www.elsevier.com/locate/caor

Acquisition planning and scheduling of computing resources

Chien-Nan Yang, Bertrand M.T. Lin, F.J. Hwang, Meng-Chun Wang

PII: S0305-0548(16)30147-2
DOI: http://dx.doi.org/10.1016/j.cor.2016.06.015
Reference: CAOR4027

To appear in: *Computers and Operation Research*

Received date: 23 November 2014
Revised date: 11 March 2016
Accepted date: 17 June 2016

Acquisition Planning and Scheduling of Computing Resources

Chien-Nan Yang[1], Bertrand M.T. Lin[1], F.J. Hwang[2], and Meng-Chun Wang[1]

[1]Institute of Information Management
Department of Information and Finance Management
National Chiao Tung University, Taiwan

[2]School of Mathematical and Physical Sciences
University of Technology Sydney, Ultimo 2007, Australia

Abstract: Cloud computing has been attracting considerable attention since the last decade. This study considers a decision problem formulated from the use of computing services over the Internet. An agent receives orders of computing tasks from his/her clients and on the other hand he/she acquires computing resources from computing service providers to fulfill the requirements of the clients. The processors are bundled as packages according to their speeds and the business strategies of the providers. The packages are rated at a certain pricing scheme to provide flexible purchasing options to the agent. The decision of the agent is to select the packages which can be acquired from the service providers and then schedule the tasks of the clients onto the processors of the acquired packages such that the total cost, including acquisition cost and scheduling cost (total weighted tardiness), is minimized. In this study, we present an integer programming model to formulate the problem and propose several solution methods to produce acquisition and scheduling plans. Ten well-known heuristics of parallel-machine scheduling are adapted to fit into the studied problem so as to provide initial solutions. Tabu search and genetic algorithm are tailored to reflect the problem nature to improve upon the initial solutions. We conduct a series of computational experiments to evaluate the effectiveness and efficiency of all the proposed algorithms. The results of the numerical experiments reveal that the proposed tabu search and genetic algorithm can attain significant improvements.
keywords: Computing service; acquisition planning; scheduling; heuristics; tabu search; genetic algorithm.

1 Introduction

In this recent decade, cloud computing has become a popular topic in many research and application areas over the Internet. Armbrust et al. [2] gave a definition from an academic perspective: "Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services." It is an attractive solution for those companies that do not have the ability and capital to build a large ad hoc computing and/or data center. Acquiring computing resources from service providers instead of establishing private IT infrastructures saves both time and cost in many aspects. As the provider is also responsible for maintenance, the clients do not have to hire specialists to be in charge of the security or sustainability problem, thus reducing the personnel cost to a considerable extent. The convenience and stability brought forth by cloud computing have stimulated the development of technologies and service models for this booming business opportunity. One of the service models is called IaaS (Infrastructure as a Service), where the vendors sell their computing resources often referred to as "virtual machines" to the clients. Amazon Elastic Compute Cloud (`http://aws.amazon.com/ec2/`) and IBM SmartCloud (`http://www.ibm.com/cloud/`) are some prominent examples of IaaS.

In contrast to a wide variety of service modes, there is an obvious lack of the pricing schemes for adaptability. Nowadays the providers sell their computing resources in similar ways, which are called pay-as-you-go or pay-per-use. This usage-based pricing scheme stems from the utility service (e.g. electricity, natural gas and water) pricing concept. Besides, a group of different prices is set for processors according to the computing capacities, such as the CPU speeds or the memory sizes. There is no minimum fee or discount on purchasing multiple instances, and only a slightly discount on purchasing an instance for a long time, e.g. a 1-year or 3-year term. However, it is insufficient for the consumers who have different and special requirements. Thus, new pricing schemes are needed for attracting those consumers. Bakos and Brynjolfsson [5] studied the strategy of bundling distinct information goods and selling them for a fixed price. Analyses revealed that this strategy often yields higher profits than that from selling them separately. Sundararajan [25] indicated that fixed-fee unlimited-usage pricing and usage-based pricing schemes should be included in different stages of information markets. They suggested that a fixed-fee pricing scheme should be included in both early-stage and mature markets. With these studies, we come up with a new scheme to bundle the processors with different speeds together as packages and rate them at different prices according to their computing capacities. A fixed-charge time interval is also given since the unlimited-usage price may be too expensive for small and medium-size enterprises. Within this time interval, consumers can fully utilize the resources in the purchased package without any extra fee. After the fixed-charge time interval, the usage-based pricing scheme applies to accumulate the expenses. This new pricing scheme is a win-win model to both parties. From the viewpoint of cloud computing service providers, the flexibility of adjusting processors makes them easily bundle processors for forming distinct packages. The providers can bundle the oldest/slowest machines and the newest/fastest machines together so that the performances could be balanced. The processors can be freely arranged to make multiple packages and fully satisfy various kinds of consumers. Also, this model can attain a large increase in profits as well as reduce the unnecessary waste of resources. From the standpoint of clients, there are more choices of packages and pricing schemes. The clients can hire packages according to their budgets and dispatch the jobs based on their urgency.

In this paper, we, from the perspective of a cloud service agent, develop a planning and scheduling model by regarding the cloud computing environments of IaaS as parallel machines. A cloud service agent acts as an intermediary between the cloud computing service providers and the clients to negotiate the contracts, bargains, and also provides additional services. As the interest in cloud computing grows, a brokerage service is necessary for the clients. Like the real estate agents or stock agents, the cloud computing agents are the connections between service providers and clients. They help the clients to select the services they need, and purchase the services from the providers. They may have multiple jobs from various clients and also purchase computing resources from different providers. Therefore, the arrangement of resource dispatching becomes an important issue. The agents have to complete the jobs of consumers as soon as possible to earn their trust and the opportunities for future cooperation. On the other hand, they have to purchase sufficient resources to process the jobs and make their own profits. Thus, as a third-party business, the management of the cloud computing agents is directly related to both service providers and clients. For simplicity of description, we assume that the resources in the purchased packages can be fully controlled and managed by the agents, and allocation of jobs to processors and the job sequence on each processor are determined by the agents as well. Since the agent aims to lower the cost of purchasing computing resources and finish the jobs on time, we formulate the studied model as a parallel-machine scheduling problem with resources acquisition planning. Since the machine speeds are usually distinct in the environment of cloud computing, we are tackling precisely a scheduling problem on uniform parallel machines. The objective function is a linear combination of the total weighted tardiness ($\sum w_j T_j$) and the total package acquisition cost ($\sum \Psi_i(L_i)$). The acquisition cost of package i is calculated based on the schedule makespan L_i and the package pricing scheme proposed above. Using the three-field notation [23], the studied problem can be denoted by $Qm||\alpha \sum w_j T_j + (1 - \alpha) \sum \Psi_i(L_i)$, where Qm represents the uniform parallel machines, and α is the weighting parameter for normalizing the two types of costs.

The rest of this paper is organized as follows. A brief review of the related literature is given in Section 2. In Section 3, problem definition and notation are provided along with an illustrated example, and an integer programming model for the studied problem is formulated. In Section 4, several heuristics are used to attain initial solutions. Details of the meta-heuristics utilized for improving the initial solutions are described in Section 5. In Section 6, computational results and related analyses are reported. Finally, Section 7 concludes this research and provides suggestions for further research.

2 Literature Review

Parallel-machine scheduling problems with different objectives and constraints have been extensively studied in the open literature. As Cheng and Sin [9] noted in their state-of-the-art review of major research results in parallel-machine scheduling problems, various job characteristics, machine configurations and performance criteria are of theoretical interest as well as practical significance. Minimizing the total tardiness with penalty weights is one of the commonly considered objectives . When the penalty weights are arbitrary positive numbers, the scheduling problem with identical parallel machines $Pm||\sum w_j T_j$ is NP-hard in the strong sense [19, 23]. When the penalty weights of all jobs are the same, the $Pm||\sum T_j$ problem is at least binary NP-hard [17]. Several studies examined the properties that lie in the structures of an optimal schedule, and developed exact algorithms for $Pm||\sum T_j$. Azizoglu and Kirca [3] presented some dominance properties for $Pm||\sum T_j$ and proposed a branch-and-bound algorithm that can solve instances with up to 15 jobs and 3 machines. They also extended some of the properties for $Qm||\sum T_j$. Yalaoui and Chu [29] developed more dominance properties and bounding rules, and showed that their branch-and-bound algorithm could obtain optimal solutions in some cases with 30 jobs and 2 machines. Shim and Kim [24] also provided dominance properties and lower bounds to show that the suggested algorithm could find optimal solutions for problems with up to 30 jobs and 5 machines in a reasonable time. As for the uniform parallel-machine problems, Dessouky et al. [10] presented algorithms for different objectives under the strong assumption that the jobs are identical, and proposed a dynamic programming algorithm for minimizing the total completion time subject to release dates.

Considering the non-classical objective functions, such as the job processing costs, several studies investigated bi-criteria scheduling problems. Leung et al. [20] addressed the bi-criteria consisting of one classical and one non-classical objective functions with two different bi-criteria structures in parallel machine scheduling. One is the hierarchical bi-criteria, i.e. optimizing the secondary objective among the schedules that the primary objective is minimized. The other is the linear combination of two objective functions, which is exactly the bi-criteria structure used in this paper. The considered classical objective is either the makespan or total completion time. Concerning a cost associated with the processing of a specific job on a particular machine, the addressed non-classical objective is the total machine assignment cost. They presented the complexity results for the considered problems. Lee et al. [18] studied the same problem with the hierarchical bi-criteria structure and developed approximation algorithms with worst -case performance analyses. A different job processing cost, which is determined by the time slots used by the job, is considered by Wan and Qi [27] for single-machine scheduling. The non-classical objective function considered in this paper is different from the above ones with respect to the following three characteristics: (i) The package acquisition cost function is a piecewise linear function; (ii) The incurred cost is associated with packages rather than jobs; (iii) The acquisition cost for each package is a function of the schedule makespan in the package.

Since the proposed $Qm||\alpha\sum w_jT_j + (1-\alpha)\sum \Psi_i(L_i)$ model is an extension of the strongly NP-hard problem $Qm||\sum w_jT_j$, developing exact solution methods, such as branch-and-bound algorithm, can only solve small or medium-size problem instances. Considering the large-scale instances in practical application, designing heuristics to derive near-optimal solutions in an acceptable time is necessary. In most heuristics, the job dispatching order is determined by a certain priority rule which is calculated by different formulae. The top-priority job is selected to be processed by the first available machine. Carroll [8] provided a rule called Cost over Time (COV) for sequencing single- or multiple-component jobs. Montagne [21] introduced Montagne's Ratio Rule (MRR) for the minimum total weighted tardiness problem. For single-machine scheduling, Baker and Bertrand [4] presented the Modified Due-Date rule (MDD) which is a combination of the Earliest Due Date first rule (EDD) and the Shortest Processing Time first rule (SPT). Morton et al. [22] designed the Apparent Urgency rule (AU) for the total weighted tardiness minimization on single and parallel machines. Ho and Chang [15] devised the Traffic Priority Index Rule (TPI), which computes the priority indices by the traffic congestion ratio (TCR), for minimizing the mean tardiness. Considering the minimization of total weighted and unweighted tardiness on parallel machines, Alidaee and Rosa [1] extended the MDD rule and compared it with other existing heuristics. Their experiment results indicate its dominance in effectiveness over other heuristics. We will further elaborate on the above heuristics in Section 4.

3 Problem Formulation and Integer Linear Programming Model

In this section, we present a formal problem definition. Cloud computing service providers offer computing packages, each of which contains multiple processors running at different speeds. A agent receives orders of computing tasks from several clients and acquires computing packages from service providers. The agent assigns the tasks of clients to the acquired processors to reflect the performance criteria set by the clients. This paper adopts the performance measure of total weighted tardiness to indicate the service quality. Other criteria can be considered in a similar way. While maintaining service quality, the agent simultaneously needs to reduce his/her acquisition cost. Therefore, the decisions include the selection of computing packages to purchase and scheduling of the commissioned tasks on the acquired processors so as to minimize a cost function that comprises acquisition cost and total weighted tardiness. Formal statements of the problem are given as follows.

Consider n jobs $J_1, J_2, \ldots, J_n$ to be scheduled on h packages $P_1, P_2, \ldots, P_h$. Package P_i has μ_i parallel processors that may run at different processing speeds. The k^{th} processor in package P_i is denoted by p_{ik} and its speed is given by s_{ik} for all $i = 1, \ldots, h$ and $k = 1, \ldots, \mu_i$. Package P_i is characterized by base time b_i, fixed-charge cost f_i, and unit cost c_i. The usage duration for each package P_i is denoted by L_i, and defined as the longest elapsed processing time among the processors in P_i. That is, L_i is the makespan of the actual schedule of the jobs on the processors of package P_i. The cost of acquiring package P_i is given by a non-decreasing step function $\Psi_i(L_i)$:

$$\Psi_i(L_i) = \begin{cases} 0, & \text{if } \; L_i = 0 \text{ (not acquired);} \\ f_i, & \text{if } \; 0 < L_i \leq b_i; \\ f_i + c_i(L_i - b_i), & \text{if } \; b_i < L_i. \end{cases}$$

Job J_j is described by computing load ℓ_j, due date d_j, and tardiness penalty weight w_j. Its actual processing time depends on the processor p_{ik} to be assigned to and is given by $t_{jik} = \lceil \ell_j / s_{ik} \rceil$. All jobs are released or available for processing at the beginning of the time horizon. The weighted tardiness penalty for job j is defined by $w_j T_j = w_j \max\{0, C_j - d_j\}$, where C_j is the completion time of job j. The computing resource of the processors are bundled and sold in packages, and the jobs can be assigned to only the processors of the purchased packages. Migration of job processing among processors or packages is not allowed. The objective is to determine an acquisition plan and a processing schedule to minimize the weighted sum of the package acquisition cost and the total weighted tardiness. The following example illustrates the problem definition.

