157816300039.

[80] H. Hakli and M. Kıran, "An improved artificial bee colony algorithm for balancing local and global search behaviors in continuous optimization," *International Journal of Machine Learning and Cybernetics*, vol. 11, pp. 2051–2076, Sep. 2020. DOI: 10.1007/s13042-020-01094-7.

[81] X. He, W. Wang, J. Jiang, and L. Xu, "An improved artificial bee colony algorithm and its application to multi-objective optimal power flow," *Energies*, vol. 8, pp. 2412–2437, Apr. 2015. DOI: 10.3390/en8042412.

[82] Y. Celik, "An enhanced artificial bee colony algorithm based on fitness weighted search strategy," *Automatika*, vol. 62, no. 3, pp. 300–310, 2021. DOI: 10.1080/00051144.2021.1938477. eprint: https://doi.org/10.1080/00051144.2021.1938477. [Online]. Available: https://doi.org/10.1080/00051144.2021.1938477.

[83] G. Zhu and S. Kwong, "Gbest-guided artificial bee colony algorithm for numerical function optimization," *Applied Mathematics and Computation*, vol. 217, no. 7, pp. 3166–

3173, 2010, ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2010.08.049. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300310009136.

[84] W.-f. Gao and S.-y. Liu, "A modified artificial bee colony algorithm," *Computers & Operations Research*, vol. 39, no. 3, pp. 687–697, 2012, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2011.06.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054811001699.

[85] L.-C. Lien and M.-Y. Cheng, "A hybrid swarm intelligence based particle-bee algorithm for construction site layout optimization," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9642–9650, 2012, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2012.02.134. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417412003971.

[86] N. Imanian, M. E. Shiri, and P. Moradi, "Velocity based artificial bee colony algorithm for high dimensional continuous optimization problems," *Engineering Applications of Artificial Intelligence*, vol. 36, pp. 148–163, 2014, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2014.07.012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197614001808.

[87] S. Kumar, V. Kumar Sharma, and R. Kumari, "A Novel Hybrid Crossover based Artificial Bee Colony Algorithm for Optimization Problem," *International Journal of Computer Applications*, vol. 82, no. 8, pp. 18–25, Nov. 2013. DOI: 10.5120/14136-2266. arXiv: 1407.5574 [cs.AI].

[88] S.-M. Chen, A. Sarosh, and Y.-F. Dong, "Simulated annealing based artificial bee colony algorithm for global numerical optimization," *Applied Mathematics and Computation*, vol. 219, no. 8, pp. 3575–3589, 2012, ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2012.09.052. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300312009514.

[89] E. López-Rubio, A. Yurtkuran, and E. Emel, "An enhanced artificial bee colony algorithm with solution acceptance rule and probabilistic multisearch," *Computational Intelligence and Neuroscience*, vol. 2016, 2016, ISSN: 1687-5265. DOI: 10.1155/2016/8085953. [Online]. Available: https://doi.org/10.1155/2016/8085953.

[90] H. F. Rahman, R. K. Chakrabortty, and M. J. Ryan, "Memetic algorithm for solving resource constrained project scheduling problems," *Automation in Construction*, vol. 111, p. 103 052, 2020, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2019.103052. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580519302766.

[91] E. W. Davis and J. H. Patterson, "A comparison of heuristic and optimum solutions in resource-constrained project scheduling," *Management Science*, vol. 21, no. 8, pp. 944–955, 1975.

[92] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[93] R. Kolisch and S. Hartmann, "Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis," in *Project Scheduling: Recent Models, Algorithms and Applications*, J. Weglarz, Ed. Boston, MA: Springer US, 1999, pp. 147–178.

[94] K. Li and R. Willis, "An iterative scheduling technique for resource-constrained project scheduling," *European Journal of Operational Research*, vol. 56, no. 3, pp. 370–379, 1992, ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(92)90320-

9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221792903209.

[95] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *European Journal of Operational Research*, vol. 174, no. 1, pp. 23–37, 2006, ISSN: 0377-2217.

[96] R. Pellerin, N. Perrier, and F. Berthaut, "A survey of hybrid metaheuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 280, no. 2, pp. 395–416, 2020, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2019.01.063. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221719300980.

[97] W. Guo, M. Vanhoucke, J. Coelho, and J. Luo, "Automatic detection of the best performing priority rule for the resource-constrained project scheduling problem," *Expert Systems with Applications*, vol. 167, p. 114 116, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.114116. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420308654.

[98] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983, ISSN: 0036-8075. DOI: 10.1126/science.220.4598.671.

[99] F. Glover and M. Laguna, *Tabu Search*. USA: Kluwer Academic Publishers, 1997, ISBN: 079239965X.

[100] J. H. Holland, "Genetic algorithms," *Scientific American*, Jul. 1992.

[101] A. Agarwal, S. Colak, and S. Erenguc, "A neurogenetic approach for the resource-constrained project scheduling problem," *Computers & Operations Research*, vol. 38, no. 1, pp. 44–50, 2011, Project Management and Scheduling, ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2010.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054810000183.

[102] D. Debels and M. Vanhoucke, "A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem," *Operations Research*, vol. 55, pp. 457–469, Jun. 2007. DOI: 10.1287/opre.1060.0358.

