A SIMPLE, FAST, AND EFFECTIVE HEURISTIC FOR THE SINGLE-MACHINE TOTAL WEIGHTED TARDINESS PROBLEM

by

HALİL ŞEN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
Spring 2010

A SIMPLE, FAST, AND EFFECTIVE HEURISTIC FOR THE SINGLE-MACHINE TOTAL WEIGHTED TARDINESS PROBLEM

APPROVED BY

Assist. Prof. Kerem Bülbül (Thesis Supervisor)	....
Prof. Gündüz Ulusoy	....
Assoc. Prof. Özgür Özlük	....
Assist. Prof. Hans Frenk	....
Assist. Prof. Hüsnü Yenigün	....

DATE OF APPROVAL: ....

to my family

Acknowledgments

I owe my deepest gratitude to my thesis advisor Assist. Prof. Kerem Bülbül for his invaluable support, supervision, advice and guidance throughout the research.

This thesis would not have been possible without my advisor's encouragement and contribution.

I am indebted to all my friends from Sabanci University for their motivation and endless friendship. My special thanks go in particular to Anıl Can for his endless encouragement and valuable support. Many special thanks go to Ömer M. Özkırımlı, Belma Yelbay, Nurşen Aydın, Mahir U. Yıldırım, İbrahim Muter, Selin Erçil, Çetin A. Suyabatmaz, and all those others who directly and indirectly helped me.

My parents deserve special mention for their invaluable support and gentle love. Lastly, I want to thank TÜBİTAK for the scholarship they gave and for their support to the science society.

A SIMPLE, FAST, AND EFFECTIVE HEURISTIC FOR THE SINGLE-MACHINE TOTAL WEIGHTED TARDINESS PROBLEM

Halil Şen

Industrial Engineering, Master of Science Thesis, 2010

Thesis Supervisor: Assist. Prof. Kerem Bülbül

Keywords: Single-machine scheduling, weighted tardiness, mathematical programming, transportation problem, heuristics, nested schedule.

Abstract

We consider the non-preemptive single-machine total weighted tardiness (TWT) problem with general weights, processing times, and due dates. We first develop a family of preemptive lower bounds for this problem and explore their structural properties. Then, we show that the solution corresponding to the least tight lower-bound among those investigated features some desirable properties that can be exploited to build excellent feasible solutions to the original non-preemptive problem in short computational times. We present results on standard benchmark instances from the literature.

TEK-MAKİNALI TOPLAM AĞIRLIKLANDIRILMIŞ GECİKME PROBLEMİ İÇİN BASİT, HIZLI VE KALİTELİ BİR SEZGİSEL YÖNTEM

Halil Şen

Endüstri Mühendisliği, Yüksek Lisans Tezi, 2010

Tez Danışmanı: Yrd. Doç. Dr. Kerem Bülbül

Anahtar Kelimeler: Tek-makinalı çizelgeleme, ağırlıklandırılmış gecikme, matematiksel programlama, ulaşım problemi, sezgisel, yuvalanmış çizelge.

Özet

Bu tezde, kesintisiz tek makinalı toplam ağırlıklı gecikme problemi genel gecikme ağırlıkları, işlem zamanları ve teslim tarihleri ile birlikte incelenmiştir. İlk olarak bu problem için bir grup kesintili gevşetilmiş alt sınır geliştirilmiş ve bunların yapısal özellikleri araştırılmıştır. Sonrasında, göz önüne alınanlar arasında en gevşek alt sınıra karşılık gelen kesintili çözümün, çok kısa hesaplama süreleri içerisinde asıl kesintisiz problem için çok kaliteli olurlu çözümler oluşturmak üzere kullanılabilecek bazı özellikler sağladığı gösterilmiştir. Literatürdeki standart denektaşı problem örnekleri çözülmüş ve bulunan sonuçlar takdim edilmiştir.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction and Motivation

Single-machine scheduling problems are one of the classical combinatorial optimization problems and are encountered commonly across the manufacturing industry and computer science. This single-machine may be a workbench, a device or a CPU, and the problem is to find the schedule of the tasks that have to be performed by the machine. Also, these tasks may carry a penalty under various objective functions. In practice, this penalty may be due to an article of an agreement or may represent a loss that arises from user dissatisfaction.

In classifying scheduling problems, we follow the three field notation of Graham et al. [24]. The single-machine total weighted tardiness (TWT) problem is represented as $1 \mid \mid \sum_j w_j T_j$ where in the first field, 1 indicates a single machine problem and the last field identifies the objective function to be minimized. It has been already shown that TWT is strongly $\mathcal{NP}$-hard by Lawler, Lenstra et al. in [35, 40].

In TWT, there are n jobs to be processed without interruption on a single-machine that cannot process more than one job at a time. Job $j = 1, \ldots, n$, becomes available for processing at time zero (i.e. the release date $r_j = 0 \ \forall j$). A job j requires a processing time $p_j > 0$ without interruption on the machine, has a due date d_j by which it should be finished and has a positive tardiness cost w_j per unit time if job j completes processing after d_j. We assume that the processing times and due dates are integral. Let s_j be the time at which job j starts processing, $C_j = s_j + p_j$ be the completion time of job j, and $T_j = \max(0,\ C_j - d_j)$ be the tardiness of job j. The objective is to minimize the weighted sum of the tardiness costs of all jobs. Then, our problem is stated as:

$$
\begin{aligned}
&(\mathbf{P1}) \qquad && \min \sum_{j=1}^{n} w_j T_j && && (1.1)\\
& && C_i \le C_j - p_j \quad \textbf{or} \quad C_j \le C_i - p_i && \forall i,\ j,\ i \neq j && (1.2)\\
& && T_j \ge C_j - d_j && \forall j && (1.3)\\
& && C_j \ge p_j + r_j && \forall j && (1.4)\\
& && T_j \ge 0 && \forall j. && (1.5)
\end{aligned}
$$

The constraints (1.2) ensure that jobs do not overlap and constraints (1.4) are the release date constraints. The tardiness of a job is computed by constraints (1.3) and (1.5).

1.1 Contributions

The aim of the study is to develop a fast and effective heuristic for the TWT problem. The following list shows the contributions of this study:

- The lower bound that we develop belongs to a well-known family of preemptive lower bounds for the single-machine weighted earliness/tardiness problems. We deliberately choose a particular relaxation within this family that does not lead to the tightest possible lower bound for the original problem; however, it exhibits structural properties that may be exploited to obtain excellent feasible non-preemptive solutions to the original single-machine weighted tardiness problem.
- The heuristic solves large-scale TWT problems in short computational times.
- The heuristic is simple, easy to implement, and fast.

1.2 Outline

The structure of the thesis is as follows. We start with the literature on TWT in Chapter 2. We introduce the proposed algorithms and heuristics and present our observations in Chapter 3. In Chapter 4, the computational results are given. The conclusions and directions for future work are presented in Chapter 5.

CHAPTER 2

Literature Survey

In 2003, an extensive survey of the research on the single-machine total tardiness problem (TT)[1] and TWT was provided by Sen et al. [54]. Other noteworthy surveys were done by Graham et al. [24] and Abdul-Razaq et al. [2] in 1979 and 1990, respectively.

According to Sen et al., the single-machine TWT problem is one of the most thoroughly investigated research problems in the machine scheduling domain. Although the first study was done more than five decades ago by McNaughton [41], the topic is still challenging for ongoing research. Studies related to this topic can be examined in two major groups as exact algorithms and heuristics. Our algorithm falls under the second group.

Holsenback et al. [27] state that "Progress in expanding the size of problems that can be solved optimally has come incrementally as new dominance properties have been identified and with improvements in computing hardware.", and this is valid for both exact and heuristic methods. The threshold of maximum size of solvable instances was 20 jobs in the late 1950s and exceeded 100 jobs after year 2000.

2.1 Exact Algorithms

Exact algorithms mainly use Branch and Bound (B&B) method and Dynamic Programming (DP) with dominance rules in order to restrict the search space.

McNaughton [41] developed rules regarding the relative positioning of tardy jobs by using the ratio r_i, where $r_i = \frac{w_i}{p_i}$, and he showed that job splitting (preemption in

[1]TT problem is a TWT problem where $w_j = 1$, $\forall j \in \{1, \ldots, n\}$.

the classical sense) has no advantage in terms of TWT in 1959. In other words, the complexity of finding an optimal preemptive schedule, where a cost is charged only to the last portion of a job, is identical to that of TWT. Schild and Fredman [52] generalized the theorem of McNaughton which was relatively restrictive.

In 1962, Held and Karp [26] and two years later, Lawler [37] presented DP formulations which consider 2^n possible subsets. No computational results were reported since this method was computationally infeasible even for 20-job problems in those years. Lawler also restated the TWT problem as an LP with $n + 2\sum p_j$ constraints.

In 1968, Elmaghraby [18] presented a network model which is similar to the backward DP algorithm where the optimal schedule is built sequentially starting from the end of the schedule. Also he introduced new dominance rules which are used by others (e.g. [8, 49]).

Emmons [19] investigated the relationships between job parameters p_j and d_j and developed three dominance theorems with a great number of corollaries in 1969. These theorems of Emmons have played major role in the TWT literature to date; many authors used these theorems in their B&B (e.g. [21, 22, 49, 51]), DP (e.g. [36, 53, 58]) and decomposition (e.g. [17, 47, 59, 60, 62]) approaches. Also, he proposed a B&B algorithm for the TT problem. Later these results were extended by Rinnooy Kan et al. [51].

From 1972 to 1976 several B&B algorithms were proposed by Shwimer [55], Gelders and Kleindorfer [22, 23], Fisher [21], Rinnooy Kan et al. [51]. Rinnooy Kan et al. also generalized Emmons' rules and the theorems they formalized have provided a stronger form of Shwimer's precedence constraints.

In 1977, Lawler [35], Lenstra et al. [40] showed that the problem is $\mathcal{NP}$-hard in the strong sense. Lawler [38] also provided a pseudo-polynomial time DP algorithm when the tardiness weights are agreeable, that is, given two jobs i and j, $p_i < p_j$ implies $w_i \geq w_j$.

Picard and Queyranne [46] developed a B&B algorithm for the Traveling Salesman Problem (TSP) which may be stated as a single-machine TWT problem with setup costs. They solved 20-job instances within 13 seconds with this method. The same year, Baker and Schrage [6] developed the "chain algorithm", which is a DP algorithm enhanced by Emmons' dominance rules. They reported that the algorithm solved 20-

job instances in an average of 3 seconds. In another paper of the same authors [53], a labeling procedure based DP algorithm was devised that dominated the previous methods till that year. One year later, in 1979, Lawler [36] came up with a faster and less memory demanding DP algorithm.

Potts and Wassenhove [49] used Lagrangian relaxation to obtain sharp lower-bounds and a DP algorithm for checking dominance rules along with the B&B algorithm they developed. With this structure, they were able to solve 40-job instances within a minute in 1985.

In 1988, Abdul-Razaq and Potts [1] presented a DP formulation of the single-machine Total Weighted Earliness-Tardiness (TWET) problem without machine idle time, which is a general case of our problem, and computed the lower-bound by a state-space relaxation of this formulation. To make the lower-bound stronger, they used penalties, state-space modifiers and additional constraints on successive jobs. Abdul-Razaq and Potts integrated this lower-bounding approach into a B&B algorithm and solved 25-job instances within 100 seconds. Three years later, Azizoglu and Kondakci [5] proposed a B&B algorithm along with the lower and upper bounding methods that they developed. They reported computational results with problems up to 20 jobs.

This problem was also studied by Ibaraki and Nakamura [29] in 1994, but they applied a Successive Sublimation Dynamic Programming (SSDP) algorithm. They solved 35-job TT instances and reported that SSDP is faster than the B&B algorithm of Abdul-Razaq and Potts. However, on the TWT problem, the B&B algorithm of Potts and Wassenhove outperforms the SSDP algorithm due to its heavy memory usage which arises because of the longer planning horizons in bigger instances. The same year, Kondakci et al. [34] proposed a B&B algorithm for TWT and reported computational results with problems up to 35 jobs.

Akturk and Yildirim [3], Kanet and Li [32], Rachamadugu [50] provided local dominance rules for determining the order of two adjacent jobs. These new dominance rules differ from the others in that they require neither agreeable nor proportional tardiness weights.

In a recent study, Kanet [31] introduced three new dominance rules and generalized some rules of Emmons [19], Rinnooy Kan et al. [51]. He also provided a B&B

scheme for how the new rules might be implemented.

In 2007, Pan and Shi [43] showed that the strongest lower-bound provided by an appropriate transportation (TR) problem and the lower-bound from the LP relaxation of the time-indexed formulation of TWT are equal. They used this new lower-bounding scheme within a B&B algorithm, and solved 100-job instances in an average of 30 minutes with a maximum of 9 hours.

Parallel to the work of Pan and Shi, Bigras et al. [8] proposed a solution approach to solve the time-indexed formulation of the problem with a column-generation technique in 2008. They decomposed the planning horizon into subperiods to solve the linear relaxation faster. With a B&B algorithm along with dominance rules, they solved 100-job instances in the OR-Library within a max of 12 hours, except for 8 instances.

The same year, Pessoa et al. [44] proposed a new arc-time indexed formulation for lower-bounding which is applicable to large instances by additional techniques such as fixing variables by reduced costs, a dual stabilization procedure to speed up column generation and others. This formulation gives better lower-bounds than the time-indexed formulation and reduces the root gap of B&B to zero in almost all OR-Library instances. Thus, in their computational experiments branching was performed in a few instances and they succeed to solve all 100-job instances in an average of 10.8 minutes with a max 142 minutes.

Finally, last year, Tanaka et al. [61] enhanced the SSDP algorithm of Ibaraki and Nakamura by a lower-bound improvement based on the dominance of two and four adjacent jobs, an adaptive step sizing method in the subgradient optimization employed for solving a series of Lagrangian relaxations of the problem, a tight upper bound computation by the enhanced DynaSearch algorithm (Congram et al. [15], Grosso et al. [25]), and choosing better state-space modifiers. With these improvements, they were able to solve 100-job instances within a max of 39 sec. and 300-job instances in 350 sec. on the average.

2.2 Heuristics

The exact algorithms assure optimality but as Abdul-Razaq et al. [2] already pointed out, before nineties, the exact algorithms struggled when the problem size exceeds 50. Even though Tanaka et al. [61] was able to solve 300-job instances within 35 minutes, this time is relatively high and solving instances with more than 100 jobs to optimality is still inefficient. Therefore, along with the exact ones, several heuristic methods have been applied by a great number of authors since the 1960s. In those studies, several dominance rule based heuristics (e.g. [30, 33, 52]), Tabu Search (TS), Simulated Annealing (SA), Genetic Algorithm (GA) and Local Search (LS) algorithms (e.g. [7, 16, 20]), and some other greedy and non-greedy heuristics (e.g. [27, 33, 42, 66]) are used.

In 1961, Schild and Fredman [52] suggested a heuristic based on the weighted shortest processing time rule, and ten years later, Wilkerson and Irwin [66] suggested a similar heuristic, but generated an initial solution with the earliest due date rule. Both heuristics start with an initial sequence and then try to improve this solution by comparing two jobs at a time according to rules.

In 1982, Morton and Rachamadugu [42] introduced a new property for adjacent jobs and they used this property to developed a new heuristic which they call "Myopic Heuristic [H3]". They also compared this heuristic to the previous ones and showed that it performs better. In the computational study, they used maximum 30-job instances due to memory and/or CPU time limitations.

In 1991, Chambers et al. [14] developed a decomposition heuristic which uses a decomposition scheme and dominance rules for shrinking the search space and also the labeling technique of Schrage and Baker [53]. The heuristic was tested and shown to be superior to others on up to 50-job instances. In the same year, Potts and Van Wassenhove [48] tested several basic and complex heuristics and noted that their SA approach is viable for TWT.

Huegler and Vasko [28] compared some interchange-based heuristics to simple heuristics. They also developed a DP-based heuristic with several subsequent improvements to this heuristic. The enhanced heuristic gave the best results on up to 500-job instances.

Crauwels et al. [16] compared several heuristics such as TS, SA, GA and descent, threshold search algorithms in 1998. Their own TS algorithm was found to be superior to the other tested search methods. One year later, Holsenback et al. [27], Volgenant and Teerhuis [64] presented new heuristics. Besten et al. [7] developed an iterated LS algorithm and were able to solve all standard benchmark instances to optimality.

In 2002, Congram et al. [15] presented a new Neighborhood Search (NS) algorithm, called DynaSearch (DS). To search exponentially-sized neighborhoods in polynomial time, they used a DP algorithm along with DS. Differently from the other LS techniques, DS is able to make more than one move in the neighborhood at each iteration. Computational results of Congram et al. showed that DS was superior to all other known LS algorithms, even to the state-of-the-art TS algorithm of Crauwels et al.. Two years later, Grosso et al. [25] integrated Generalized Pairwise Interchange (GPI) operators to the algorithm of Congram et al. and reported significantly better computational results on the OR-Library instances. Later, the LS approach of Congram et al. has been applied to the TWT problem with start time dependent processing times by Angel and Bampis [4].

Kanet and Li [33] introduced a new rule, called Weighted Modified Due Date (WMDD), and in their simulation study with other plausible rules for 40-job instances showed that WMDD clearly had an advantage over other rules by its simplicity and performance.

In 2006, Tasgetiren et al. [63] presented two metaheuristics, particle swarm optimization and differential evolution algorithms, and embedded Variable Neighborhood Search (VNS) in both algorithms. They succeeded to find optimal solutions of all standard benchmark instances with both algorithms within an average of 9 seconds. The same year, Bozejko et al. [10] proposed a TS algorithm with compound moves, and they solved all benchmark instances 4.5 times faster on average then Congram et al. within a max of 2.5 seconds. One year later, Bilge et al. [9] proposed a TS algorithm with four different versions but they were not able to solve all instances to optimality and their CPU times were very high. Ferrolho and Crisostomo [20] developed a GA-based tool, called HybFlexGA, along with new genetic operators for scheduling problems. They concluded that HybFlexGA had good performance and efficiency on standard benchmark instances. Jouglet et al. [30] proposed a TS algorithm with neighborhood search algorithm to cover new dominance rules that they described. They solved up to 250-job instances with this method effectively.

Last year, Wang and Tang [65] noted that VNS, which is applied to many other combinatorial problems such as TSP, the continuous location allocation problem and so on, has not been used much in TWT problem, and they presented a population-based VNS (PVNS) algorithm for the problem and pointed out that PVNS outperforms the VNS of Tasgetiren et al. [63] in terms of optimality gaps but in terms of CPU time VNS is much more faster than PVNS.

At present, the best exact algorithm is the SSDP algorithm of Tanaka et al. [61]; they are able to solve up to 300-job instances in 350 seconds on the average and 100-job instances in an average of 6.42 seconds with a maximum of 39 seconds. In the domain of heuristics, the best is the GPI-DynaSearch of Grosso et al. [25]. GPI-DynaSearch solves 100-job instances to optimality in an average of 0.11 seconds with a maximum of 3.91 seconds.