Example. Consider a set of eight jobs to be scheduled in two available packages, each of which has two processors. The parameter values are shown in Table 1, and two example schedules are presented in Figure 1.

Table 1: Example parameters.

job	1	2	3	4	5	6	7	8
ℓ_j	8	8	7	8	4	6	6	5
d_j	4	7	3	6	7	7	6	6
w_j	8	3	5	9	4	4	6	7

Package	P_1	P_2
c_i	25	15
f_i	15	10
b_i	10	15
s_{ik}	1.8, 1.9	1.2, 1.3

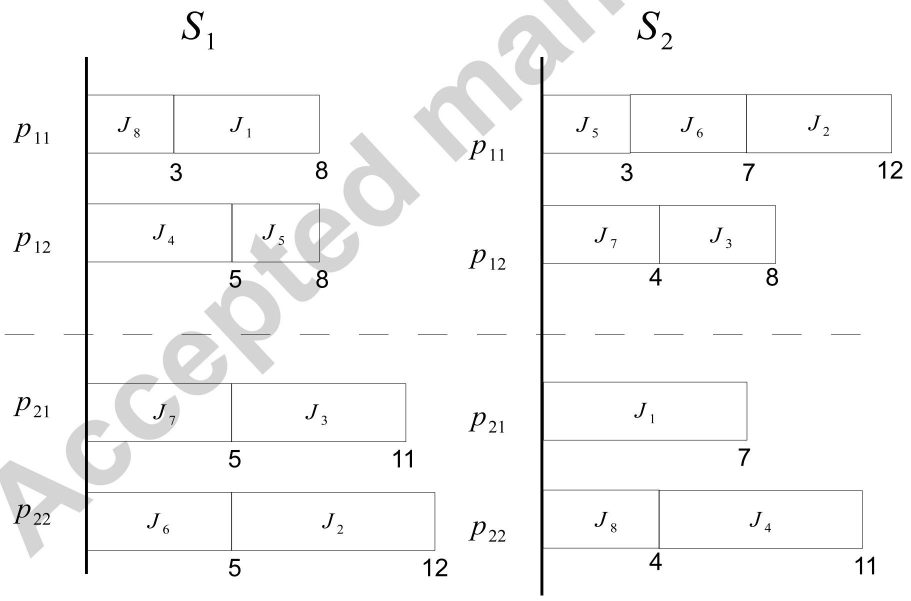


Figure 1: Example schedules.

As shown in Figure 1, the package elapsed times of schedule S_1 are 8 for package P_1 and 12 for package P_2, both of which are within the base time. Thus, the acquisition cost of schedule S_1 is 25, the sum of the fixed-charge costs. However, in schedule S_2, the elapsed time of P_1 is 12, which exceeds the base time 10. Therefore, P_1 incurs two extra time units and results in an acquisition cost of 75 for S_2. Schedule S_1 assigns four jobs to each package and has a total weighted tardiness of 91. The other schedule S_2 dispatches five jobs to package P_1 and three jobs to package P_2, and attains a total weighted tardiness of 109. The decisions involving the package acquisition planning as well as job dispatching and sequencing exhibit the complex structure of the studied problem which demands tailored solution approaches.

The following assumptions and constraints are incorporated in this paper:

1. All jobs are available for processing from time zero onward, i.e. no release date is assumed;
2. Every job can be processed by any processor in the packages;
3. Any processor can process at most one job at a time;
4. The processing time of a job depends on the speed of the processor it is assigned to;
5. No migration is assumed, i.e. if a job is started on some processor, then it must be processed on that processor until its completion;
6. Preemption is not allowed on any processor.

Under the first assumption, the performance of a single-thread execution mode is not worse than that of a multi-thread execution mode. Therefore, we also make the fifth and the sixth assumptions.

Before presenting the mathematical programming model for the studied problem, we define two binary decision variables x_{jikl} and y_i as follows. If job J_j is sequenced in the l^{th} position, where $l \in \{1, \ldots, n\}$, in processor p_{ik}, then $x_{jikl} = 1$; otherwise, $x_{jikl} = 0$. If package P_i is purchased, then $y_i = 1$; otherwise, $y_i = 0$. A binary integer linear program (ILP) can be formulated as follows.

$$\text{Minimize} \quad Z = \alpha \sum_{j=1}^{n} w_j T_j + (1-\alpha) \sum_{i=1}^{h} \Psi_i(L_i)$$

$$\sum_{i=1}^{h}\sum_{k=1}^{\mu_i}\sum_{l=1}^{n} x_{jikl} = 1, \qquad \forall j, \tag{1}$$

$$\sum_{j=1}^{n} x_{jikl} \leq 1, \qquad \forall i,k,l, \tag{2}$$

$$L_i = \max_{1\leq k\leq \mu_i} \{\sum_{l=1}^{n}\sum_{j=1}^{n} x_{jikl}t_{jik}\}, \qquad \forall i, \tag{3}$$

$$T_j \geq S_{i,k,l} + x_{jikl}t_{jik} - (1-x_{jikl})M - d_j, \qquad \forall j,i,k,l, \tag{4}$$

$$S_{i,k,l+1} - S_{i,k,l} = x_{jikl}t_{jik}, \qquad \forall j,i,k,l, \tag{5}$$

$$y_i \geq x_{jikl}, \qquad \forall j,i,k,l, \tag{6}$$

$$\Psi_i(L_i) = y_i f_i + c_i \max\{0, (L_i - b_i)\}, \qquad \forall i, \tag{7}$$

$$T_j \geq 0, \qquad \forall j, \tag{8}$$

$$x_{jikl} \in \{0,1\}, \qquad \forall j,i,k,l, \tag{9}$$

$$y_i \in \{0,1\}, \qquad \forall i. \tag{10}$$

In the above formulation, the objective function is a weighted sum of two terms with the weighting parameter α. The first term is the total weighted tardiness, and the second term is the acquisition cost. Eq. (1) enforces that each job is to be processed in exactly one position of a specific processor of its package. Eq. (2) ensures that a position of a specific processor in its package can accommodate at most one job. Eq. (3) states the duration time of each package, i.e. the longest completion time among processors in each package. The tardiness can be expressed by Eq. (4) and (8), where $S_{i,k,l}$ is the starting time of the job sequenced in the l^{th} position in processor p_{ik}, and M is a sufficiently large positive number. If job J_j is processed in the l^{th} position in processor p_{ik}, then $x_{jikl} = 1$. On the other hand, when $x_{jikl} = 0$, the inequality will be satisfied due to the large negative value $-M$. Eq. (5) enforces that the jobs on the same processor do not overlap. Eq. (6) specifies whether the package is purchased or not. If a job is assigned to any processor of package P_i, the package has to be purchased ($y_i = 1$). Eq. (7) is the pricing function of package P_i. If P_i is not purchased, then $y_i = 0$ and $L_i = 0$. The cost is thus zero. Eq. (9) and (10) are given for the the binary restriction on the decision variables.

The commercial software CPLEX was deployed to solve the ILP model for the example instance given in Table 1. Given $\alpha = 0.5$, an optimal solution $x_{1111} = x_{2112} = x_{3121} = x_{4221} = x_{5123} = x_{6212} = x_{7211} = x_{8122} = 1, y_1 = 1, y_2 = 1$ with $L_1 = 10, L_2 = 10, T_1 = 1, T_2 = 3, T_3 = 1, T_4 = 1, T_5 = 3, T_6 = 3, T_7 = 0, T_8 = 1, \Psi_1 = 15, \Psi_2 = 10$ is yielded. The corresponding optimal schedule S_{opt} is shown in Figure 2. The derived schedule has a total weighted tardiness of 62 and an acquisition cost of 25. Compared to the schedules in Figure 1, the optimal schedule S_{opt} dispatches five jobs to the faster package P_1 and fully utilizes its base-time interval. Thus, the CPLEX solver reduces the total weighted tardiness and leads to an optimal schedule with objective value of 43.5 under the condition $\alpha = 0.5$.

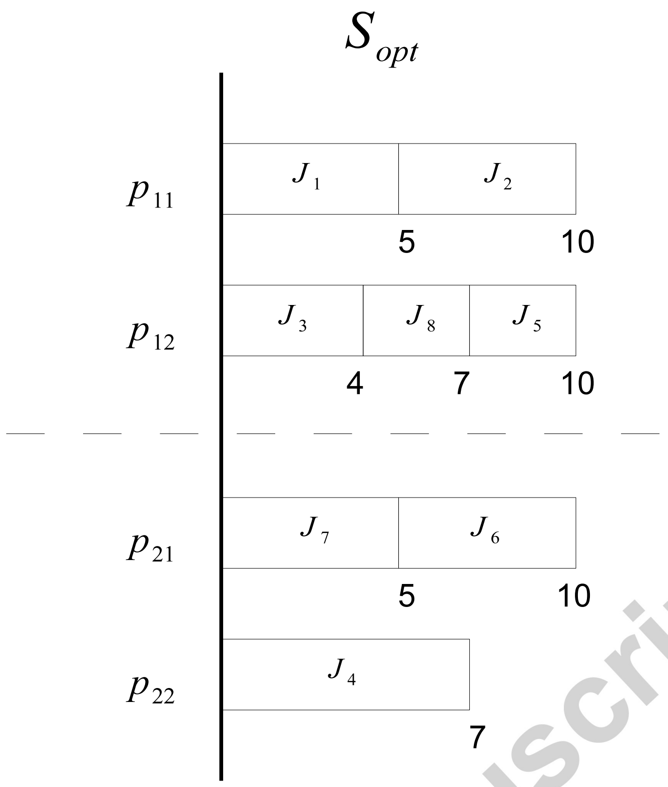


Figure 2: Optimal solution with the objective value 43.5 ($\alpha = 0.5$).

4 Heuristics

The proposed model is an extension of total tardiness minimization in parallel-machine scheduling, which is NP-hard [19, 23], implying that developing exact solution methods, such as branch-and-bound algorithm, to derive optimal solutions in polynomial time is very unlikely. Our preliminary study suggests that deploying the commercial software CPLEX can solve the proposed ILP for only small-size instances with n no greater than eight. We thus circumvent to adopt heuristics to find approximate solutions and then apply meta-heuristics to further improve upon the solutions. In this section, we adapt all the heuristics introduced in literature review [1, 4, 8, 15, 21, 22] for parallel-machine scheduling to solve the studied acquisition planning and scheduling problem. In the identical parallel-machine setting, job processing times can be simply given by job loads, i.e. $t_{jik} = \ell_j/1$. For the uniform parallel-machine scheduling considered in this study, the job processing time is determined by the speed of the processor, i.e. $t_{jik} = \lceil \ell_j/s_{ik} \rceil$. In order to adapt the heuristics originally proposed for identical machines to the studied problem, some adjustments on job processing times are necessary. Some of these heuristics may use the processing times to identify the priority indices among the remaining unscheduled jobs. Since the processing time is unknown until it has been allocated to a processor, we use the slowest speed among all the processors to reflect the priority indices. The estimated maximum processing time for job j is denoted by $\bar{t}_j = \lceil \ell_j / s_{\min} \rceil$, where $s_{\min} = \min_{i,k}\{s_{ik}\}$. A challenge arising in managing the machines is due to the different numbers of machines included in the packages.

In parallel-machine scheduling, the number of machines is usually denoted by m. In the studied model, the total number of machines is the summation of processors in all the packages, i.e, $m = \sum_{i=1}^{h} \mu_i$. We explain the original heuristics in the following section, while substituting some major components addressed here with different notation. These heuristics can be classified into two groups, *static* and *dynamic*, according to the time dependency.

4.1 Static Dispatching

Static dispatching consists of job dispatching rules and package selection rules that are irrelevant to the completion time of the last job residing on each machine. A static rule determines the job priority indices simply by the static data of the jobs and/or packages. After the order has been determined, the package selection rule chooses a processor to accommodate the job of the highest priority, until all jobs are dispatched.