[103] A. Lim, H. Ma, B. Rodrigues, S. Tan, and F. Xiao, "New meta-heuristics for the resource-constrained project scheduling problem," *Flexible Services and Manufacturing Journal*, vol. 25, pp. 48–73, Jun. 2011. DOI: 10.1007/s10696-011-9133-0.

[104] M. Cervantes, A. Lova, P. Tormos, and F. Barber, "A dynamic population steady-state genetic algorithm for the resource-constrained project scheduling problem," in *New Frontiers in Applied Artificial Intelligence*, N. T. Nguyen, L. Borzemski, A. Grzech, and M. Ali, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 611–620.

[105] I. Ismail and M. Barghash, "Diversity guided genetic algorithm to solve the resource constrained project scheduling problem," *Int. J. of Planning and Scheduling*, vol. 1, pp. 147–170, Jan. 2012. DOI: 10.1504/IJPS.2012.050125.

[106] J. Alcaraz, C. Maroto, and R. Ruiz, "Improving the performance of genetic algorithms for the rcps problem," *Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pp. 40–43, 2004, Cited By :39.

[107] H. Wang, T. Li, and D. Lin, "Efficient genetic algorithm for resource-constrained project scheduling problem," *Transactions of Tianjin University*, vol. 16, no. 5, pp. 376–382, 2010, ISSN: 1995-8196. DOI: 10.1007/s12209-010-1495-y.

[108] R. Zamani, "A competitive magnet-based genetic algorithm for solving the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 229, no. 2, pp. 552–559, 2013, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.

ejor.2013.03.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221713002130.

[109] V. Valls, F. Ballestín, and S. Quintanilla, "A hybrid genetic algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 185, pp. 495–508, Mar. 2008. DOI: 10.1016/j.ejor.2006.12.033.

[110] D. Debels and M. Vanhoucke, "A bi-population based genetic algorithm for the resource-constrained project scheduling problem," in *Computational Science and Its Applications – ICCSA 2005*, O. Gervasi, M. L. Gavrilova, V. Kumar, *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 378–387.

[111] G. J. Fernando, R. M. G. C., and M. J. J. M., "A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem," *Journal of Heuristics*, vol. 17, no. 5, pp. 467–486, 2011, ISSN: 1572-9397. DOI: 10.1007/s10732-010-9142-2.

[112] S. Chand, H. K. Singh, and T. Ray, "A heuristic algorithm for solving resource constrained project scheduling problems," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2017, pp. 225–232. DOI: 10.1109/CEC.2017.7969317.

[113] L. Deng, V. Lin, and M. Chen, "Hybrid ant colony optimization for the resource-constrained project scheduling problem," *Journal of Systems Engineering and Electronics*, vol. 21, no. 1, pp. 67–71, 2010. DOI: 10.3969/j.issn.1004-4132.2010.01.012.

[114] D. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm," *Technical Note, Manufacturing Engineering Centre, Cardiff University, UK*, pp. 44–48, 2005.

[115] R. Akbari, A. Mohammadi, and K. Ziarati, "A novel bee swarm optimization algorithm for numerical function optimization," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, no. 10, pp. 3142–3155, 2010, ISSN: 1007-5704.

[116] D. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm technical note," *Manufacturing Engineering Centre, Cardiff University, UK*, pp. 1–57, Sep. 2005.

[117] "A novel bee swarm optimization algorithm for numerical function optimization," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, no. 10, pp. 3142–3155, 2010, ISSN: 1007-5704. DOI: https://doi.org/10.1016/j.cnsns.2009.11.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S100757040900570X.

[118] F. Berthaut, R. Pellerin, A. Hajji, and N. Perrier, "A path relinking-based scatter search for the resource-constrained project scheduling problem," *International Journal of Project Organisation and Management*, vol. 10, no. 1, pp. 1–36, 2018. DOI: 10.1504/IJPOM.2018.090372.

[119] H. Gu, A. Schutt, and P. J. Stuckey, "A lagrangian relaxation based forward-backward improvement heuristic for maximising the net present value of resource-constrained projects," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, C. Gomes and M. Sellmann, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 340–346, ISBN: 978-3-642-38171-3.

[120] A. Etminaniesfahani., H. Gu., and A. Salehipour., "An efficient relax-and-solve algorithm for the resource-constrained project scheduling problem," in *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - ICORES 2022*, 2022, pp. 271–277, ISBN: 978-989-758-548-7. DOI: 10.5220/0010772400003117.

[121] N. Nouri, S. Krichen, T. Ladhari, and P. Fatimah, "A discrete artificial bee colony algorithm for resource-constrained project scheduling problem," in *2013 5th International*

Conference on Modeling, Simulation and Applied Optimization (ICMSAO), vol. 0, 2013, pp. 1–6.

[122] R. Pellegrini, A. Serani, G. Liuzzi, F. Rinaldi, S. Lucidi, and M. Diez, "Hybridization of multi-objective deterministic particle swarm with derivative-free local searches," *Mathematics*, vol. 8, Apr. 2020. DOI: 10.3390/math8040546.

[123] R. Zamani, "An accelerating two-layer anchor search with application to the resource-constrained project scheduling problem," vol. 14, no. 6, pp. 975–984, 2010.

[124] A. Sprecher and A. Drexl, "Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm1supported by the deutsche forschungsgemeinschaft.1," *European Journal of Operational Research*, vol. 107, no. 2, pp. 431–450, 1998, ISSN: 0377-2217.