CHAPTER 3

Proposed Algorithm

McNaughton [41, Theorem 2.2] shows that preemption in the classical sense, that is, splitting job j into any number of parts where the process time of the parts should be integer and assigning a weighted tardiness cost only to the completion time of the last part of the job, is not useful. Such a preemptive schedule can easily be converted into a non-preemptive schedule with no larger cost. That is, $1 \mid \mid \sum_j w_j T_j$ is unary $\mathcal{NP}$-hard. However, if we break job j into p_j unit-jobs, allow jobs to be preempted at integer points in time and assign a cost to the completion time of each unit-job, this preemptive relaxation of TWT boils down to a transportation problem (TR) as discussed later, and is solvable in pseudo-polynomial time.

Our solution approach relies on the idea that the second type of preemptive relaxation of P1 as discussed above has some desirable properties that allows us to build excellent feasible solutions to the original non-preemptive problem in short computational times. The key issue here is that the information contained in the optimal solution of the preemptive relaxation reveals sufficient structure about near-optimal job processing sequences for P1. To determine possible sequences for the jobs and convert a preemptive schedule into a non-preemptive schedule, one may use the first, last or average completion time of the unit-jobs as an information. At this point, the structure of the preemptive schedule is important for constructing the non-preemptive schedule easily.

Both schedules in Figure 3.1 are preemptive but schedule in Figure 3.1(b) has a special structure which can be useful for building non-preemptive schedules easily. In the schedule in Figure 3.1(a) job j preempts job i and gets preempted by job i.

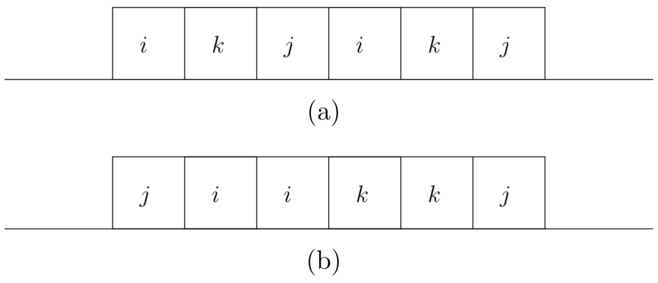


Figure 3.1: Structure of preemptions.

On the contrary, in the other schedule a job does not resume processing until the job that preempts it completes processing. Thus, a job j may be preempted by job i no more than once. Also, since second schedule has less preemption, it may be converted into a non-preemptive schedule more easily and may yield a solution which is closer to the optimal.

To find a type of preemptive schedule with the above properties, first in Section 3.1.1 we show that the optimal objective value of an appropriate TR is a lower bound on the optimal objective value of P1, and then in Section 3.2 we show that the cost coefficients we are using in TR yield an optimal preemptive solution which can always be converted into a solution with the above properties by Algorithm 1. In Section 3.4, we present our heuristic to turn this optimal solution of the TR problem into a feasible solution for the original problem P1.

3.1 A Lower Bound for P1

An approximation to P1 could be obtained by dividing each job j into p_j unit-jobs, allowing preemption at integer points in time, associating costs with each unit-job and planning for a horizon consisting of $P = \sum_{j=1}^{n} p_j$ time periods.

This type of preemption-based relaxation is used before by Bülbül et al. [11], Sourd and Kedad-Sidhoum [57]. In their studies they approximate total weighted earliness tardiness problem (TWET) as a transportation problem and determine appropriate cost coefficients for the jobs. In a similar vein, in order to find a preemptive schedule whose objective value is a lower bound on P1, with the properties discussed above, we approximate P1 by TR with appropriate cost coefficients.

3.1.1 Transportation Problem

Here, we reformulate P1 as a time-indexed formulation while allowing preemptions at integer time points.

$$
\begin{aligned}
&\textbf{(TR)} \qquad && \min \sum_j \sum_{t \in H} c_{jt} x_{jt} && && (3.1)\\
& && \sum_{t \in H} x_{jt} = p_j && \forall j && (3.2)\\
& && \sum_j x_{jt} = 1 && \forall t \in H && (3.3)\\
& && x_{jt} \in \{0, 1\} && \forall j,\ \forall t \in H. && (3.4)
\end{aligned}
$$

This new formulation is equivalent to a transportation problem (TR) where x_{jt} is the decision variable and it equals to 1 if a unit-job of job j processed in period t, otherwise it equals to 0 and c_{jt} is the cost coefficient associated with job j in time period t corresponding to the time interval $(t-1, t]$. The objective is to minimize the total assignment cost of all jobs in the planning horizon $H = [1, P]$, where $P = \sum_j p_j$. The constraints (3.2) ensure that the number of scheduled unit-jobs of job j equals to p_j and constraints (3.3) assure that exactly one unit-job is processed in a period t. Since the binary constraints do not need to be stated explicitly, the problem can be solved very efficiently as an LP or by the transportation simplex algorithm.

The cost coefficients used in this formulation are of great importance. They have to provide a lower bound on the optimal objective value of P1 and there has to exist a "nested" optimal solution to TR with them. Now, we introduce the concepts and definitions related to "nestedness" and then discuss the development of appropriate cost coefficients.

In the presentation below, a feasible solution (schedule) of the transportation problem is denoted by S_{TR}, where an optimal solution is marked by an $*$ in the superscript. $S_{\text{TR}}(t), t = 1, \ldots, P$, represents the job processed in period t and $S_{\text{TR}}(t_1, t_2)$ represents the ordered set of jobs processed in periods $t_1, \ldots, t_2$, and $j(t_1, t_2)$ is the ordered set of all time periods $t_1 \leq t \leq t_2$ so that $S_{\text{TR}}(t) = j$. Similarly, if J denotes a set of jobs, then $J(t_1, t_2)$ is the ordered set of all time periods $t_1 \leq t \leq t_2$ so that $S_{\text{TR}}(t) \in J$. The rth elements of $j(t_1, t_2)$ and $J(t_1, t_2)$ are referred to by $j(t_1, t_2)[r]$ and $J(t_1, t_2)[r]$, respectively. A unit job of job j performed in period t is referred to as u_{jt}. Furthermore, the cost of S_{TR} is computed as $C_{TR}(S_{\text{TR}}) = \sum_{t=1}^{P} c_{S_{\text{TR}}(t)t}$.

Definition 3.1 *A job j is said to be preempted by job k at time t_1, if there exist two time periods t_1 and t_2 such that $1 \leq t_1 < t_2 < P$, $S_{\text{TR}}(t_1) = j$, $S_{\text{TR}}(t_2) = k$, $\mid j(t_1 + 1, t_2 - 1) \mid = 0$, and $\mid j(t_2 + 1, P) \mid \geq 1$.*

In other words, if at least one unit job of job k appears between two successive unit jobs of job j, then job k is said to preempt job j. Under this definition, job k may preempt job j even if these two jobs are never processed in two adjacent time slots.

Definition 3.2 *A feasible schedule S_{TR} for TR is said to be preemptive, if it contains at least one preempted job.*

Definition 3.3 *A feasible preemptive schedule S_{TR} is nested, if for any pair of jobs j and k and any three time periods $t_1 < t_2 < t_3$ such that $S_{\text{TR}}(t_1) = j$, $S_{\text{TR}}(t_2) = k$, and $S_{\text{TR}}(t_3) = j$ implies that all unit jobs of job k are processed in the periods $t_1 + 1, \ldots, t_3 - 1$; that is, $\mid k(t_1 + 1, t_3 - 1) \mid = p_k$.*

In a nested schedule in which job k preempts job j, all unit jobs of job k are processed before job j is resumed. Equivalently, if job k preempts job j, then job j cannot preempt job k. In this thesis, we develop an algorithm that can transform any feasible schedule of TR into another feasible schedule with no larger cost. In particular, we prove that the proposed algorithm converts an optimal schedule of TR into a nested optimal schedule. These results are only valid if our cost coefficients described next are used in TR.

3.1.2 Cost Coefficients

When we study the structure of the preemptions, we observe that in a preemptive schedule, if a job with higher priority needs to be scheduled then this job may preempt jobs with lower priority. This priority is determined by the cost coefficients of the jobs in the time period in which the preemption occurs.

To obtain a nested structure similar to the structure in Figure 3.1(b), we need to determine suitable cost coefficients for this purpose. Since a preemption is related to the priority between two jobs, we have to select the cost coefficients in a way that, a lower priority job is preempted by a higher priority job at most once. This property can be achieved by selecting cost coefficients that lie on a piecewise linear function with a single breakpoint. In Figure 3.2(a), the priority of job i_1 is higher than the priority of job i_2 in the entire planning horizon and in the next figure, the relative priorities of job i_1 and i_2 change at time period d_{i_1}. Job i_2 is scheduled before job i_1 in the time interval $[1, d_{i_1}]$ and after time period d_{i_1}, the priority of job i_1 is higher than the priority of job i_2.

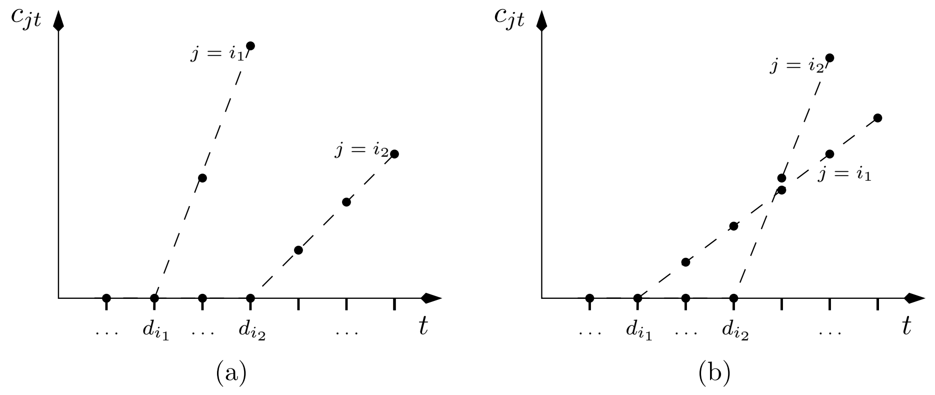


Figure 3.2: Cost functions.

The idea underlying our proposed cost structure is intuitive. Each unit-duration portion of each job has a cost coefficient given by the ratio of the tardiness weight of the job to its processing time.

$$c_{jt} = \begin{cases} 0 & t \leq d_j \\ \dfrac{w_j}{p_j}(t - d_j) & t > d_j \end{cases} \quad \forall j, t \in H. \tag{3.5}$$

Below, we provide a proof that the solution of TR with the coefficients given above provides a lower bound on the optimal objective function value of P1. Furthermore, we study the cost coefficients of Bülbül et al. [11], Sourd and Kedad-Sidhoum [57] and give a counterexample to show that the TR problem does not necessarily admit a nested optimal solution with these cost coefficients. The proof of Theorem 3.4 follows closely that of Bülbül et al. [11, Theorem 3.2].

Theorem 3.4 *The optimal objective value of TR, $C_{TR}(S^*_{TR})$, is a lower bound on the optimal objective value $C_{WT}(S^*_{P1})$ of P1.*

Proof. We show that for any optimal solution S^*_{P1} for P1, there exists a corresponding feasible schedule S_{TR} for TR such that $C_{TR}(S_{TR}) \leq C_{WT}(S^*_{P1})$. In particular, we consider a solution S_{TR} for TR constructed by converting S^*_{P1} into a feasible solution of TR. This is accomplished by dividing each job in S^*_{P1} into contiguous unit-duration segments. We demonstrate that for a schedule S_{TR} constructed in this manner, $C_{TR}(S_{TR}) \leq C_{WT}(S^*_{P1})$. Clearly, an optimal solution S^*_{P1} for P1 exists in which all job completion times belong to $H = \{k | k \in \mathbb{Z},\ k \in [1, P]\}$ which is the same time horizon considered in problem TR. Our strategy is to consider each job in S^*_{P1} separately. If $C_j \leq d_j$ in S^*_{P1}, then the cost that job j incurs in S_{TR} is 0 as in S^*_{P1}.

If job j is tardy in S^*_{P1}, then we need to distinguish between two cases. If $C_j \geq d_j + p_j$, then the cost that job j incurs in S_{TR} is given by:

$$\begin{aligned}
\sum_{k=C_j-p_j+1}^{C_j} c_{jk} &= \frac{w_j}{p_j} \sum_{k=C_j-p_j+1}^{C_j} (k - d_j) = \frac{w_j}{p_j} \sum_{k=1}^{p_j} (C_j - p_j - d_j) + k \\
&= \frac{w_j}{p_j} \left[p_j(t - d_j) - p_j^2 + \sum_{k=1}^{p_j} k \right] = w_j(t - d_j) + \left[\frac{p_j(p_j+1)}{2} - \frac{2p_j^2}{2} \right] \frac{w_j}{p_j} \\
&= w_j(t - d_j) + \left[\frac{(p_j+1) - 2p_j}{2} \right] w_j = w_j(t - d_j) - \left[\frac{(p_j - 1)}{2} \right] w_j \\
&\leq w_j(t - d_j)
\end{aligned}$$

(because $p_j \geq 1$).

However, if $d_j + 1 \leq C_j \leq d_j + p_j - 1$ when $p_j \geq 2$, then $x = C_j - d_j$ unit jobs of job j incur a tardiness cost in S_{TR} while the remaining $(p_j - x)$ unit jobs incur zero cost as in S^*_{P1}. In this case, the cost incurred by job j in S_{TR} is given by:

$$\sum_{k=C_j-p_j+1}^{C_j} c_{jk} = \sum_{k=d_j+x-p_j+1}^{d_j+x} c_{jk} = \sum_{k=d_j+1}^{d_j+x} c_{jk},$$

where $1 \leq x \leq p_j - 1$.

The costs incurred by the unit jobs completed after d_j is:

$$\sum_{k=d_j+1}^{d_j+x} c_{jk} = \frac{w_j}{p_j} \sum_{k=d_j+1}^{d_j+x} (k - d_j) = \frac{w_j}{p_j} \frac{x\,(x+1)}{2} < w_j \frac{(x+1)}{2} \leq w_j x$$

(because $x \leq p_j - 1$ and $x \geq 1$).

Therefore, we have $\sum_{k=C_j-p_j+1}^{C_j} c_{jk} < w_j x = w_j(C_j - d_j)$ when $d_j + 1 \leq C_j \leq d_j + p_j - 1$. Finally, summing over all jobs, we obtain $C_{TR}(S^*_{TR}) \leq C_{TR}(S_{TR}) \leq C_{WT}(S^*_{P1})$ as desired because the cost incurred by any job j in S_{TR} is no larger than that in S^*_{P1}. $\square$

Bülbül et al. [11] propose a lower bound for the problem $1 \mid\mid \sum_j \epsilon_j E_j + w_j T_j$ based on a similar transportation problem with the following cost coefficients:

$$c'_{jk} = \begin{cases} \frac{\epsilon_j}{p_j}\left[(d_j - \frac{p_j}{2}) - (k - \frac{1}{2})\right] & k \leq d_j \\ \frac{w_j}{p_j}\left[(k - \frac{1}{2}) - (d_j - \frac{p_j}{2})\right] & k > d_j. \end{cases} \tag{3.6}$$

These cost coefficients can be applied to TR where the earliness cost ϵ_j equals to 0 for all j. Since they satisfy $\sum_{k=C_j-p_j+1}^{C_j} c'_{jk} = w_j T_j$ for the completion times $C_j \geq d_j + p_j$, this set of cost coefficients gives better lower bounds then the cost coefficients in (3.5). Note that, our cost coefficients satisfy $\sum_{k=C_j-p_j+1}^{C_j} c_{jk} = w_j T_j - [\frac{(p_j-1)}{2}]w_j < w_j T_j$ for $C_j \geq d_j + p_j$ and $p_j \geq 2$ and with equality only for $p_j = 1$. However, the TR problem does not necessarily have a nested optimal solution with these cost coefficients. The only optimal solution to the instance below with the cost coefficients in (3.6) is $S^*_{\text{TR}}(1) = 1$, $S^*_{\text{TR}}(2) = 2$, $S^*_{\text{TR}}(3) = 1$, $S^*_{\text{TR}}(4) = 2$, $S^*_{\text{TR}}(5) = 2$ and this solution is not nested according to the Definition 3.3.

i	p_i	d_i	w_i
1	2	1	1
2	3	2	1

Table 3.1: Transportation problem does not necessarily have a nested solution with the cost coefficients in (3.6) and (3.7)

Sourd and Kedad-Sidhoum [57] propose a similar lower bound based on the transportation problem for $1 \mid\mid \sum_j \epsilon_j E_j + w_j T_j$. The cost coefficients in TR are given by:

$$c^{''}_{jk} = \begin{cases} \left\lfloor \frac{(d_j-k)}{p_j} \right\rfloor \epsilon_j & k \leq d_j - p_j \\ 0 & d_j - p_j + 1 \leq k \leq d_j \\ \left\lceil \frac{(k-d_j)}{p_j} \right\rceil w_j & k \geq d_j + 1. \end{cases} \tag{3.7}$$

These cost coefficients satisfy $\sum_{k=C_j-p_j+1}^{C_j} c^{''}_{jk} = w_j T_j$ for all completion times C_j and give better lower bounds than those by our cost coefficients. These coefficients form a (discrete) step function, and they stay constant for p_j consecutive periods. That is, they do not have a two piecewise linear structure. The only optimal solution to the instance in Table 3.1 with these cost coefficients is the same solution with the cost coefficients in (3.6). Thus, the TR problem does not admit a nested optimal solution with these cost coefficients.

TR with cost coefficients (3.6), (3.7) provides tighter lower bounds compared to TR with cost coefficients (3.5). However, as we demonstrate in Section 4.3, higher quality feasible solutions are obtained from TR under the cost coefficients (3.5).

3.2 Nester Algorithm

In this section, we present Algorithm 1 which is called Nester Algorithm and show that it converts any feasible schedule S_{TR} of TR into a feasible schedule S'_{TR} with no larger cost. Furthermore, we prove that Nester Algorithm constructs a "nested" optimal schedule, this optimal schedule S'_{TR} when applied to any optimal schedule S^*_{TR}. A direct corollary of this result is that there exists a nested optimal solution to TR under our cost coefficients.

Algorithm 1 performs two types of tasks. First, it rearranges the current schedule so that the unit jobs of job j succeed all unit jobs of the jobs with no larger due dates over the time periods $1, \ldots, d_j$. We denote this set of jobs by $J^j_{prec} = \{k| \ k < j\}$, where we assume that the jobs are sorted and re-labeled in non-decreasing order of their due dates in the rest of our presentation. (See Steps 3-4 of Algorithm 1.) Second, we define $J^j_{succ} = \{k| \ \frac{w_j}{p_j} > \frac{w_k}{p_k}\} \cup \{k < j| \ \frac{w_j}{p_j} = \frac{w_k}{p_k}\}$, and Algorithm 1 ensures that the unit jobs of the jobs in J^j_{succ} appear following all unit jobs of job j over the time periods $d_j + 1, \ldots, P$. (See Steps 12-13 of Algorithm 1.)

