# Time-indexed Mathematical Models for Order Acceptance and Scheduling Problems

by

Saeed Saffari

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Industrial Engineering

Koç University

June, 2015

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a M.Sc. thesis by

Saeed Saffari

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Prof. Dr. Ceyda Oğuz (Advisor)

Assoc. Prof. Dr. F. Sibel Salman

Prof. Dr. Ümit Bilge

Date: _____

*Dedicated to my parents*

# ABSTRACT

Scheduling has been an active area of research for decades. A substantial amount of work has been done to define, classify, and solve scheduling problems. Most of these problems are computationally difficult to solve, and incorporating other decisions into them increases the complexity of these problems. This thesis focuses on order acceptance and scheduling (OAS) problems, and proposes time-indexed mixed integer and linear programming (MILP) models for the first time.

An OAS problem can be defined with parameters of processing time, release date, due date, and deadline for each order. There are also a maximum revenue that will be brought by each order and an importance weight of each order. Furthermore, there could be a setup time between orders if they are processed consecutively.

In the thesis, both the general OAS problem (OAS 2) that will be defined with all above parameters, and a special case (OAS 1) that will exclude the release dates and setup times are considered.

After developing a time-indexed MILP for both OAS 1 and OAS 2, their efficiency and effectiveness were tested computationally. To enhance both the efficiency and the effectiveness of OAS 1, three dominance properties were suggested, and it was observed that while the optimality gap was decreased, the computational time was also reduced considerably so that large instances can be solved.

Since OAS 2 is more challenging than OAS 1, the time-indexed MILP developed could not solve problem instances to optimality in a reasonable time. Hence, different methods were proposed to find near optimal lower bounds and upper bounds for the problem. To obtain good upper bounds, a Lagrangian Relaxation method as well as a linear programming (LP) relaxation method with three new valid inequalities were proposed, and it is shown that Lagrangian Relaxation finds only loose upper bounds,

but LP relaxation with valid inequalities improves almost all upper bounds for all sizes of instances. To obtain lower bounds, a simple heuristic method and a revised MILP formulation were presented, and then, it is shown that they are able to solve only small instances to optimality.

# ÖZETÇE

Uzun yıllardır üzerinde çalışılan bir konu olarak, çizelgeleme problemlerini tanımlayan, sınıflandıran ve çözen önemli sayıda çalışma yapılmıştır. Çizelgeleme problemlerinin büyük bir çoğunluğu, hesaplama karmaşıklığı yüksek olduğundan, zordur ve farklı problemlerle birleştirmek hesaplama karmaşıklığını daha çok arttırmaktadır. Bu tez, çizelgeleme problemini siparişleri kabul/red kararıyla birleştiren Sipariş Kabul ve Çizelgeleme Problemleri (SKÇ) üzerine odaklanmakta ve bu problem için ilk kez zaman endeksli Karışık Tam Sayılı Doğrusal Programlama (KTSDP) modelleri sunmaktadır.

Sipariş kabul ve çizelgeleme problemi işlem süresi, serbest bırakılma tarihi, teslim tarihi ve son teslim tarihi parametreleriyle tanımlanabilir. Ayrıca, her siparişin maksimum gelir ve önem ağırlığı bulunmaktadır. Ek olarak, ardışık şekilde işlenen siparişler arasında hazırlama süresi bulunabilir.

Bu tezde, yukarıdaki parametrelerle tanımlanan, genel sipariş kabul ve çizelgeleme problemi (SKÇ 2) ile serbest bırakılma tarihi ve hazırlama sürelerini dahil etmeyen özel hali (SKÇ 1) ele alınmıştır.

Zaman endeksli KTSDP modeli SKÇ 1 ve SKÇ 2 için geliştirildikten sonra, bu modellerin yeterliliği ve etkinliği sayısal olarak test edilmiştir. SKÇ 1 modelinin yeterliliği ve etkinliğini arttırmak için, üç baskınlık özelliği önerilmiştir. Bu baskınlık özellikleri sonucunda hem eniyilik aralığının hem de çözüm zamanının azaldığı gözlemlenmiştir. Bu sayede büyük test örnekli problemler çözülebilmiştir.

SKÇ 2, SKÇ1 e kıyasla daha zor bir model olduğu için, zaman endeksli KTSDP modeli, problemi makul zaman çerçevesinde eniyi sonucu verecek şekilde çözememiştir. Bu sebeple, eniyi çözüme alt ve üst sınır bulabilmek amacıyla farklı yöntemler sunulmuştur. İyi bir üst sınır elde etmek amacıyla Lagrangian Gevşetmesi yanı sıra, üç adet yeni geçerli eşitsizlik içeren Doğrusal Programlama Gevşetmesi önerilmiştir. Yapılan testler

sonucunda Lagrangian Gevşetmesi sadece gevşek üst sınırlar bulurken, geçerli eşitsizlik içeren Doğrusal Programlama Gevşetmesi yaklaşık olarak bütün üst sınırları tüm test örnekleri için geliştirmiştir. Alt sınır elde etmek için, basit bir sezgisel yöntem ve yeniden düzenlenmiş KTSDP formülü sunulmuş ve bu yöntemlerin sadece küçük test örnekli problemleri en iyi şekilde çözebildiği gösterilmiştir.

# ACKNOWLEDGMENTS

My sincere appreciations go to my supervisor, Prof. Ceyda Oğuz, whose vast knowledge guided me during the path of pursuing my M.Sc. I am thankful to her for providing me the chance to be a member of her group at Koç University, and I am thankful for her trust in me. Things could not evolve without her trust and support. And a special gratitude for teaching me how to be a successful person and leader both in academia and real life.

I also would like to thank to my thesis committee members, Dr. F. Sibel Salman and Dr. Ümit Bilge for their valuable advice.

Finally, I would like to express my deepest gratitude to my family, who has always believed in me and supported me throughout my life. I would not be able to accomplish this achievement without them.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# NOMENCLATURE

| | | |
|---|---|---|
| **OAS** | : | Order Acceptance and Scheduling |
| **MILP** | : | Mixed Integer Linear Programming |
| **SA** | : | Simulated Annealing |
| **GA** | : | Genetic Algorithm |
| **TS** | : | Tabu Search |
| **CPU** | : | Central Processing Unit |
| **IGAL** | : | Improved Genetic Algorithm with Local search |
| **DCGA** | : | Diversity Controlling Genetic Algorithm |
| **DP** | : | Dominance Property |
| **VI** | : | Valid Inequality |
| **LR** | : | Lagrangian Relaxation |
| **LB** | : | Lower Bound |
| **UB** | : | Upper Bound |
| **HLP** | : | Heuristic based on LP relaxation |
| **LPVI** | : | LP relaxation with Valid Inequalities |
| **API** | : | Average Percentage Improvement |

# Chapter 1

# INTRODUCTION

Available resources should be utilized in an efficient manner for effective performance in any system. This can be made possible only if an excellent scheduling system is in place. Scheduling can be defined as the allocation of resources over a period of time to perform a collection of tasks. Any scheduling problem involves devising a plan to carry out a number of activities that require resources, which are limited, along with various constraints to optimize one or more objectives. Developing a schedule can be the solution to many problems including job scheduling in a production facility, lecture and teacher scheduling at a university, vehicle scheduling in transportation networks, or nurse scheduling at a hospital.