[125] V. Zeighami, R. Akbari, I. Akbari, and Y. Biletskiy, "An abc-genetic method to solve resource constrained project scheduling problem," *Artificial Intelligence Research (AIR) Journal, SCIEDU, Canada*, vol. 1, no. 2, pp. 185–197, 2012.

[126] M. M. Ahmadian, A. Salehipour, and T. Cheng, "A meta-heuristic to solve the just-in-time job-shop scheduling problem," *European Journal of Operational Research*, vol. 288, no. 1, pp. 14–29, 2021.

[127] M. M. Ahmadian, A. Salehipour, and M. Kovalyov, "An efficient relax-and-solve heuristic for open-shop scheduling problem to minimize total weighted earliness-tardiness," *Available at SSRN 3601396*, 2020.

[128] A. Salehipour, "A heuristic algorithm for the aircraft landing problem," in *22nd International Congress on Modelling and Simulation*, Modelling, Simulation Society of Australia, and New Zealand Inc.(MSSANZ), 2017.

[129] A. Salehipour, M. Ahmadian, and D. Oron, "Efficient and simple heuristics for the aircraft landing problem," in *Matheuristic 2018 International Conference*, 2018.

[130] V. Maniezzo, M. Boschetti, and T. Stützle, *Matheuristics: Algorithms and Implementations*. Jan. 2021, ISBN: 978-3-030-70276-2. DOI: 10.1007/978-3-030-70277-9.

[131] P. Laborie, "An update on the comparison of mip, cp and hybrid approaches for mixed resource allocation and scheduling," in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Cham: Springer International Publishing, 2018, pp. 403–411, ISBN: 978-3-319-93031-2.

[132] C. Maleck, G. Nieke, K. Bock, D. Pabst, and M. Stehli, "A comparison of an cp and mip approach for scheduling jobs in production areas with time constraints and uncertainties," in *2018 Winter Simulation Conference (WSC)*, 2018, pp. 3526–3537. DOI: 10.1109/WSC.2018.8632404.

[133] E. Kelareva, S. Brand, P. Kilby, S. Thiebaux, and M. Wallace, "Cp and mip methods for ship scheduling with time-varying draft," *ICAPS 2012 - Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, Jan. 2012.

[134] O. Liess and P. Michelon, "A constraint programming approach for the resource-constrained project scheduling problem," *Annals of Operations Research*, vol. 157, no. 1, pp. 25–36, 2008.

[135] A. Schutt, T. Feydy, P. Stuckey, and M. Wallace, "Explaining the cumulative propagator," *Constraints*, vol. 16, pp. 250–282, Jul. 2011. DOI: 10.1007/s10601-010-9103-2.

[136] A. Schutt, T. Feydy, P. J. Stuckey, and M. G. Wallace, "A satisfiability solving approach," in *Handbook on Project Management and Scheduling Vol.1*. Cham: Springer International Publishing, 2015, pp. 135–160, ISBN: 978-3-319-05443-8. DOI: 10.1007/978-3-319-05443-8_7. [Online]. Available: https://doi.org/10.1007/978-3-319-05443-8_7.

[137] S. Kreter, A. Schutt, and P. Stuckey, "Using constraint programming for solving rcpsp/max-cal," *Constraints*, vol. 22, Jul. 2017. DOI: 10.1007/s10601-016-9266-6.

[138] N. Absi and W. van den Heuvel, "Worst case analysis of relax and fix heuristics for lot-sizing problems," *European Journal of Operational Research*, vol. 279, no. 2, pp. 449–458, 2019, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2019.06.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221719304850.

[139] L. F. Escudero and C. P. Romero, "On solving a large-scale problem on facility location and customer assignment with interaction costs along a time horizon," *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, vol. 25, no. 3, pp. 601–622, 2017. DOI: 10.1007/s11750-017-0461-4.

[140] S. Helber and F. Sahling, "A fix-and-optimize approach for the multi-level capacitated lot sizing problem," *International Journal of Production Economics*, vol. 123, no. 2, pp. 247–256, 2010.

[141] A. Etminaniesfahani., H. Gu., and A. Salehipour., "An efficient relax-and-solve algorithm for the resource-constrained project scheduling problem," pp. 271–277, 2022, ISSN: 2184-4372. DOI: 10.5220/0010772400003117.

[142] F. B. Talbot, "Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case," *Management Science*, vol. 28, no. 10, pp. 1197–1210, 1982. [Online]. Available: https://EconPapers.repec.org/RePEc:inm:ormnsc:v:28:y:1982:i:10:p:1197-1210.

[143] V. V. Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, 2010, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2009.03.034. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037722170900191X.

[144] R. Słowiński, "Two approaches to problems of resource allocation among project activities — a comparative study," *Journal of the Operational Research Society*, vol. 31, no. 8, pp. 711–723, 1980. DOI: 10.1057/jors.1980.134.

[145] Y. Li, H. Li, and T. Yang, "Solving mrcpsp by constraint programming," in *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, vol. 1, 2011, pp. 312–315. DOI: 10.1109/ICM.2011.251.

[146] A. Schnell and R. F. Hartl, "On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs," *Operations Research Perspectives*, vol. 4, pp. 1–11, 2017, ISSN: 2214-7160. DOI: https://doi.org/10.1016/j.orp.2017.01.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214716016300458.