Algorithm 1: Converting a feasible schedule into a feasible schedule with no larger cost

```
   input : A feasible schedule $S_{\mathrm{TR}}$ for TR. Without loss of generality, assume
           that $d_1 \leq d_2 \leq \ldots \leq d_n$.
   output: A feasible schedule $S'_{\mathrm{TR}}$ for TR, where $C_{TR}(S'_{\mathrm{TR}}) \leq C_{TR}(S_{\mathrm{TR}})$.

 1 for $j = 1$ to $n$ do
 2     if $j > 1$ then
 3         $n_j =\mid j(1, d_j) \mid$;        // # of unit jobs of $j$ processed in time periods $1, \ldots, d_j$
 4         if $n_j > 0$ then
 5             $J^j_{prec} = \{k \mid k < j\}$;     // Jobs in $J^j_{prec}$ precede $j$ in periods $1, \ldots, d_j$.
 6             if $\mid J^j_{prec}(1, d_j) \mid > 0$ then
 7                 $J = \{j\} \cup J^j_{prec}$;
                   /* Move unit jobs of jobs in $J^j_{prec}$ before those of $j$ in
                      time periods $1, \ldots, d_j$ in the next two loops.        */
 8                 for $r = 1$ to $\mid J(1, d_j) \mid - n_j$ do
 9                     $S_{\mathrm{TR}}(J(1, d_j)[r]) = S_{\mathrm{TR}}(J^j_{prec}(1, d_j)[r])$;
10                 for $r =\mid J(1, d_j) \mid - n_j + 1$ to $\mid J(1, d_j) \mid$ do
11                     $S_{\mathrm{TR}}(J(1, d_j)[r]) = j$;
12     $n_j =\mid j(d_j + 1, P) \mid$;        // # of unit jobs of $j$ processed in time periods $d_j + 1, \ldots, P$
13     if $n_j > 0$ then
14         $J^j_{succ} = \{k \mid \frac{w_j}{p_j} > \frac{w_k}{p_k}\} \cup \{k < j \mid \frac{w_j}{p_j} = \frac{w_k}{p_k}\}$; /* Jobs in $J^j_{succ}$ succeed $j$ in periods $d_j + 1, \ldots, P$. */
15         if $\mid J^j_{succ}(d_j + 1, P) \mid > 0$ then
16             $J = \{j\} \cup J^j_{succ}$;
               /* Move unit jobs of jobs in $J^j_{succ}$ after those of $j$ in
                  time periods $d_j + 1, \ldots, P$ in the next two loops.        */
17             for $r =\mid J^j_{succ}(d_j + 1, P) \mid$ to $1$ do
18                 $S_{\mathrm{TR}}(J(d_j + 1, P)[n_j + r]) = S_{\mathrm{TR}}(J^j_{succ}(d_j + 1, P)[r])$;
19             for $r = 1$ to $n_j$ do
20                 $S_{\mathrm{TR}}(J(d_j + 1, P)[r]) = j$;
```

Lemma 3.5 *The cost of the final schedule S'_{TR} obtained by Algorithm 1 is no larger than that of the initial feasible solution S_{TR}, that is, $C_{TR}(S'_{\text{TR}}) \leq C_{TR}(S_{\text{TR}})$.*

Proof. Algorithm 1 performs two types of actions for job j. First, in Steps 3-4 it moves the unit jobs of job j after the unit jobs of jobs $k \in J^j_{prec}$ in the time interval $[0, d_j]$. Thus, the total cost incurred by the unit jobs of job j before and after the move is zero. Furthermore, for any job $k = 1, \ldots, n$, the assignment costs $c_{kt}, t = 1, \ldots, P$, are non-decreasing in t and the unit jobs of job $k \in J^j_{prec}$ can only be shifted earlier by this operation. Thus, the cost of the schedule cannot increase in Steps 3-4. Second, in Steps 12-13 the schedule in the time periods $d_j + 1, \ldots, P$, is modified by shifting some of the unit jobs of the jobs $k \in J^j_{succ}$ later while the unit jobs of job j are moved earlier. Note that the updates to the schedule in the `for`-loops in Steps 17-19 can also be regarded as a series of (not necessarily adjacent) pairwise interchanges between the unit jobs of job j and the unit jobs of jobs $k \in J^j_{succ}$. Therefore, in order to complete the proof we only need to argue that such a pairwise interchange is not cost increasing. To this end, consider a swap of u_{kt_1}, $k \in J^j_{succ}$, and u_{jt_2}, where $d_j < t_1 < t_2$. For job j, the decrease in cost is $c_{jt_2} - c_{jt_1} = \frac{w_j}{p_j}(t_2 - t_1)$ because the unit job of job j completes after its due date in either case. On the other hand, the increase in cost for the unit job of job k is given by $c_{kt_2} - c_{kt_1}$ which is bounded from above by $\frac{w_k}{p_k}(t_2 - t_1)$. The actual increase depends on the relative location of d_k with respect to t_1 and t_2. In any case, we have $c_{jt_2} - c_{jt_1} \geq c_{kt_2} - c_{kt_1}$ because $\frac{w_j}{p_j} \geq \frac{w_k}{p_k}$ for any job $k \in J^j_{succ}$. ∎

The proof of Lemma 3.5 clearly demonstrates that there is no step in Algorithm 1 that ever increases the total cost of the current schedule. This observation leads to the corollary below.

Corollary 3.6 *When applied to an optimal schedule S^*_{TR}, Algorithm 1 preserves optimality at every step of the algorithm and terminates with an optimal solution S'_{TR}.*

Lemma 3.7 *When applied to an optimal schedule S^*_{TR}, Algorithm 1 satisfies two properties.*

a. In Steps 3-4, no unit job u_{kt1}, where $k \in J^j_{prec}$ and $t_1 \geq d_k + 1$, is moved to a time period $t_2 < t_1$.

b. *In Steps 12-13, a unit job* u_{kt_1}*, where* $k \in J^j_{succ}$*, may be moved to a time period* $t_2 > t_1 > d_j$ *only if* $d_k \leq t_1$ *and* $\frac{w_j}{p_j} = \frac{w_k}{p_k}$.

Proof. According to Corollary 3.6, starting with an optimal solution Algorithm 1 preserves optimality at every step because it modifies the solution while guaranteeing that the total cost does not increase. Both Parts a and b of Lemma 3.7 follow from this invariant property of Algorithm 1.

For Part a, observe that the unit jobs of jobs $k \in J^j_{prec}$ can only be completed earlier after the current schedule is modified in Steps 3-4 of Algorithm 1. Since the total cost incurred by the unit jobs of job j is constant at zero, the total cost would strictly decrease if a unit job of a job $k \in J^j_{prec}$ that is currently processed in a period $t_1 \geq d_k + 1$ is moved to a time period $t_2 < t_1$ as a result of the updates to the schedule in Steps 3-4. This would contradict the optimality of the current schedule.

For Part b, note that the modifications of the schedule in Steps 12-13 may also be regarded as a series of (not necessarily) pairwise interchanges between the unit jobs of job j and the unit jobs of the jobs in $k \in J^j_{succ}$. By Corollary 3.6, the total cost is not affected by any of these interchanges. Now, consider a swap of u_{kt_1} and u_{jt_2}, where $k \in J^k_{succ}$. The decrease in the the total cost is calculated as $(c_{jt_2} - c_{jt_1}) - (c_{kt_2} - c_{kt_1}) \geq 0$ because $\frac{w_j}{p_j} \geq \frac{w_k}{p_k}$ and $d_j < t_1 < t_2$. The inequality holds as equality only if $d_k \leq t_1 < t_2$ and $\frac{w_j}{p_j} = \frac{w_k}{p_k}$ as required. ∎

Lemma 3.7 helps us prove the next result which is instrumental in characterizing the nature of preemptions in an optimal schedule constructed by Algorithm 1.

Lemma 3.8 *When applied to an optimal schedule* S^*_{TR}*, the optimal schedule* S'_{TR} *constructed by Algorithm 1 satisfies two properties at termination:*

a. *All unit jobs of job* j *succeed all unit jobs of the jobs in the set* $J^j_{prec} = \{k|\ k < j\}$ *over the time periods* $1, \ldots, d_j$.

b. *All unit jobs of job* j *precede all unit jobs of the jobs in the set* $J^j_{succ} = \{k|\ \frac{w_j}{p_j} > \frac{w_k}{p_k}\} \cup \{k < j|\ \frac{w_j}{p_j} = \frac{w_k}{p_k}\}$ *over the time periods* $d_j + 1, \ldots, P$.

Proof. We carry out this proof by induction by showing that these two properties hold for jobs $k = 1, \ldots, j$, at the end of iteration j of the main loop of Algorithm 1 in Steps 1-1. Both of the properties clearly hold for job 1 at the end of iteration 1. Thus, our induction hypothesis asserts that both of the properties are satisfied for jobs $k = 1, \dots, j-1$, at the end of iteration $j-1$ of the main loop in Steps 1-1. In the induction step, we prove that we extend these properties to job j in the jth iteration while preserving them for jobs $k = 1, \dots, j-1$.

Steps 3-4 of iteration j ensure that Lemma 3.8a is satisfied for job j. Furthermore, for any job $k < j$ we have $(\{k\} \cup J^k_{prec}) \subseteq J^j_{prec}$, and thus the relative order of the unit jobs of job k and the unit jobs of the jobs in J^k_{prec} is preserved during these steps in the entire schedule although some of them may be shifted earlier. Thus, Lemma 3.8a may only be violated for job k during Steps 3-4 if a unit job u_{lt_1}, where $l < k$ and $t_1 > d_k$, is shifted earlier into a time period $t_2 \leq d_k$ as a result of these updates to the schedule. Lemma 3.7a guarantees that this never happens since $l < k$ implies that $d_l \leq d_k < t_1$. Furthermore, in Steps 12-13 the algorithm modifies the schedule only in the time periods $d_j + 1, \dots, P$, where $d_j \geq d_k$; and thus, Lemma 3.8a holds for all jobs $1, \dots, j$, upon completion of the jth iteration of Steps 1-1 in Algorithm 1.

In Steps 12-13 of the jth iteration of the main loop of Algorithm 1, Lemma 3.8b is satisfied for job j. Thus, in order to complete the proof we need to argue that Lemma 3.8b still holds for any job $k < j$ following the completion of the jth iteration of the main loop. To this end, we note that during the course of Algorithm 1 we never encounter a schedule in which a unit job of a job l with $\frac{w_l}{p_l} < \frac{w_k}{p_k}$ precedes a unit job of job k over the time periods $d_k + 1, \dots, P$. Such a solution would not be optimal and contradict Corollary 3.6. In other words, Corollary 3.6 establishes Lemma 3.8b for the unit jobs of the jobs in the set $\{l | \frac{w_k}{p_k} > \frac{w_l}{p_l}\}$. Consequently, here it is sufficient to prove that if $l < k < j$ and $\frac{w_l}{p_l} = \frac{w_k}{p_k}$ then all unit jobs of job k appear before those of l in the time periods $d_k + 1, \dots, P$, when the jth iteration of the main loop of Algorithm 1 terminates. In Steps 3-4, some unit jobs of jobs l and k may be shifted earlier because both l and k belong to J^j_{prec}. However, these operations cannot lead to a violation of Lemma 3.8b for job k because the relative order of these unit jobs is maintained throughout the schedule and the unit jobs of job k already precede those of job l over $d_k+1, \dots, P$. Steps 12-13 result in two cases. If $\frac{w_l}{p_l} = \frac{w_k}{p_k} \neq \frac{w_j}{p_j}$, then the unit jobs of jobs k and l remain intact in their current positions in the schedule because Lemma 3.7b implies that a unit job of a job $k < j$

may only be moved around in Steps 12-13 if $\frac{w_k}{p_k} = \frac{w_j}{p_j}$. Otherwise, if $\frac{w_l}{p_l} = \frac{w_k}{p_k} = \frac{w_j}{p_j}$ and $l, k \in J^j_{succ}$ then any updates to the schedule in periods $d_j + 1, \ldots, P$, does not change the relative order of the unit jobs of jobs k and l over the time periods $d_k + 1, \ldots, P$, where $d_k \leq d_j$. Thus, Lemma 3.8b remains valid for job k. This argument completes the final piece of the proof. ∎

In the next lemma, we establish the structure of preemptions in an optimal schedule constructed by Algorithm 1. Note that if job j is preempted by job k at time t_1, Definition 3.1 asserts that there exist three unit jobs u_{jt_1}, u_{kt_2}, and u_{kt_3}, where $t_1 < t_2 < t_3$.

Lemma 3.9 *A job j can only be preempted by a job $k > j$ with $\frac{w_k}{p_k} \geq \frac{w_j}{p_j}$ in a time period $t_1 \leq d_k$ in an optimal schedule S'_{TR} constructed by Algorithm 1 starting from an arbitrary optimal solution S^*_{TR}.*

Proof. We consider four cases:

Case 1 Consider two jobs $j < k$, where $\frac{w_j}{p_j} > \frac{w_k}{p_k}$. The optimality of S'_{TR} requires that $t_3 \leq d_j$. However, in this case u_{kt_2} appears before u_{jt_3} over the time periods $1, \ldots, d_k$, and this violates Lemma 3.8a for job k.

Case 2 Consider two jobs $j < k$, where $\frac{w_j}{p_j} \leq \frac{w_k}{p_k}$. In this case, we need to rule out the possibility that $t_1 > d_k$. If $d_k < t_1 < t_2 < t_3$, u_{kt_2} succeeds u_{jt_1} over the time periods $d_k + 1, \ldots, P$, which contradicts Lemma 3.8b for job k.

Case 3 Consider two jobs $j > k$, where $\frac{w_k}{p_k} > \frac{w_j}{p_j}$. The optimality of S'_{TR} requires that $t_2 \leq d_k$. However, in this case u_{jt_1} appears before u_{kt_2} over the time periods $1, \ldots, d_j$, and this violates Lemma 3.8a for job j.

Case 4 Consider two jobs $j > k$, where $\frac{w_k}{p_k} \leq \frac{w_j}{p_j}$. If $t_2 \leq d_j$, then this would contradict Lemma 3.8a for job j. On the other hand, $t_2 > d_j$ is ruled out by Lemma 3.8b for job j. ∎

The possible ways that a job $k > j$ with $\frac{w_k}{p_k} \geq \frac{w_j}{p_j}$ may preempt job j is illustrated in Figure 3.3. Note that we also need $t_3 > d_k$; otherwise, Lemma 3.8a would not hold for job k.

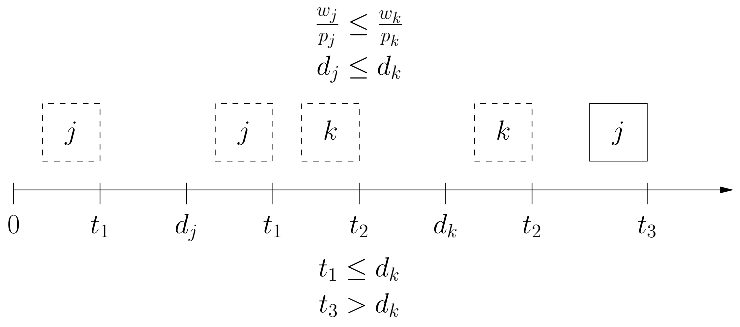


Figure 3.3: Structure of preemptions in an optimal schedule S'_{TR} constructed by Algorithm 1.

Finally, we present our main result which stipulates that the optimal solution S'_{TR} for TR constructed by Algorithm 1 is nested according to the Definition 3.3.

Theorem 3.10 *When applied to an arbitrary optimal schedule S^*_{TR}, Algorithm 1 constructs a nested optimal schedule S'_{TR}.*

Proof. Suppose that the optimal schedule S'_{TR} constructed by Algorithm 1 is not nested. Then, there must exist at least one pair of jobs j and k and two time periods t_1 and t_2 such that job k preempts job j at time t_1 and job j preempts job k at time t_2. This is clearly not possible by Lemma 3.9 because we either have $j > k$ or $k > j$. ∎

Corollary 3.11 *There exists a nested optimal solution to TR under the cost coefficients in* (3.5).

Proof. We can always apply Algorithm 1 to an arbitrary optimal solution S^*_{TR} of TR and obtain a nested optimal solution S'_{TR}. ∎

3.3 Further Remarks on the Structure of TR and Nester Algorithm

In this section, we characterize a dominance rule for TR which is not necessarily valid for TWT. We demonstrate that Nester Algorithm is oblivious to this dominance rule and may convert a preemptive schedule that observes this dominance rule into a preemptive schedule in which the dominance rule is violated. Also, we give an example to show that a non-preemptive optimal solution of TR is not necessarily optimal for TWT.

Property 3.12 *If $j < k$ ($d_j \leq d_k$) and $\frac{w_j}{p_j} \geq \frac{w_k}{p_k}$, then there exists an optimal solution to TR in which all unit jobs of job j are scheduled before those of job k.*

Proof. Suppose that we are given an optimal solution S^*_{TR} for an instance of TR in which there exist two jobs j and k that satisfy the conditions of this property, but some unit jobs of job k appear before those of job j. We complete the proof by demonstrating that an alternate optimal schedule may be constructed by modifying S^*_{TR} appropriately so that no unit job of job k is processed before all unit jobs of job j are performed. This task is accomplished in two steps. First, we set $J = \{j, k\}$ and identify all time periods $J(1, P)$ in the optimal schedule S^*_{TR} in which we either process job j or job k. Then, we assign the unit jobs of job j to the first p_j time periods in the set $J(1, P)$ while the unit jobs of job k are processed in the remaining time periods of $J(1, P)$. These changes to S^*_{TR} do not lead to an increase in the total cost and we obtain an alternate optimal solution because $c_{kt} \leq c_{jt}$ for all $t \in J(1, P)$ and the unit jobs of the other jobs are clearly not affected. ∎

However, our algorithm is oblivious to this dominance rule and may even convert a non-preemptive optimal solution to TR that obeys the dominance rule in Property 3.12 into a nested preemptive optimal solution. Consider the instance given in Table 3.2. There exists a non-preemptive optimal solution to this instance of TR in which all unit jobs of 1 are scheduled before those of 2. However, our algorithm converts this non-preemptive optimal solution into the following nested preemptive optimal solution: $S^*_{\text{TR}}(1) = 1$, $S^*_{\text{TR}}(2) = 1$, $S^*_{\text{TR}}(3) = 1$, $S^*_{\text{TR}}(4) = 2$, $S^*_{\text{TR}}(5) = 2$, $S^*_{\text{TR}}(6) = 2$, $S^*_{\text{TR}}(7) = 1$.

Job	1	2
w_j	4	3
p_j	4	3
d_j	2	3

Table 3.2: Our algorithm does not observe Property 3.12.

An interesting question regarding the optimality structure of TR is whether the nested structure of preemptive optimal solutions may be employed to decompose the original non-preemptive problem into smaller independent subproblems. In other words, we investigate whether it is possible to solve the original non-preemptive problem to optimality by solving one relatively smaller non-preemptive single-machine weighted tardiness problem for each parenthesis structure present in a nested preemptive optimal solution and then appending the optimal solutions of these subproblems one after another. Consider the instance in Table 3.3. Assume that the optimal solution of TR retrieved from the solver is $S^*_{\text{TR}}(1) = 1$, $S^*_{\text{TR}}(2) = 2$, $S^*_{\text{TR}}(3) = 2$, $S^*_{\text{TR}}(4) = 2$, $S^*_{\text{TR}}(5) = 2$ with an objective value of 10.5. This optimal solution is also obtained by applying our algorithm to an alternate optimal solution $S^*_{\text{TR}}(1) = 2$, $S^*_{\text{TR}}(2) = 1$, $S^*_{\text{TR}}(3) = 2$, $S^*_{\text{TR}}(4) = 2$, $S^*_{\text{TR}}(5) = 2$. In this case, each non-preemptively scheduled job forms its own parenthesis, and the decomposition approach described above would yield a sequence $1 \rightarrow 2$ for the original non-preemptive problem with an objective value of 21. We can easily verify that $2 \rightarrow 1$ is the optimal sequence for the original non-preemptive problem with an associated objective value of 20.