In modern manufacturing, scheduling problems have become an interesting topic of research. Satisfying the daily demand of product with top quality and on-time delivery leads manufacturing industries to invest money and time in solving scheduling problems. The objective of the scheduling problem may vary according to industry needs. Potential objectives include minimizing the total weighted tardiness, total completion time or maximizing total revenue. Maximizing total revenue is the objective for this research.

In a firm that strives to align its functions so that profit is maximized, the coordination of capacity with demand may require some orders to be turned down. Which orders are accepted and which are rejected may depend on the strategic direction of the firm, the current status of capacity already allocated, and the profitability of the order in question. In particular, there is a trade-off between the revenue brought in by a particular order, and all of its associated costs of processing, which may include

delay costs for other orders, as well as any penalties incurred if this order is delivered after its agreed-upon due date.

The problem of order acceptance and scheduling (OAS) is defined as the joint decision of which orders to accept for processing and how to schedule them. The decision-maker is faced with a collection or stream of independent orders $O$ ($|O|= n$) whose combined processing requirements would exceed available capacity, and has the option of rejecting some of those orders. The capacity available for processing the accepted orders is limited and each order is characterized by a known processing time $p_i$, release date $r_i$, due date $d_i$, revenue $e_i$ and a weight $w_i$ representing a penalty per unit-time delay beyond the due date, the deadline $\bar{d}_i$ after which order $i$ cannot be produced ($\bar{d}_i \geq d_i$), the sequence dependent setup time $s_{ij}$ which is required when order $i$ is scheduled before order $j$. The tardiness of order $i$ is defined as the difference between the completion time of order $i$ and its due date, if the difference is greater than zero, and the tardiness of order $i$ is zero, if the its processing is finished before its due date. As our aim is to maximize the profit gained from the accepted orders, which is defined as the total revenues minus the total weighted tardiness, our objective function is then defined by max $\sum_{i \in S} R_i = \sum_{i \in S}(e_i - T_i \times w_i)$ where S is the set of accepted orders, $w_i$ is the weight for the tardiness of order $i$ and $R_i$ is the profit gained from the accepted order $i$. The profit $R_i$ reflects the fact that if an order $i$ is tardy, the customer will receive a discount. However we note that the early delivery is not penalized as the manufacturer is assumed to have unlimited capacity for finished product storage.

The scope of this study is restricted to a single machine environment. Single machine environment is one of the most widely used environments in scheduling literature. Moreover, in the problem, sequence dependent setup times among the orders to be processed are present. Setups generalize the work needed on the machine or on the order before the order becomes available for the main processing. Inspecting materials, cleaning, dye changes, tool changes are some examples of setups. Although setup times or costs are negligible or can be considered within the processing time

for make-to-stock products, make-to-order systems require separate consideration of setup times/costs. Inclusion of sequence dependent setup times increases the hardness of the underlying scheduling problem.

In this research, we focus on finding the acceptance, allocations and schedules of the orders for a make-to-order system within a pre-specified period of time while considering maximum possible revenue as our objective. We carry out the research with two different types of order acceptance and scheduling problems. These problems can be stated as $1 \mid d_j, e_j, w_j \mid \sum R_j$ problem (Chapter 3) and $1 \mid r_j, d_j, e_j, w_j, s_{ij} \mid \sum R_j$ problem (Chapter 4) by using the three-field notation of Graham et al. [61]. It consists of three fields: $\alpha$, $\beta$ and $\gamma$. Each field may be a comma separated list of words. The $\alpha$ field describes the machine environment, $\beta$ the job characteristics, and $\gamma$ the objective function.

Several techniques can be used to develop a solution for scheduling models. Some of them include mathematical programming, analytical methods, graph coloring approaches, artificial intelligence techniques, heuristics and metaheuristics like simulated annealing, tabu search, genetic algorithms and ant colony optimization. Most of these techniques are meant for a specific type of problem and they need to be adapted to different problems. Some techniques are more general (e.g., mathematical programming) but if there is a larger dataset, one would have to develop clever heuristics to deal with such a problem. One of the most popular types of mathematical programming is integer programming. Algorithmic advances in recent years have greatly increased the use of integer programming as a practical technique for solving scheduling problems. In this research, we use time-indexed mixed integer linear programming (MILP) to model our problems. Furthermore, we develop some heuristic methods based on the solution of our MILP models or the relaxed version MILP models. Moreover, we find dominance properties and valid inequalities to ease the processes of finding solutions.

### *1.1 Outline of the Thesis*

In Chapter 2, we give an extensive literature review for the OAS problem. In this review, all previous studies are divided to three main sections: 1. Time-indexed formulations, 2. Valid inequalities, dominance properties, and heuristic ideas, and 3. Order Acceptance and Scheduling (OAS) problems. In Chapter 3, first, we define the OAS problem $1 \,|\, d_j, e_j, w_j \,|\, \sum R_j$ in detail, and give the MILP model as well as dominance properties. Next, we discuss the computational experiments. In Chapter 4, we consider general version of order acceptance and scheduling problem, that is $1 \,|\, r_j, d_j, e_j, w_j, s_{ij} \,|\, \sum R_j$. Then, we propose our MILP model as well as our methods for finding upper and lower bounds. Afterwards, we show the computational studies for this problem using our methods. Finally in Chapter 5, we mention future study areas on the problem.

### *1.2 Contributions to the Literature*

In this thesis, we consider two different versions of order acceptance and scheduling problem for which the literature studies are highly potent for mathematical model formulations.

Regarding the first problem, discussed in Chapter 3, Slotnick and Morton [40] developed some heuristics which are able to solve small instances. Their heuristic algorithm cannot solve all instances to optimality, and they have an optimality gap of about 5 percent most of the times. Using their data, MILP1, proposed in this thesis (Section 3.2) can solve for all instances to optimality in less than one second. Rom and Slotnick [42] also constructed two heuristic algorithms based on genetic algorithm to solve larger instances. We solve their instances by our MILP, explained in Section 3.2, with and without our three dominance properties, discussed in Section 3.3. We show that we could improve all the optimality gaps.

Regarding the second problem, discussed in Chapter 4, Oguz et al. [43] gave the first MILP formulation and two heuristic methods to solve the problem with a reasonable

optimality gap and within a reasonable time. Cesaret et al. [45] developed a Tabu Search algorithm which could be solved in a very small CPU time, and improved the optimality gaps. Then, an artificial bee colony based algorithm is proposed by Lin and Ying [46] which could improve the optimality gaps for some instances compared to previous Tabu Search. Chen et al. [58] then proposed an improved genetic algorithm with local search (IGAL) for this problem. Finally, Chen et al. [47] proposed a Diversity Controlling Genetic Algorithm (DCGA), and they could obtain best lower bounds with less optimality gaps compared to all previous methods. Our contribution for this problem is related to our work for finding better upper bounds. In Section 4.3, we discuss our relaxation methods as well as our new valid inequalities, and in Section 4.6.3, we show our improvements in finding better and more near optimal upper bounds.