(1)*Job dispatching rules*

- *Earliest Due Date* (EDD): The jobs are ordered in the non-decreasing order of d_j.
- *Weighted Minimum Load* (WML): The jobs are ordered in the non-increasing order of w_j/ℓ_j. The rule is adapted from the Weighted Shortest Processing Time first (WSPT) rule.
- *Largest Load First* (LLF): The jobs are ordered in the non-increasing order of ℓ_j. The rule is adapted from the Longest Processing Time first (LPT) rule.
- *Slack Time* (SLK): The jobs are ordered in the non-decreasing order of $d_j - \bar{t}_j$.
- *Montagne's Ratio Rule* (MRR) [21]: The jobs are ordered in the non-decreasing order of $\bar{t}_j/(t - d_j)$, where t is the sum of estimated maximum processing times divided by the number of processors, i.e. $t = \sum_{j=1}^{n} \bar{t}_j / m$.
- *Traffic Priority Index Rule* (TPI) [15]: This rule is defined as follows. Set *traffic congestion ratio (TCR)* $= \bar{t}n/\bar{d}m$, where $\bar{t}$ and $\bar{d}$ are the average of estimated processing times and the average of due dates, respectively. Also, let

$$M_d = \max_{1 \le j \le n} \{d_j\},$$

$$M_p = \max_{1 \le j \le n} \{\bar{t}_j\},$$

a constant factor $K = 3$ as suggested by Ho and Chang [15],

$$W_d = \max\{\min\{0.5 + \frac{K - TCR}{TCR}, 1\}, 0\},$$
$$W_p = 1 - W_d,$$
$$R_j = (d_j \times \frac{W_d}{M_d}) + (\bar{t_j} \times \frac{W_p}{M_p}), \quad \forall j.$$

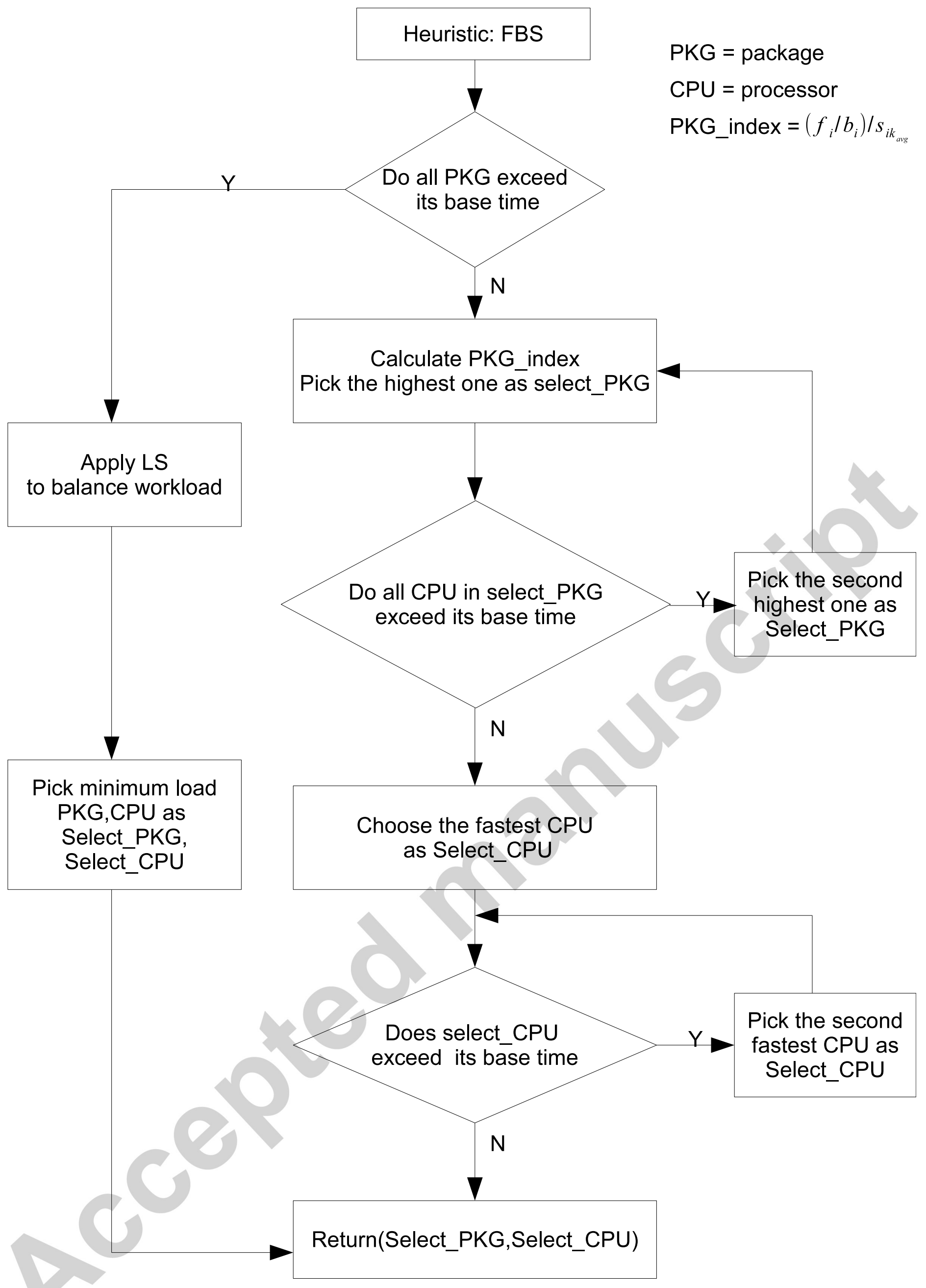
Then the jobs are ordered in the non-decreasing order of R_j.

(2) *Package selection rules*

- *List Scheduling* (LS): This rule selects the processor which has the shortest processing time among all processors to process the top priority job.
- *Ratio of Fixed, Base, Speed* (FBS): We develop a heuristic, called FBS, to select the worthiest processor by the ratio of the cost per unit time to the average speed. We use $(f_i/b_i)/s_{ik_{avg}}$ as a factor to select package P_i, where $s_{ik_{avg}}$ is the average speed of processors in package P_i. Then we purchase the highest ratio package and assign the job to the fastest processors in this package. Before we purchase another package, we seek to fill up the base time of processors of the purchased package in order to minimize total cost. If the base time of all packages is filled, we again use LS to schedule the remaining jobs for balancing the workload among processors. The reason behind using LS instead of using $(c_i/s_{ik_{avg}})$ is to avoid selecting the same (highest ratio) package, which will increase the tardiness rapidly. The flow chart of FBS is shown in Figure 3.
- LS': This rule selects the processor that can finish the top priority job at the earliest time.
- FBS': The selection rule is the same as FBS, except for using LS' instead of LS for scheduling the remaining jobs to balance the workload among processors.

4.2 Dynamic Dispatching Rules

In contrast to static dispatching rules, dynamic dispatching rules are time dependent, which means that the procedures for determining the job sequence and package selection are related to the completion time of the last job on each processor. Under these rules, the jobs are scheduled to a processor one by one. Once a job is scheduled, the priority indices of the remaining unscheduled jobs are recalculated. These rules select a processor according to LS, except for MDD rules, then dispatch the top priority amongst the unscheduled jobs. To facilitate the description of dispatching rules, we assume that LS selects processor p_{ik} so that the processing time of job J_j is t_{jik} and C_{ik} is the completion time of the last job on that processor.



Heuristic: FBS
PKG = package
CPU = processor
PKG_index = $(f_i/b_i)/s_{ik_{avg}}$
Do all PKG exceed its base time
Y
N
Calculate PKG_index Pick the highest one as select_PKG
Apply LS to balance workload
Do all CPU in select_PKG exceed its base time
Y
Pick the second highest one as Select_PKG
N
Pick minimum load PKG,CPU as Select_PKG, Select_CPU
Choose the fastest CPU as Select_CPU
Does select_CPU exceed its base time
Y
Pick the second fastest CPU as Select_CPU
N
Return(Select_PKG,Select_CPU)

Figure 3: Flow chart of FBS.

- *Cost over Time* (COV) [8]: This rule chooses the next job to be processed according to the largest ratio of π_j defined by

$$\pi_j = \begin{cases} w_j/t_{jik}, & \text{if } \ d_j \leq t_{jik} + C_{ik}; \\ (w_j/t_{jik})(t-d_j)/(t-C_{ik}-t_{jik}), & \text{if } \ C_{ik}+t_{jik} < d_i < t; \\ 0, & \text{if } \ t < d_j, \end{cases}$$

where t is the sum of processing times divided by the number of processors, i.e. $t = \sum_{j=1}^{n} t_{jik}/m$.

- *Apparent Urgency* (AU) [22]: This rule chooses the next job to be processed according to the largest ratio of π_j defined by

$$\pi_j = (w_j/t_{jik}) \exp(-(d_j - C_{ik} - t_{jik})^+/\kappa\bar{t}),$$

where $(x)^+ = \max\{0, x\}$, $\bar{t}$ is the average of processing times, and κ is a look-ahead parameter. We set $\kappa = 2$ in the computational experiment.

- *Minimum Slack Time* (MS): This rule chooses the next job for processing according to the largest ratio of π_j defined by

$$\pi_j = \max\{d_j - t_{jik} - C_{ik}, 0\}.$$

- *Modified Due Date* (MDD) [4]: The MDD rule is a combination of the SPT rule and the EDD rule, proposed by Baker and Bertrand [4], for the single-machine tardiness problem. Alidaee and Rosa [1] extended this rule to parallel-machine tardiness problem. This generalized rule can be described as follows.

Step 1 Calculate the completion time of the last job on each processor, i.e. $C_{11}, C_{12}, \ldots, C_{h\mu_h}$.

Step 2 For each processor p_{ik}, repeat the following steps as on a single machine and derive the job G_{ik} and the value $\delta_{G_{ik}}$.

Step 2-1 For the set of unscheduled jobs, u, partition them into u_1 and u_2 such that

$$u_1 = \{j \in u : C_j = C_{ik} + t_{jik} > d_j\}, \text{ and}$$
$$u_2 = u - u_1 = \{j \in u : C_j = C_{ik} + t_{jik} \leq d_j\}.$$

Step 2-2 Find two sets, γ and λ, of the jobs such that

$$\gamma = \{j \in u_1 : w_j/t_{jik} = \max_{q \in u_1}\{w_q/t_{qik}\}\}, \text{ and}$$

$$\lambda = \{j \in u_2 : d_j = \min_{q \in u_2}\{d_q\}\}.$$

Step 2-3 Find two jobs, a and b, such that

$$a = \arg\min_{q \in \gamma}\{w_q C_q\}, \text{ and}$$

$$b = \arg\min_{q \in \lambda}\{w_q d_q\}.$$

Step 2-4 Choose the next job, $G_{ik} \in \{a, b\}$, for the schedule that satisfies

$$\delta_{G_{ik}} = \min\{w_a C_a, w_b d_b\}.$$

Step 3 Schedule the next job G_{yz} on the processor p_{ik} such that

$$\delta_{G_{yz}} = \min_{1 \le i \le h, 1 \le k \le \mu_i} \{\delta_{G_{ik}}\}.$$

Combining the above rules, we have 28 heuristics, as listed in Table 2, for producing approximate solutions.

Table 2: List of proposed heuristics.

	Job sequence	PKG selection	Total
Static	EDD WML LLF SLK MRR TPI	LS LS' FBS FBS'	24
Dynamic	COV AU MS	LS	4
	MDD		

5 Meta-heuristics

In this section, two meta-heuristics, tabu search and genetic algorithm, are presented for solution finding.

5.1 Tabu Search

Tabu search (TS) [11, 12] has been extensively used to solve many combinatorial optimization problems [6, 7]. TS starts with an initial solution derived randomly or from other heuristics, then moves iteratively to another solution, and stops when certain stopping criteria are satisfied. During each iteration, TS picks one move from the neighbors of the incumbent solution to perform. This move is the best one that is not tabu-active or the next one that satisfies the aspiration criteria. Once the move is chosen, the reverse direction of the chosen move is recorded in the tabu list. Any entry in tabu list is inaccessible until the tabu status has been relaxed. By managing the tabu list, the search procedure can efficiently seek the solution space without revisiting the solutions already encountered. The TS mechanism can guide the search space toward a promising area and find a better solution. The remainder of this section describes the key components of TS.

5.1.1 Initial solutions

We use the solutions generated by heuristics in section 4 as the initial solution. Since the heuristics differ from one another in their solution qualities, preliminary tests are conducted to distinguish the ability of TS on improving different initial solutions.

5.1.2 Neighborhood structures

In each iteration, a set of neighbor solutions is derived from the incumbent solution. We use two types of most frequently used moves to generate neighbors.

- Swap: The swap operations are achieved by exchanging the positions of two jobs regardless of whether they reside in the same processor or not. The sequences of all the other jobs on any processors remain the same. The size of the neighborhood solutions of this type is $n \times (n-1)/2$.

- Insert: The insertion neighbors are accomplished by removing a job from its processor, and inserting it to a random position of a different processor. The neighborhood size of this type is $n(\sum_{i=1}^{m} \mu_i - 1)$.

Instead of using a single type of neighbors, our neighborhood strategy is to hybridize these two types into the iteration process at a specific ratio θ. For example, assume that TS totally runs for 100 iterations, and the ratio of using the first type neighborhood is 30% ($\theta = 0.3$). Then, we will have 30 iterations to generate neighbors with swap moves and the remaining iterations with insertion moves.

5.1.3 Tabu list

Tabu list is designed as a first-in-first-out queue and is used to avoid cycling to solutions that have been encountered in earlier iterations. Any move that is tabu-active is not allowed to take, unless it satisfies the aspiration criteria. In each iteration, TS selects the best but not tabu-active move to perform. If the best move is tabu-active and does not satisfy the aspiration criteria, then TS selects the next move to examine. When a move is made, its reverse move is entered into the tabu list. The oldest entry would be deleted if the queue is full. The size of the tabu list, called the tabu tenure, is usually set to be between five to twelve as suggested in [11].

Since there are two types of moves to generate neighbors, we can use a single tabu list to record two types of moves. Thus, the type of moves that can improve the solution quality is more frequently recorded. However, we can use double tabu lists to record the prohibit moves separately. In the our experiments, different parameters of tabu list will be examined, including single or double lists and tabu tenure.

5.1.4 Perturbation

We apply perturbation to the processors for adjusting the sequences of jobs while keeping the allocation of jobs to processors. Since the acquisition cost is determined by the makespan of processors in each package, changing the job precedence in each processor will not increase the total acquisition cost but may reduce the total weighted tardiness. We adjust the job sequences by the following two rules:

- EDD: The jobs on each processor are rescheduled in non-decreasing order of d_j.
- *Apparent Tardiness Cost* (ATC): Vepsalainen and Thomas [26] devised ATC that merges the characteristics of WSPT and MS rules. The jobs on processor p_{ik} are rescheduled according to the following index:

$$I_j(t) = (w_j/t_{jik}) \exp(-(d_j - C_{ik} - t_{jik})^+/\kappa \bar{t}_u),$$

  where $\bar{t}_u$ stands for the average processing time of the unscheduled jobs on that processor, and the look-ahead parameter $\kappa = 2$.

An example using the parameter values in Table 1 is given in Figure 4. The acquisition costs of three schedules are all 25. The original schedule S_0 has a weighted tardiness of 91. After using EDD to reschedule each processor, the sequences of processors p_{11} and p_{21} are reversed, reducing the weighted tardiness to 84. In schedule S_{ATC} yielded by applying ATC to S_0, only the sequence of p_{11} is updated and the total weighted tardiness is reduced to 82. The above example clearly indicates that applying perturbation could decrease the total weighted tardiness without increasing the acquisition cost.