Job	1	2
w_j	2	7
p_j	1	4
d_j	2	2

Table 3.3: A non-preemptive optimal solution to TR is not necessarily optimal with respect to the original non-preemptive problem.

3.4 Heuristics

Now, we present three simple heuristics based on job statistics along with the development of Non-Preemptive Heuristic (NPH) which converts a nested preemptive schedule into a feasible schedule for TWT.

3.4.1 Simple Heuristics

These three greedy heuristics only use completion time of the unit-jobs of jobs as information to construct a job processing sequence.

- ACT: Sequence jobs in non-decreasing order of the average completion time of their unit-jobs, where the average completion time of the unit-jobs of job j is defined as $\overline{C_j} = \sum\limits_{l \in \{k : x_{jk}=1\}} l/p_j$.
- LCT: Sequence jobs in non-decreasing order of the completion time of their last unit-jobs, where the completion time of the last unit-job of job j is defined as $\max\{k : x_{jk} = 1\}$.
- MCT: Sequence jobs in non-decreasing order of the median completion time of their unit-jobs.

The last two heuristics are related to the α-point concept which is introduced by Phillips et al. [45]. An α-point of a job defined as the first time point where α-portion of the job is completed.

3.4.2 Non-Preemptive Heuristic

A nested schedule may be represented as a parenthesis structure, where the first and last unit-jobs of a job are denoted by opening and closing parentheses, respectively. $(\quad l \quad)(\quad i \quad (\quad m \quad)(\quad j \quad (\quad k \quad) \quad j \quad) \quad i \quad)(\quad n \quad)$ is the parenthesis representation of the schedule in Figure 3.4. Observe that, there is only one job (e.g. l, n) inside some parenthesis, this job is a non-preempted job. The second parenthesis (i.e. parenthesis i), has two more parentheses inside. These are parenthesis m and j, parenthesis m is non-preemptive however, parenthesis j has one more parenthesis inside.

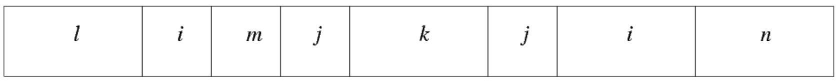


Figure 3.4: A nested schedule of TR

The non-preemptive heuristic (NPH) described in Algorithm 2, basically, finds a preempted parenthesis p and locates all unit-jobs of job p between other parentheses non-preemptively. While performing these placement operations, NPH calculates weighted tardiness costs and then schedules job p in the place with minimum cost.

Algorithm 2: Converting a nested schedule into a non-preemptive schedule.

```
input          : A nested feasible schedule $S'_{\mathrm{TR}}$ of TR.
output         : A feasible schedule $S_{\mathrm{WT}}$ for TWT.
initialization: $t = 1$, $S_{\mathrm{WT}} = S'_{\mathrm{TR}}$.

1  while $t \leq P$ do
2      $j = S_{\mathrm{WT}}(t)$;
3      $C_j = \max\{m \mid S_{\mathrm{WT}}(m) = j\}$ ;                    // Completion time of job $j$.
4      if $C_j - t = p_j - 1$ then              // If job $j$ is scheduled non-preemptively.
5          $t = t + p_j$;
6      else                                                   // If job $j$ is preempted.
7          $Cost = \infty$;
8          $S = S_{\mathrm{WT}}(t, C_j)$;
9          Delete all unit-jobs of job $j$ from the set $S$;
10         $S' = \emptyset$;
11         for $l = 1$ to $p_j$ do
12             $S'(l) = j$;
13         $l =\mid S \mid$;
14         while $l \geq 0$ do
               /* Generating schedule part $S'$ to find the place for job $j$ with minimum cost.  */
15             $S'' = S(1, l) \cup S' \cup S(l + 1, \mid S \mid)$;
16             $l = \min\{m \mid S(m) = S(l)\} - 1$ ;  // Index of the last job of the previous
               parenthesis.
17             if $C_{WT}(S'', t - 1) < Cost$ then
18                 $Cost = C_{WT}(S'', t - 1)$;
19                 $S_{best} = S''$;
20         $S_{\mathrm{WT}}(t, C_j) = S_{best}$;
```

In the description of the non-preemptive heuristic, $S_{\mathrm{WT}}(t)$ denotes the tth unit-job in the schedule S_{WT}, S is the set of unit-jobs except those of job j in the parenthesis j, $S_{\mathrm{WT}}(t_1, t_2)$ and $S(t_1, t_2)$ represent the ordered set of jobs processed in the periods $t_1, \ldots, t_2$, and S'' is the schedule part corresponding to $S_{\mathrm{WT}}(t, C_j)$, which is modified in order to make the job j non-preemptive. NPH calculates weighted tardiness cost of a schedule part S'' (i.e. a parenthesis) that starts in time period k, with Procedure $C_{WT}(S,k)$ (Algorithm 3). This procedure first finds the last completion time sequence of the jobs, since there may be another parenthesis in S''. Then it calculates the weighted tardiness cost of the sequence.

Algorithm 3: Procedure $C_{WT}(S'',t)$

```
input          : $S''$ and $t$.
output         : Tardiness cost of schedule part $S''$.
initialization: $S''_{seq}$ = LCT sequence of the jobs in schedule part $S''$, $C_{WT} = 0$.
1 for $i = 1$ to $| S''_{seq} |$ do
2     $j = S''_{seq}(i)$;
3     $t = t + p_j$;
4     $C_{WT} = C_{WT} + w_j \max\{0, t - d_j\}$;
```

In Figure 3.5, there is an example of how NPH finds the position with minimum weighted tardiness cost for job i. When the heuristic comes across a preempted parenthesis i, starting from the end, it locates all unit-jobs of job i between other parentheses non-preemptively (i.e. (1), (2) and (3)). Then it schedules job i in the position with minimum cost (i.e. (2)).

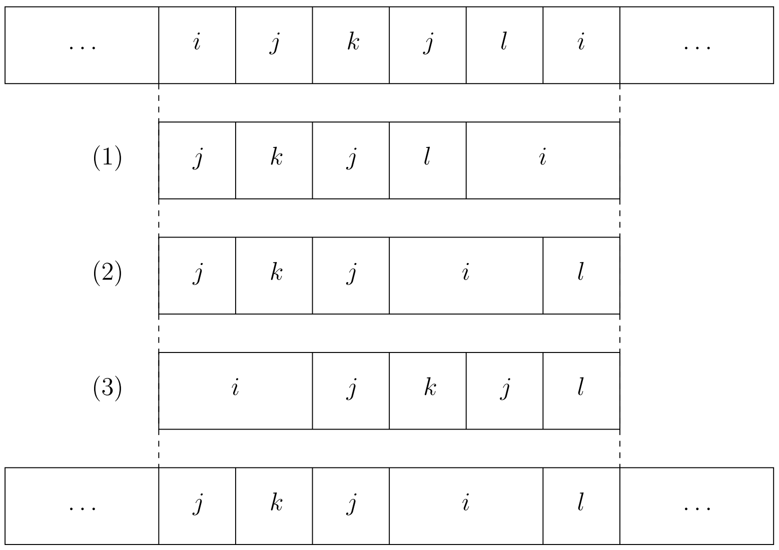


Figure 3.5: An iteration of NPH.

The heuristic's outer loop performs at most $n+\alpha$ iterations where α is the number of preempted parentheses in the schedule. The total number of placement operations on those α iterations cannot exceed $n+\alpha$; more precisely it is less than or equal to $\alpha\left\lfloor \frac{n}{\alpha}+1\right\rfloor$.

3.5 Common Due Date

The single-machine common due date total weighted tardiness problem (CDD) is a special case of TWT, where the due dates are identical for each job. Therefore, the same notation and formulation may be used for the common due date case.

Our solution approach relies on the fact that, with our cost coefficients, TR approximation of CDD admits an easy solution. This solution can be obtained by inspection and since it is non-preemptive, it is feasible for the original common due date problem.

To this end, we first restate TR as a Linear Sum Assignment problem (LSAP) which is a special case of TR. Then, we show that, with our cost coefficients, the cost matrix C of LSAP fulfills the Monge property which is defined as:

$$c_{ij}+c_{kl}\le c_{il}+c_{kj} \tag{3.8}$$

for all $1\le i<k\le P$ and $1\le j<l\le P$ in [12, Definition 5.5], where $P=\sum_j p_j$ is the size of the matrix. Thus, the optimal solution of LSAP is the identical permutation (i.e. $x_{ij}=1$ for only $i=j$) [12, Proposition 5.7].

Now, we restate TR as LSAP by treating each unit-job as a separate job. Assume that, jobs are ordered and relabeled in non-increasing order of their $\frac{w_i}{p_i}$ values. By introducing binary variables x_{ijt} such that

$$x_{ijt}=\begin{cases}1 & \text{if } j\text{th task of job } i \text{ assigned to time period } t,\\ 0 & \text{otherwise,}\end{cases}$$

and a $P\times P$ cost matrix C such that

$$c_{\left(\sum_{n=1}^{i-1}p_n+j\ \ t\right)}=\begin{cases}0 & t\le d,\\ \frac{w_i}{p_i}\left[t-d_i\right] & t>d,\end{cases} \tag{3.9}$$

where $j \leq p_i$ and $c_{\left(\sum\limits_{n=1}^{i-1} p_n + j \;\; t\right)}$ represents the cost coefficient associated with jth task of ith job in time period t. And the objective is to minimize the total assignment cost over the planning horizon $H = 1 \dots P$.

$$(\mathbf{LSAP}) \qquad \min \sum_{i=1}^{n} \sum_{j=1}^{p_i} \sum_{t \in H} c_{\left(\sum\limits_{n=1}^{i-1} p_n + j \;\; t\right)} x_{ijt} \tag{3.10}$$

$$\sum_{t \in H} x_{ijt} = 1 \qquad \forall i,\ j \tag{3.11}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{p_i} x_{ijt} = 1 \qquad \forall t \in H \tag{3.12}$$

$$x_{ijt} \in \{0, 1\} \qquad \forall i,\ j,\ t \in H. \tag{3.13}$$

Lemma 3.13 *The solution to the LSAP with the cost coefficients in* (3.9)*, is the identical permutation (i.e. $x_{ijt} = 1$ for only $\sum\limits_{n=1}^{i-1} p_n + j = t$).*

Proof. To prove this lemma, it is enough to show that the cost matrix C of LSAP with the cost coefficients in (3.9), is a Monge matrix, since it is already shown that the solution to the LSAP whose cost matrix is a Monge matrix, is the identical permutation by Burkard et al. [12, Proposition 5.7].

To show that C is a Monge matrix, we show that C is a so-called ordered product matrix which forms a subclass of Monge matrices. A matrix D is said to be ordered product matrix if $D = (d_{ij}) = u_i v_j$ where $u_1 \geq u_2 \ldots \geq u_n \geq 0$ and $0 \leq v_1 \leq v_2 \ldots \leq v_n$ hold [13, p. 499].

When we take $u_1 \geq u_2 \ldots \geq u_P \geq 0$ and $0 \leq v_1 \leq v_2 \ldots \leq v_P$ as below:

$$\begin{aligned} v_j &= \max\{0, j - d\} && \text{for } j = 1, \ldots, P \\ u_{\left(\sum\limits_{n=1}^{i-1} p_n + j\right)} &= \frac{w_i}{p_i} && \text{for } i = 1, \ldots, n \text{ and } j = 1, \ldots, p_i, \end{aligned}$$

then the cost coefficients $c_{ij} = u_i v_j$ and the cost matrix created with these cost coefficients is a Monge matrix. Thereby, solution to the linear sum assignment problem with these cost coefficients is the order of the jobs with respect to the ratio $\frac{w_i}{p_i}$. ∎

Since LSAP relaxation is non-preemptive, it is feasible for the original single-machine common due date weighted tardiness problem.

It turns out that, the solution procedure we described in this section for the single-machine common due date weighted tardiness problem boils down to a well-known heuristic rule, weighted shortest processing time (WSPT) first described by Smith [56] in 1956. McNaughton further showed the cases in which WSPT gives the optimal schedule for the original scheduling problem TWT.

CHAPTER 4

Computational Results

In order to evaluate the performance of the proposed solution approach, we solved the standard benchmark instances in the OR-Library[1] and the instances that are generated by Tanaka et al. [61] in a similar way to the OR-Library instances[2]. The main objectives of our computational experiments are to show that the proposed algorithms find excellent feasible solutions in short computational times. To generate a feasible schedule for TWT, we obtain an optimal schedule of TR and convert this schedule into a nested one with the Nester Algorithm and then use the heuristics described in Section 3.4 to obtain a feasible solution of TWT. In general, the optimal solution of TR is a preemptive schedule and the factors that affect preemption are due dates, processing times, and the number of jobs. In particular, among these factors, we are interested in the behavior of our algorithms with respect to the due date factors and the number of jobs. In Section 4.3.4, we also investigate whether the nested structure may help us to decompose TWT time-wise into smaller subproblems.

4.1 Data of TWT

The data set consists of seven sets of instances with problem sizes n = 40, 50, 100, 150, 200, 250 and 300. Each instance is generated by assigning a processing time p_j and a tardiness weight w_j for each job j from a discrete uniform distribution $U[1, 100]$ and $U[1, 10]$, respectively.

The due dates are generated from the interval $[(1 - TF - RDD/2)P, (1 - TF +$

[1]http://people.brunel.ac.uk/∼mastjjb/jeb/info.html
[2]http://turbine.kuee.kyoto-u.ac.jp/ tanaka/SiPS/SiPS.html

n	p_j	TF	RDD	w_j
40, 50, 100, 150, 200, 250, 300	$U[1, 100]$	$\{0.2, 0.4, 0.6, 0.8, 1\}$	$\{0.2, 0.4, 0.6, 0.8, 1\}$	$U[1, 10]$

Table 4.1: Data parameters

$RDD/2)P]$, where $P = \sum_j p_j$, and RDD and TF are two parameters. TF is referred to as tardiness factor since it determines the tightness of the average due date. RDD controls the variability in the due dates. There are 25 combinations of TF and RDD, and for each combination of TF and RDD, five instances are generated. Therefore, there are 125 instances for each problem size. The optimal solutions of all instances are found by Tanaka et al. [61] in 2009.

4.2 Other Cost Coefficients for TR

In order to evaluate the performance of the cost coefficients in (3.5), we modify the cost coefficients of Bülbül et al. [11] and Sourd and Kedad-Sidhoum [57] originally proposed for the single-machine total weighted earliness tardiness problem. We obtain the following cost coefficients, by setting the earliness cost to zero:

$$c_{jk}^{BUL} = \begin{cases} 0 & k \leq d_j \\ \frac{w_j}{p_j}\left[(k - \frac{1}{2}) - (d_j - \frac{p_j}{2})\right] & k > d_j, \end{cases} \tag{4.1}$$

$$c_{jk}^{SOU} = \begin{cases} 0 & k \leq d_j \\ \left\lceil \frac{(k-d_j)}{p_j} \right\rceil w_j & k \geq d_j + 1. \end{cases} \tag{4.2}$$

Along with the cost coefficients in (3.5), TR is solved with these cost coefficients as well. After the preemptive schedules are obtained, only simple heuristics (SH) are applied to the schedules which are obtained with the cost coefficients in (4.1) and (4.2), because the TR problem does not necessarily have a nested optimal solution with these cost coefficients as we demonstrate in Section 3.1.2.

4.3 Summary of Results

The solution procedure for the transportation problem and all proposed algorithms are implemented in C++ using `CPLEX 12.1` Concert Technology. All computational results are obtained on a Windows PC with a 2.33 GHz Intel Core2 Quad CPU and 3.46 GB RAM.

The structure of this section is as follows. We present the effects of the Nester Algorithm and the number of jobs in the next section and then the effects of the tardiness and range of due date factors are presented. In Section 4.3.3, statistics related to the parentheses in the nested schedules are given and after that, the results obtained by solving the parentheses with an IP to optimality are presented.

4.3.1 Effect of the Nester Algorithm and Number of Jobs

Tables 4.2 and 4.3 show the performance changes of the heuristics and lower bounds in terms of gaps and CPU times, with respect to the number of jobs. The effect of the Nester Algorithm is reported in columns 8 and 11-13. In these columns, the numbers in the parentheses indicate the values that are obtained from a nested schedule. Column 2 indicates the cost coefficients used in TR. For each cost coefficient c_{ij}, the first row indicates the average and the second row indicates the worst case performance measures. In columns 3-6, performance measures related to CPU times are reported. The computation time for obtaining the optimal solution of TR is reported in column TR. The total time required to compute the best solution (after obtaining a nested optimal schedule to TR) from three simple heuristics is given in column BSH ("Best of Simple Heuristic"). The computation time required to obtain a nested optimal schedule from the optimal solution of TR and a feasible schedule for TWT from the nested schedule by our proposed algorithm are reported in columns NA ("Nester Algorithm") and NPH ("Non-Preemptive Heuristic"), respectively. The number of times TR solution, the best simple non-preemptive heuristic solution, and the non-preemptive heuristic solution from NPH match the optimal solution of TWT are represented under # of Optimal Solutions in columns 7-9, respectively. In column BSH, the numbers in parentheses refer to the number of optimal solutions when the heuristics are applied to a nested optimal solution of TR.

n	c_{ij}	CPU time (millisecond)				# of Optimal Solutions			Percentage Gap (%)					
		TR[1]	BSH	NA	NPH	TR	BSH	NPH	TR	ACT	MCT	LCT	BSH	NPH
40	(3.5)	**0.5**	**0.5**	**5**	**0.6**	**18**	**(41) 34**	**54**	**-14**	**(8.7) 47**	**(4.0) 32**	**(2.8) 2.8**	**1.94**	**1.20**
		0.9	16	16	16				-96	(303) 1113	(76) 889	(82) 82	63	42
	(4.1)	**0.5**	**0.5**			**18**	**35**		**-4.9**	**67.7**	**50.5**	**7.10**	**6.08**	
		1	16						-91	3006	1926	274	274	
	(4.2)	**0.3**	**0.6**			**18**	**23**		**-6.8**	**70.2**	**26.4**	**14.8**	**7.97**	
		0.5	16						-92	3530	745	551	551	
50	(3.5)	**0.7**	**0.5**	**9**	**2**	**17**	**(34) 30**	**47**	**-13**	**(8.4) 256**	**(5.3) 173**	**(2.1) 2.1**	**1.51**	**0.42**
		1.3	16	16	32				-98	(102) 15400	(102) 13500	(34) 34	34	13
	(4.1)	**0.7**	**0.4**			**17**	**28**		**-3.9**	**280**	**227**	**3.02**	**1.92**	
		1.3	16						-60	20475	19650	34	34	
	(4.2)	**0.5**	**0.6**			**20**	**22**		**-4.8**	**436**	**333**	**23.5**	**20.1**	
		0.9	16						-64	46950	34000	2100	2100	
100	(3.5)	**3.1**	**0.5**	**47**	**10**	**18**	**(33) 32**	**40**	**-9.6**	**(4.2) 472**	**(2.6) 85**	**(1.1) 1.1**	**0.73**	**0.21**
		6.3	16	79	188				-99	(116) 24815	(79) 4676	(30) 30	12	8
	(4.1)	**3**	**1.1**			**18**	**24**		**-3.5**	**730**	**55.7**	**26.2**	**25.5**	
		5.7	16						-60	47738	2525	2225	2225	
	(4.2)	**2.7**	**1.3**			**19**	**19**		**-4.4**	**564**	**74.5**	**35.6**	**32.0**	
		4.1	16						-63	35125	4475	2225	2225	

[1]The results in this column are in seconds.