In summary, the first time-indexed mathematical model formulations are given in this thesis. Several dominance properties are developed to improve the optimality gap of these models for larger instances. For the general OAS problem, better upper bounds are also obtained with the methodology proposed in this thesis.

# Chapter 2

# LITERATURE REVIEW

The literature on mathematical formulations for scheduling problems, which are related to the problems considered in this thesis, is extensive and can be divided into three parts. Some researchers have proposed time-indexed formulations in their model and investigate different kinds of objectives; some others have looked to maximize total revenue either in flow shop environment or in a single machine environment, whereas a few of them have tried to find useful valid inequalities, dominance properties, and heuristic ideas which may be applicable to scheduling problems and reduce running times.

## *2.1 Time-indexed formulations*

To have a better comparison, we classify the papers of machine scheduling problems which used time indexed formulation according to the criterion (objective function) they considered, and then we investigate them one by one.

### *2.1.1 Total (weighted) tardiness*

Time-indexed mathematical models for scheduling problems have started in 1990. Dyer and Wolsey [1] gave a motivation to people to work with time-indexed models in scheduling problems. At first, they used the completion time as their main variable, and they assumed that just release date is the restriction. The objective function was minimization of total weighted tardiness. They developed their model under this assumption and solved using some relaxations, and finally they presented a time-indexed model. Consequently, they showed that with the relaxation of time-indexed models, they could obtain better lower bounds compared to other mathematical

models.

Bigras et al. [2] introduced several integer programming formulations for single machine scheduling problems with sequence dependent setups. Considering instances of $1|s_{ij}|\sum C_j$ and $1|s_{ij}|\sum T_j$, they proposed a branch-and-bound algorithm in which they used linear programming (LP) relaxation bounds.

Bansal et al. [3] used some O(1)-speed O(1)-approximation algorithms for non-preemptive scheduling problems. An s-speed c-approximation algorithm A has a property that the value of the objective function of the schedule that A produces with processors of speed $s \geq 1$ is at most c times the optimal value of the objective function for speed 1 processors [4]. An LP based online algorithm is an O(1)-speed O(1)-approximation algorithm. Bansal et al. [3] used these algorithms for $1|r_j|\sum w_j F_j$ (weighted flow time), $1|r_j|\sum T_j$ (total tardiness), $1|r_j|\sum \bar{U}_j$ (throughput maximization), and $1|r_j|\sum w_j T_j$ (weighted tardiness) problems, and they proposed an integer programming formulation whose relaxation is sufficiently close to the integer optimum, and which can be transformed to a schedule on a faster machine.

Using a Lagrangian decomposition, Babu et al. [5] first presented a lower bound for solving the $1||\sum w_j T_j$ scheduling problem. They then proposed a branch-and-bound algorithm, based on a the binary time-indexed formulation presented by Van den Akker et al. [6].

Pan and Shi [8] considered a single machine min-sum scheduling problem and presented new results to find the strong bound from the family of lower bounds provided by transportation problem relaxations of the single machine scheduling problem. They showed the equivalence of that strong bound and the bound provided by the LP relaxation of the time-indexed integer programming formulation. They also proposed a new scheme to approximate the time-indexed bound by branch-and-bound algorithm.

Bigras et al. [9] improved the mathematical model proposed by Van den Akker et al. [6] for the total weighted tardiness problem. They used a time-decomposition approach in order to reduce the solution time of the LP relaxation of time-indexed

formulations for the $1||\sum w_j T_j$ problems while finding worse lower bounds. Moreover, they applied this strategy in a branch-and-bound algorithm.

With the objective of minimizing the total job completion cost, Tanaka et al. [10] proposed an exact algorithm for the general single machine scheduling problem by improving the previous algorithm based on the a successive sublimation dynamic programming (SSDP) method [11] to reduce both the memory usage and computational efforts. It was also shown that their algorithm outperformed the existing specialized algorithms for the single machine total weighted tardiness problem and the single machine total weighted earliness-tardiness problem without machine idle time.

Altunc and Keha [12] used an interval-indexed formulation for the single machine total weighted tardiness problem. At first, they reduced the well-known time-indexed formulation by using a set of feasible solutions which was guaranteed to give a solution that was at least as good as the solutions that they started with. They then introduced the interval-indexed formulation which could be solved much faster than the reduced time-indexed formulation. Interval-indexed formulation was also guaranteed to give a solution at least as good as the solution given by the reduced time-indexed formulation when the intervals are created by using the same feasible schedules that are used to reduce the time-indexed formulation. They also introduced two post-processing algorithms based on optimizing smaller subsequences. Finally, their computational experiments showed that their heuristic gives promising results.

### *2.1.2 Regular objective function ($\sum F_j(C_j)$)*

Baptiste and Sadykov [13] considered a function of completion times as their objective function which can be total (weighted) completion time, total (weighted) tardiness, earliness and tardiness, etc. They introduced a Mixed Integer Program (MIP) based on time-interval decomposition closely related to the well-known time-indexed MIP formulation. They compared both the time-indexed and interval-indexed formulations for different examples. Their numerical experiments showed that the choice of the formulation should be made based on the properties of the given instance to solve.

They showed that the larger the processing times of jobs are the more likely that the interval-indexed formulation provides better results. In the conclusion, they believed that the direct application of those formulations was useful for solving medium size instances.

### *2.1.3 Total earliness-tardiness*

Sourd and Kedad-Sidhoum [14], presented a new branch-and-bound algorithm based on the combination of a Lagrangian relaxation of resource constraints as well as new dominance rules for the one-machine earliness-tardiness problem. Their algorithm outperformed previous ones because the lower bound was better so that fewer nodes were required in the search tree. Moreover, their new dominance rules could help in reducing the size of the search tree. Their experimental results suggested that the algorithm could be improved to solve special cases where the range between due dates is narrow and where earliness penalties are significantly smaller than tardiness penalties.

Bulbul et al. [15] considered a single machine earliness–tardiness scheduling problem with general weights, release times and due dates. They developed a tight lower bound for $1|r_j|\sum \epsilon_j E_j + \pi_j T_j$ (minimizing weighted total earlinesses and tardinesses) based on a preemptive solution that was easy to compute by solving a time-indexed problem. Then, they presented a heuristic that used the information from the optimal solution of the preemptive relaxation to create a feasible solution for the general problem.

Baptiste and Sadykov [16] worked on airborne radars scheduling and introduced a generalization for the classic earliness-tardiness single machine scheduling problem. After giving the general time-indexed formulation for this problem, they changed it according to radar scheduling problem, since in radar scheduling problem the radar is a single machine and a radar task is a job to be scheduled on the machine. A job consists of a chain of operations with identical processing times. The operations of the same job should be ideally scheduled with a given frequency, and any deviation from this frequency is penalized using a V-shape function. Their objective was then to

minimize the total penalty. They used a branch-and-price algorithm, and a heuristic was executed at each node to obtain an upper bound. Then they used LP relaxations in order to measure the quality of their solutions.