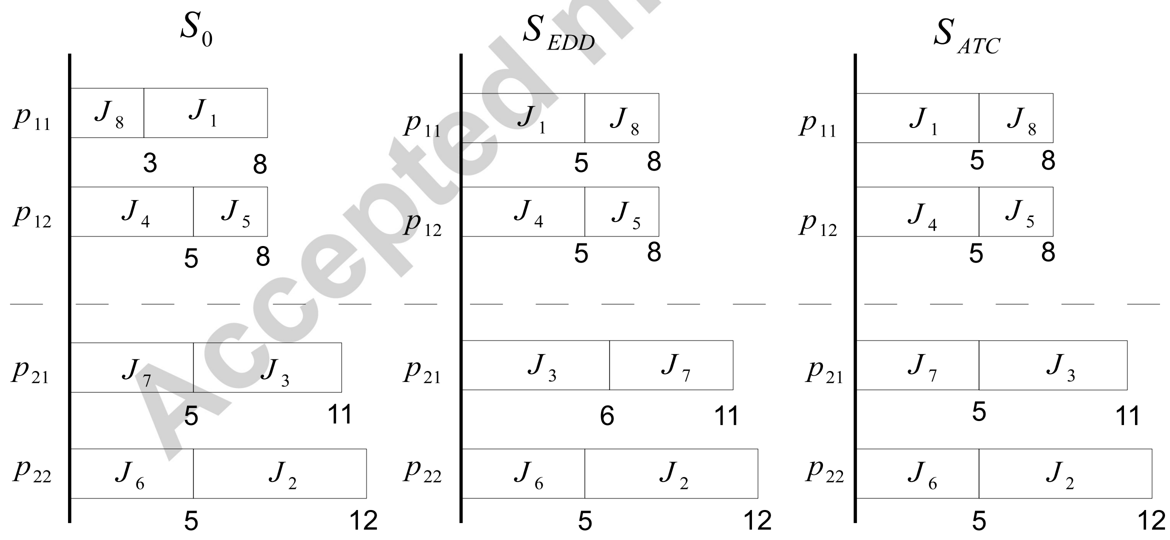


Figure 4: Example of perturbation results.

5.1.5 Aspiration and stopping criterion

The aspiration criteria are used for TS to overrule the tabu-active restriction so as to escape from local optima. The objective value is used to measure the aspiration level of a neighborhood solution in our program. The stopping criterion for the search process used in our experiment is that the number of iterations reaches 100.

5.2 Genetic Algorithm

Unlike TS that manipulates one solution with its neighbors in one iteration, Genetic Algorithm (GA) [16] handles a population containing multiple possible solutions. The members of a population, called individuals, are represented abstractly as chromosomes. It starts with an initial population and generates successive populations through genetic operators. The operators combine or mutate these chromosomes to generate new individuals for the next iteration. This procedure iteratively works until the stopping criteria are met, and each iteration is referred to as a generation. In each generation, the better individuals are more likely to survive or to mate with others while the least fit individuals tend to vanish. For each individual, we use the objective value to represent the fitness value. In our computational experiments, the stopping criterion is that the number of generations reaches 100, which is determined through extensive preliminary tests. The major elements are elaborated in the following sections.

5.2.1 Encoding scheme

We use two chromosomes to represent one individual. The first chromosome, *Seq*, indicates the job sequence on each of the m processors. Each bit in chromosome *Seq*, called gene, represents an index of a job. The second chromosome, *Num*, shows the total number of jobs on each processor. For example, schedule S_1 in Figure 1 is represented as $\{8,1,4,5,7,3,6,2\}$ for chromosome *Seq*, and $\{2,2,2,2\}$ for chromosome *Num*. Schedule S_2 is represented as $\{1,2,3,8,5,7,6,4\}$ for chromosome *Seq*, and $\{2,3,2,1\}$ for chromosome *Num*. Using these two chromosomes can represent a unique individual and can easily check the feasibility. The chromosome *Seq* is a permutation of consecutive integers from one to n. If any repetitive number occurs in *Seq*, it is infeasible. The summation of every gene in chromosome *Num* is equal to n. If the summation is not equal to n, it is infeasible.

5.2.2 Initial population

We use the solutions generated by the heuristics addressed in section 4 to construct the initial population. In order to avoid the case that some individuals with high fitness values are eliminated too early, we reproduce the initial solutions four times. Also, we set the population size equaling 112 in the reproduction step.

5.2.3 Selection

Many selection methods have been proposed in the literature, and the remainder stochastic sampling without replacement [13] is used in our experiment settings of GA. Each individual i is selected $\lfloor {}^{\varphi_i}/_{\bar{\varphi}} \rfloor$ times to be included in the next generation, where φ_i is the fitness value of individual i and $\bar{\varphi}$ is the average fitness value of the population in the current generation. The remainder of the offspring is chosen from the individuals with the highest offspring counts $({}^{\varphi_i}/_{\bar{\varphi}})$. Once the individual is selected, its offspring count is set to be zero. This process repeats until the next population is full.

5.2.4 Genetic operators

The genetic operators are analogous to biological behavior. They are applied to the individuals after the selection procedure is completed. The next generation is then bred by these operators. The most commonly used genetic operators in the new generation formation are crossover and mutation.After these two operators complete their task, the old individuals are eliminated and replaced by the new individuals for the next generation. Thus, we can derive multiple new solutions in one iteration. The following subsections give the details of these operators.
(1) *Crossover*
To vary the chromosomes for potential diversity in the next generation, crossover operators combine two individuals as parents to produce another two individuals as offspring. Crossover in each generation is applied with a probability $Prob_c$, that is, some parents may not breed any offspring. We use $Prob_c \in \{0.7, 0.8, 0.9\}$ in the program. The new individuals in the next generation are composed of the offspring and the parents that did not breed. The crossover operators for *Seq* and *Num* are given as follows:

(i) Crossover operator of *Seq*:

- Partially matched crossover (PMX): Two randomly selected points are chosen, and PMX is invoked by pairwise interchanges. The matching pair inside the two crossover points exchange their values position by position and repair the repetitive value to a proper one.

- Order crossover (OX): The portion between two randomly selected crossover points is replaced by another parent. The rest is filled up by the remaining genes with its original order.

(ii) Crossover operator of *Num*:

- Single point crossover: A randomly selected point is assigned to both parents. The data in front of this point, is swapped between the two parents.

- Two point crossover: Swapping the data between two randomly chosen points.

- Uniform crossover: Preserving the even bits of the original parents for the offspring, and swapping the odd bits.

The crossover operator applied on *Seq* and *Num* are different due to the structures of the parameter values. The operator of *Num* could generate some infeasible solutions, i.e. the summation of every gene in *Num* is not equal to n. To fix them, we have some repairing functions which will be introduced later. In addition, a hybrid operator is used to perform crossover in the experiment, and each operator type has the same probability to be selected. For instance, in the hybrid crossover operator of *Seq*, PMX and OX both have a probability of 50% to be chosen.

(2) *Mutation*

To introduce new information into individuals, mutation may be applied after crossover. In our experiment, mutation operates on a generation with a probability $Prob_m \in \{0.01, 0.1\}$. Three simple mutation operators are adopted in the framework of GA.

- Swap: Randomly assign two genes in one offspring, and change the values of these genes.
- Inverse: The sequence of data between two randomly selected points in a chromosome will be reversed to generate a new chromosome.
- Shift: Rotate as if the left and right ends of this chromosome were connected until a randomly selected gene reaches the first gene position.

These operators simply change the sequence of genes within a chromosome. Thus, infeasibility will not arise. Like the hybrid strategy of crossover, we also use the same hybrid mechanism for mutation.

(3) *Repairing function*

While crossover operators could generate some infeasible solutions, a feasibility repairing function is needed. There are obviously two cases of infeasible solutions: (i) Too large: The summation of every gene in *Num* is larger than n; (ii) Too small: The summation of every gene in *Num* is smaller than n. Both of these two cases can be easily repaired by subtracting or adding random gene(s) until the summation is equal to n.

5.2.5 Perturbation

We use the same perturbation mechanism as in TS to reduce the total weighted tardiness of each processor. However, applying perturbation to the individuals of a population could reduce the diversity and lead to inferior improvements. Therefore, we only apply perturbation to the solutions for predicting the fitness value that each individual can reach under the condition that the allocation of jobs to processors remains unaltered.

6 Computational Experiments

This section is dedicated to the computational study conducted for examining the performance of the heuristics and meta-heuristics. The platform of the experiments is a desktop computer with an Intel Core2 Quad 2.40GHz CPU and 2GB RAM. The operating system is Microsoft Windows 7 and the programs are coded in Microsoft C# .NET Framework 4.0 environment. The data generation scheme of the experiments followed by the results and the analyses are described in the following sections.

6.1 Data Generation Scheme

Since the studied problem is different from the traditional parallel-machine scheduling problem, we do not adopt the existing benchmark instances but generate ad hoc test instances. We use several probability distribution functions, as suggested in [14], to sample the job data and a simple pricing scheme to describe the price of each package. We set three problem sizes $(n, h, \mu_i) \in \{(20, 2, 2), (30, 3, 2), (40, 4, 2)\}$, and apply the following schemes to generate the instances.

6.1.1 Characteristics of job parameters

Three probability distributions, including uniform distribution, normal distribution and beta distribution, with different parameters are used to generate job loads, as follows:

- Uniform, $U[lb, ub]$: The job loads are uniformly drawn over the interval $[lb, ub]$.
- Normal, $N(\nu, \sigma)$: The job loads generated by the normal distribution with two parameters ν and σ. The mean ν measures the location of the distribution, and the standard deviation σ measures its spread.
- Beta, $B(a_1, a_2, lb, ub)$: The job loads are approximately reflected by the beta distribution with shape parameters a_1 and a_2 over the interval $[lb, ub]$. The shape of the beta density function depends on the values of a_1 and a_2. When $a_1 > a_2$, the curve skews toward upper bound ub, and the curve skews toward lower bound lb while $a_1 < a_2$.

In addition, we separate the job loads into three groups by their upper bounds. This classification is used to discriminate the abilities of the proposed methods in dealing with different loads. The loads of the first group are less than or equal to 10, and those of the second group are not greater than 50. The upper bound of the third group is 100. The weights of jobs are generated by the distributions as mentioned above with the upper bounds equal to 10. Another method to sample the weights is scaling proportionally the loads to the interval $[1, 10]$, namely, the jobs with larger loads are more important than smaller ones.

The due dates of jobs are sampled according to job loads in order to give reasonable ranges. The first method is to generate the estimate processing times, i.e. ℓ_j/s_{min}, plus the additional tolerant range produced by a uniform distribution with the upper bound equal to 10 and a normal distribution with mean and standard deviation equal to 5 and 2, respectively. The second method is to use a function of processing times to compute the upper and lower bounds of an uniform distribution. We use two functions to compute the bounds as follows.

- H_{est}: The due dates are randomly drawn from $U[\beta_1 H_{est}, \beta_2 H_{est}]$. We let $H_{est} = \sum_j(\ell_j/1.5)$ and $(\beta_1, \beta_2) = (0.05, 0.2)$ in the experiment.
- Due factor: The due dates were randomly drawn from $U[1, 2H_{est}/(mq)]$, where $q \in \{2, 8\}$ is a constant factor for controlling the tightness of the due-date range.

The above generation schemes are summarized in Table 3.

Table 3: Generation of job parameters.

Load	Group 1	$U[1, 10]$	$N(5, 2)$	B(1,5,1,10) B(5,1,1,10)
	Group 2	$U[1, 50]$	$N(25, 4)$ $N(25, 8)$	B(1,5,1,50) B(5,1,1,50)
	Group 3	$U[1, 100]$	$N(50, 8)$ $N(50, 16)$	B(1,5,1,100) B(5,1,1,100)
Weight	$\propto \ell_j$	$U[1, 10]$	$N(5, 2)$	B(1,5,1,10) B(5,1,1,10)
Due	$U[1, 10]+\ell_j/1.5$	$N(5, 2)+\ell_j/1.5$	$U[\beta_1 H_{est}, \beta_2 H_{est}]$	Tight, $q = 8$ Loose, $q = 2$

6.1.2 Characteristics of package parameters

The speeds of processors are randomly drawn from the log-normal distribution with its mean and standard deviation respectively equal to 0.5 and 0.2, which produces the values within the interval $[1, 2]$. For the simplicity in method comparison, we set the price according to the average speed of processors in the package. The average speeds for all packages are categorized into four groups. The packages in the group with a lower processing capacity are cheaper in unit cost c_i and fixed cost f_i, but has a longer base time b_i. On the other hand, the package price for the group with the fastest processors is higher than others. The pricing scheme of the experiments is shown in Table 4.

Using these methods, we generate 10 independent sets for each problem size $(n, h, \mu_i) \in \{(20, 2, 2), (30, 3, 2), (40, 4, 2)\}$ and 45 instances in each set, thus a total of 1350 instances. The instances and the complete instance generation schemes can be found in the website (`http://people.cs.nctu.edu.tw/~cnyang/data.php`).

6.2 Experiment Results

In the experiments on the capability of heuristic, 28 heuristics were applied to all instances with $\alpha = 0.5$. The results are presented in Tables 5 and 6. The column entitled

Table 4: Price characteristics of packages.

Average speed	[1,1.25)	[1.25,1.5)	[1.5,1.75)	[1.75,2]
Unit cost (c_i)	2	4	6	8
Fixed-charge cost (f_i)	25	36	45	48
Base time (b_i)	25	18	15	12

"best%" records the percentage with which the heuristic achieved the minimal objective value among all results. We can see from the results that the dynamic dispatching rules outperform other heuristics, especially COV and AU. The results could be attributed to the facts that the dynamic rules takes more factors into account for computing the priority indices than the static rules, and that the dynamic rules incorporate the completion time of the last job on each processor. The priority index of a job may increase or decrease at different time points. Thus, dynamic rules could deal with the urgent jobs earlier. On the contrary, static rules determine the job dispatching sequence at the beginning, and cannot adjust the sequence according to the urgency of jobs.