Table 4.2: Effect of n and Nester Algorihm: CPU times, number of optimal solutions, and average and worst case gaps of lower bounds and heuristics

n	c_{ij}	CPU time (millisecond)				# of Optimal Solutions			Percentage Gap (%)					
		TR[1]	BSH	NA	NPH	TR	BSH	NPH	TR	ACT	MCT	LCT	BSH	NPH
150	(3.5)	**12**	**2.4**	**127**	**50.8**	**20**	**(33) 30**	**38**	**-8**	**(12.66) 391**	**(11.8) 161**	**(1.52) 1.54**	**1.10**	**0.58**
		20	16	188	500				-91	(509) 19136	(509) 13097	(56) 56	56	56
	(4.1)	**12**	**2.7**			**20**	**29**		**-3**	**510**	**287**	**2.4**	**1.6**	
		20	16						-45	36087	23917	83	83	
	(4.2)	**7.6**	**3.1**			**22**	**25**		**-3**	**492**	**321**	**9.0**	**4.3**	
		17	16						-50	30133	27080	546	156	
200	(3.5)	**23**	**4.2**	**259**	**143**	**20**	**(26) 25**	**34**	**-7**	**(6.59) 969**	**(3.39) 123**	**(1.40) 1.46**	**0.99**	**0.3**
		41	32	407	1750				-98	(171) 104246	(122) 8715	(24) 24	19	13
	(4.1)	**23**	**4.9**			**20**	**25**		**-3**	**356**	**109**	**2.4**	**1.6**	
		44	16						-90	13677	7008	62	61	
	(4.2)	**14**	**5.1**			**20**	**21**		**-4**	**233**	**138**	**5.7**	**4.4**	
		22	16						-85	5892	9638	115	115	
250	(3.5)	**48**	**4.7**	**440**	**175**	**21**	**(29) 27**	**35**	**-6**	**(8.24) 1291**	**(4.96) 305**	**(1.48) 1.50**	**1.16**	**0.43**
		101	31	656	2391				-95	(233) 113456	(174) 15656	(43) 43	43	41
	(4.1)	**47**	**5**			**21**	**25**		**-2**	**1482**	**465**	**2.0**	**1.3**	
		101	31						-45	134556	23056	43.0	43.0	
	(4.2)	**26**	**5.7**			**22**	**23**		**-2**	**942**	**308**	**3.4**	**2.1**	
		57	16						-34	64489	18456	75.7	65.3	
300	(3.5)	**78**	**9.3**	**712**	**348**	**20**	**(26) 24**	**28**	**-6**	**(4.12) 538**	**(3.67) 234**	**(1.24) 1.29**	**0.93**	**0.58**
		154	47	1047	4031				-91	(111) 21786	(142) 10085	(28) 28	28	28
	(4.1)	**75**	**9.1**			**20**	**23**		**-2**	**607**	**277**	**4.1**	**3.4**	
		161	32						-47	23129	11120	259	259	
	(4.2)	**39**	**7.6**			**21**	**20**		**-3**	**481**	**220**	**13.1**	**12.0**	
		71	32						-44	18229	11285	558	558	

[1]The results in this column are in seconds.

Table 4.3: Effect of n and Nester Algorihm: CPU times, number of optimal solutions, and average and worst case gaps of lower bounds and heuristics

The gaps for the lower bound and heuristics are reported in columns 10-15, where the percentage gap for a cost C is computed as $\frac{C-C_{OPT}}{C_{OPT}}$. The column TR indicates the gap of the lower bound obtained from the transportation problem. The remaining columns report the optimality gaps of the heuristics.

The results indicate that NPH is always the best with respect to both the number of optimal solutions and optimality gaps, and is followed by the last completion time, median completion time, and average completion time heuristics. The optimality gaps and the number of optimal solutions of the heuristics clearly show that the Nester Algorithm improves the results. Observe that, the optimality gaps of the LCT heuristic are not affected by NA. One intuitive explanation is that, the reason for a job to be preempted and scheduled after its due date is, the priority of this job is smaller than those of that made the preemption. Since NA operates with the relative priorities and the preempted tardy jobs are scheduled in that order because of their relative priorities, the algorithm cannot change the last completion time of the jobs which are scheduled after their due dates.

Note that the performance of the heuristics with our cost coefficients improves as the number of jobs increases. This result cannot be seen with the other cost coefficients and observe that there is a great difference between the optimality gaps obtained with cost coefficients (4.1) and (4.2), and the optimality gaps obtained with our cost coefficients, given in (3.5). However our cost coefficients yield worse lower bounds than those yielded by the others. The reason for this result is explained in Section 3.1.2. As a matter of fact, although the set of cost coefficients in (4.2) satisfies $\sum_{k=C_j-p_j+1}^{C_j} c_{jk} = w_j T_j$ for all possible completion times C_j, the set of cost coefficients in (4.1) gives better lower bounds and optimality gaps, recall that these cost coefficients satisfy the equality only for completion times $C_j \geq d_j + p_j$.

Observe that, the CPU times of Nester Algorithm and Non-Preemptive Heuristic grow more rapidly compared to those of the Simple Heuristics. On the other hand, these CPU times are negligible compared to those of TR.

In Figure 4.1, we present the distribution of the optimality gaps for all problem sizes. This figure clearly indicates the excellent performance of the Non-Preemptive Heuristic.

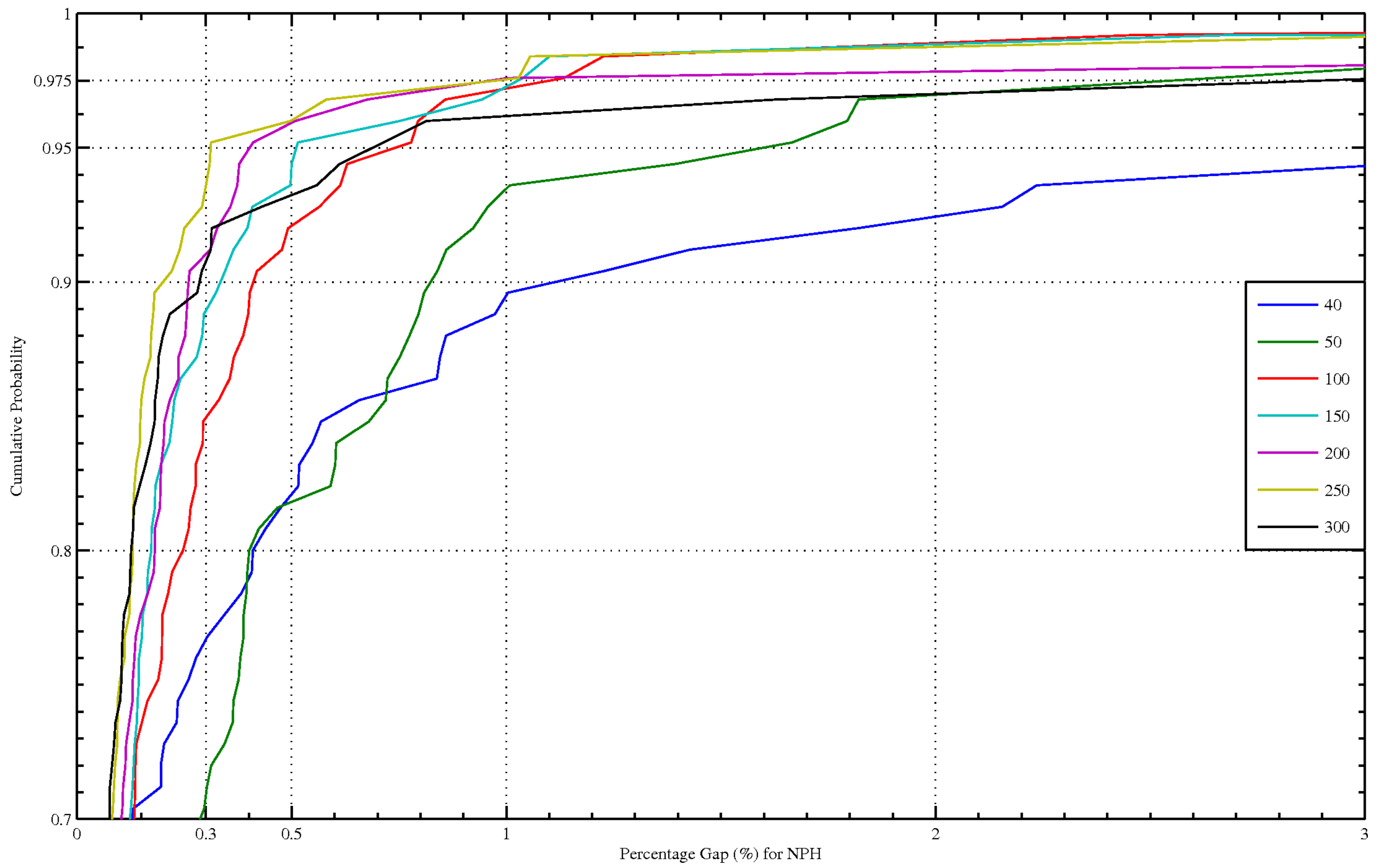


Figure 4.1: Probability distribution of the optimality gaps.

Note that, with probability 0.700, the optimality gaps of 300 and 250-job instances are smaller than 0.09%, those of 200, 150, 100 and 40-job instances are smaller than 0.14%, and that of 50-job instances is smaller than 0.30%. Observe that, the cumulative probability of optimality gaps being smaller than 1.20% is higher than 0.975 for 100, 150, 200, and 250-job instances, and higher than 0.965, 0.940, and 0.900 for 300, 50, and 40-job instances, respectively.

Table 4.4 shows the performance changes of the Non-Preemptive Heuristic in terms of average and worst case optimality gaps, and the last three columns show the number of instances with optimality gaps larger than 1%, 5% and 10%, respectively. The performance improvement as the number of jobs increases is much clearer when the instances with gaps larger than 5% are excluded from the table. Only 24 instances of 875 instances have gaps larger than 5% and 12 of those are higher than 10%. When these 24 instances excluded, the average of optimality gaps improve to between $0.28 - 0.11\%$ from between $1.20 - 0.21\%$ for NPH.

n	Percentage Gap (%)		# Instances w. Gap		
	ave.	max	>1%	$> 5\%$	>10%
40	0.22	3.08	14	7	5
50	0.28	3.45	9	2	1
100	0.15	2.47	4	1	0
150	0.14	2.70	4	1	1
200	0.13	4.40	4	2	1
250	0.11	3.19	4	1	1
300	0.12	3.07	5	3	3

Table 4.4: Effect of n: average and worst case gaps of NPH for instances with gap smaller than or equals to 5%.

4.3.2 Effects of Tardiness and Range of Due Date Factors

In Tables 4.5-4.11, we explore the effect of tardiness (TF) and range of due date (RDD) factors on the percentage gaps of the transportation problem and best heuristics (BH) for all problem sizes, and in Table 4.12 and 4.13, we explore the same effect on the CPU times required to solve TR for problem size $n = 100$ and 300, respectively. In each cell of these tables, results for 5 problem instances are reported. The values in the parenthesis are the worst case performance measures.

	Percentage Gap (%) for TR					Percentage Gap (%) for BH				
TF	RDD					RDD				
	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
0.2	-23.5(-31.6)	-83(-96.0)	0(0)	0(0)	0(0)	3.36(10.1)	6.67(33.3)	0(0)	0(0)	0(0)
	-4.07(-10.5)	-54.7(-91.0)	0(0)	0(0)	0(0)	5.18(14.3)	101(274)	0(0)	0(0)	0(0)
	-7.11(-12.5)	-53.9(-92.2)	0(0)	0(0)	0(0)	13.4(37.6)	110(551)	0(0)	0(0)	0(0)
0.4	-11.5(-12.3)	-13.4(-16.0)	-22.1(-31.3)	-54.9(-72.3)	-18.3(-49.4)	0.23(0.86)	0(0)	2.56(12.8)	10.1(41.7)	2.25(11.2)
	-0.99(-1.54)	-1.99(-3.08)	-6.52(-11.5)	-28.6(-48.8)	-6.09(-16.5)	1.89(4.46)	0.74(2.04)	6.69(14.6)	16.8(27.8)	4.29(15.9)
	-4.30(-5.08)	-4.39(-5.56)	-7.49(-11.3)	-26.2(-43.6)	-6.18(-16.5)	2.74(6.35)	3.05(5.73)	7.45(25.7)	32.3(83.3)	6.12(16.7)
0.6	-9.07(-10.9)	-10.1(-13.5)	-12.5(-14.5)	-17.5(-21.1)	-15.5(-19.6)	0.23(0.66)	1.09(3.08)	0.90(2.24)	0.99(1.82)	0.58(1.43)
	-0.86(-1.67)	-1.86(-3.43)	-2.76(-4.42)	-5.60(-8.18)	-2.78(-3.60)	1.55(2.21)	3.95(11.4)	1.93(5.05)	3.52(5.88)	2.27(4.02)
	-4.23(-5.74)	-4.89(-7.08)	-5.37(-6.66)	-8.07(-10.7)	-6.51(-8.09)	1.42(2.85)	4.05(11.5)	2.31(3.44)	4.06(7.09)	4.04(9.34)
0.8	-5.82(-6.19)	-6.68(-7.27)	-7.39(-7.92)	-8.83(-10.7)	-9.44(-12.0)	0.11(0.34)	0.18(0.41)	0.16(0.3)	0.15(0.47)	0.20(0.52)
	-0.31(-0.44)	-0.88(-1.13)	-0.93(-1.08)	-1.26(-1.92)	-1.12(-1.78)	0.49(0.93)	0.50(1.32)	0.32(0.58)	0.19(0.49)	0.60(1.05)
	-3.02(-3.17)	-3.40(-3.94)	-3.63(-4.12)	-3.97(-5.22)	-4.20(-4.96)	0.58(0.71)	1.62(3.74)	0.99(2.36)	2.23(4.43)	1.26(1.91)
1	-4.74(-5.30)	-5.15(-5.66)	-5.76(-6.64)	-5.13(-6.22)	-6.08(-7.25)	0.01(0.04)	0.03(0.07)	0.06(0.11)	0.01(0.03)	0.20(0.56)
	-0.09(-0.16)	-0.17(-0.35)	-0.31(-0.61)	-0.25(-0.41)	-0.47(-0.77)	0.02(0.07)	0.06(0.2)	0.05(0.11)	0.06(0.12)	0.26(0.56)
	-2.51(-2.84)	-2.66(-2.80)	-3.01(-3.43)	-2.79(-3.53)	-3.15(-3.90)	0.14(0.22)	0.18(0.37)	0.34(0.48)	0.24(0.46)	0.62(1.21)

Table 4.5: Effect of TF and RDD: average and (worst case) percentage gaps of optimality and lower bounds for $n = 40$

	Percentage Gap (%) for TR					Percentage Gap (%) for BH				
TF	RDD					RDD				
	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
0.2	-20.3(-24.0)	-83(-98.4)	0(0)	0(0)	0(0)	0.43(1.39)	0.53(2.65)	0(0)	0(0)	0(0)
	-3.46(-8.29)	-45.9(-59.6)	0(0)	0(0)	0(0)	5.17(14.1)	7.71(33.5)	0(0)	0(0)	0(0)
	-7.59(-10.3)	-20.1(-63.6)	0(0)	0(0)	0(0)	11.1(18.6)	428(2100)	0(0)	0(0)	0(0)
0.4	-9.9(-13.5)	-12.3(-19.5)	-19.4(-26.0)	-38.8(-68.4)	-19.4(-33.9)	0.52(1.82)	0(0.60)	2.16(5.33)	0.35(0.72)	2.73(12.8)
	-1.06(-2.63)	-2.04(-5.85)	-6.29(-9.77)	-14.5(-23.9)	-7.96(-15.9)	0.28(0.72)	1.00(2.63)	8.36(27.5)	3.28(8.09)	8.27(20.3)
	-4.36(-6.57)	-4.59(-8.45)	-8.57(-12.8)	-16.5(-24.6)	-7.79(-16.4)	1.48(3.45)	3.72(14.9)	9.66(27.3)	3.48(14.0)	26.2(66.9)
0.6	-5.92(-6.46)	-7.9(-8.83)	-10.3(-12.8)	-13.5(-16.3)	-17.3(-23.4)	0.27(0.8)	0.08(0.22)	0.82(1.79)	0.31(0.68)	0.58(0.96)
	-0.28(-0.40)	-0.95(-1.60)	-2.24(-2.67)	-3.42(-4.74)	-3.67(-5.83)	0.76(2.66)	1.10(2.55)	3.53(8.07)	2.46(5.60)	2.46(5.70)
	-2.86(-3.62)	-3.76(-4.10)	-4.61(-5.65)	-5.40(-6.20)	-6.39(-7.91)	1.12(2.08)	1.29(2.56)	4.42(7.08)	2.47(4.05)	2.07(3.90)
0.8	-4.92(-5.33)	-5.95(-6.35)	-6.90(-7.58)	-7.67(-9.87)	-6.98(-7.94)	0.27(0.72)	0.18(0.36)	0.27(0.39)	0.55(1.01)	0.08(0.23)
	-0.31(-0.50)	-0.79(-1.10)	-1.10(-1.29)	-1.20(-1.79)	-0.92(-1.15)	0.32(0.75)	0.40(0.65)	0.49(0.77)	1.57(3.23)	0.22(0.48)
	-2.72(-2.95)	-2.97(-3.31)	-3.23(-3.44)	-3.79(-4.61)	-3.21(-3.97)	0.91(1.62)	1.12(2.49)	0.93(1.24)	2.43(4.81)	0.87(1.17)
1	-3.67(-4.49)	-4.05(-4.27)	-4.51(-5.30)	-4.47(-4.77)	-5.40(-6.17)	0.02(0.06)	0.02(0.09)	0.04(0.06)	0.10(0.20)	0.12(0.24)
	-0.07(-0.09)	-0.13(-0.17)	-0.26(-0.57)	-0.24(-0.41)	-0.52(-0.75)	0.05(0.09)	0.06(0.13)	0.15(0.39)	0.13(0.20)	0.24(0.70)
	-2.09(-2.36)	-2.27(-2.40)	-2.42(-2.92)	-2.41(-2.53)	-2.87(-3.05)	0.11(0.23)	0.21(0.32)	0.53(1.08)	0.29(0.40)	0.98(1.66)

Table 4.6: Effect of TF and RDD: average and (worst case) gaps of optimality and lower bounds for $n = 50$