### *2.1.4 Total (weighted) flow time*

Phillips et al. [17] considered a class of scheduling problems which includes a variety of problems that are exceedingly difficult to approximate. They gave a pseudo–polynomial-time algorithm that schedules a constant fraction of $n$ jobs within a constant factor times the optimal value (OPT). Then, for analyzing the performance of algorithm, they considered some single machine scheduling problems which they used for approximation algorithms they built earlier [18, 19], and they tried to minimize average weighted flow time. They showed that for the instances that were hardest empirically for traditional approximation algorithms, a large fraction of the set of jobs can be scheduled using these techniques, with good performance. Actually, Savelsbergh et al. [18] applied these techniques from single machine problems to obtain good solutions for actual instances of hard real problems. Phillips et al. [17] gave the first constant-factor approximation algorithms for scheduling jobs with release dates and due dates on a single machine so as to maximize the (weighted) number of on-time jobs.

### *2.1.5 Total (weighted) completion time*

Considering a time-indexed formulation of nonpreemptive single-machine scheduling problems, Van den Akker et al. [20] gave complete characterizations of all facet-inducing inequalities with integral coefficients for the convex hull of the set of feasible partial schedules, i.e., schedules in which not all jobs have to be started. Furthermore, they identified conditions under which those facet-inducing inequalities are also facet-inducing for the original polytope, which is the convex hull of the set of feasible complete schedules, i.e., schedules in which all jobs have to be started. To obtain insight on the effectiveness of those classes of facet-inducing inequalities, they developed

a branch-and-cut algorithm based on them. They evaluated its performance on the strongly NP-hard single machine scheduling problem of minimizing the weighted sum of the job completion times subject to release dates.

Van den Akker et al. [6] considered a simple time-indexed formulation and minimized the total cost related to starting time of each job without considering any parameters except processing time. They used Dantzig-Wolfe decomposition to decrease the number of constraints, and they obtained a good lower bound by LP relaxation of this reformulation. Moreover, they used column generation and compared the solutions with two other techniques for 20-jobs instances. According to the results of [21, 22], they knew that the bound provided by LP relaxation can be improved significantly. Therefore, they applied column generation algorithm in which LP relaxations are solved after cuts. Considering characterization of all facet-inducing inequalities which they presented already in Van den Akker et al. [20], they could combine column and cut generation and solve some special benchmark instances of 20-job and 30-job in acceptable CPU times and reach good lower bounds.

Savelsbergh et al. [18] studies the design and analysis of approximation algorithms for $1|r_j|\sum w_j c_j$ problems which had been constructed in [1, 21, 23], and developed some techniques with improved computational efficiency. Then, they studied both the quality of lower bounds given for this problem by two different linear programming relaxations, and the quality of upper bounds delivered by a number of approximation algorithms based on them. The best algorithms, on almost all instances, came within a few percent of the optimal average weighted completion time.

Hall et al. [23] was motivated by the results of Sousa and Wolsey [21] and Van den Akker et al. [22] papers. They used a model in which main variable indicates finishing period time of a certain job. They considered just release times and precedence constraints and tried to minimize total weighted completion times. Then they designed an approximation algorithm and called it schedule-by-$\alpha$-point with good worst-case performance guarantees. After that, they showed how they extend their model for the $m$-parallel identical machines problems. Finally, they presented two other techniques

considering presence of release dates, but no precedence constraint.

Goemans et al. [24] showed that the time-indexed formulation for $1|r_j|\sum w_j c_j$ introduced by Dyer and Wolsey [1] can be solved to optimality using the LP formulations. They also used a completion time relaxation and proved its equivalence to the time-indexed formulation. They also dealt with approximation algorithms derived from the LP relaxations. Then, by the inspiration of Hall et al. [23], they described two simple randomized approximation algorithms, which were guaranteed to deliver feasible schedules with expected objective function value within factors of 1.7451 and 1.6853, respectively, of the optimum.

Crauwels et al. [25] considered the problem of scheduling families of jobs on a single machine to minimize the total weighted completion time, where a set-up time is incurred whenever the machine switches from processing a job in one family to a job in another family. They could reach a new lower bound based on a Lagrangian relaxation of machine capacity constraints, and they proposed a heuristic method which is applied at the root node of the branch-and-bound search tree to find an upper bound UB on the minimum value of the total weighted completion time. They built three branch-and-bound algorithms: one algorithm with forward branching and objective splitting, other one with forward branching and multiplier adjustment, and the last one with binary branching and subgradient optimization. Finally, they compared them in aspects of average computation time, average number of nodes in the branch-and-bound search tree, and number of unsolved problems.

Avella et al. [26] worked on the $1|r_j|\sum w_j c_j$ problem and considered the time-indexed formulation and a new lagrangian heuristic, based on the observation that lagrangian relaxation of job constraints leads to a weighted stable set problem on an interval graph. They proposed a dynamic programming algorithm which can solve the relaxed problem in polynomial time. In their computational experience, they showed that instances up to 400 jobs and maximum processing time 50 (around 5,000,000 variables) can be solved in less than 40 minutes on a personal computer, yielding duality gaps never exceeding 3%. They also tested a set of hard instances, built to

produce bad performances where they yielded duality gaps less than 5%.

### *2.1.6 Total (weighted) number of tardy jobs*

Peridy et al. [27] presented an exact method for solving the $1|r_j|\sum w_j U_j$ problem. They used a time-indexed formulation to define a Lagrangian lower bound which was based on the relaxation of the multiplicity constraint that enforces jobs to be processed at most once on their time windows. In order to soften that relaxation, they employed a short-term memory that stores some of the last jobs processed to avoid repetitions inside that buffer. Two Lagrangian heuristics and some elimination rules were also derived. Their computational experiments showed that the use of those tools within a branch-and-bound method can solve to optimality problems with up to 100 jobs. Their results seemed to be competitive in comparison with previous approaches.

## *2.2 Valid inequalities, dominance properties, and heuristic ideas*

Sousa and Wolsey [21] tried to investigate the behavior of time-indexed formulations. They proposed three valid inequalities and implemented a cutting plane algorithm based on them. They tested the algorithm for small problems considering four different objective functions. Finally, by their limited computational experience, they concluded that the combination of a time-indexed formulation with the cutting plane approach can provide a useful tool in solving various relatively small machine sequencing problems. Their promising computational results stimulated Van den Akker et al. [28] to study those three classes of inequalities. They worked on facet-inducing inequalities and they could make these classes more comprehensive, while Sousa and Wolsey [21] had just worked on some particular examples.