[147] M. Speranza and C. Vercellis, "Hierarchical models for multi-project planning and scheduling," *European Journal of Operational Research*, vol. 64, no. 2, pp. 312–325, 1993, Project Management and Scheduling, ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(93)90185-P.

[148] A. Sprecher and A. Drexl, "Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm1supported by the deutsche forschungsgemeinschaft.1," *European Journal of Operational Research*, vol. 107, no. 2, pp. 431–450, 1998, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(97)00348-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221797003482.

[149] A. Sprecher, S. Hartmann, and A. Drexl, "An exact algorithm for project scheduling with multiple modes," *Operations-Research-Spektrum*, vol. 19, pp. 195–203, 1997.

[150] G. Zhu, J. F. Bard, and G. Yu, "A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem," *INFORMS Journal on Computing*, vol. 18, no. 3, pp. 377–390, 2006. DOI: 10.1287/ijoc.1040.0121. [Online]. Available: https://doi.org/10.1287/ijoc.1040.0121.

[151] F. Zaman, S. Elsayed, R. Sarker, and D. Essam, "Hybrid evolutionary algorithm for large-scale project scheduling problems," *Computers & Industrial Engineering*, vol. 146, p. 106 567, 2020, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106567. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220303016.

[152] F. F. Boctor, "A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes," *European Journal of Operational Research*, vol. 90, no. 2, pp. 349–361, 1996, ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(95)00359-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221795003592.

[153] A. Lova, M. Tormos, and F. Barber, "Multi-mode resource constrained project scheduling: Scheduling schemes, priority rules and mode selection rules," *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, vol. 10, pp. 69–86, Dec. 2006. DOI: 10.4114/ia.v10i30.947.

[154] R. Slowinski, B. Soniewicki, and J. Weglarz, "Dss for multiobjective project scheduling," *European Journal of Operational Research*, vol. 79, no. 2, pp. 220–229, 1994, ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(94)90353-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221794903530.

[155] F. F. Boctor, "Resource-constrained project scheduling by simulated annealing," *International Journal of Production Research*, vol. 34, no. 8, pp. 2335–2351, 1996. DOI: 10.1080/00207549608905028. [Online]. Available: https://doi.org/10.1080/00207549608905028.

[156] J. Jozefowska, M. Mika, R. Rozycki, G. Waligora, and J. Weglarz, "Simulated annealing for multimode resource-constrained project scheduling," *Annals of Operations Research - Annals OR*, vol. 102, pp. 137–155, Feb. 2001. DOI: 10.1023/A:1010954031930.

[157] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268–281, 2003, Sequencing and Scheduling, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(02)00761-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221702007610.

[158] K. Nonobe and T. Ibaraki, "Formulation and tabu search algorithm for the resource constrained project scheduling problem," in *Essays and Surveys in Metaheuristics*, 2002.

[159] M. Ranjbar, B. De Reyck, and F. Kianfar, "A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling," *European Journal of Operational Research*, vol. 193, no. 1, pp. 35–48, 2009, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2007.10.042. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221707010661.

[160] A. E. F. Muritiba, C. D. Rodrigues, and F. A. da Costa, "A path-relinking algorithm for the multi-mode resource-constrained project scheduling problem," *Computers & Operations Research*, vol. 92, pp. 145–154, 2018, ISSN: 0305-0548. DOI: https://doi.org/

10.1016/j.cor.2018.01.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054818300017.

[161] K. Fleszar and K. S. Hindi, "Solving the resource-constrained project scheduling problem by a variable neighbourhood search," *European Journal of Operational Research*, vol. 155, no. 2, pp. 402–413, 2004, Financial Risk in Open Economies, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(02)00884-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221702008846.

[162] M. Mori and C. C. Tseng, "A genetic algorithm for multi-mode resource constrained project scheduling problem," *European Journal of Operational Research*, vol. 100, no. 1, pp. 134–141, 1997, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(96)00180-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221796001804.

[163] L. Ozdamar, "A genetic algorithm approach to a general category project scheduling problem," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 29, pp. 44–59, Mar. 1999. DOI: 10.1109/5326.740669.

[164] S. Hartmann, "Project scheduling with multiple modes: A genetic algorithm," *Annals of Operations Research*, vol. 102, no. 1, pp. 111–135, 2001, ISSN: 1572-9338. DOI: 10.1023/A:1010902015091. [Online]. Available: https://doi.org/10.1023/A:1010902015091.

[165] J. Alcaraz, C. Maroto, and R. Ruiz, "Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms," *Journal of the Operational Research Society*, vol. 54, no. 6, pp. 614–626, 2003. DOI: 10.1057/palgrave.jors.2601563. [Online]. Available: https://doi.org/10.1057/palgrave.jors.2601563.

[166] M. R. Afshar, V. Shahhosseini, and M. H. Sebt, "A genetic algorithm with a new local search method for solving the multimode resource-constrained project scheduling problem," *International Journal of Construction Management*, vol. 22, no. 3, pp. 357–365, 2022. DOI: 10.1080/15623599.2019.1623992. [Online]. Available: https://doi.org/10.1080/15623599.2019.1623992.

[167] A. Lova, P. Tormos, M. Cervantes, and F. Barber, "An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes," *International Journal of Production Economics*, vol. 117, no. 2, pp. 302–316, 2009, ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2008.11.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925527308003654.