	Percentage Gap (%) for TR					Percentage Gap (%) for BH				
TF	RDD					RDD				
	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
0.2	-12.1(-13.7)	-72.0(-99.0)	0(0)	0(0)	0(0)	0.23(1.13)	0(0)	0(0)	0(0)	0(0)
	-1.57(-2.95)	-39.2(-59.7)	0(0)	0(0)	0(0)	1.11(3.87)	598(2225)	0(0)	0(0)	0(0)
	-4.39(-7.14)	-40.8(-62.5)	0(0)	0(0)	0(0)	4.60(8.82)	723(2225)	0(0)	0(0)	0(0)
0.4	-4.90(-5.03)	-6.91(-7.76)	-9.97(-11.7)	-55.7(-88.5)	-24.0(-84.2)	0.16(0.39)	0.64(2.47)	0.21(0.57)	1.50(7.5)	0.08(0.42)
	-0.22(-0.36)	-1.07(-1.58)	-2.10(-3.06)	-25.3(-45.0)	-10.1(-34.9)	0.18(0.55)	0.69(1.34)	3.28(8.56)	24.1(88.6)	2.90(8.89)
	-2.43(-2.65)	-3.11(-3.52)	-3.50(-4.47)	-16.8(-32.7)	-11.3(-37.5)	0.69(1.36)	1.97(4.8)	5.68(8.64)	39.8(111)	8.36(36.2)
0.6	-3.32(-3.60)	-3.95(-4.43)	-5.21(-7.09)	-8.46(-10.3)	-7.13(-10.0)	0.06(0.2)	0.07(0.14)	0.52(0.86)	0.57(1.23)	0.34(0.61)
	-0.19(-0.25)	-0.43(-0.53)	-0.98(-1.69)	-2.03(-2.60)	-1.40(-2.17)	0.07(0.11)	0.34(0.52)	1.04(3.45)	2.32(3.96)	1.12(1.82)
	-1.91(-2.02)	-1.92(-2.09)	-2.36(-3.32)	-3.56(-4.62)	-3.21(-4.31)	0.32(0.81)	0.72(1.7)	3.75(4.81)	4.14(5.9)	2.25(3.22)
0.8	-2.45(-2.61)	-3.03(-3.47)	-3.54(-4.09)	-3.72(-4.46)	-3.57(-3.81)	0.04(0.08)	0.16(0.4)	0.15(0.28)	0.24(0.37)	0.15(0.2)
	-0.14(-0.17)	-0.41(-0.53)	-0.50(-0.73)	-0.43(-0.63)	-0.41(-0.46)	0.15(0.23)	0.51(0.79)	0.33(0.48)	0.57(1.22)	0.43(0.76)
	-1.51(-1.56)	-1.75(-1.98)	-1.91(-2.09)	-1.96(-2.40)	-1.90(-1.98)	0.29(0.64)	0.75(1.21)	0.92(1.59)	1.12(1.67)	0.75(1.01)
1	-1.96(-2.10)	-1.99(-2.06)	-2.18(-2.28)	-2.31(-2.53)	-2.63(-2.87)	0.01(0.01)	0.03(0.04)	0.03(0.05)	0.03(0.08)	0.07(0.12)
	-0.03(-0.04)	-0.06(-0.07)	-0.08(-0.11)	-0.10(-0.16)	-0.19(-0.22)	0.01(0.04)	0.04(0.06)	0.08(0.16)	0.10(0.21)	0.15(0.25)
	-1.25(-1.33)	-1.27(-1.31)	-1.39(-1.48)	-1.45(-1.56)	-1.56(-1.67)	0.08(0.13)	0.10(0.23)	0.13(0.2)	0.19(0.29)	0.44(0.63)

Table 4.7: Effect of TF and RDD: average and (worst case) gaps of optimality and lower bounds for $n = 100$

	Percentage Gap (%) for TR					Percentage Gap (%) for BH				
TF	RDD					RDD				
	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
0.2	-9.01(-11.0)	-57.7(-86.4)	0(0)	0(0)	0(0)	0.95(2.7)	11.2(56.2)	0(0)	0(0)	0(0)
	-0.99(-1.66)	-28.0(-44.6)	0(0)	0(0)	0(0)	1.82(3.87)	16.7(83.3)	0(0)	0(0)	0(0)
	-3.43(-4.84)	-22.7(-50.0)	0(0)	0(0)	0(0)	5.09(8.43)	29.4(83.3)	0(0)	0(0)	0(0)
0.4	-3.49(-4.17)	-4.30(-5.00)	-7.90(-9.57)	-62.3(-90.9)	-10.5(-52.3)	0.02(0.07)	0.29(0.50)	0.29(1.04)	0(0)	0(0)
	-0.20(-0.35)	-0.49(-0.62)	-1.80(-3.06)	-28.2(-44.0)	-6.82(-34.1)	0.15(0.28)	1.07(2.25)	1.09(1.89)	5.00(25.0)	3.87(19.4)
	-1.86(-2.01)	-2.01(-2.11)	-2.99(-4.32)	-13.3(-27.4)	-7.37(-36.8)	0.60(1.46)	2.58(5.43)	5.84(19.8)	42.6(156)	9.08(45.4)
0.6	-2.27(-2.33)	-2.65(-3.31)	-3.46(-3.80)	-5.55(-7.53)	-7.48(-8.75)	0.09(0.16)	0.04(0.15)	0.16(0.32)	0.37(0.75)	0.32(0.51)
	-0.12(-0.17)	-0.29(-0.66)	-0.66(-0.88)	-1.35(-1.80)	-2.12(-2.66)	0.26(0.37)	0.18(0.44)	0.69(1.9)	1.76(2.09)	2.06(2.87)
	-1.38(-1.42)	-1.49(-1.93)	-1.71(-2.00)	-2.48(-2.99)	-2.97(-3.47)	0.32(0.59)	0.39(0.99)	1.01(1.76)	1.56(2.52)	3.07(3.62)
0.8	-1.63(-1.69)	-2.07(-2.18)	-2.64(-2.83)	-3.15(-3.51)	-4.11(-4.74)	0.02(0.04)	0.12(0.18)	0.11(0.23)	0.15(0.22)	0.16(0.24)
	-0.07(-0.08)	-0.27(-0.32)	-0.44(-0.54)	-0.66(-0.75)	-0.95(-1.19)	0.09(0.14)	0.38(0.96)	0.41(0.59)	0.83(1.82)	1.14(1.58)
	-1.04(-1.12)	-1.20(-1.29)	-1.48(-1.58)	-1.69(-1.92)	-1.97(-2.14)	0.18(0.19)	0.64(1.41)	0.81(0.99)	1.63(3.08)	1.36(2.2)
1	-1.34(-1.38)	-1.59(-1.64)	-1.76(-1.87)	-2.23(-2.40)	-2.40(-2.70)	0.02(0.04)	0.02(0.05)	0.03(0.05)	0.08(0.11)	0.12(0.14)
	-0.04(-0.05)	-0.10(-0.14)	-0.16(-0.20)	-0.33(-0.41)	-0.41(-0.50)	0.06(0.1)	0.07(0.12)	0.16(0.39)	0.45(0.71)	0.62(1.24)
	-0.92(-0.95)	-1.04(-1.08)	-1.11(-1.15)	-1.34(-1.47)	-1.34(-1.46)	0.07(0.09)	0.19(0.29)	0.27(0.58)	0.64(1.08)	0.99(1.86)

Table 4.8: Effect of TF and RDD: average and (worst case) percentage gaps of optimality and lower bounds for $n = 150$

TF	Percentage Gap (%) for TR					Percentage Gap (%) for BH				
	RDD					RDD				
	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
0.2	-6.21(-7.33)	-76.47(-97.7)	0(0)	0(0)	0(0)	0.05(0.19)	3.42(12.7)	0(0)	0(0)	0(0)
	-0.39(-0.64)	-55.0(-89.6)	0(0)	0(0)	0(0)	1.14(1.77)	7.26(23.6)	0(0)	0(0)	0(0)
	-2.57(-3.41)	-52.3(-84.6)	0(0)	0(0)	0(0)	2.61(6.07)	54.6(115)	0(0)	0(0)	0(0)
0.4	-2.70(-3.06)	-3.62(-3.89)	-6.75(-7.63)	-37.1(-47.3)	0(0)	0.09(0.22)	0.08(0.31)	0.19(0.33)	1.92(9.62)	0(0)
	-0.14(-0.27)	-0.40(-0.67)	-1.38(-1.56)	-17.0(-30.0)	0(0)	0.50(0.77)	0.65(1.75)	1.20(3.07)	18.68(60.9)	0(0)
	-1.48(-1.62)	-1.76(-1.97)	-3.05(-3.69)	-16.9(-31.1)	0(0)	0.82(1.21)	1.42(2.43)	1.31(2.06)	33.22(64.6)	0(0)
0.6	-1.75(-1.81)	-2.00(-2.23)	-2.47(-2.81)	-4.23(-5.24)	-8.76(-12.0)	0.03(0.06)	0.14(0.26)	0.09(0.2)	0.34(0.51)	0.51(1)
	-0.08(-0.15)	-0.23(-0.26)	-0.42(-0.49)	-1.00(-1.16)	-2.47(-3.46)	0.13(0.23)	0.49(0.69)	0.79(2.7)	0.96(1.56)	4.11(9.05)
	-1.10(-1.16)	-1.16(-1.30)	-1.20(-1.33)	-1.95(-2.58)	-3.68(-4.70)	0.23(0.42)	1.09(2.86)	1.48(2.81)	1.96(2.93)	5.46(8.35)
0.8	-1.23(-1.24)	-1.55(-1.66)	-1.88(-1.92)	-2.42(-2.56)	-3.09(-3.70)	0.03(0.07)	0.03(0.08)	0.09(0.2)	0.11(0.18)	0.17(0.25)
	-0.06(-0.06)	-0.18(-0.27)	-0.30(-0.35)	-0.45(-0.50)	-0.71(-0.93)	0.15(0.3)	0.23(0.69)	0.52(0.88)	0.30(0.39)	1.06(2.69)
	-0.83(-0.85)	-0.94(-0.97)	-1.07(-1.10)	-1.25(-1.30)	-1.49(-1.78)	0.22(0.37)	0.44(0.84)	0.71(0.95)	0.67(0.99)	1.42(2.62)
1	-1.01(-1.05)	-1.12(-1.16)	-1.34(-1.41)	-1.58(-1.70)	-1.91(-2.03)	0.01(0.01)	0.02(0.02)	0.04(0.06)	0.07(0.11)	0.15(0.24)
	-0.03(-0.03)	-0.07(-0.08)	-0.12(-0.15)	-0.20(-0.26)	-0.30(-0.35)	0.05(0.12)	0.05(0.08)	0.13(0.18)	0.27(0.65)	0.49(0.88)
	-0.71(-0.75)	-0.75(-0.78)	-0.85(-0.94)	-0.96(-1.03)	-1.10(-1.14)	0.04(0.06)	0.09(0.12)	0.23(0.28)	0.34(0.50)	0.64(0.92)

Table 4.9: Effect of TF and RDD: average and (worst case) gaps of optimality and lower bounds for $n = 200$

TF	Percentage Gap (%) for TR					Percentage Gap (%) for BH				
	RDD					RDD				
	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
0.2	-5.47(-5.68)	-58.0(-94.6)	0(0)	0(0)	0(0)	0.23(0.98)	0(0)	0(0)	0(0)	0(0)
	-0.54(-0.90)	-29.2(-45.2)	0(0)	0(0)	0(0)	2.32(5.69)	0(0)	0(0)	0(0)	0(0)
	-2.40(-2.85)	-13.1(-31.6)	0(0)	0(0)	0(0)	4.34(10.7)	7.90(36.8)	0(0)	0(0)	0(0)
0.4	-2.14(-2.28)	-2.96(-3.49)	-4.78(-6.01)	-48.7(-59.7)	0(0)	0.01(0.04)	0.15(0.31)	0.13(0.30)	9.04(41.0)	0(0)
	-0.08(-0.18)	-0.34(-0.53)	-0.84(-1.44)	-23.9(-31.1)	0(0)	0.38(1.03)	1.21(2.76)	1.37(3.25)	19.7(43.0)	0(0)
	-1.16(-1.23)	-1.43(-1.62)	-2.05(-2.82)	-24.2(-34.1)	0(0)	0.50(0.8)	1.36(2.89)	1.82(3.86)	24.5(65.3)	0(0)
0.6	-1.39(-1.43)	-1.68(-1.76)	-2.15(-2.33)	-3.56(-4.16)	-5.77(-7.47)	0.01(0.02)	0.06(0.16)	0.10(0.18)	0.23(0.5)	0.27(0.58)
	-0.06(-0.08)	-0.22(-0.27)	-0.39(-0.53)	-0.86(-0.95)	-1.72(-2.25)	0.20(0.29)	0.35(0.76)	0.33(0.72)	1.29(2.22)	3.12(5.02)
	-0.90(-0.93)	-0.96(-1.03)	-1.11(-1.19)	-1.60(-2.02)	-2.40(-3.07)	0.22(0.51)	0.63(1.00)	0.78(1.09)	1.86(2.31)	3.10(4.65)
0.8	-1.01(-1.07)	-1.30(-1.35)	-1.63(-1.80)	-1.97(-2.11)	-2.38(-2.80)	0.01(0.01)	0.06(0.07)	0.09(0.15)	0.09(0.13)	0.16(0.22)
	-0.05(-0.06)	-0.15(-0.18)	-0.26(-0.29)	-0.38(-0.44)	-0.51(-0.58)	0.06(0.08)	0.19(0.24)	0.37(0.54)	0.51(1.67)	0.78(1.32)
	-0.69(-0.73)	-0.81(-0.84)	-0.92(-0.99)	-1.04(-1.13)	-1.15(-1.38)	0.12(0.15)	0.38(0.5)	0.57(0.91)	0.65(1.21)	1.29(1.54)
1	-0.79(-0.81)	-0.94(-0.97)	-1.09(-1.11)	-1.27(-1.38)	-1.53(-1.59)	0.01(0.02)	0.02(0.04)	0.05(0.13)	0.04(0.07)	0.07(0.13)
	-0.02(-0.02)	-0.06(-0.07)	-0.11(-0.14)	-0.15(-0.17)	-0.25(-0.26)	0.03(0.05)	0.07(0.12)	0.21(0.39)	0.22(0.36)	0.33(0.46)
	-0.57(-0.59)	-0.65(-0.67)	-0.71(-0.73)	-0.78(-0.85)	-0.86(-0.90)	0.05(0.07)	0.12(0.25)	0.29(0.48)	0.42(1.11)	0.54(0.84)

Table 4.10: Effect of TF and RDD: average and (worst case) gaps of optimality and lower bounds for $n = 250$

	Percentage Gap (%) for TR					Percentage Gap (%) for BH				
TF	RDD					RDD				
	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
0.2	-4.18(-4.37)	-65.1(-90.7)	0(0)	0(0)	0(0)	0.27(0.56)	9.24(27.7)	0(0)	0(0)	0(0)
	-0.28(-0.48)	-30.9(-42.7)	0(0)	0(0)	0(0)	0.77(1.35)	12.7(35.8)	0(0)	0(0)	0(0)
	-1.78(-1.96)	-21.0(-44.4)	0(0)	0(0)	0(0)	2.14(4.95)	197(558)	0(0)	0(0)	0(0)
0.4	-1.78(-1.90)	-2.45(-2.63)	-4.84(-6.21)	-47.9(-87.2)	0(0)	0.07(0.2)	0.04(0.10)	0.41(1.63)	3.10(12.3)	0(0)
	-0.07(-0.11)	-0.20(-0.28)	-0.95(-1.70)	-25.0(-46.9)	0(0)	0.14(0.22)	0.26(0.68)	2.51(6.68)	62.4(259)	0(0)
	-1.03(-1.10)	-1.31(-1.38)	-2.23(-2.84)	-25.5(-43.8)	0(0)	0.69(1.21)	0.68(2.01)	3.35(6.46)	85.8(259)	0(0)
0.6	-1.17(-1.19)	-1.34(-1.51)	-1.76(-1.85)	-3.00(-3.93)	-4.82(-5.90)	0.03(0.08)	0.05(0.13)	0.05(0.08)	0.28(0.43)	0.47(0.81)
	-0.04(-0.05)	-0.13(-0.19)	-0.32(-0.38)	-0.73(-1.10)	-1.42(-1.86)	0.22(0.41)	0.10(0.14)	0.32(0.49)	1.52(3.05)	2.67(5.23)
	-0.77(-0.79)	-0.79(-0.85)	-0.93(-0.96)	-1.37(-1.94)	-2.07(-2.73)	0.30(0.53)	0.33(0.64)	0.61(0.87)	2.45(5.14)	3.79(7.00)
0.8	-0.85(-0.88)	-1.12(-1.20)	-1.31(-1.38)	-1.59(-1.73)	-1.95(-2.04)	0.01(0.03)	0.06(0.13)	0.05(0.08)	0.09(0.12)	0.13(0.19)
	-0.04(-0.05)	-0.15(-0.19)	-0.18(-0.22)	-0.30(-0.37)	-0.37(-0.42)	0.03(0.04)	0.19(0.31)	0.15(0.29)	0.37(0.62)	0.50(1.23)
	-0.59(-0.61)	-0.69(-0.75)	-0.76(-0.81)	-0.84(-0.92)	-0.99(-1.07)	0.09(0.11)	0.40(0.57)	0.32(0.46)	0.49(0.97)	0.86(1.41)
1	-0.68(-0.70)	-0.77(-0.81)	-0.92(-0.98)	-1.11(-1.19)	-1.34(-1.39)	0.01(0.02)	0.01(0.02)	0.03(0.04)	0.05(0.07)	0.06(0.09)
	-0.02(-0.02)	-0.04(-0.05)	-0.09(-0.12)	-0.13(-0.17)	-0.22(-0.25)	0.03(0.05)	0.07(0.14)	0.16(0.31)	0.14(0.19)	0.28(0.44)
	-0.49(-0.51)	-0.54(-0.55)	-0.61(-0.63)	-0.69(-0.72)	-0.79(-0.83)	0.06(0.1)	0.09(0.14)	0.21(0.32)	0.27(0.37)	0.34(0.48)

Table 4.11: Effect of TF and RDD: average and (worst case) gaps of optimality and lower bounds for $n = 300$

For each value of TF, the first row indicates the measures that are obtained with our cost coefficients, and the second and third rows indicate those obtained with cost coefficients in (4.1) and (4.2), respectively.

Tables 4.5-4.11 indicate that for all cost coefficients, for a given value of RDD, the quality of both average and worst case of percentage gaps for TR improves as TF increases, except for the cells with $TF = 0.2$ and $RDD = \{0.6, 0.8, 1\}$. In a similar manner, for a given value of TF, the quality of those improves as RDD decreases, except for the cells with $TF = 0.4$ and $RDD = 0.8$, and $TF = 0.2$ and $RDD = \{0.6, 0.8, 1\}$. On the other hand, such general statements cannot be made for the gaps of BH.

In Table 4.12 and 4.13, the CPU times required to solve TR for $n = 100$ and 300 appear. Observe that, the instances with $TF = 0.2$ and $RD = \{0.6, 0.8, 1\}$ are the easiest instances in terms of the gaps and the CPU times. Optimal objective values of these instances are 0, and the maximum and average CPU times of these instances are the shortest ones.