Queyranne and Schulz [29] investigated some polyhedral approaches to non-preemptive machine scheduling problems which were studied in [21, 28]. Considering time-indexed formulations based on linear ordering, start time and completion time variables, on assignment and positional date variables, and on TSP variable, they showed important role of supermodular polyhedral and greedy algorithms and discussed

about separation algorithms for several classes of inequalities, and their potential applicability in generating cutting planes for the practical solution of such scheduling problems.

Van den Akker et al. [22] reported the complete characterizations of all facet-inducing inequalities based on [21, 28, 29]. To obtain insight in the effectiveness of those classes of facet-inducing inequalities, they developed a branch-and-cut algorithm based on them and evaluated its performance on the strongly NP-hard single machine scheduling problem of minimizing the weighted sum of the job completion times subject to release dates. Finally, they showed the strength of the presented branch-and-cut algorithms which can be applied successfully to a wide range of single-machine scheduling problems, but its weakness was its limitation to instances with a relatively small number of jobs and relatively small processing times, because otherwise the time to solve the linear programs became prohibitive.

Waterer et al. [30] focused on the facet-inducing inequalities for the convex hull of the set of feasible partial schedules which were defined by Sousa and Wolsey [21] already, and they used similarity between single machine scheduling problem and node packing problem, and showed that the characteristic vectors of that set of schedules and those of the set of feasible node packings in a graph whose structure is determined by the parameters of the scheduling problem are equivalent. In their result, they concluded that those facet-inducing inequalities which have integral coefficients and right hand side 1 or 2 are the maximal clique inequalities and the maximally and sequentially lifted 5-hole inequalities of the convex hull of the set of feasible node packings, respectively.

## *2.3 Order Acceptance and Scheduling (OAS) problems*

There are lots of papers related to OAS problem defined on either stochastic or deterministic parameters, and considered different objectives. Since this area is much expanded, we consider papers related to those problems with deterministic parameters and variables as well as an objective function trying to maximize the profit on both

single and multiple machine scheduling problems.

### *2.3.1 Single machine scheduling problems*

Stern et al. [31] worked on a case study of a textile firm. The $1 \mid d_j, e_j, w_j \mid \sum R_j$ problem is changed to a binary integer program with the objective of maximizing revenues while meeting due dates. They proposed three greedy heuristics and tested them using the data from the real problem. A project acceptance and scheduling problem $(1 \mid d_j, e_j, w_j \mid \sum R_j)$ is considered by Gupta et al. [32] so as to maximize the net present value of the total return. They first consider the scheduling problem assuming that all orders are accepted and then proposed an efficient polynomial dynamic programming method for solving the problem. Aspvall et al. [33] studied the same problem and presented an optimal method for OAS by developing an analogous rule. Considering this problem, Kyparisis et al. [34] showed that the optimal order (sequence) of projects does not depend on the particular subset of selected projects and hence the overall problem is reduced to a multi-constraint nonlinear integer (binary) problem. They therefore proposed two heuristic methods for the single-constraint case and the multi-constraint case, respectively. Slotnick et al. [35] considered OAS problem with the objective of maximizing revenue less penalties due to lateness. Developing an optimal algorithm and two heuristic procedures, they showed that their heuristic demonstrates near-optimal performance in a fraction of the CPU time of the optimal benchmark. Ghosh et al. [36] showed that this problem is NP-hard and then proved that the problem can be solved in polynomial time if either the job processing times or the job weights for the lateness penalty are equal. They used this attribute and provided a fully polynomial time approximation scheme whose solution was near optimal. Lewis et al. [37] presented an optimal dynamic programming algorithm for large problems and a number of myopic heuristics for smaller problems. The same problem where the manufacturer has the flexibility to choose lead times is investigated by Charnsirisakskul et al. [38]. Through numerical analyses, they compared the benefits of lead time flexibility and the flexibility to partially fulfill orders in different

demand environments. Charnsirisakskul et al. [39] added the manufacturer ability to set prices to this problem, and presented decision models that integrate pricing and production decisions. Finally, they concluded the benefits of price customization, lead time, and inventory flexibilities, in various demand environments. Slotnick et al. [40] proposed a straightforward application of an approach which separates sequencing and job acceptance. They then developed an optimal branch-and-bound procedure as well as some heuristics to perform the sequencing and job acceptance decisions jointly. Yang et al. [41] added another decision related to reducing processing times at a cost, and proposed an algorithm that maximizes schedule profit for a given sequence of jobs as well as two heuristic approaches for generating good job sequences. Using an upper bound based on an assignment relaxation, Rom et al. [42] compared the performance of a myopic heuristic and a genetic algorithm based on a probabilistic local search by which obtained results almost as good as exhaustive local search with much shorter running times.

Oguz et al. [43] studied the general OAS problem for the first time and presented the first MILP formulation for $1 \mid r_j, d_j, e_j, w_j, s_{ij} \mid \sum R_j$ problem which can be solved to optimality for smaller problems as well as three heuristic algorithms to solve large sized problems. Considering this problem in a pool consisting of firm planned orders as well as potential orders, Nobibon et al. [44] proposed two linear formulations and two exact branch-and-bound procedures by which they were able to solve instances with up to 50 jobs within reasonable CPU times. The mathematical models presented in [43,44] are not time-indexed models, hence there is a room for developing such models and testing their performance for OAS problems. Cesaret et al. [45] presented a tabu search algorithm for this problem and compared their solutions with two heuristics proposed by Oguz et al. [43] in aspects of running time as well as optimal gap percent. Lin et al. [46] proposed a new algorithm based on Artificial Bee Colony (ABC) for this problem, and showed that their results are sometimes better than Oguz et al. [43] and Cesaret et al. [45]. Working on a diversity controlling genetic algorithm (DCGA), Chen et al. [47] improved the solution qualities for this problem. Nguyen et al. [48]

developed a new branch-and-bound approach as well as new genetic programming method to find optimal solutions for the general OAS. Thevenin et al. [49] proposed and compared a constructive heuristic, local search methods, and population-based algorithms on realistic instances for the general OAS in a single machine environment.

### *2.3.2 Multiple machine problems*

Ono et al. [50] considered a deterministic job shop environment and used two heuristic rules as well as three scheduling rules to obtain near optimal solutions. In the same environment, Pourbabi [51, 52] tried to maximize the profit using an MILP and a dispatching rule for scheduling. Kolisch [53] formulates an MILP for a multi period knapsack problem, with the objective of maximizing the revenue of accepted orders subject to some standard constraints. A heuristic based on LP was then constructed to obtain near optimal solutions for large problems. Considering multiple machine scheduling problem, Mestry et al. [54] developed an MILP which was solvable for small problems as well as a branch-and-price algorithm with Lagrangian bounds and an approximate branching scheme for larger problems.

Xiao et al. [55] formulated an integer programming model for the permutation flow shop scheduling problem with order acceptance and weighted tardiness. They then proposed a heuristic algorithm named Simulated Annealing Based on Partial Optimization (SABPO) and showed that the SABPO algorithm exhibits good optimality for small-sized problems and robustness for medium/large-sized problems compared with benchmarks.