[168] M. Ayodele, J. McCall, and O. Regnier-Coudert, "Bpga-eda for the multi-mode resource constrained project scheduling problem," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 3417–3424. DOI: 10.1109/CEC.2016.7744222.

[169] P. Tormos and A. Lova, "A competitive heuristic solution technique for resource-constrained project scheduling," *Annals of Operations Research*, vol. 102, no. 1, pp. 65–81, 2001.

[170] P. Larranaga and J. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Jan. 2002. DOI: 10.1007/978-1-4615-1539-5.

[171] H. Zhang, C. Tam, and H. Li, "Multimode project scheduling based on particle swarm optimization," *Computer-Aided Civil and Infrastructure Engineering*, vol. 21, no. 2, pp. 93–103, 2006, ISSN: 1093-9687. DOI: 10.1111/j.1467-8667.2005.00420.x.

[172] B. Jarboui, N. Damak, P. Siarry, and A. Rebai, "A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems," *Applied Mathematics and Computation*, vol. 195, no. 1, pp. 299–308, 2008, ISSN: 0096-3003. DOI: https://doi.org/10.1016/j.amc.2007.04.096. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S009630030700553X.

[173] L. Zhang, Y. Luo, and Y. Zhang, "Hybrid particle swarm and differential evolution algorithm for solving multimode resource-constrained project scheduling problem," *Journal of Control Science and Engineering*, vol. 2015, Oct. 2015. DOI: 10.1155/2015/923791.

[174] H. Li and H. Zhang, "Ant colony optimization-based multi-mode scheduling under renewable and nonrenewable resource constraints," *Automation in Construction*, vol. 35, pp. 431–438, 2013, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2013.05.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580513000976.

[175] S. Asta, D. Karapetyan, A. Kheiri, E. Özcan, and A. J. Parkes, "Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem," *Information Sciences*, vol. 373, pp. 476–498, 2016, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2016.09.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025516307502.

[176] A. Kheiri, E. Özcan, and A. J. Parkes, "Hysst: Hyper-heuristic search strategies and timetabling," 2012.

[177] E. Ozcan and A. Kheiri, "A hyper-heuristic based on random gradient, greedy and dominance," 2012, pp. 557–563. DOI: 10.1007/978-1-4471-2155-8-71.

[178] J. Blazewicz, J. Lenstra, and A. Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983, ISSN: 0166-218X.

[179] A. A. Fernandez, R. L. Armacost, and J. J. Pet-Edwards, "The role of the nonanticipativity constraint in commercial software for stochastic project scheduling," *Computers & Industrial Engineering*, vol. 31, no. 1, pp. 233–236, 1996, Proceedings of the 19th International Conference on Computers and Industrial Engineering, ISSN: 0360-8352.

[180] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar, "Application of a technique for research and development program evaluation," *Operations Research*, vol. 7, no. 5, pp. 646–669, 1959.

[181] B. Dodin, "Determining the k most critical paths in PERT networks," *Operations Research*, vol. 32, no. 4, pp. 859–877, 1984.

[182] S. Elmaghraby, A. Ferreira, and L. Tavares, "Optimal start times under stochastic activity durations," *International Journal of Production Economics*, vol. 64, no. 1, pp. 153–164, 2000, ISSN: 0925-5273.

[183] W. Herroelen and R. Leus, "Robust and reactive project scheduling: A review and classification of procedures," *International Journal of Production Research*, vol. 42, no. 8, pp. 1599–1620, 2004.

[184] F. Deblaere, E. Demeulemeester, and W. Herroelen, "Proactive policies for the stochastic resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 214, no. 2, pp. 308–316, 2011, ISSN: 0377-2217.

[185] M. Brčić, M. Katić, and N. Hlupić, "Planning horizons based proactive rescheduling for stochastic resource-constrained project scheduling problems," *European Journal of Operational Research*, vol. 273, no. 1, pp. 58–66, 2019, ISSN: 0377-2217.

[186] S. Van de Vonder, F. Ballestín, E. Demeulemeester, and W. Herroelen, "Heuristic procedures for reactive project scheduling," *Computers & Industrial Engineering*, vol. 52, no. 1, pp. 11–28, 2007, ISSN: 0360-8352.

[187] R. K. Chakrabortty, H. F. Rahman, K. M. Haque, S. K. Paul, and M. J. Ryan, "An event-based reactive scheduling approach for the resource constrained project scheduling problem with unreliable resources," *Computers & Industrial Engineering*, vol. 151, p. 106 981, 2021, ISSN: 0360-8352.

[188] E. Demeulemeester, W. Herroelen, and R. Leus, "Proactive-reactive project scheduling," in *Resource-Constrained Project Scheduling*. John Wiley & Sons, Ltd, 2008, ch. 13, pp. 203–211, ISBN: 9780470611227.

[189] M. Davari and E. Demeulemeester, "The proactive and reactive resource-constrained project scheduling problem," *Journal of Scheduling*, vol. 22, Apr. 2019.

[190] Z. Chen, E. Demeulemeester, S. Bai, and Y. Guo, "Efficient priority rules for the stochastic resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 270, no. 3, pp. 957–967, 2018, ISSN: 0377-2217.

[191] S. Rostami, S. Creemers, and R. Leus, "New strategies for stochastic resource-constrained project scheduling," *Journal of Scheduling*, vol. 21, pp. 1–10, Jun. 2018.