The average objective values of COV and AU are lower than others, but EDD and WML can also attain better values in comparison to LLF. The results of LLF are worst under the condition $\alpha = 0.5$. However, if we only consider the acquisition cost, i.e. $\alpha = 0$, LLF can contribute a better solution than others. Since LLF is modified from LPT rule which is usually used to minimize makespan, the elapsed time of each package is shorter than other heuristics. Note that the running times required by the heuristics are negligible and not shown in the tables.

In the experiments on the performances of TS and GA, we conducted 100 independent runs for each instance with each parameter setting. The independent runs were designed to avoid the statistical error that might be produced by the randomness in the execution course of TS and GA. We chose 45 instances (15 for each problem size (n, h, μ_i)) from the instance pool, which were all generated by the uniform distribution. We use the average improvement percentage (Avg impr%), the maximum improvement percentage (Max impr%), the total execution time in seconds of 100 runs (Time), and the standard deviation in run time of 100 runs (Std) to evaluate the results of different parameter settings. The improvement of meta-heuristic X over the initial solution is computed by

$$\frac{\mathrm{Z}_{Initial} - \mathrm{Z}_X}{\mathrm{Z}_{Initial}} \times 100\%,$$

where $\mathrm{Z}_{Initial}$ and Z_X are the objective value of the initial solutions and that yielded by X, respectively.

6.2.1 GA

In the preliminary test, different types of crossover and mutation operators did not introduce significant discrimination to the results in the aspects of Time, Std or impr%.

Table 5: Results of static dispatching rules ($\alpha = 0.5$).

$\alpha = 0.5$	n=20		n=30		n=40		Average	
	Average Z	best%	Average Z	best%	Average Z	best%	Average Z	best%
EDD + LS	1381.34	1.1%	2193.94	1.8%	3001.98	1.3%	2192.42	1.4%
EDD + FBS	1394.80	0.7%	2223.94	0.4%	3033.59	1.1%	2217.44	0.7%
EDD + LS'	1362.76	2.9%	2169.13	6.7%	2961.87	4.9%	2164.59	4.8%
EDD + FBS'	1379.50	0.7%	2200.95	0.9%	3019.71	0.9%	2200.05	0.8%
WML + LS	1170.15	5.6%	1855.37	5.1%	2515.54	3.3%	1847.02	4.7%
WML + FBS	1184.69	3.3%	1876.78	4.9%	2551.56	3.8%	1871.01	4.0%
WML + LS'	1150.55	17.3%	1837.14	11.3%	2477.63	18.0%	1821.77	15.6%
WML + FBS'	1168.58	5.1%	1866.39	5.1%	2529.17	5.1%	1854.71	5.1%
LLF + LS	2323.67	-	3690.70	-	5019.07	-	3677.82	-
LLF + FBS	2320.28	-	3684.37	-	5012.95	-	3672.53	-
LLF + LS'	2320.87	-	3688.84	-	5009.43	-	3673.05	-
LLF + FBS'	2318.49	-	3684.38	-	5005.88	-	3669.58	-
SLK + LS	1603.61	0.2%	2539.54	-	3276.33	-	2473.16	0.1%
SLK + FBS	1606.70	-	2548.08	-	3297.69	-	2484.16	-
SLK + LS'	1583.96	-	2519.06	-	3236.58	-	2446.54	-
SLK + FBS'	1597.46	-	2539.23	-	3281.65	-	2472.78	-
MRR + LS	1707.46	-	2814.04	-	3796.87	-	2772.79	-
MRR + FBS	1714.84	-	2830.39	-	3812.56	0.2%	2785.93	0.1%
MRR + LS'	1686.24	-	2786.32	-	3739.34	-	2737.30	-
MRR + FBS'	1698.45	-	2812.45	-	3774.77	-	2761.89	-
TPI + LS	1692.73	-	2813.46	-	3782.38	-	2762.86	-
TPI + FBS	1695.24	-	2831.36	-	3801.18	-	2775.93	-
TPI + LS'	1682.83	-	2785.23	-	3723.56	-	2730.54	-
TPI + FBS'	1690.07	-	2809.30	-	3757.01	-	2752.12	-

Table 6: Results of dynamic dispatching rules ($\alpha = 0.5$).

$\alpha = 0.5$	n=20		n=30		n=40		Average	
	Average Z	best%	Average Z	best%	Average Z	best%	Average Z	best%
COV	1148.67	20.7%	1811.25	21.6%	2450.93	22.7%	1803.62	21.6%
AU	1136.31	32.0%	1801.71	30.4%	2444.38	26.0%	1794.13	29.5%
MS	1627.30	1.3%	2522.95	4.2%	3412.23	2.2%	2520.82	2.6%
MDD	1189.51	9.1%	1885.59	7.6%	2555.00	10.4%	1876.70	9.0%

Therefore, we use hybrid operators of both crossover and mutation throughout the computational experiments.

In the experiments, we compare the different combinations of $Prob_c$,$Prob_m$ and perturbation strategies in order to find an optimal parameter setting for GA in the studied problem. The numerical results are given in Tables 7 to 8 and Figures 5 to 10. We use AU as the initial solution to compute the impr% since AU has been included in the initial population.

Table 7 lists the Avg impr% and Max impr% of all combinations. The charts shown in Figures 5 to 7 are the average results of all the instances with different parameter settings and different perturbation strategies. The results reveal that no single parameter setting is dominant except the setting $(Prob_c, Prob_m) = (0.7, 0.1)$, which performs slightly better than others. However, applying perturbation can make positive contributions to impr%. The charts shown in Figures 8 to 10 are the average results of all the parameter settings for each group under different perturbation strategies. The impr% after perturbation can increase 4% for group 1, and 1% for other groups in comparison with those without perturbation. The causes for this may lie in the fact that the genetic operators involves a certain level of randomness. As long as the points of crossover and mutation operators are not properly randomized, the operators may lead to worse solutions. Also, GA can barely keep the good solution structures. Through the genetic operators, more than half of the jobs could be dispatched to another processor, thus breeding offspring that are drastically different from their parent chromosomes. Consequently, the good schedules of each processor can easily be ruined. Therefore, applying perturbation to every processor may rearrange the solution structures to a better way, and attain more improvement.

The results of Time and Std are provided in Table 8. The relationship between probabilities ($Prob_c$ and $Prob_m$) and execution times can be clearly observed from the table. The higher the probabilities are, the more time we need to finish 100 generations of evolution. If the probabilities are higher, then the chances that individuals conduct crossover and mutation are higher. Thus, the program consumes more time. Applying perturbation strategies also costs more times due to we have to rearrange the sequence of each processor.

The final concern of the discussion is about Std. From the computational results, Std is higher when applying perturbation for the most of the settings. One explanation can

Table 7: Results of genetic algorithm (impr%).

		Group 1, init = 218.77			Group 2, init = 1526.93			Group 3, init = 3227.53		
		None	EDD	ATC	None	EDD	ATC	None	EDD	ATC
$Prob_c$	$Prob_m$	Avg impr%								
0.7	0.01	3.99%	4.26%	8.80%	1.57%	1.57%	1.94%	2.07%	2.17%	2.70%
0.7	0.05	4.27%	4.73%	8.85%	1.63%	1.64%	1.99%	2.21%	2.32%	2.77%
0.7	0.1	4.65%	4.77%	8.99%	1.63%	1.63%	2.00%	2.21%	2.30%	2.78%
0.8	0.01	3.75%	4.00%	8.67%	1.51%	1.51%	1.88%	2.00%	2.13%	2.65%
0.8	0.05	3.98%	4.29%	8.78%	1.56%	1.57%	1.90%	2.09%	2.21%	2.69%
0.8	0.1	4.32%	4.45%	8.83%	1.57%	1.59%	1.95%	2.06%	2.24%	2.72%
0.9	0.01	3.43%	3.69%	8.40%	1.46%	1.44%	1.82%	1.89%	2.03%	2.56%
0.9	0.05	3.76%	4.02%	8.57%	1.51%	1.50%	1.84%	1.90%	2.09%	2.61%
0.9	0.1	3.88%	4.02%	8.67%	1.51%	1.53%	1.84%	1.94%	2.06%	2.60%
		Max impr%								
0.7	0.01	11.22%	11.64%	14.51%	2.55%	2.59%	3.29%	3.37%	3.44%	4.11%
0.7	0.05	11.43%	12.71%	15.00%	2.75%	2.80%	3.61%	3.73%	3.60%	4.20%
0.7	0.1	12.84%	12.43%	15.04%	2.79%	2.80%	3.61%	3.87%	3.79%	4.19%
0.8	0.01	9.81%	10.84%	13.82%	2.26%	2.38%	3.23%	3.50%	3.40%	4.06%
0.8	0.05	11.57%	12.29%	15.04%	2.69%	2.51%	3.46%	3.54%	3.44%	4.20%
0.8	0.1	12.59%	12.16%	14.89%	2.57%	2.76%	3.51%	3.50%	3.74%	4.22%
0.9	0.01	10.88%	11.89%	13.99%	2.37%	2.22%	3.11%	3.29%	3.30%	4.00%
0.9	0.05	11.89%	11.73%	14.23%	2.43%	2.50%	3.13%	3.25%	3.43%	3.96%
0.9	0.1	11.13%	11.47%	14.24%	2.42%	2.55%	3.24%	3.40%	3.37%	4.28%

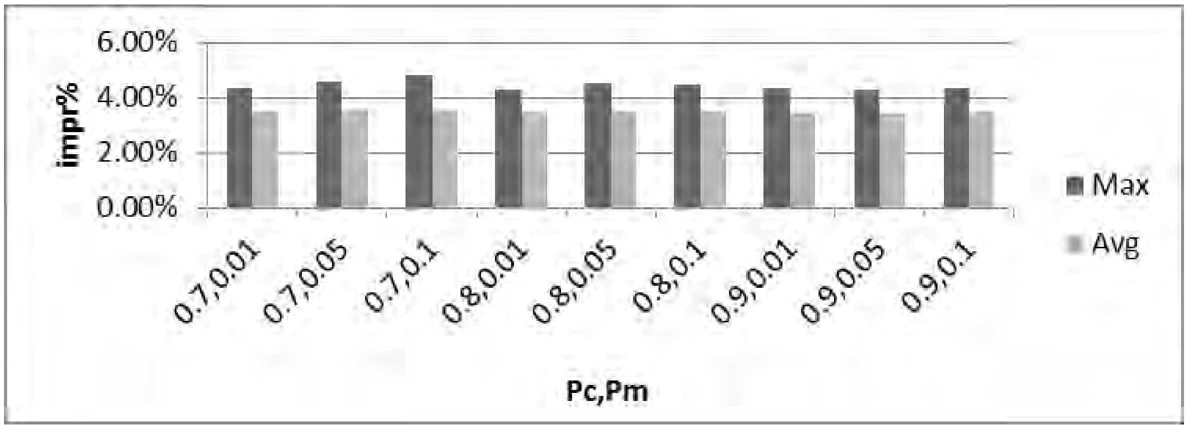


Figure 5: Average impr% of GA without perturbation strategy.

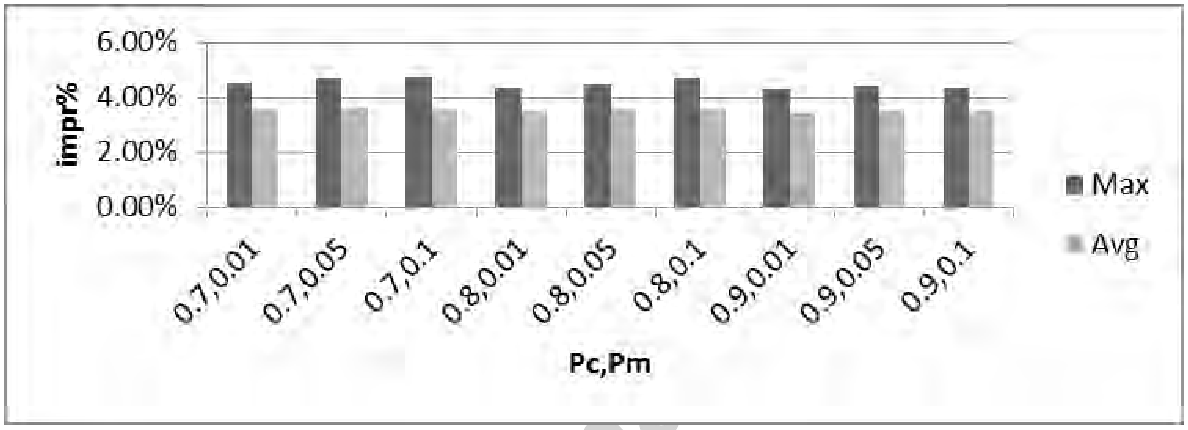


Figure 6: Average impr% of GA with EDD.

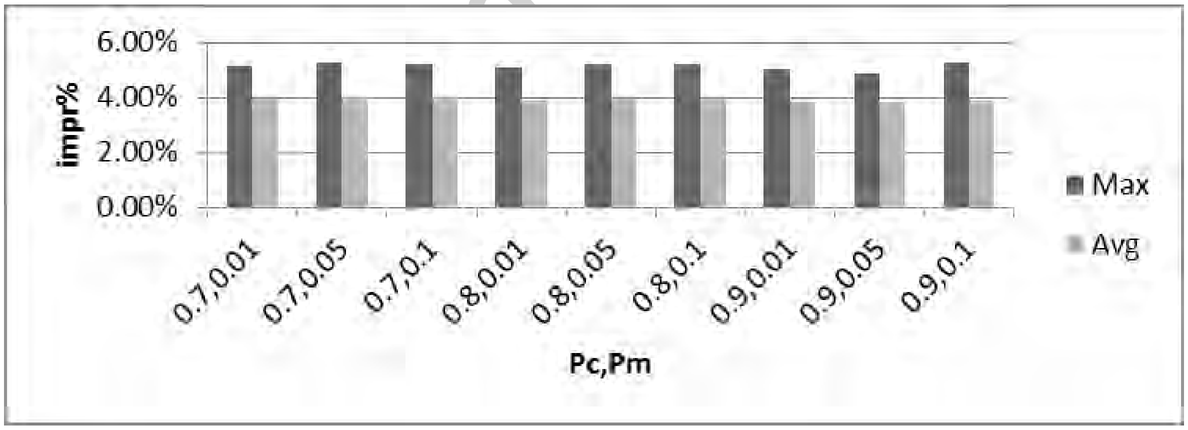


Figure 7: Average impr% of GA with ATC.