Wang et al. [56] presented a modified artificial bee colony algorithm for a two-stage make-to-order production system characterized by limited production capacity and tight order due dates to generate good solutions for large-scale problem instances. Wang et al. [57] worked on a two-machine flow shop, and formulated the problem as mixed-integer linear programming models, and developed a heuristic and a branch-and-bound algorithm based on some derived dominance rules and relaxation techniques.

# Chapter 3

# OAS PROBLEM $1 \mid d_j, e_j, w_j \mid \sum R_j$

The system under consideration along this thesis is a make-to-order system where each of the orders has different characteristics such as the maximum revenue that can be gained and the required processing time to complete a specific order. In the problem, as in majority of the make-to-order systems, products are sophisticated and specially designed for each customer. Thus, no work-in-process inventory can be carried for the incoming orders. However, it is assumed that the manufacturer has no inventory holding limit for the completed orders and incurs no holding cost.

The organization of this chapter is as follows: In Section 3.1 we give a definition for Order Acceptance and Scheduling Problem $1 \mid d_j, e_j, w_j \mid \sum R_j$. In Section 3.2, we introduce our mixed integer linear programming model and formulation. Then, in Section 3.3, we introduce three dominance properties which can reduce the running time of the mathematical model. Finally, in Section 3.4, we discuss the computational experiments in detail.

## *3.1 Problem Definition*

In this chapter, we discuss order and acceptance scheduling problem in a make-to-order system, where each incoming order has different features and our objective is maximizing the total profit. The orders' features include the time each order needs to be processed, the profit gained if we can deliver the order to the customer before the due date, the weight by which we lose the profit for each day of tardiness, and the deadline that if the processing of the certain order finishes after it, there would be no profit.

Considering a single machine environment, the customers submit their orders $O$ with their features at the beginning of the time horizon. Then, two different and simultaneous decisions needs to be made by us. Since our time horizon and the capacity are limited, we normally cannot accept all orders to process. Therefore, an acceptance or rejection decision must be made for each order. Other than that, we need to find a feasible schedule for the accepted orders by which we gain the maximum profit.

The whole time horizon is divided into $H$ equal time periods. For each order $i \in O$, there are the data including processing time $p_i$, due date $d_i$, maximum revenue $e_i$, and importance weight $w_i$. The manufacturer can gain maximum revenue of order $i$, if he can deliver it to the customer before the due date $d_i$. If the processing of order $i$ is finished after due date, the customer would consider a penalty of importance weight $w_i$ for each day of tardiness, which is equal to the difference between its finish time and its due date, if its processing is finished after its due date, and is equal to zero, otherwise. We assume that all orders are released in the first day. No setup time is considered between processing of two consecutive orders. It is also assumed that when the manufacturer finishes an order, he can deliver it to the customer instantly.

Considering a given schedule $\sigma_S$ including $S$ independent selected orders out of total $n$ existing orders, we can calculate the starting time of each order $i \in S$ by a binary time-indexed variable $x_{it}$ which is one, if processing of order $i$ is started at time period $t$. After scheduling, the revenue of order $i$, $R_i$, can be calculated by subtracting penalty cost from the maximum revenue. To calculate tardiness cost of order $i$, we need to calculate its tardiness, $T_i$. Before modeling the problem, we also need to determine total number of time periods, $H$. We can consider any big number for total number of time periods, but to reduce the number of decision variables as well as the number of constraints, we find the minimum possible total number of time periods. $T_i$ and $H$ are calculated as belows:

$$t = 1, ..., H$$
$$T_i = max\{0, \sum_{t=1}^{H} t x_{it} + p_i - 1 - d_i\}$$

$$H = min\{\sum_{i=1}^{n} p_i, \max_{i=1,\dots,n} (d_i + [\tfrac{e_i}{w_i}])\}$$

Then, summation of all revenues is what the manufacturer gains from this certain schedule, i.e., $\sum_{i \in S} R_i = \sum_{i \in S} (e_i - T_i \times w_i)$ where S is the set of accepted orders.

### *3.2 Mathematical Model (MILP1)*

Using mixed integer linear programming, we modeled the problem as following:

## Indices:

$i$ : orders.

$t, s$ : time periods.

## Sets:

$O$ : set of all orders.

$S$ : set of selected orders.

$\sigma_S$ : processing sequence of selected order set $S$.

## Parameters:

$H$ : number of time periods.

$n$ : number of orders.

$p_i$ : processing time of order $i$, $i \in O$.

$d_i$ : due date of order $i$, $i \in O$.

$e_i$ : maximum revenue of order $i$, $i \in O$.

$w_i$ : importance weight of order $i$, $i \in O$.

## Decision Variables:

$T_i$ : total tardiness of order $i$, $i \in S$.

$R_i$ : revenue gained from order $i$, $i \in S$.

$$x_{it} = \begin{cases} 1 & \text{If processing of order } i \text{ starts at time period } t, \\ 0 & \text{Otherwise.} \end{cases}$$

$$I_i = \begin{cases} 1 & \text{If order } i \text{ is accepted,} \\ 0 & \text{Otherwise.} \end{cases}$$

# MILP1:

$$max \sum_{i=1}^{n} R_i$$

s.t.

$$\sum_{i=1}^{n} x_{it} \leq 1 \qquad t = 1, \ldots, H \tag{3.1}$$

$$\sum_{t=1}^{H} x_{it} = I_i \qquad i \in O \tag{3.2}$$

$$\sum_{i=1}^{n} \sum_{s=max(t-p_i+1,1)}^{t} x_{is} \leq 1 \qquad t = 1, \ldots, H \tag{3.3}$$

$$T_i \geq \sum_{t=1}^{H} t.x_{it} + p_i I_i - d_i - 1 \qquad i \in O \tag{3.4}$$

$$T_i \geq 0 \qquad i \in O \tag{3.5}$$

$$R_i = e_i I_i - w_i T_i \qquad i \in O \tag{3.6}$$

$$x_{it} = \{0, 1\}, I_i = \{0, 1\} \qquad i \in O;\ t = 1, \ldots, H \tag{3.7}$$

Constraint set 3.1 prohibits assigning more than one order to each time period. Constraint set 3.2 implies that if an order is accepted, exactly one time period is needed to be allocated to this order. In case of rejection, no time period must be assigned to the order. Constraint set 3.3 ensures that if an order is started from a time period, no other order can be started from previous time periods unless it can be processed and finished until that certain time period. Constraint sets 3.4 and 3.5

calculate tardiness time of each order so that if the order is finished before its due date, its tardiness would be greater than a negative value, according to constraint set 3.4, and since all tardiness values should be nonnegative, according to constraint set 3.5, its tardiness would be zero. On the other hand, if an order is completed after its deadline, constraint set 3.4 calculates its tardiness, which is a positive value, and constraint set 3.5 becomes redundant in this case. By constraint set 3.6, the model can calculate the revenue for each order, by subtracting the tardiness penalty, if there is any, from maximum revenue of the order. Constraint set 3.7 defines the domain of the variables.