[192] B. Ashtiani, R. Leus, and M.-B. Aryanezhad, "New competitive results for the stochastic resource-constrained project scheduling problem: Exploring the benefits of pre-processing," *J. Scheduling*, vol. 14, pp. 157–171, Apr. 2011.

[193] F. Zaman, S. Elsayed, R. Sarker, D. Essam, and C. A. Coello Coello, "An evolutionary approach for resource constrained project scheduling with uncertain changes," *Computers & Operations Research*, vol. 125, p. 105 104, 2021, ISSN: 0305-0548.

[194] Y. Zhou, J. Miao, B. Yan, and Z. Zhang, "Stochastic resource-constrained project scheduling problem with time varying weather conditions and an improved estimation of distribution algorithm," *Computers & Industrial Engineering*, vol. 157, p. 107 322, 2021, ISSN: 0360-8352.

[195] F. Ballestín, "When it is worthwhile to work with the stochastic RCPSP?" *J. Scheduling*, vol. 10, pp. 153–166, Jun. 2007.

[196] R. Kolisch, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation," *European Journal of Operational Research*, vol. 90, no. 2, pp. 320–333, 1996, ISSN: 0377-2217.

[197] G. Igelmund and F. J. Radermacher, "Preselective strategies for the optimization of stochastic project networks under resource constraints," *Networks*, vol. 13, no. 1, pp. 1–28, 1983.

[198] R. H. Möhring and F. Stork, "Linear preselective policies for stochastic project scheduling," vol. 52, no. 3, pp. 501–515, 2000.

[199] R. L. Graham, "Bounds for certain multiprocessing anomalies," *The Bell System Technical Journal*, vol. 45, no. 9, pp. 1563–1581, 1966.

[200] E. L. Demeulemeester and W. S. Herroelen, *Project scheduling: a research handbook*. Springer Science & Business Media, 2006, vol. 49.

[201] J. Choi, M. J. Realff, and J. H. Lee, "Dynamic programming in a heuristically confined state space: A stochastic resource-constrained project scheduling application," *Computers & Chemical Engineering*, vol. 28, no. 6, pp. 1039–1058, 2004, ISSN: 0098-1354.

[202] S. Creemers, "Minimizing the expected makespan of a project with stochastic activity durations under resource constraints," *Journal of Scheduling*, vol. 18, pp. 1–11, Jun. 2015.

[203] F. Xie, H. Li, and Z. Xu, "An approximate dynamic programming approach to project scheduling with uncertain resource availabilities," *Applied Mathematical Modelling*, vol. 97, pp. 226–243, 2021, ISSN: 0307-904X.

[204] R. Kolisch and A. Drexl, "Local search for nonpreemptive multi-mode resource-constrained project scheduling," *IIE Transactions*, vol. 29, no. 11, pp. 987–999, 1997. DOI: 10.1080/07408179708966417.

[205] A. Etminaniesfahani, H. Gu, L. Naeni, and A. Salehipour, "An efficient approximate dynamic programming approach for resource-constrained project scheduling with un-

certain task duration," in *Proceedings of the 13th International Conference on Operations Research and Enterprise Systems*, 2024, pp. 261–268, ISBN: 978-989-758-681-1. DOI: 10.5220/0012356200003639.

[206] V. Gergel, "A Global Optimization Algorithm for Multivariate Functions with Lipschitzian First Derivatives," *Journal of Global Optimization*, vol. 10, pp. 257–281, 1997.

[207] T. Nguyen, Z. Li, S. Zhang, and T. Truong, "A hybrid algorithm based on particle swarm and chemical reaction optimization," *Expert Systems with Applications: An International Journal*, vol. 41, pp. 2134–2143, Apr. 2014. DOI: 10.1016/j.eswa.2013.09.012.

[208] H. T. Reiner Horst, *Global Optimization*, 3rd. Springer-Verlag Berlin Heidelberg, 1996.

[209] T. C. Immanuel Bomze, *Developments in Global Optimization*, 1st. Springer US, 1997.

[210] D. S. W. G. G. Rizk-Allah Rizk M. EI-Sehiemy R A., "Hybridizing sine cosine algorithm with multi-orthogonal search strategy for engineering design problems," *A novel fruit fly framework for multi-objective shape design of tubular linear synchronous motor*, vol. 73, no. 3, pp. 1–22, 2016.

[211] R. M. Rizk-Allah, R. A. El-Sehiemy, and G.-G. Wang, "A novel parallel hurricane optimization algorithm for secure emission/economic load dispatch solution," *Applied Soft Computing*, vol. 63, pp. 206–222, 2018, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2017.12.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494617307160.

[212] "Line search methods," in *Numerical Optimization*, J. Nocedal and S. J. Wright, Eds. New York, NY: Springer New York, 1999, pp. 34–63, ISBN: 978-0-387-22742-9. DOI: $10.1007/0-387-22742-3_3$. [Online]. Available: $https://doi.org/10.1007/0-387-22742-3_3$.

[213] E. Chong and S. Zak, *An Introduction to Optimization* (Wiley Series in Discrete Mathematics and Optimization). Wiley, 2013, ISBN: 9781118515150. [Online]. Available: https://books.google.nl/books?id=iD5s0iKXHP8C.

[214] S. Das, P. Koduru, Min Gui, *et al.*, "Adding local search to particle swarm optimization," in *2006 IEEE International Conference on Evolutionary Computation*, 2006, pp. 428–433.