TF	RDD				
	0.2	0.4	0.6	0.8	1
0.2	2.24(2.44)	2.20(2.91)	1.62(1.72)	1.61(1.75)	1.65(1.78)
	2.22(2.50)	2.23(2.92)	1.61(1.69)	1.61(1.77)	1.64(1.78)
	2.00(2.19)	2.17(2.72)	1.61(1.70)	1.61(1.75)	1.64(1.77)
0.4	2.51(2.74)	2.48(2.72)	2.91(3.81)	2.99(4.00)	2.62(3.97)
	2.48(2.58)	2.49(2.75)	2.91(3.59)	3.03(4.36)	2.64(3.80)
	2.30(2.47)	2.32(2.55)	2.59(3.16)	2.82(3.81)	2.51(3.25)
0.6	3.39(3.86)	3.11(3.59)	3.81(4.30)	4.74(5.42)	5.53(6.28)
	3.30(3.95)	3.06(3.75)	3.60(4.36)	4.49(5.11)	5.22(5.73)
	2.73(2.98)	2.65(3.20)	3.17(3.42)	3.63(4.03)	3.78(4.03)
0.8	3.61(4.24)	4.06(4.28)	3.70(4.00)	4.27(5.02)	4.24(5.03)
	3.50(4.17)	3.79(4.17)	3.59(4.13)	3.98(4.52)	4.05(4.69)
	2.88(3.33)	3.15(3.39)	2.88(3.16)	3.14(3.50)	3.25(4.08)
1	2.23(2.78)	2.48(2.78)	2.60(2.99)	3.57(4.23)	3.90(4.72)
	2.15(2.70)	2.44(2.92)	2.59(2.91)	3.39(4.00)	3.65(4.56)
	2.26(2.84)	2.46(2.98)	2.52(2.88)	3.23(3.91)	3.10(3.63)

Table 4.12: Effect of TF and RDD: average and (maximum) of the CPU times required to solve TR, $n = 100$

TF	*RDD*				
	0.2	0.4	0.6	0.8	1
0.2	38.4(43.2)	24.8(27.6)	16.9(17.8)	16.4(16.7)	16.8(17.4)
	40.0(46.6)	24.4(26.8)	16.8(17.9)	16.4(16.8)	16.7(17.3)
	26.3(29.5)	24.5(28.4)	17.0(18.0)	16.5(16.9)	17.5(19.6)
0.4	69.7(83.9)	57.4(62.7)	51.6(64.3)	45.1(56.5)	23.5(27.9)
	69.7(81.7)	55.9(57.0)	50.5(64.4)	44.3(51.2)	23.9(26.9)
	34.0(37.0)	34.2(35.9)	37.6(48.9)	43.7(53.3)	23.7(26.0)
0.6	90.3(107)	81.8(94.8)	88.4(102)	77.8(90.9)	82.3(105)
	85.1(92.3)	77.7(93.6)	82.7(92.0)	74.4(87.8)	78.0(95.2)
	35.7(41.0)	37.1(39.4)	46.1(51.9)	51.1(64.2)	60.1(71.5)
0.8	127(143)	115(128)	97.7(105)	95.8(106)	90.8(113)
	129(157)	112(129)	103(111)	82.5(90.1)	82.8(111)
	45.6(53.9)	43.8(49.3)	44.5(50.2)	45.8(48.1)	49.1(55.6)
1	140(153)	131(138)	131(154)	113(124)	116(135)
	140(148)	133(144)	131(161)	108(129)	106(125)
	48.5(51.8)	48.3(52.1)	49.3(57.2)	44.8(47.8)	48.1(55.5)

Table 4.13: Effect of TF and RDD: average and (maximum) of the CPU times required to solve TR, $n = 300$

4.3.3 Statistics on Parentheses

In this section, we investigate whether the number of jobs, the tardiness and range of due date factors effect the parenthesis structure of the instances. We use three informations based on parenthesis statistics to explore the effects of the factors. These are AVJP, MAJP, and MIJP (i.e. the average, maximum, and minimum number of jobs in the parentheses of an instance). Also, we present the empirical distribution of the sizes of parentheses.

Effect of the Factors

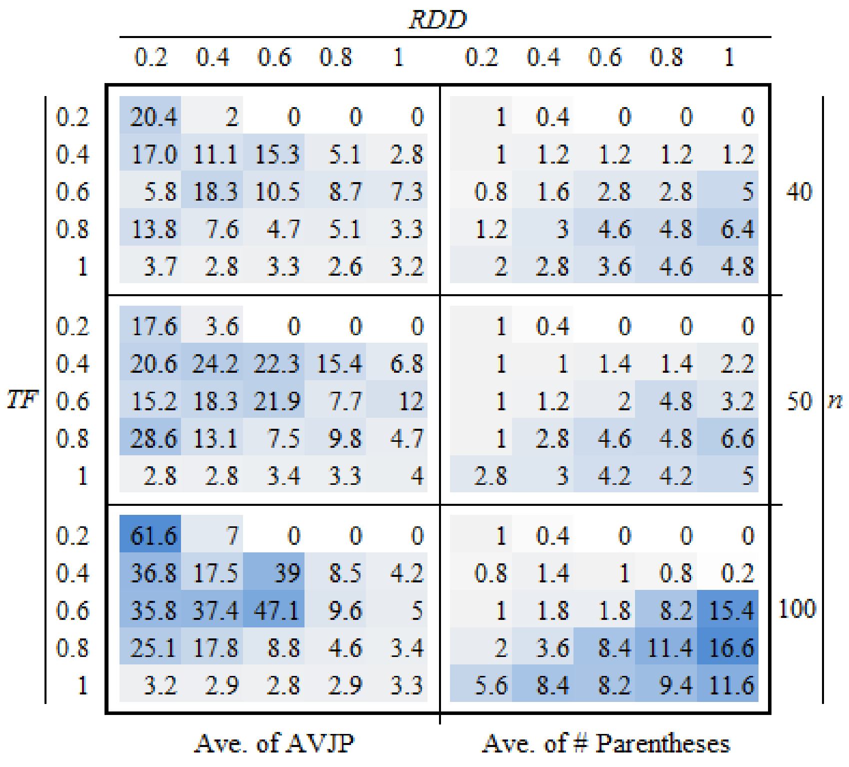
In Table 4.14, we explore the effect of number of jobs. In column 2 and 3, the number of instances that have at least one parenthesis, and the average and maximum number of jobs in those parenthesis are reported. The last two columns are the average of AVJP and MAJP. Although all the parameters grow almost linearly with n, the correlation of the maximum number of jobs in the parentheses and n is much more significant than the others.

In Figure 4.2, we investigate the relation between the factors, and the average of

n			Average of	
	# Ins.	# Par.	AVJP	MAJP
40	103.00	2.82(7)	8.5	11.1
50	103.00	2.89(9)	12.9	16.4
100	100.00	5.95(18)	19.2	25.3

Table 4.14: Effect of n: number of instances that have parenthesis, average and (maximum) number of parenthesis per instance, and average number of AVJP and MAJP

AVJP and number of parentheses. The result clearly indicates the relation that for a given problem size n, while TF and RDD are decreasing, the average numbers of parentheses and AVJP are generally increasing and decreasing, respectively.



		Ave. of AVJP					Ave. of # Parentheses				
n	*TF* \ *RDD*	0.2	0.4	0.6	0.8	1	0.2	0.4	0.6	0.8	1
40	0.2	20.4	2	0	0	0	1	0.4	0	0	0
40	0.4	17.0	11.1	15.3	5.1	2.8	1	1.2	1.2	1.2	1.2
40	0.6	5.8	18.3	10.5	8.7	7.3	0.8	1.6	2.8	2.8	5
40	0.8	13.8	7.6	4.7	5.1	3.3	1.2	3	4.6	4.8	6.4
40	1	3.7	2.8	3.3	2.6	3.2	2	2.8	3.6	4.6	4.8
50	0.2	17.6	3.6	0	0	0	1	0.4	0	0	0
50	0.4	20.6	24.2	22.3	15.4	6.8	1	1	1.4	1.4	2.2
50	0.6	15.2	18.3	21.9	7.7	12	1	1.2	2	4.8	3.2
50	0.8	28.6	13.1	7.5	9.8	4.7	1	2.8	4.6	4.8	6.6
50	1	2.8	2.8	3.4	3.3	4	2.8	3	4.2	4.2	5
100	0.2	61.6	7	0	0	0	1	0.4	0	0	0
100	0.4	36.8	17.5	39	8.5	4.2	0.8	1.4	1	0.8	0.2
100	0.6	35.8	37.4	47.1	9.6	5	1	1.8	1.8	8.2	15.4
100	0.8	25.1	17.8	8.8	4.6	3.4	2	3.6	8.4	11.4	16.6
100	1	3.2	2.9	2.8	2.9	3.3	5.6	8.4	8.2	9.4	11.6

Figure 4.2: Effect of the factors: average of AVJP and number of parentheses

Empirical Distribution of Sizes of the Parentheses

In Figures 4.3-4.5, we present the distributions of the sizes of the parentheses for the problem size n equal to 40, 50, and 100-jobs, respectively. In these figures, the Min, Max, and Ave. refer to the distribution of MIJP, MAJP, and AVJP, respectively. We note that the probability of an instance having a parenthesis with $\frac{n}{4}$ or less jobs is between 60 – 80% and with probability 0.18, NA yields a non-preemptive schedule.

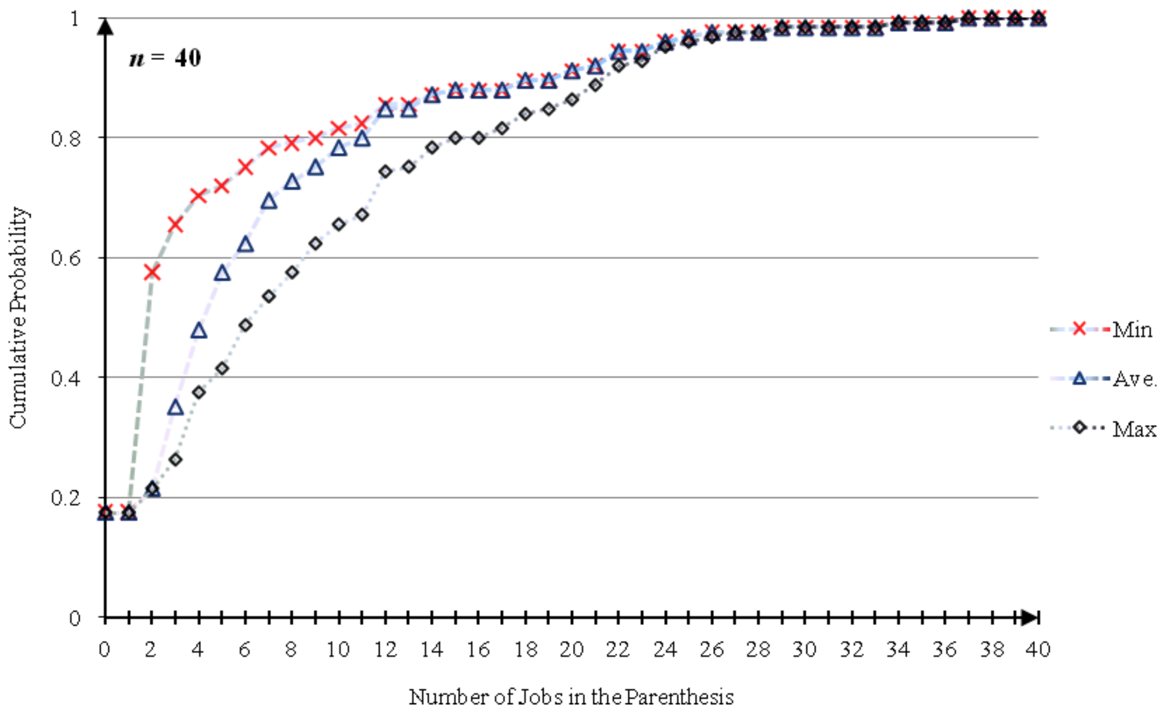


Figure 4.3: Distribution of sizes of the parentheses, $n = 40$

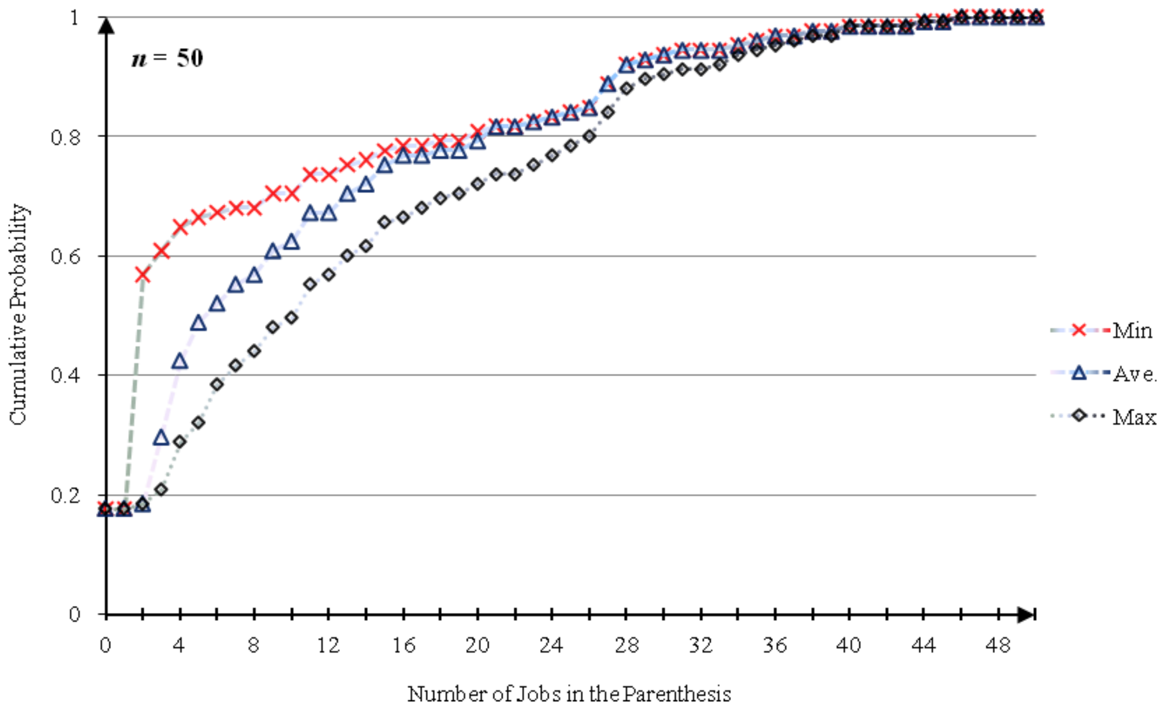


Figure 4.4: Distribution of sizes of the parentheses, $n = 50$

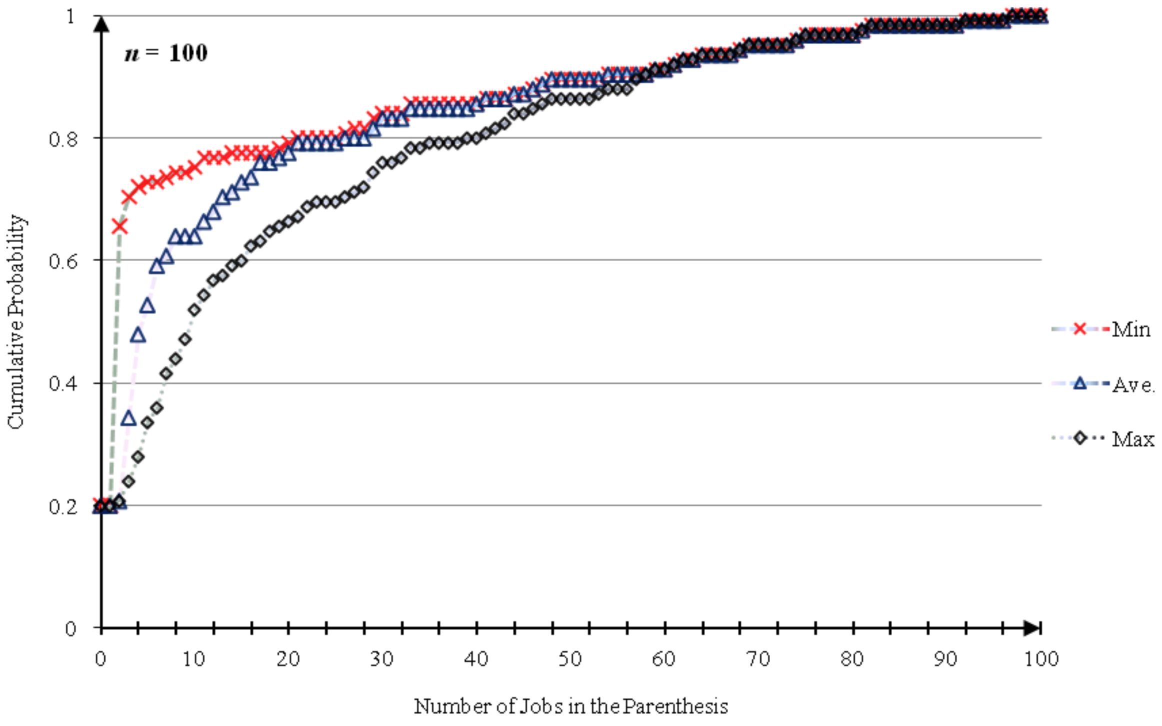


Figure 4.5: Distribution of sizes of the parentheses, $n = 100$

4.3.4 Time Decomposition of TWT

As we mention at the beginning of this chapter, after obtaining the nested schedule, we solve an IP to obtain the optimal schedules of the parentheses, in order to investigate whether our solution approach may be used as a decomposition method. The IP is stated as:

$$\textbf{(IP1)} \qquad \min \sum_{j=1}^{n} \sum_{t=r_j}^{T_{\max}-p_j+1} w_j \left(t + p_j - 1 - d_j\right)^+ x_{jt} \tag{4.3}$$

$$\sum_{t=r_j}^{T_{\max}-p_j+1} x_{jt} = 1 \qquad \forall j \tag{4.4}$$

$$\sum_{j=1}^{n} \sum_{\substack{s= \\ \max(0,t-p_j+1)}}^{t} x_{js} \leq 1 \qquad \forall t \in H \tag{4.5}$$

$$x_{jt} \in \{0,1\} \qquad \forall j, t \in H, \tag{4.6}$$

where $(x)^+ = \max(0, x)$, x_{jt} is a decision variable equal to 1 if a job j starts processing in period t, and equal to 0 otherwise. The planning horizon is defined as $H =$

$[\min_j r_j, T_{\max}]$, where $T_{\max} = P + \max_j r_j$ and P is the sum of all processing times. The objective is to minimize the total weighted tardiness. The constraints (4.4) ensure that all jobs are scheduled. The constraints (4.5) and (4.6) together ensure that at most one job is processed at any point in time and that processing is non-preemptive.

The gaps that are obtained by solving parentheses to optimality are in Table 4.15. In columns 2 and 4, we report the number of instances with 25 or less jobs in the largest parenthesis (i.e. MAJP) and the number of instances whose optimal schedule is found among those, respectively.

n	# Ins.[1]	Ave. Opt. Gaps	# Opt.
40	120	0.891%	74
50	97	0.121%	55
100	87	0.089%	33

[1] Instances with 25 or less jobs in the parentheses are solved.

Table 4.15: Solving Parentheses with IP

Although the average optimality gaps are less than 1% and 0.1% for 40 and 100-job instances, respectively, we cannot find all optimal schedules with this approach, the reason for that is, as we discussed in Section 3.3, TR is vulnerable to the Property 3.12 which is not necessarily valid for the TWT problem. Thus, even though we find the optimal schedule for the preemptive jobs in a nested schedule in order to make it non-preemptive, this resulting schedule is not necessarily optimal for TWT because the non-preemptive parts of the nested schedule are obtained from TR.