### *3.3 Dominance Properties*

In this part, we develop three dominance properties and add them to the MILP1 separately. Next, we find the cases in which each dominance property improves running time of the MILP1.

1. If $(p_i + p_j > d_i + \frac{e_i}{w_i}$ and $p_i + p_j < d_j + \frac{e_j}{w_j}$ and $e_j \geq e_i$ and $p_j \leq p_i)$ then: $I_i \leq I_j \qquad \forall i, j = 1, ..., n.$

   Here, $d_i + \frac{e_i}{w_i}$, and $d_j + \frac{e_j}{w_j}$ logically defines deadline for order $i$ and $j$, respectively. This property says that if processing both orders $i$ and $j$ needs time more than the deadline of order $i$, as it is shown in Figure 3.1, and if the revenue of order with later deadline is more than the revenue of the order with earlier deadline, and if processing time of order $i$ is greater than processing of order $j$, then our preference must be accepting order $j$. This property does not cut the optimal solution, because if we can find two orders $i$ and $j$ such that:

   - The deadline of order $j$ is greater than the deadline of order $i$,
   - The time order $j$ needs to be processed is less than the time order $i$ needs to be processed,
   - The revenue gained from order $j$ is more than the revenue gained from order $i$,

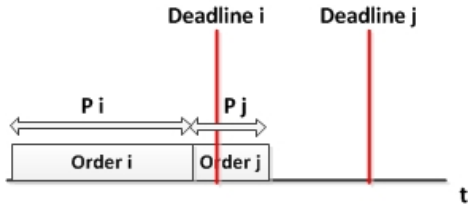


Figure 3.1: First dominance property conditions

in the case, all parameters related to order $j$ are better than those related to order $i$, and that's why order $j$ is preferred.

2. If $(p_i + p_j > d_j + \frac{e_j}{w_j})$ then: $\sum_{t=1}^{H} t.x_{jt} \leq \sum_{t=1}^{H} t.x_{it} + H(1 - I_i) \qquad \forall i, j = 1, ..., n.$

   This property ensures that if processing of both orders $i$ and $j$ needs time more than the deadline of order $j$, and if order $i$ is accepted, then the schedule should set order $j$ before $i$. This property finds the cases the deadline of order $j$ is later than time we need at least for processing both orders $i$ and $j$. Here, we investigate all cases that may happen:

   - Order $i$ is accepted, and order $j$ is rejected: Since order $j$ is rejected, all $x_{jt}$ values are zero, so the left hand side of the property becomes zero. Order $i$ is accepted, so $H(1 - I_i) = H(1 - 1) = 0$, and the right hand side becomes $\sum_{t=1}^{H} t.x_{it}$ which is a positive number. Hence, the property works.

   - Order $i$ is rejected, and order $j$ is accepted: The left hand side of the property becomes a positive number between 0 and $H - p_j$, and the right hand side becomes $0 + H$. Therefore, this inequality holds, and so, the property works.

   - Both orders $i$ and $j$ are rejected: The inequality becomes: $0 \leq 0 + H$. The inequality holds, and hence, the dominance property works in this case as well.

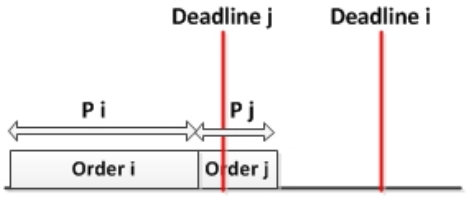


Figure 3.2: Second dominance property conditions

- Both orders $i$ and $j$ are accepted, and $i$ is processed before $j$: This case cannot happen for this inequality, because if order $j$ is processed later, according the fact that $p_i + p_j > d_j + \frac{e_j}{w_j}$, processing of order $j$ will be finished later than its deadline, as Figure 3.2 shows it, which implies that order $j$ is rejected.

- Both orders $i$ and $j$ are accepted, and $j$ is processed before $i$: The inequality becomes $\sum_{t=1}^{H} t.x_{jt} \leq \sum_{t=1}^{H} t.x_{it}$, which holds according to our assumption in this case.

3. If $(\frac{e_j}{w_j} > H - d_j)$ then: $I_j = 1 \qquad \forall j = 1, ..., n.$

   This property implies that if deadline of order $j$ is later than the time we need for processing all orders, then order $j$ must be accepted. This inequality works, because if deadline of order $j$ is later than the time we need for processing all orders, whenever we process order $j$, we will finish its processing earlier than its deadline, so we will gain a positive revenue.

These three dominance properties were coded in CPLEX and fed into the MILP1 model as additional constraints. Next, both the MILP1 model with dominance properties added and the MILP1 model were solved with the benchmark data generated as explained in Section 3.4.1 to test the performance of these models.

## *3.4 Computational Study*

We performed computational experiments to analyze the performance of the proposed mathematical model (MILP1) and dominance properties. In this section, we first describe the benchmark data used in the experiments. For this problem, we use the benchmark instances generated by Slotnick and Morton [40] for 10- and 20-order instances, and by Rom and Slotnick [42] for 75- and 100-order instances.

### *3.4.1 Data Generation*

#### *3.4.1.1 Slotnick and Morton [40] Data*

In the first set of benchmark data, 100 10-order and 100 20-order problems are randomly generated by Slotnick and Morton [40]. Weights and processing times are generated from a uniform distribution $(0, 1)$, then adjusted by a constant, in order to vary the difficulty of the problem in terms of profit margin and order size. Revenue is randomly generated from a lognormal distribution with an underlying normal distribution with mean 0 and standard deviation one. Due dates are generated from a uniform distribution, adjusted to the magnitude of processing times in a particular problem. This choice of distributions reflects a situation where order characteristics are similar, but revenue may vary more widely.

#### *3.4.1.2 Rom and Slotnick [42] Data*

In the second set of benchmark data, two sets of 120 problems are randomly generated, one with the size 75 and other with the size 100 by Rom and Slotnick [42]. As for the pilot study (but using different random number seeds) weights and processing times are drawn from a uniform distribution $(0, 1)$, and due dates are drawn from a uniform distribution, adjusted to the magnitude of processing times in each problem. Each due date equals the generated uniform number plus an adjustment factor: $2 \times [p_i/(1+\tau)] - 1$. The problems thus generated typify a scenario where job characteristics are similar, but revenue may vary more widely. Three different settings of $\tau$ $(0.3, 2.0, 3.0)$ are

considered and generated revenues for half of the problems without correlating due date and revenue (average correlation across all problems was 0.003), and for the other half, revenues are generated that are inversely correlated with each due date for each order (average correlation $-0.187$). This resulted in six sets of orders with different order characteristics. To call these groups easier, group numbers 1 and 2 are used for $\tau = 0.3$ without and with correlation, respectively. Group numbers 3 and 4 are used for $\tau = 2.0$ without and with correlation, respectively. Group numbers 5 and 6 are used for $\tau = 3.0$ without and with correlation, respectively.