[215] J. Wang and Y. Zhou, "Quantum-behaved particle swarm optimization with generalized local search operator for global optimization," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, D.-S. Huang, L. Heutte, and M. Loog, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 851–860.

[216] Junying Chen, Zheng Qin, Yu Liu, and Jiang Lu, "Particle swarm optimization with local search," in *2005 International Conference on Neural Networks and Brain*, vol. 1, 2005, pp. 481–484.

[217] H. M. Alshamlan, G. H. Badr, and Y. A. Alohali, "Genetic bee colony (gbc) algorithm: A new gene selection method for microarray cancer classification," *Computational Biology and Chemistry*, vol. 56, pp. 49–60, 2015, ISSN: 1476-9271. DOI: https://doi.org/10.1016/j.compbiolchem.2015.03.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S147692711500047X.

[218] K. Premalatha and A. Natarajan, "Hybrid pso and ga for global maximization," *Int J Open Prob Compt Math*, vol. 2, Jan. 2009.

[219] S. Fidanova, M. Paprzycki, and O. Roeva, "Hybrid ga-aco algorithm for a model parameters identification problem," in *2014 Federated Conference on Computer Science and Information Systems, FedCSIS 2014*, Sep. 2014, pp. 413–420. DOI: 10.15439/2014F373.

[220] Z. Meng, J.-S. Pan, and H. Xu, "Quasi-affine transformation evolutionary (quatre) algorithm: A cooperative swarm based algorithm for global optimization," *Knowledge-Based*

Systems, vol. 109, pp. 104–121, 2016, ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2016.06.029. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705116302003.

[221] A. Sharma, A. Sharma, S. Choudhary, R. Pachauri, A. Shrivastava, and D. Kumar, "A review on artificial bee colony and it's engineering applications," *Journal of Critical Reviews*, vol. 7, p. 2020, Aug. 2020. DOI: 10.31838/jcr.07.11.558.

[222] B. T. Tezel and A. Mert, "A cooperative system for metaheuristic algorithms," *Expert Systems with Applications*, vol. 165, p. 113 976, 2021, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113976. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417420307545.

[223] S. Elmaghraby, *Activity Networks: Project Planning and Control by Network Models* (A Wiley-Interscience publication). Wiley, 1977, ISBN: 9780471238614.

[224] H. Mohammed and T. Rashid, "A novel hybrid gwo with woa for global numerical optimization and solving pressure vessel design," *Neural Computing and Applications*, pp. 14 701–14 718, Mar. 2020. DOI: 10.36227/techrxiv.11916369.v1.

[225] M. Jamil and X. Yang, "A literature survey of benchmark functions for global optimisation problems," *ArXiv*, vol. abs/1308.4008, 2013.

[226] K. Hussain, M. Salleh, S. Cheng, and R. Naseem, "Common benchmark functions for metaheuristic evaluation: A review," *International Journal on Informatics Visualization*, vol. 1, pp. 218–223, Nov. 2017. DOI: 10.30630/joiv.1.4-2.65.

[227] K. V. Price, N. H. Awad, M. Z. Ali, and P. N. Suganthan, "The 100-Digit Challenge: Problem Definitions and Evaluation Criteria for the 100-Digit Challenge Special Session and Competition on Single Objective Numerical Optimization," School Elect. Electron. Eng., Nanyang Technol. Univ., Singapore, Tech. Rep., Nov. 2018.

[228] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied Mathematics and Computation*, vol. 214, no. 1, pp. 108–132, 2009, ISSN: 0096-3003.

[229] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *Annals of Mathematical Statistics*, vol. 11, pp. 86–92, 1940.

[230] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

[231] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2011.02.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650211000034.

[232] Q. Jia and Y. Seo, "Solving resource-constrained project scheduling problems: Conceptual validation of flp formulation and efficient permutation-based abc computation," *Computers & Operations Research*, vol. 40, no. 8, pp. 2037–2050, 2013, ISSN: 0305-0548.

[233] Y.-j. Shi, F.-Z. Qu, W. Chen, and B. Li, "An artificial bee colony with random key for resource-constrained project scheduling," in *Life System Modeling and Intelligent Computing*, K. Li, M. Fei, L. Jia, and G. W. Irwin, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 148–157.

[234] Q. Jia and Y. Guo, "Hybridization of abc and pso algorithms for improved solutions of rcpsp," *Journal of the Chinese Institute of Engineers*, vol. 39, no. 6, pp. 727–734, 2016.

[235] J. Czogalla and A. Fink, "Particle swarm topologies for resource constrained project scheduling," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2008).*

Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 61–73, ISBN: 978-3-642-03211-0. DOI: 10.1007/978-3-642-03211-0_6. [Online]. Available: https://doi.org/10.1007/978-3-642-03211-0_6.

[236] A. Fahmy, T. M. Hassan, and H. Bassioni, "Improving rcpsp solutions quality with stacking justification – application with particle swarm optimization," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5870–5881, 2014, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.03.027. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741741400164X.

[237] M. M. Nasiri, "A pseudo particle swarm optimization for the rcpsp," *The International Journal of Advanced Manufacturing Technology*, vol. 65, no. 5, pp. 909–918, 2013, ISSN: 1433-3015. DOI: 10.1007/s00170-012-4227-8.