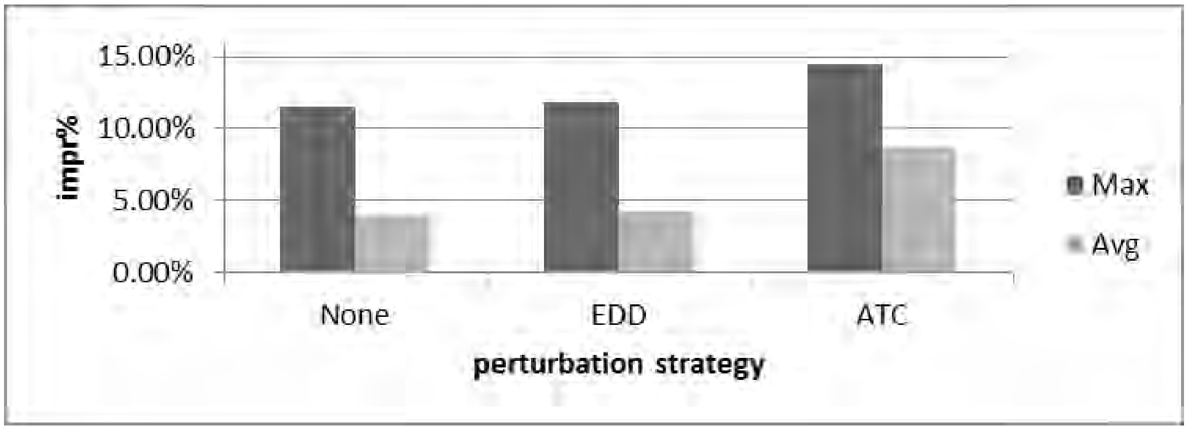


Figure 8: Average impr% of GA with different perturbation strategies (group 1).

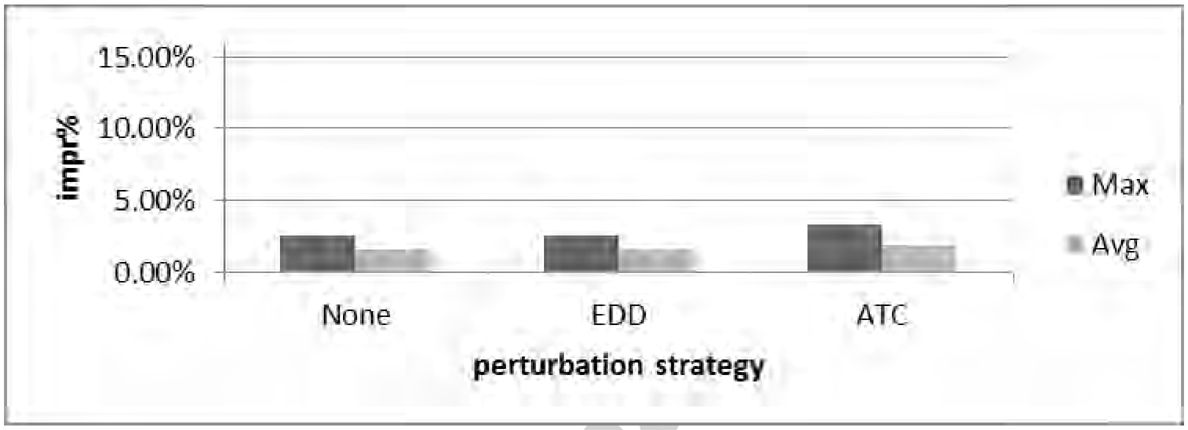


Figure 9: Average impr% of GA with different perturbation strategies (group 2).

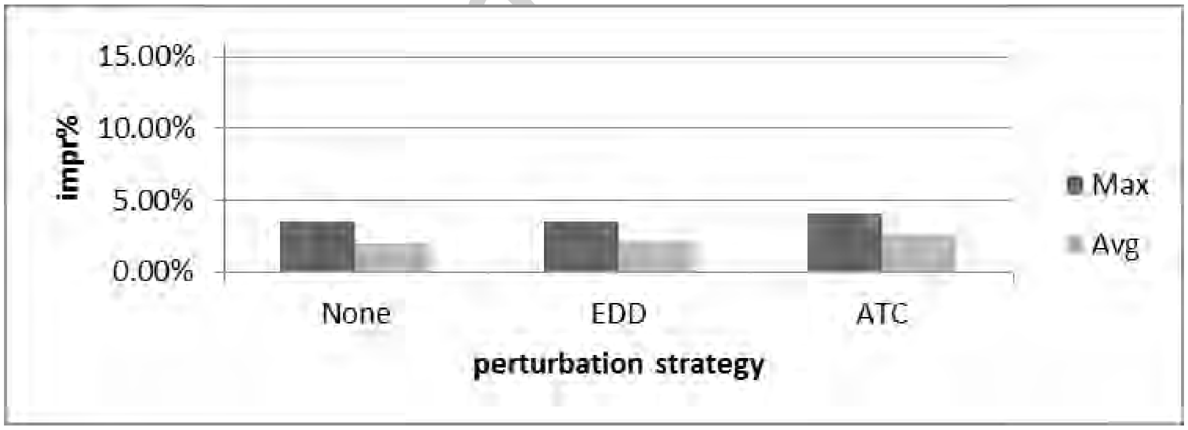


Figure 10: Average impr% of GA with different perturbation strategies (group 3).

be derived from observing some specific instances. One of the instance results is shown in Table 9. From the table, we can see that GA cannot yield any improvement without perturbation, and Std is lower since all the objective values of 100 trials were almost the same. However, applying perturbation may let GA gain improvement through reducing the total weighted tardiness. Thus, the objective values varied and increased the Std.

Table 8: Results of genetic algorithm (Time and Std).

		Group 1			Group 2			Group 3		
		None	EDD	ATC	None	EDD	ATC	None	EDD	ATC
$Prob_c$	$Prob_m$	Time (sec.)								
0.7	0.01	49.87	129.14	171.13	49.76	129.23	172.31	50.65	132.08	174.98
0.7	0.05	148.90	222.41	264.44	150.15	225.83	267.18	152.41	227.90	270.22
0.7	0.1	243.46	319.48	359.94	246.40	320.33	361.71	248.84	325.01	366.48
0.8	0.01	338.20	413.19	455.59	341.42	416.33	457.66	346.16	422.88	464.43
0.8	0.05	433.63	508.91	550.19	436.47	510.80	552.00	441.82	518.01	559.25
0.8	0.1	528.46	602.10	643.19	530.38	604.91	646.68	537.62	613.44	655.33
0.9	0.01	623.26	694.73	735.42	625.71	701.72	742.17	633.23	709.40	751.35
0.9	0.05	713.43	787.42	828.23	720.97	795.67	836.88	729.36	806.03	846.83
0.9	0.1	806.33	881.09	921.74	815.24	888.91	929.98	824.71	901.51	942.51
		Std (sec.)								
0.7	0.01	4.08	4.30	3.69	2.97	3.06	4.82	5.86	6.17	8.64
0.7	0.05	4.34	4.89	3.86	3.49	3.76	5.78	7.46	7.56	9.96
0.7	0.1	5.06	4.80	4.08	3.74	3.84	6.16	8.37	7.32	10.36
0.8	0.01	3.82	3.83	3.50	2.42	2.53	4.86	6.28	5.77	8.50
0.8	0.05	4.22	4.32	3.79	3.19	2.97	5.19	6.93	6.33	9.17
0.8	0.1	1.80	4.56	3.84	3.93	3.51	5.69	7.16	7.64	10.02
0.9	0.01	3.69	3.73	3.31	2.26	1.85	3.99	5.48	5.05	7.73
0.9	0.05	4.18	3.91	3.35	2.55	2.58	4.10	5.91	5.75	7.56
0.9	0.1	3.98	3.89	3.42	2.43	2.67	4.36	5.83	5.41	9.29

Table 9: One instance result of $n = 30$.

		None	EDD	ATC	None	EDD	ATC
$Prob_c$	$Prob_m$	Avg impr%			Max impr%		
0.7	0.01	-	-	0.04%	-	-	2.23%
0.7	0.05	-	-	0.07%	-	-	2.05%
0.7	0.1	-	0.01%	0.01%	-	0.90%	0.62%
0.8	0.01	-	-	0.04%	-	0.23%	1.13%
0.8	0.05	-	-	0.02%	-	-	0.98%
0.8	0.1	-	0.02%	0.04%	-	1.55%	0.76%
0.9	0.01	-	-	0.03%	-	-	1.55%
0.9	0.05	-	-	0.02%	-	-	0.93%
0.9	0.1	-	-	0.02%	-	-	0.99%

6.2.2 Preliminary tests of TS

The preliminary tests were carried out to evaluate the impacts of tabu list and initial solutions. We chose 27 instances (9 for each problem size (n, h, μ_i)) from the instance set, which was generated by uniform distribution, to run the preliminary experiments with $\theta = 0.5$ and no perturbation strategy applied. Different settings of tabu list were examined, including the sizes and the usage of single or double list. We use the difference percentage of two lists (Diff%) and the absolute difference of objective values over the initial solution, which is computed by

$$\frac{|Z_{List_1} - Z_{List_2}|}{Z_{Initial}} \times 100\%,$$

to evaluate the results.

Table 10 shows the comparisons between different types of tabu lists with the same size, same type of tabu lists with different size, and tabu lists with similar sizes. The differences between all the results were within 0.5%, indicating that the influence of tabu list can be neglected. Therefore, we use the single tabu list with size equal to seven in the following experiments.

Figure 11 sketches out the convergence behavior of different initial solutions. We had two initial solutions, LLF and AU. The objective value of LLF is 7070, whereas that of AU is 3505.5. Both of them got the results around 3400 to 3500 in 50 iterations of TS. The results indicated that the qualities of initial solutions had no significant effect on the final results. Thus, we used AU as the initial solution in the following experiments.

6.2.3 TS

In the experiments on the performance of TS, the impact of different values of θ and perturbation strategies were analysed. The computational results are summarized in

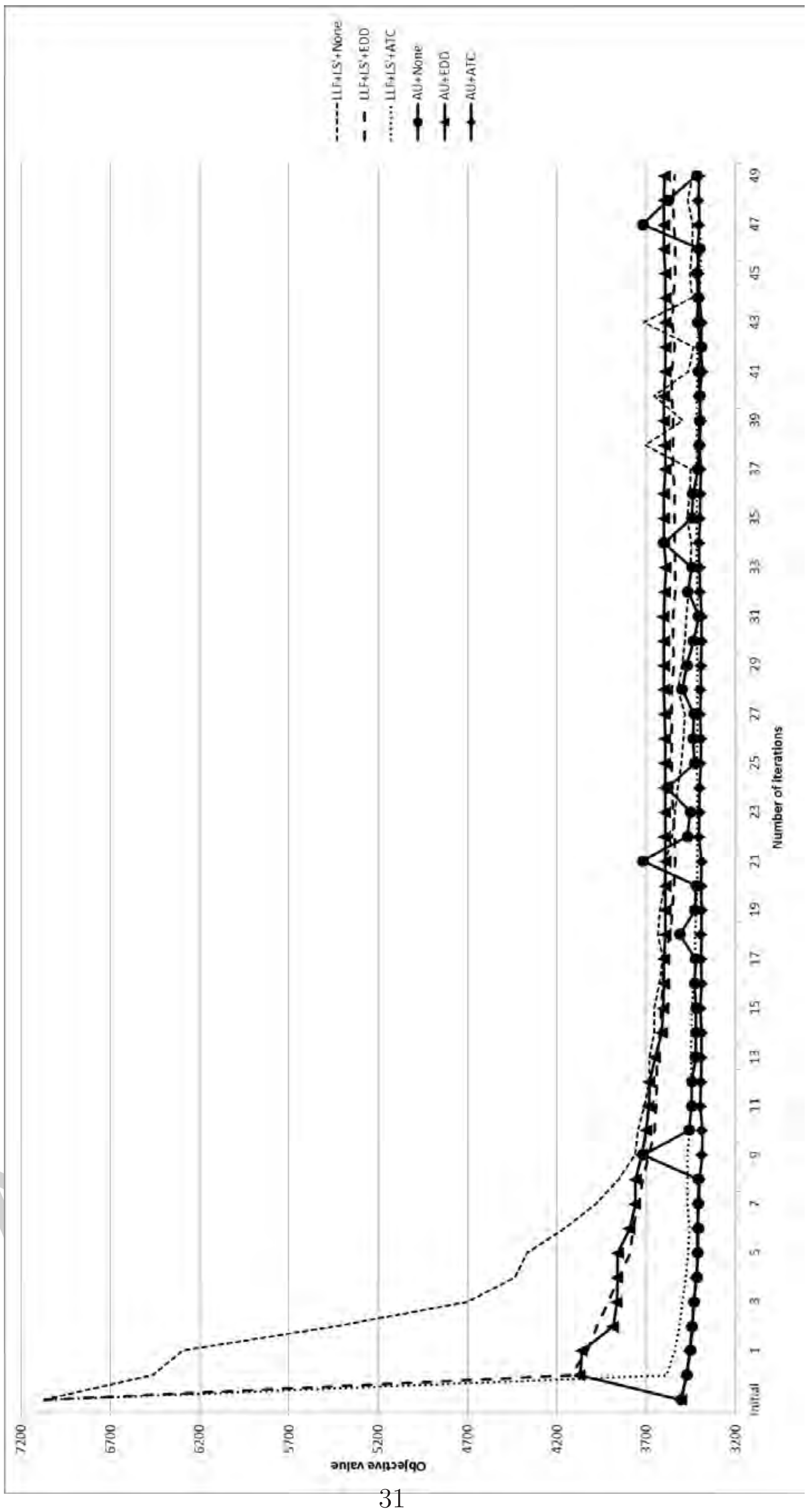


Figure 11: Convergence behaviors.