Figure 4.6 shows the minimum, average, and maximum CPU times for solving IP1 with respect to the number of jobs in the maximal parenthesis. Obviously solving an IP model becomes computationally expensive with respect to the average CPU times when the size of the problem exceeds 20 jobs.

4.4 Common Due Date

In order to evaluate the performance of the proposed heuristic, we create and solve 125 instances for each problem size $n = \{40, 50, 100\}$. The data set is generated with

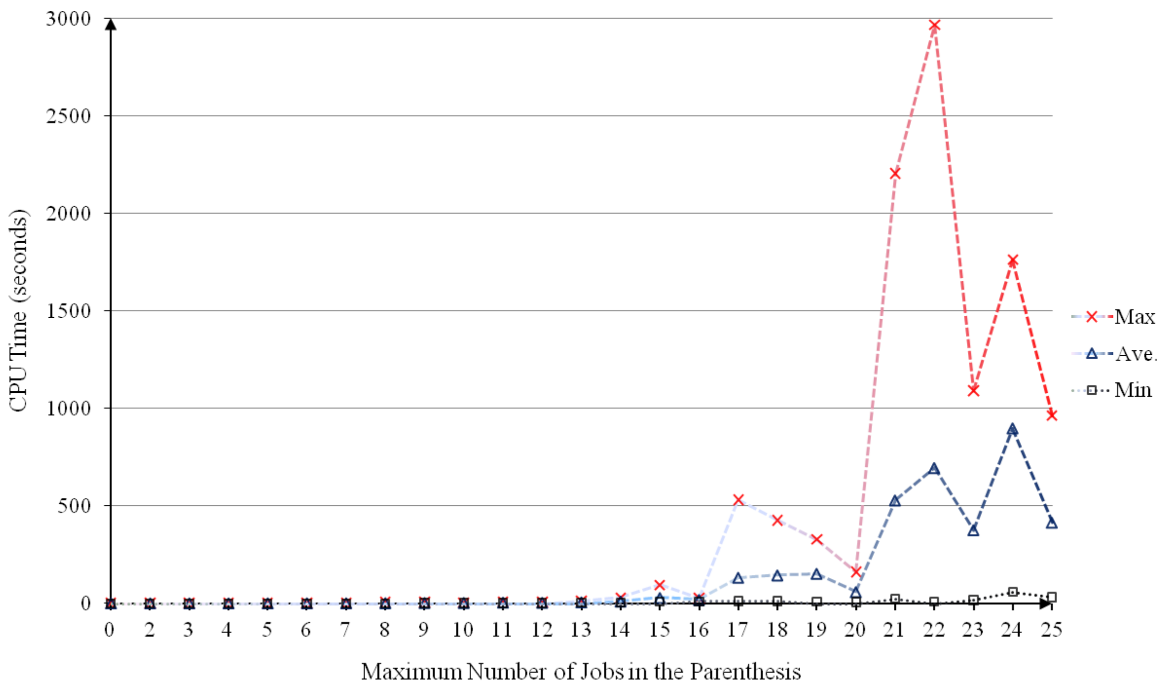


Figure 4.6: CPU times for solving the IP model

the same manner of the OR-Library, except a common due date is assigned to all jobs in an instance, and the data parameters in Table 4.1 are used. The optimal solutions of the CDD problem are obtained by using the DP algorithm[3] proposed by Lawler and Moore [39].

4.4.1 Summary of Results

For the common due date problem, we investigate the effect of the number of jobs, and tardiness and due date range factors on the number of optimal solutions, and on the average and worst case optimality gaps.

Effects of Number of Jobs

In Table 4.16, we present the effect of number of jobs. The number of times in which the solution of the proposed heuristic matches the optimal solution is indicated under # Opt. ("Number of Optimal") in column 2. In column 3 and 4, the average and worst case optimality gaps are presented, respectively. The average optimality gap for all 375 instances is less than half a percent. Although maximum gaps are between

[3]This algorithm is implemented in `MATLAB`.

2.5 and 5%; only 28 instances have gaps larger than 1% and 7 of those are higher than 2%.

n	# Opt.	Percentage gap (%)	
40	31	0.43	(4.85)
50	32	0.29	(2.57)
100	23	0.16	(3.01)

Table 4.16: Effect of n: number of optimal solution, and average and (maximum) gaps of the heuristic

Note that the optimality gaps decrease as the number of jobs increases. Even though the worst case gap of 100-job instances is larger than that of 50-job instances, there are only 3 100-job instances with the worst case optimality gap larger than 1% but for 50-job instances this number is 8. When the number of jobs insreases, the number of optimal solutions found by the heuristic decreases, which is common in the heuristics domain. The distribution of # Opt. may be seen in Table 4.18.

Effects of Tardiness and Range of Due Date Factors

The worst cases of the optimality gaps come from the instance with $TF = 0.2$ and $RDD = \{0.2, 0.6\}$, this may be seen in the Table 4.17. The reason for this effect is that, as the TF decreases, less jobs be tardy, and the objective function values of these instances are relatively smaller than the others, so that even one misplaced job may cause relatively big optimality gap increase.

Observe that the gaps for $TF = 1$ are relatively smaller than those for the other values of TF, the same effect also may be seen in Table 4.18. The due dates of these instances are relatively small, recall that due dates are generated from the interval $[(1 - TF - RDD/2)P, (1 - TF + RDD/2)P]$. Since in the heuristic, we basically order the jobs non-increasingly with respect to their $\frac{w_j}{p_j}$ ratios, when the due date is small, the heuristic schedules more number of jobs in its optimal place. In the next table, each cell corresponds to 15 instances and represents the number of instances whose optimal solution is obtained by the heuristic. Also, these results are consistent with the reasoning above.

n	TF	RDD				
		0.2	0.4	0.6	0.8	1
40	0.2	**2.14**(4.85)	**0.59**(2.26)	**0.76**(2.79)	**0.37**(1.43)	**0.35**(1.15)
	0.4	**0.50**(1.02)	**0.24**(0.83)	**0.17**(0.65)	**0.70**(1.79)	**0.51**(1.53)
	0.6	**0.55**(0.88)	**0.65**(1.20)	**0.58**(1.82)	**0.49**(1.60)	**0.46**(1.14)
	0.8	**0.15**(0.31)	**0.20**(0.51)	**0.40**(0.84)	**0.23**(0.56)	**0.42**(1.53)
	1	**0.01**(0.05)	**0.05**(0.12)	**0.16**(0.25)	**0.03**(0.10)	**0.10**(0.18)
50	0.2	**0.70**(2.16)	**0.44**(0.49)	**0.85**(2.57)	**0.11**(0.36)	**0.08**(0.24)
	0.4	**0.53**(0.94)	**0.41**(0.71)	**0.41**(0.96)	**0.43**(0.65)	**0.39**(1.38)
	0.6	**0.23**(0.52)	**0.23**(0.38)	**0.32**(0.69)	**0.45**(1.25)	**0.26**(0.49)
	0.8	**0.13**(0.27)	**0.26**(0.34)	**0.12**(0.32)	**0.34**(0.98)	**0.31**(1.14)
	1	**0.01**(0.03)	**0**(0)	**0.03**(0.15)	**0.02**(0.09)	**0.17**(0.45)
100	0.2	**0.58**(1.03)	**0.20**(0.57)	**0.85**(3.01)	**0.09**(0.22)	**0.11**(0.35)
	0.4	**0.23**(0.36)	**0.24**(0.35)	**0.14**(0.17)	**0.22**(0.48)	**0.36**(0.67)
	0.6	**0.07**(0.09)	**0.07**(0.11)	**0.10**(0.19)	**0.08**(0.26)	**0.41**(1.61)
	0.8	**0.03**(0.05)	**0.02**(0.03)	**0.07**(0.16)	**0.05**(0.17)	**0.04**(0.10)
	1	**0.01**(0.02)	**0.01**(0.03)	**0.02**(0.04)	**0.02**(0.10)	**0.04**(0.15)

Table 4.17: Effect of TF and RDD: average and (maximum) percentage gaps (%) of the heuristic.

TF	RDD				
	0.2	0.4	0.6	0.8	1
0.2	2	3	6	7	8
0.4	2	1	2	1	1
0.6	1	-	-	-	-
0.8	1	1	-	3	5
1	10	10	5	10	7

Table 4.18: Effect of TF and RDD: number of optimal solutions.

CHAPTER 5

Conclusion and Future Work

In this thesis, we proposed a simple, fast, and effective heuristic method to solve the strongly $\mathcal{NP}$-hard single-machine total weighted tardiness problem. Even though the transportation problem has been used for lower bounding before, as Bülbül et al. [11] said in 2007, "we investigated a relatively unexplored path" by studying the structure of the preemptions in the schedule obtained by the relaxation. We constructed a set of cost coefficients and proved that with these cost coefficients, the preemptive relaxation of the total weighted tardiness problem has a nested optimal schedule according to the Definition 3.3. Also, we showed that with this set of cost coefficients, the cost matrix of the relaxation of the common due date problem is a Monge matrix.

We demonstrated that the proposed solution approach yields excellent results in the computational experiments both in terms of the optimality gaps and CPU times.

We also note that with our cost coefficients the cost matrix of the transportation problem has a special structure. We hope to explore this structure in depth to find some desirable properties that may be used to develop a specialized algorithm for solving the transportation problem faster. A possible extension of our research could be to develop an algorithm algorithm that exploits the nested structure. Also, the proposed solution approach may used in meta-heuristics for generating the solution pool by modifying the behavior of the non-preemptive heuristic.

Bibliography

[1] Abdul-Razaq, T. and Potts, C. (1988). Dynamic programming state-space relaxation for single-machine scheduling. *J. of the Operational Research Society*, 39(2):141–152.

[2] Abdul-Razaq, T. S., Potts, C. N., and Van Wassenhove, L. N. (1990). A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Appl. Math.*, 26(2-3):235–253.

[3] Akturk, M. S. and Yildirim, M. B. (1998). A new dominance rule for the total weighted tardiness problem. *Computers & Operations Research*, 25(4):265–278.

[4] Angel, E. and Bampis, E. (2005). A multi-start dynasearch algorithm for the time dependent single-machine total weighted tardiness scheduling problem. *European J. of Operational Research*, 162(1):281–289.

[5] Azizoglu, M. and Kondakci, S.; Kirca, O. (1991). Bicriteria scheduling problem involving total tardiness and total earliness penalties. *International Journal of Production Economics*, 23(1–3):17–24.

[6] Baker, K. R. and Schrage, L. E. (1978). Finding an optimal sequence by dynamic programming: An extension to precedence-related tasks. *Operations Research*, 26(1):111–120.

[7] Besten, M. D., Stutzle, T., and Dorigo, M. (2001). Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In *In Proceedings of EvoWorkshops, LNCS*, pages 441–452. Springer.

[8] Bigras, L., Gamache, M., and Savard, G. (2008). Time-indexed formulations and the total weighted tardiness problem. *INFORMS J. on Computing*, 20(1):133–142.

[9] Bilge, U., Kurtulan, M., and Kirac, F. (2007). A tabu search algorithm for the single machine total weighted tardiness problem. *European J. of Operational Research*, 176(3):1423–1435.

[10] Bozejko, W., Grabowski, J., and Wodecki, M. (2006). Block approach–tabu search algorithm for single machine total weighted tardiness problem. *Computers & Industrial Engineering*, 50(1-2):1–14.

[11] Bülbül, K., Kaminsky, P., and Yano, C. (2007). Preemption in single machine earliness/tardiness scheduling. *J. of Scheduling*, 10(4-5):271–292.

[12] Burkard, R., Dell'Amico, M., and Martello, S. (2009). *Assignment Problems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[13] Burkard, R. E., Deineko, V. G., van Dal, R., van der Veen, J. A. A., and Woeginger, G. J. (1998). Well-solvable special cases of the traveling salesman problem: A survey. *SIAM Review*, 40(3):496–546.

[14] Chambers, R. J., Carraway, R. L., Lowe, T. J., and Morin, T. L. (1991). Dominance and decomposition heuristics for single machine scheduling. *Operations Research*, 39(4):639–647.

[15] Congram, R. K., Potts, C. N., and van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS J. on Computing*, 14(1):52–67.

[16] Crauwels, H. A. J., Potts, C. N., and Wassenhove, L. N. V. (1998). Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS J. on Computing*, 10(3):341–350.

[17] Della Croce, F., Tadei, R., Baracco, P., and Grosso, A. (1998). A new decomposition approach for the single machine total tardiness scheduling problem. *J. of the Operational Research Society*, 49(10):1101–1106.

[18] Elmaghraby, S. (1968). The one machine scheduling problem with delay costs. *Journal of Industrial Engineering*, 19:105–108.

[19] Emmons, H. (1969). One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, 17(4):701–715.

[20] Ferrolho, A. and Crisostomo, M. (2007). Single machine total weighted tardiness problem with genetic algorithms. *Computer Systems and Applications, ACS/IEEE International Conference on*, pages 1–8.

[21] Fisher, M. L. (1976). A dual algorithm for the one-machine scheduling problem. *Mathematical Programming*, 11:229–251.

[22] Gelders, L. and Kleindorfer, P. R. (1974). Coordinating aggregate and detailed scheduling decisions in the one-machine job shop: Part I. Theory. *Operations Research*, 22(1):46–60.

[23] Gelders, L. and Kleindorfer, P. R. (1975). Coordinating aggregate and detailed scheduling in the one-machine job shop: II−Computation and Structure. *Operations Research*, 23(2):312–324.

[24] Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E. J. and Korte, B., editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier.

[25] Grosso, A., Croce, F. D., and Tadei, R. (2004). An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, 32(1):68–72.

[26] Held, M. and Karp, R. M. (1962). A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210.

[27] Holsenback, J. E., Russell, R. M., Markland, R. E., and Philipoom, P. R. (1999). An improved heuristic for the single-machine, weighted-tardiness problem. *Omega*, 27(4):485 – 495.

[28] Huegler, P. A. and Vasko, F. J. (1997). A performance comparison of heuristics for the total weighted tardiness problem. *Computers & Industrial Engineering*, 32(4):753–767.

[29] Ibaraki, T. and Nakamura, Y. (1994). A dynamic programming method for single machine scheduling. *European J. of Operational Research*, 76(1):72–82.

[30] Jouglet, A., Savourey, D., Carlier, J., and Baptiste, P. (2008). Dominance-based heuristics for one-machine total cost scheduling problems. *European J. of Operational Research*, 184(3):879–899.

[31] Kanet, J. J. (2007). New precedence theorems for one-machine weighted tardiness. *Mathematics of Operations Research*, 32(3):579–588.

[32] Kanet, J. J. and Li, X. (2003). On adjacent job precedence for $1 \mid\mid w_j T_j$. Working paper, University of Dayton, Dayton, OH.

[33] Kanet, J. J. and Li, X. (2004). A weighted modified due date rule for sequencing to minimize weighted tardiness. *J. of Scheduling*, 7(4):261–276.

[34] Kondakci, S., mer Kirca, and Azizoglu, M. (1994). An efficient algorithm for the single machine tardiness problem. *International Journal of Production Economics*, 36(2):213–219.

[35] Lawler, E. (1977a). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 1:331–342.

[36] Lawler, E. (1979). Effcient implementation of dynamic programming algorithms for sequencing problems. Technical Report BW 106, Mathematisch Centrum, Amsterdam, The Netherlands.

[37] Lawler, E. L. (1964). On scheduling problems with deferral costs. *Management Science*, 11(2):280–288.

[38] Lawler, E. L. (1977b). A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In P.L. Hammer, E.L. Johnson, B. K. and Nemhauser, G., editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 331–342. Elsevier.

[39] Lawler, E. L. and Moore, J. M. (1969). A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84.

[40] Lenstra, J., Rinnooy Kan, A., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

[41] McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12.

[42] Morton, T. E. and Rachamadugu, R. M. V. (1982). Myopic heuristics for the single machine weighted tardiness problem. Technical Report 83-9, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA.

[43] Pan, Y. and Shi, L. (2007). On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming*, 110(3):543–559.

[44] Pessoa, A., Uchoa, E., Poggi de Arago, M., and Rodrigues, R. (2008). Algorithms over arc-time indexed formulations for single and parallel machine scheduling problems. Technical Report RPEP Vol. 8 no. 8, Universidade Federal Fluminense, Engenharia de Producao, Niteroi, Brazil.

[45] Phillips, C., Stein, C., and Wein, J. (1998). Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223.

[46] Picard, J.-C. and Queyranne, M. (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, 26(1):86–110.

[47] Potts, C. and van Wassenhove, L. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters*, 1(5):177–181.

[48] Potts, C. N. and Van Wassenhove, L. N. (1991). Single machine tardiness sequencing heuristics. *IIE Transactions*, 23(4):346–354.

[49] Potts, C. N. and Wassenhove, L. N. v. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33(2):363–377.

[50] Rachamadugu, R. M. V. (1987). Technical note–A note on the weighted tardiness problem. *Operations Research*, 35(3):450–452.

[51] Rinnooy Kan, A. H. G., Lageweg, B. J., and Lenstra, J. K. (1975). Minimizing total costs in one-machine scheduling. *Operations Research*, 23(5):908–927.

[52] Schild, A. and Fredman, I. J. (1961). On scheduling tasks with associated linear loss functions. *Management Science*, 7(3):280–285.

[53] Schrage, L. and Baker, K. R. (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, 26(3):444–449.

[54] Sen, T., Sulek, J. M., and Dileepan, P. (2003). Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *International Journal of Production Economics*, 83(1):1 – 12.

[55] Shwimer, J. (1972). On the n-job, one-machine, sequence-independent scheduling problem with tardiness penalties: A branch-bound solution. *Management Science*, 18(6):B301–B313.

[56] Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66.

[57] Sourd, F. and Kedad-Sidhoum, S. (2003). The one-machine problem with earliness and tardiness penalties. *J. of Scheduling*, 6(6):533–549.

[58] Srinivasan, V. (1971). A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. *Naval Research Logistics Quarterly*, 18(3):317–327.

[59] Szwarc, W., Della Croce, F., and Grosso, A. (1999). Solution of the single machine total tardiness problem. *J. of Scheduling*, 2(2):55–71.

[60] Szwarc, W., Grosso, A., and Della Croce, F. (2001). Algorithmic paradoxes of the single-machine total tardiness problem. *J. of Scheduling*, 2(4):93–104.

[61] Tanaka, S., Fujikuma, S., and Araki, M. (2009). An exact algorithm for single-machine scheduling without machine idle time. *J. of Scheduling*, 12(6):575–593.

[62] Tansel, B., Kara, B., and Sabuncuoglu, I. (2001). An efficient algorithm for the single machine total tardiness problem. *IIE Transactions*, 33(8):661–674.

[63] Tasgetiren, M. F., Liang, Y.-C., Sevkli, M., and Gencyilmaz, G. (1999). Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem. *International J. of Production Research*, 44(22):4737–4754.

[64] Volgenant, A. and Teerhuis, E. (1999). Improved heuristics for the n-job single-machine weighted tardiness problem. *Comp. & Operations Research*, 26(1):35–44.

[65] Wang, X. and Tang, L. (2009). A population-based variable neighborhood search for the single machine total weighted tardiness problem. *Comp. & Operations Research*, 36(6):2105–2110.

[66] Wilkerson, L. J. and Irwin, J. D. (1971). An improved method for scheduling independent tasks. *AIIE Transactions*, 3(3):239–245.