[238] G. Koulinas, L. Kotsikas, and K. Anagnostopoulos, "A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem," *Information Sciences*, vol. 277, pp. 680–693, 2014, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2014.02.155. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025514002771.

[239] L. F. Muller, "An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem," in *Proceedings of the VIII Metaheuristics International Conference (MIC)*, 2009.

[240] D. Debels, B. De Reyck, R. Leus, and M. Vanhoucke, "A hybrid scatter search/electromagnetism meta-heuristic for project scheduling," *European Journal of Operational Research*, vol. 169, no. 2, pp. 638–653, 2006, Feature Cluster on Scatter Search Methods for Optimization, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2004.08.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221704005594.

[241] S. Elsayed, R. Sarker, T. Ray, and C. C. Coello, "Consolidated optimization algorithm for resource-constrained project scheduling problems," *Information Sciences*, vol. 418-419, pp. 346–362, 2017, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.08.023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025517308745.

[242] B. Roy and A. K. Sen, "A novel swarm intelligence approach for the constrained scheduling problem," *International Journal of Management Science and Engineering Management*, vol. 0, no. 0, pp. 1–13, 2023. DOI: 10.1080/17509653.2023.2248058.

[243] A. Bockmayr and J. N. Hooker, "Constraint programming," in *Discrete Optimization*, ser. Handbooks in Operations Research and Management Science, K. Aardal, G. Nemhauser, and R. Weismantel, Eds., vol. 12, Elsevier, 2005, pp. 559–600. DOI: https://doi.org/10.1016/S0927-0507(05)12010-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927050705120106.

[244] A. Fréville, "The multidimensional 0–1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, 2004, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(03)00274-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221703002741.

[245] G. A. Fernandes and S. R. de Souza, "A matheuristic approach to the multi-mode resource constrained project scheduling problem," *Computers & Industrial Engineering*, vol. 162, p. 107 592, 2021, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2021.107592.

[246] R. Zamani, "An effective mirror-based genetic algorithm for scheduling multi-mode resource constrained projects," *Computers & Industrial Engineering*, vol. 127, pp. 914–924, 2019, ISSN: 0360-8352.

[247] J. Coelho and M. Vanhoucke, "Multi-mode resource-constrained project scheduling using rcpsp and sat solvers," *European Journal of Operational Research*, vol. 213, no. 1, pp. 73–82, 2011, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2011.03.019.

[248] M. Vanhoucke and J. Coelho, "A tool to test and validate algorithms for the resource-constrained project scheduling problem," *Computers & Industrial Engineering*, vol. 118, pp. 251–265, 2018, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2018.02.001.

[249] A. Pritsker, L. J. Waiters, and P. Wolfe, "Multiproject scheduling with limited resources: A zero-one programming approach," *Management Science*, vol. 16, pp. 93–108, 1969.

[250] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[251] R. E. Bellman, *Dynamic programming*. Princeton university press, 2010.

[252] D. Bertsekas, J. Tsitsiklis, and C. Wu, "Rollout algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 3, pp. 245–262, Nov. 1997.

[253] S. M. Pour, J. H. Drake, L. S. Ejlertsen, K. M. Rasmussen, and E. K. Burke, "A hybrid constraint programming/mixed integer programming framework for the preventive signaling maintenance crew scheduling problem," *European Journal of Operational Research*, vol. 269, no. 1, pp. 341–352, 2018, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2017.08.033. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221717307646.

[254] R. Kolisch and S. Hartmann, "Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis," in *Project Scheduling: Recent Models, Algorithms and Applications*. Boston, MA: Springer US, 1999, pp. 147–178, ISBN: 978-1-4615-5533-9. DOI: 10.1007/978-1-4615-5533-9_7. [Online]. Available: https://doi.org/10.1007/978-1-4615-5533-9_7.

[255] P. Vilím, P. Laborie, and P. Shaw, "Failure-directed search for constraint-based scheduling," in *Integration of AI and OR Techniques in Constraint Programming*, L. Michel, Ed., Cham: Springer International Publishing, 2015, pp. 437–453, ISBN: 978-3-319-18008-3.

[256] T. Feydy and P. J. Stuckey, "Lazy clause generation reengineered," in *Principles and Practice of Constraint Programming - CP 2009*, I. P. Gent, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 352–366, ISBN: 978-3-642-04244-7.

[257] M. Bofill, J. Coll, J. Suy, and M. Villaret, "Smt encodings for resource-constrained project scheduling problems," *Computers & Industrial Engineering*, vol. 149, p. 106 777, 2020, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106777. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220304873.

[258] K. Ziarati, R. Akbari, and V. Zeighami, "On the performance of bee algorithms for resource-constrained project scheduling problem," *Applied Soft Computing*, vol. 11, no. 4, pp. 3720–3733, 2011, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2011.02.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494611000627.

[259] J. Liu, Y. Liu, Y. Shi, and J. Li, "Solving resource-constrained project scheduling problem via genetic algorithm," *Journal of Computing in Civil Engineering*, vol. 34, no. 2, p. 04 019 055, 2020. DOI: 10.1061/(ASCE)CP.1943-5487.0000874.

[260] R. Kolisch and A. Sprecher, "PSPLIB - A project scheduling problem library: OR Software - ORSEP Operations Research Software Exchange Program," *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997, ISSN: 0377-2217.