Table 11 and Figures 12 to 17. The column entitled "None" contains the results of TS without a perturbation strategy. We first examine the improvement in percentages from Figures 12 to 14 which are the average results of all the instances with different parameter settings and different perturbation strategies. When $\theta = 0$ and $\theta = 1$, the impr% are worse than those with other θ-values. The two trends could be attributed to the neighborhood structure. Weakness of TS emerged when only a single type of moves is used to generate neighbors. Swap moves can only exchange the positions of two jobs, but the number of jobs on each processor remains the same as in the initial solution. Thus, the variation of the solution structure is minor. Insert moves can adjust the allocation of jobs to processors, but the ability to reduce the objective value is weaker than swap moves. Swap move can be viewed as a combination of two insert moves, and insert move may increase the objective value at its first step. Thus, insert moves could be terminated and combining the two moves can diminish the disadvantages. As for the ratio θ, there is no clear dominance among $\{0.7, 0.5, 0.3\}$ even though the setting $\theta = 0.7$ exhibits slight advantage than others.

The charts shown in Figures 15 to 17 are the average results of all the hybrid neighborhood settings for each group with different perturbation strategies. The impr% after applying perturbation showed a different outcome when compared to GA. In the results of GA, applying all types of perturbation strategies could achieve more improvement. In TS, however, applying EDD as perturbation could even lead to worse results. Because the moves in TS are smaller than GA, only one or two jobs could be moved to another processor. Since the moves in TS can preserve the solution structures well, we had to apply a better perturbation strategy to adjust the sequence on each processor in order to make significant improvements. ATC is a promising heuristic in the single-machine weighted tardiness problem, and is also a better perturbation strategy for the studied problem.

Second observation is made on the execution time, and the relationship is also clearly showed. The execution times grew when we use more swap moves. The reason may be that a swap move is a combination of two insert moves. The time required to execute a swap move is therefore longer than an insert move. Applying perturbation also takes more times and ATC is much more time-consuming than EDD since the time complexity of ATC is $O(n^2)$ and EDD is $O(n \log n)$. However, comparing the execution times of TS with GA, it is obvious that TS cost less time to satisfy the stopping criterion even though TS may have to check more solutions within an iteration (generation). The main reason would be the genetic operators in GA which have to applied to all individuals when the probability criteria are met. The second reason is due to the perturbation when it is activated. In GA, the perturbation was applied to every solution of every generation to predict whether it is better than the best solution observed so far. However, TS only used perturbation when the move is performed to change the incumbent solution to its best neighbor, and then applied it to the best neighbor. In other words, perturbation performed 112×100 times in GA; but in TS, only 100 times perturbation were performed. This is the reason why TS works more efficiency than GA.

Finally, the last part of the computational study on TS indicates the importance of deploying perturbation strategies. The resulting Std without perturbation were larger than those with perturbation, especially for ATC and group 3. The fact that Std is smaller suggests that the objective values obtained by TS with perturbation are more consistent and less affected by the random factors.

Summarizing the above discussions, we suggest that using the setting $\theta = 0.3$ and adopting ATC as perturbation could be the best scenario setting for TS while taking both impr% and Time into consideration.

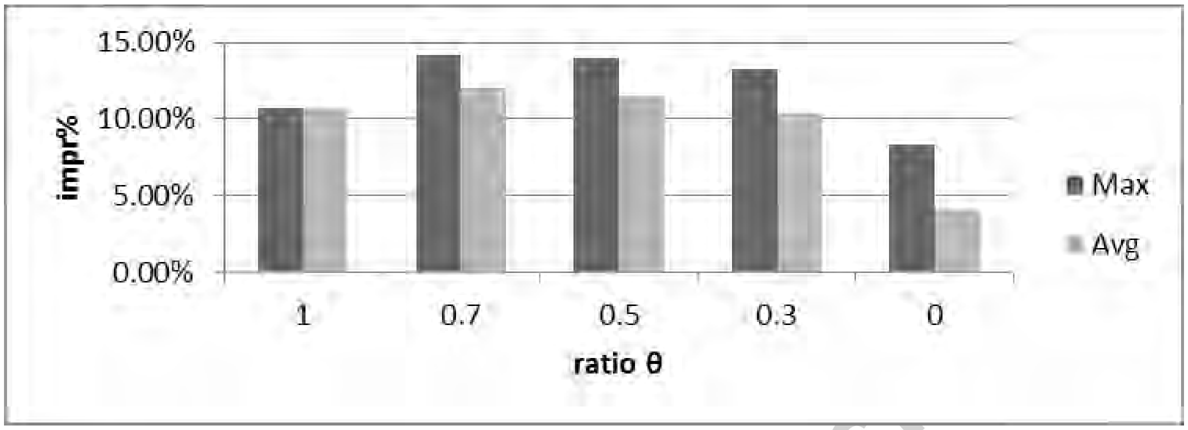


Figure 12: Average impr% of TS without pertubation strategy.

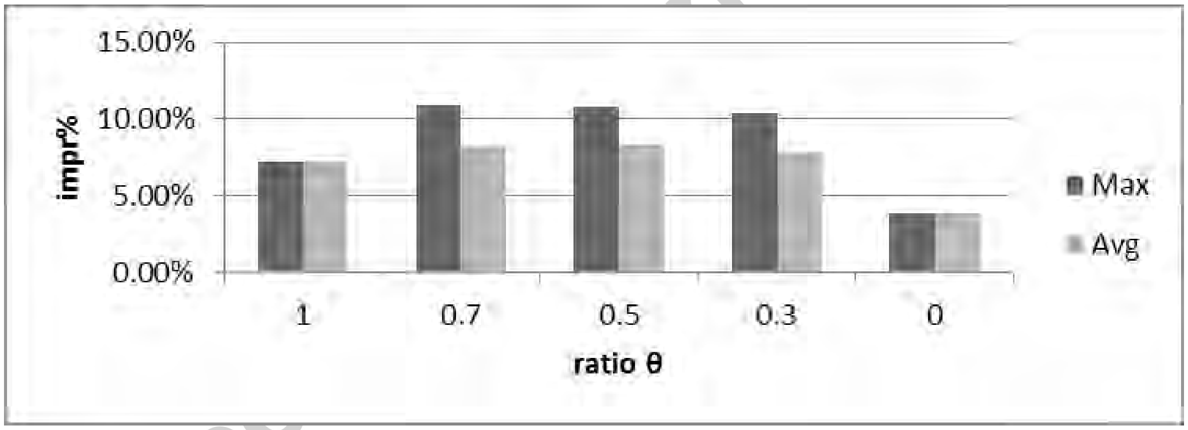


Figure 13: Average impr% of TS with EDD.

6.2.4 Comparisons between GA & TS

In the above experiments, there exist large differences in impr% among the groups. The cause may lie in the data generation scheme. The due-date range ($U[1, 10] + \ell_j/1.5$) is too small for group 2 and group 3. The lack of flexibility in the sequence of each processor

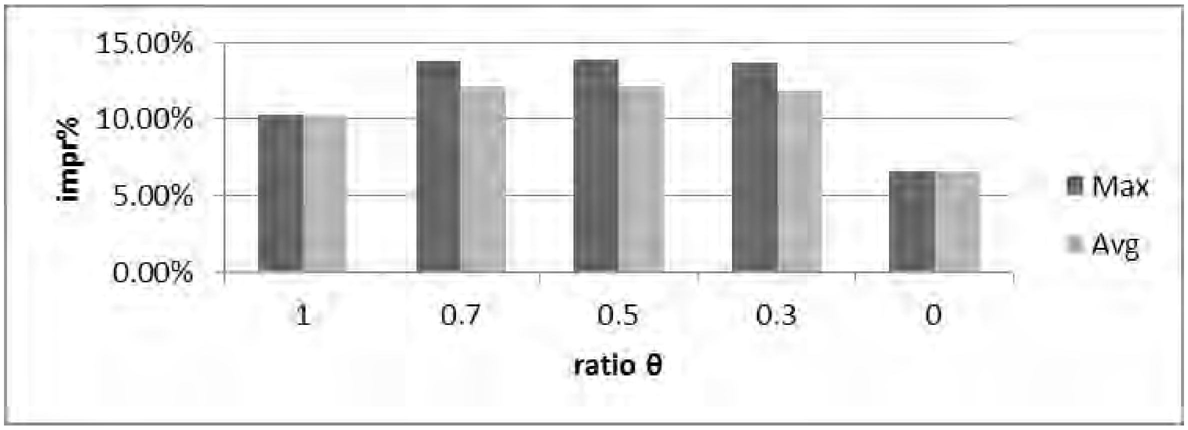


Figure 14: Average impr% of TS with ATC.

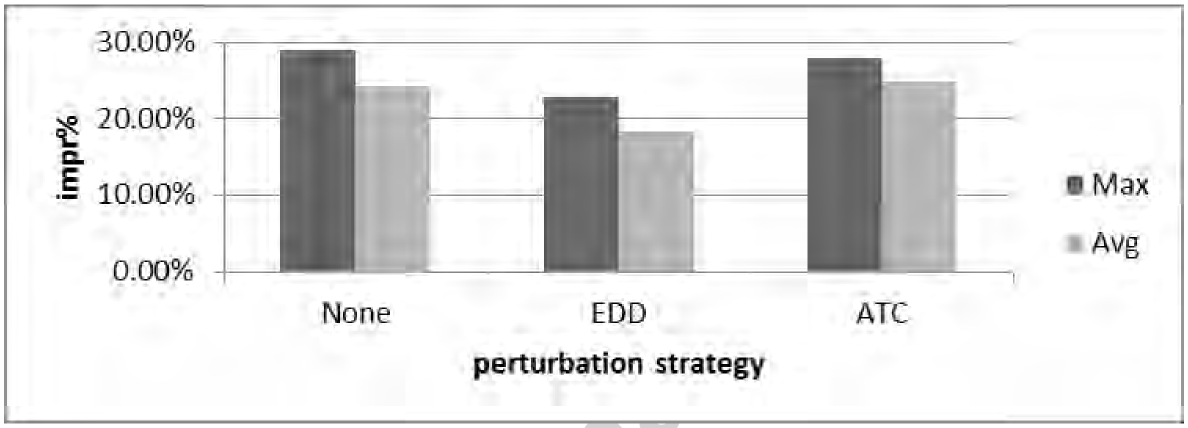


Figure 15: Average impr% of TS with different perturbation strategies (group 1).

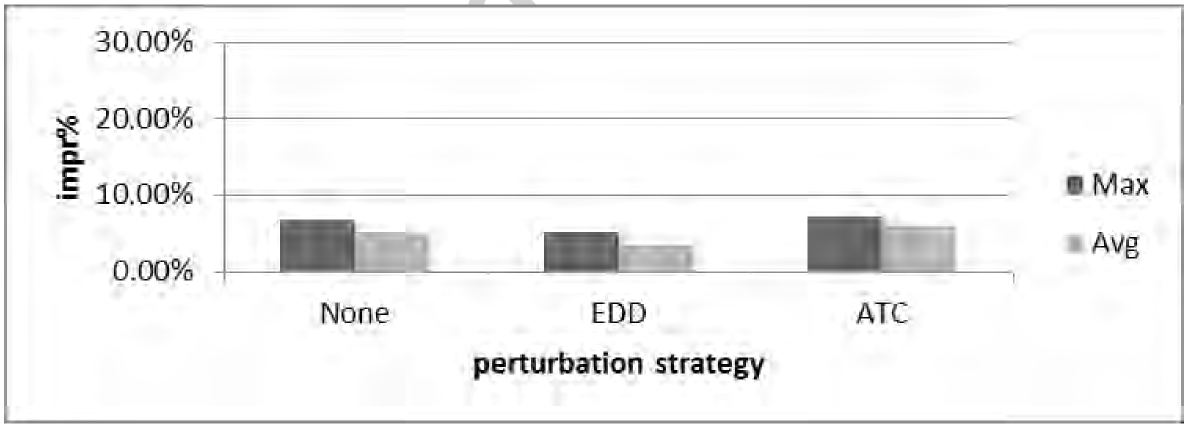


Figure 16: Average impr% of TS with different perturbation strategies (group 2).

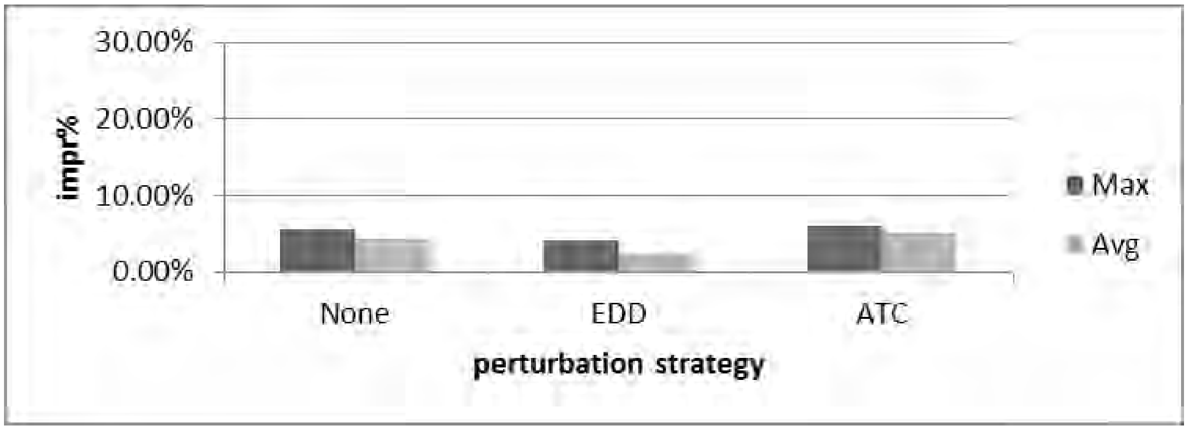


Figure 17: Average impr% of TS with different perturbation strategies (group 3).

makes impr% smaller than group 1. Therefore, we used $U[1, \ell_j/2]+\ell_j/1.5$ to compute new due-date range and produce another 18 instances, and also conducted further experiments to test the impr% for all groups. The experiments were also used for the comparison of TS and GA. In this experiment, the focus is not on the parameter settings. Hence, we only adopt $\theta = 0.3$ with ATC to run TS and $\{Prob_c, Prob_m\} = \{0.7, 0.1\}$ with ATC for GA. The results are presented in Tables 12 and 13.

The computational results shown in Table 12 are the average objective values, Std and Time of the six instances for each problem size. In Table 13, the results are the average and maximum impr% of the six instances for each group. From the tables, the superiority of TS over GA is clear. TS can obtain better solutions within a much shorter time. Moreover, the variance of the TS solutions over multiple trials is also much smaller. As for the aspect of different groups, the differences in impr% are smaller compared to the previous instances. Therefore, as long as the due-date range is reasonable, the proposed methods were able to provide the same improvement.

7 Conclusions

This paper proposed and formulated a new model of parallel-machine scheduling associated with computing resource acquisition decisions. In the model, computing service providers sell computing capacity in packages, each of which bundles several machines together, and charge the customers at different prices according to a new pricing scheme. To a client who needs cloud computing as a tool, reducing the acquisition cost and the tardiness penalty are equally important. Therefore, the objective function is a linear combination of the acquisition cost and the total weighted tardiness.

We provided an integer programming formulation of the studied problem. As enumerating all the solutions is exceedingly time-consuming, we adopted approximation methods to derive near-optimal solutions. Heuristics and meta-heuristics were adopted to tackle the computationally challenging problem. The heuristics used several scheduling rules for computing the priority indices of the jobs and allocating the selected jobs to suitable processors. The computational results indicated that dynamic dispatching rules can exhibit a better solution quality than the static ones. The obtained solutions were then further improved by meta-heuristics (GA and TS). In the meta-heuristics, we used two problem-specific perturbation strategies (EDD and ATC) to rearrange the sequence on each processor for reducing the total weighted tardiness without sacrificing the acquisition cost. With further improvement by meta-heuristics, we were able to get better solutions within a short time.

Future research could be focused on development of exact algorithms, like dynamic programming or branch-and-bound algorithms, to attain optimal solutions. Developing other pricing schemes of the studied model is also an interesting subject to investigate. As indicated by Sundararajan [25], unlimited-usage and usage-based pricing schemes should be mixed and provided. Another area which is worth considering is about the profits of both clients and providers. The clients who want to adopt cloud computing to serve as their platform need to estimate their budgets before purchasing computing resources. Therefore, how to compute the lower bound or upper bound of costs is an important issue of acquisition planning. From the viewpoint of cloud computing service providers, they aim to maximize profits. Consequently, the strategy for bundling the processors with different speeds as a package is another topic to be explore. Examining the profit brought by different pricing schemes, such as non-linear pricing, logarithmic-curve pricing or dynamic pricing schemes, is helpful to the providers. Customized bundle strategies, as suggested by Wu et al. [28], for clients or customers of different groups with large orders is absolutely necessary for attracting more clients. A further research direction is to consider other managerial measures in the same problem setting. For example, in the application context of information retrieval, the quality of service is commonly addressed by the total weighted waiting (or, response) time.

Acknowledgements

The authors are grateful to the anonymous referees for their constrictive comments that have improved the presentation of the paper. This paper was partially supported by the Ministry of Science and Technology of Taiwan under grants NSC100-2410-H-009-015-MY2 and MOST102-2410-H-009-048-MY2.

References

[1] B. Alidaee and D. Rosa (1997), Scheduling parallel machines to minimize total weighted and unweighted tardiness, *Computers & Operations Research*, Vol. 24, No. 8, pp. 775-788.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia (2010), A view of cloud computing, *Communications of the ACM* , Vol. 53, No. 4, pp. 50-58.

[3] M. Azizoglu and O. Kirca (1998), Tardiness minimization on parallel machines, *International Journal of Production Economics*, Vol. 55, No. 2, pp. 163-168.

[4] K.R. Baker and J.M.W. Bertrand (1982), A dynamic priority rule for scheduling against due-dates, *Journal of Operations Management*, Vol. 3, No. 1, pp. 37-42.

[5] Y. Bakos and E. Brynjolfsson (1999), Bundling information goods: Pricing, profits, and efficiency, *Management Science*, Vol. 45, No. 12, pp. 1613-1630.

[6] Ü. Bilge, F. Kıraç, M. Kurtulan and P. Pekgün (2004), A tabu search algorithm for parallel machine total tardiness problem, *Computers & Operations Research*, Vol. 31, No. 3, pp. 397-414.

[7] D. Cao, M.Y. Chen and G.H Wan (2005), Parallel machine selection and job scheduling to minimize machine cost and job tardiness, *Computers & Operations Research*, Vol. 32, No. 8, pp. 1995-2012.

[8] D.C. Carroll (1965), Heuristic sequencing of single and multiple component jobs, Ph.D. Dissertation, Sloan School of Management, M.I.T, Cambrige, MA.

[9] T.C.E. Cheng and C.C.S Sin (1990), A state-of-the-art review of parallel-machine scheduling research, *European Journal of Operational Research*, Vol. 47, No. 3, pp. 271-292.

[10] M. Dessouky, B. Lageweg, J.K. Lenstra and S.L. Van De Velde (1990), Scheduling identical jobs on uniform parallel machines, *Statistica Neerlandica*, Vol. 44, pp. 115-123.

[11] F. Glover (1990), Tabu search: A tutorial, *Interfaces*, Vol. 20, No. 4, pp. 74-94.

[12] F. Glover and M. Laguna (1990), *Tabu Search*, Kluwer Academic Publisher, Hingham, MA.

[13] D.E. Goldberg (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Boston, MA.

[14] N.G. Hall and M.E. Posner (2001), Generating experimental data for computational testing with machine scheduling spplications, *Operations Research*, Vol. 49, pp. 845-865.

[15] J.C. Ho and Y.L. Chang (1991), Heuristics for minimizing mean tardiness for m parallel machines, *Naval Research Logistics*, Vol. 38, No. 3, pp. 367-381.

[16] J.H. Holland (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MA.

[17] C. Koulamas (1994), The total tardiness problem: Review and extensions, *Operations Research*, Vol. 42, No. 6, pp. 1025-1041.

[18] K. Lee, J.Y-T. Leung, Z. Jia, W. Li, M. L. Pinedo and B.M.T. Lin (2014), Fast approximation algorithms for bi-criteria scheduling with machine assignment costs, *European Journal of Operational Research*, Vol. 238, No. 1, pp. 54-64.

[19] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Bruker (1977), Complexity of machine scheduling problems, *Annals of Discrete Mathematics*, Vol. 1, pp. 343-362.

[20] J.Y-T. Leung, K. Lee and M. L. Pinedo (2012), Bi-criteria scheduling with machine assignment costs, *International Journal of Production Economics*, Vol. 139, No. 1, pp. 321-329.

[21] E.R. Jr. Montagne (1969), Sequencing with time delay costs. *Arizona State University Industrial Engineering Research Bulletin*, Vol. 5, pp. 20-31.

[22] T.E. Morton, R.M. Rachamadugu and A. Vepsalainen (1984), Accurate myopic heuristics for tardiness scheduling, GSIA Working Paper No. 36-83-84, Carnegie-Mellon University, PA.

[23] M.L. Pinedo (2002), *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, Englewood Cliffs, NJ.

[24] S.O. Shim and Y.D. Kim (2007), Scheduling on parallel identical machines to minimize total tardiness, *European Journal of Operational Research*, Vol. 177, No. 1, pp. 135-146.

[25] A. Sundararajan (2004), Nonlinear pricing of information goods, *Management Science*, Vol. 50, No. 12, pp. 1660-1673.

[26] A. Vepsalainen and E.M. Thomas (1987), Priority rules for job shops with weighted tardiness costs, *Management Science*, Vol. 33, No. 8, pp. 1035-1047.

[27] G. Wan and X. Qi (2010), Scheduling with variable time slot costs, *Naval Research Logistics*, Vol. 57, No. 2, pp. 159-171.

[28] S.Y. Wu, L.M. Hitt, P.Y. Chen and G. Anadalingam (2008), Customized bundle pricing for information goods: A nonlinear mixed-integer programming approach, *Management Science*, Vol. 54, No. 3, pp. 608-622.

[29] F. Yalaoui and C.B. Chu (2002), Parallel machine scheduling to minimize total tardiness, *International Journal of Production Economics*, Vol. 76, No. 3, pp. 265-279.

Table 10: Preliminary tests on tabu list.

Comparing type	$List_1 - List_2$	Avg Diff%	Max Diff%
Same size	(single,5) - (double,5)	0.13%	0.37%
	(single,7) - (double,7)	0.14%	0.30%
	(single,12) - (double,12)	0.20%	0.27%
Single with different size	(single,5) - (single,7)	0.10%	0.21%
	(single,5) - (single,12)	0.14%	0.34%
	(single,7) - (single,12)	0.09%	0.30%
Double with different size	(double,5) - (double,7)	0.11%	0.34%
	(double,5) - (double,12)	0.21%	0.33%
	(double,7) - (double,12)	0.19%	0.29%
Similar size	(single,7) - (double,5)	0.09%	0.25%
	(double,5) - (single,7)	0.07%	0.24%
	(single,12) - (double,7)	0.09%	0.38%

Table 11: Results of tabu search.

	Group 1, init = 218.77			Group 2, init = 1526.93			Group 3, init = 3227.53		
	None	EDD	ATC	None	EDD	ATC	None	EDD	ATC
θ	Avg impr%								
1	23.13%	14.32%	21.45%	4.82%	3.94%	5.03%	4.18%	3.45%	4.53%
0.7	25.87%	18.56%	25.18%	5.57%	3.50%	6.06%	4.77%	2.75%	5.20%
0.5	24.70%	18.64%	25.13%	5.40%	3.63%	6.10%	4.57%	2.73%	5.21%
0.3	22.31%	18.00%	24.45%	4.79%	3.29%	5.94%	4.01%	2.32%	5.09%
0	8.68%	8.14%	13.84%	1.85%	1.65%	3.25%	1.94%	1.76%	2.82%
	Max impr%								
1	23.13%	14.32%	21.45%	4.82%	3.94%	5.03%	4.18%	3.45%	4.53%
0.7	30.02%	22.86%	28.16%	6.82%	5.54%	7.19%	5.84%	4.37%	6.04%
0.5	29.32%	22.98%	28.19%	6.93%	5.19%	7.26%	5.72%	4.33%	6.12%
0.3	27.88%	22.61%	27.79%	6.57%	4.70%	7.19%	5.41%	3.83%	6.00%
0	17.09%	8.14%	13.84%	4.18%	1.65%	3.25%	3.88%	1.76%	2.82%
	Time (sec.)								
1	135.64	173.61	269.98	135.76	172.89	266.35	138.71	176.67	268.82
0.7	102.69	131.21	203.90	102.79	131.02	201.45	105.36	134.28	203.93
0.5	80.79	102.90	160.07	80.94	103.13	158.17	83.51	105.86	161.07
0.3	58.95	75.30	116.50	58.75	75.03	114.56	61.37	77.63	117.59
0	26.01	32.92	50.36	25.97	32.81	49.93	27.97	35.24	52.81
	Std								
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.7	4.33	4.63	3.38	8.55	14.49	7.43	15.89	17.92	11.74
0.5	4.66	4.55	3.20	10.23	9.64	7.85	17.56	20.11	12.36
0.3	5.57	4.84	3.46	12.76	8.47	8.08	24.54	17.76	13.05
0	6.95	0.0	0.0	13.27	0.0	0.0	26.28	0.0	0.0

Table 12: Comparison results of GA and TS.

	$n = 20,\ m = 2,\ \mu_i = 2$			
Initial =850.33	Min	Avg	Std	Time (sec.)
TS, $\theta = 0.3$, ATC	784.58	796.33	6.52	22.87
GA, 0.7, 0.1, ATC	802.58	824.28	9.52	246.04
	$n = 30,\ m = 3,\ \mu_i = 2$			
Initial =1686.50	Min	Avg	Std	Time (sec.)
TS, $\theta = 0.3$, ATC	1519.83	1535.21	6.91	91.10
GA, 0.7, 0.1, ATC	1614.58	1645.08	7.82	360.01
	$n = 40,\ m = 4,\ \mu_i = 2$			
Initial =1392.92	Min	Avg	Std	Time (sec.)
TS, $\theta = 0.3$, ATC	1191.17	1213.50	8.27	236.01
GA, 0.7, 0.1, ATC	1310.33	1350.94	11.33	479.17
	Total average			
Initial =1309.92	Min	Avg	Std	Time (sec.)
TS, $\theta = 0.3$, ATC	1165.19	1181.68	1.23	116.66
GA, 0.7, 0.1, ATC	1242.50	1273.44	9.56	361.74

Table 13: Comparison results (by group).

Average	Group 1, init = 281.83		Group 2, init = 1258.17		Group 3, init = 2389.75	
	Min	Avg	Min	Avg	Min	Avg
TS, $\theta = 0.3$, ATC	14.87%	13.36%	11.00%	9.50%	10.62%	9.52%
GA, 0.7, 0.1, ATC	6.68%	3.05%	5.27%	3.33%	4.90%	2.47%