### *3.4.2 Computational Platform*

All of the runs throughout these computational experiments were performed on a laptop with an Intel Xeon processor, 2.70 Ghz speed, and 8 GB of RAM. In MILP1 with and without dominance properties runs, we have used ILOG CPLEX 12.5.1.0. Time limit was 900 seconds for total CPU time for CPLEX runs. All of the solution methods developed within the scope of this thesis were coded in CPLEX.

### *3.4.3 Computational Experiments with MILP1 model*

Here, branch-and-bound process is stopped in CPU time of 900 seconds. The upper bound and the lower bound obtained from this process are taken and called as UB and LB, respectively. Then, gaps are calculated as below:

$$\%\ Optimality\ gap = \frac{UB - LB}{UB} \times 100$$

For 10-order and 20-order problems, MILP1 is solved and the results are presented in Tables 3.1 and 3.2, respectively . In these tables, the lower bound, upper bound, and CPU time for all 100 10-order and 20-order problems are reported. The result related to previous attempts by Slotnick and Morton [40] are given and compared. The platform used by Slotnick and Morton [40] is very old, and we wanted to run their methods for all instances by our platform. However, they did not publish their algorithms' details, and hence, we used their CPU times in Tables 3.1 and 3.2. It

is observed that MILP1 can solve all 10-order and 20-order problems to optimality in less than one second, while best previous effort could obtain average optimality gap of 0.06% and at least 12.21% for 10-order and 20-order problems, respectively. For 20-order instances, since Slotnick and Morton [40]'s branch-and-bound algorithm could not solve any instance to optimality, they used a Beam search/RBVA method as their benchmark solution, and then, they compared their other methods solutions with Beam search/RBVA solutions. For this reason, we put $\geq$ and $\leq$ signs in the table.

For 75-order and 100-order problems, there are six groups of data, each having 20 instances. For all instances, MILP1 is first solved, and then, three dominance properties are added to the problem one by one, and in the last effort, MILP1 with the second and third dominance properties are solved with maximum CPU time of one hour. In the literature, Rom and Slotnick [42] constructed two heuristic methods, Myopic algorithm and Genetic algorithm, to solve the problem with small optimality gaps in a reasonable time.

Regarding the first group of instances, according to Table 3.4 and Table 3.8, Myopic method can be solved in much less CPU time compared to all others, and after that, Genetic algorithm and MILP1 without dominance properties perform better in 75- and 100-order instances, respectively. Regarding optimality gap, according to Table 3.3, all methods except Myopic can solve all 75-order instances to optimality. Table 3.7 shows that MILP1 with and without dominance properties can solve all 100-order instances to optimality, but Myopic and Genetic cannot. So we can conclude that MILP1 without dominance properties performs best if we want the optimal solution in a reasonable time.

Second group of 75- and 100-order instances are also solved by Myopic, Genetic, and our methods. Table 3.4 shows that the best time performance is for MILP1 with the first dominance property. In aspect of optimality gap, MILP1 with and without dominance properties solved this group of instances to optimality. In 100-order instances, although MILP1 with the first property does not have the minimum CPU time, we can arrive at the conclusion that MILP1 with the first dominance property

performs best in both aspects of time and optimality gap together, since its running time is not that different than that of Myopic and Genetic algorithms which are not able to solve all 100-order instances to optimality.

Myopic algorithm is the best with respect to time for solving the third group of 75 and 100-order instances, according to Table 3.4 and Table 3.8. However, the average deviation of solutions solved by MILP1 with the second and third dominance properties for 75-order instances, and MILP1 with the second dominance property for 100-order instances are the minimum among all methods according to Table 3.3 and Table 3.7. Therefore, as a conclusion, either only second or both second and third dominance properties can be used to obtain a very near optimal solution.

For the fourth group, it seems that Myopic algorithm performs best in aspect of time based on Table 3.4 and Table 3.8. MILP1 with the third dominance property, and MILP1 with the second and the third dominance properties work better to find solutions more close to optimal solutions in 75 and 100-order instances, respectively. Therefore, the decision maker should consider whether the solution quality ot the computational time is more important when choosing the appropriate method. Hence, if the solution quality is important, then we suggest using MILP1 after adding the third or both second and third dominance properties. Otherwise, Myopic algorithm can be chosen by the decision maker.

Regarding the fifth group, Myopic algorithm again performs better than all others in aspect of time based on Table 3.4 and Table 3.8. MILP1 with the third dominance property, and MILP1 with the second dominance property work better to find more near optimal solutions in 75 and 100-order instances, respectively. Therefore, choosing which method is better again depends on which factor is more important for the manufacturer. MILP1 after adding the third or both second and third dominance properties is suggested when solution quality is important, and Myopic algorithm is suggested when computational time is the important factor.

Considering the sixth group of instances, Myopic method is the fastest method like other groups of instances, according to Table 3.4 and Table 3.8. However, since MILP1

with or without dominance properties can solve all 75-order instances to optimality in two to three minutes, using MILP1 with the second dominance property is suggested. Using MILP1 with the second dominance property, we obtain the minimum possible gaps for 100-order instances in average among all methods.

Tables 3.5 and 3.9 show the number of instances whose solution is optimal out of 120 instances for 75-order and 100-order instances, respectively. Tables 3.6 and 3.10 show the maximum and minimum optimality gap for different groups of 75-order and 100-order instances, respectively.

#### *3.4.3.1 Statistical Hypothesis Testing*

Statistical analysis is a powerful tool to apply when evaluating the performance of heuristics algorithms. Often the results of computational studies are reported as tables of operation counts, CPU times, solution qualities, etc. that are used to support the authors contentions. In this section, to check the performance of our dominance properties, a t-test which is the two-sample version of Friedmans test is used. In the start of the procedure, there are two hypotheses $H_0$: "the dominance property(s) added to the MILP1 improves the CPU time", and $H_1$: "the dominance property(s) added to the MILP1 does not improve CPU time". The first one is called null hypothesis, and is for the time being accepted. The second one is called alternative (hypothesis). It is the hypothesis one hopes to support. The significance level ($\alpha$), which a probability threshold below which the null hypothesis will be rejected, is considered as 5%. The observed value $t_{obs}$ of the test statistic T for the case where dominance property $i$ is added to MILP1 is calculated as below:

$$t_{obs} = \frac{\overline{CPU}_i - \overline{CPU}_{MILP1}}{\sqrt{\frac{\overline{S^2}_i}{n_i} - \frac{\overline{S^2}_{MILP1}}{n_{MILP1}}}} \tag{3.8}$$

In the equation 3.8, $\overline{CPU}_i$ and $\overline{CPU}_{MILP1}$, show the average CPU time of all instances in a certain group solved by adding dominance property $i$ to MILP1, and the average CPU time of all instances in a certain group solved by MILP1, respectively. $\overline{S^2}_i$

and $\overline{S^2}_{MILP1}$ show the variance of all CPU times in a certain group solved by adding dominance property $i$ to MILP1, and the variance of all CPU times in a certain group solved by adding MILP1, respectively. $n_i$, and $n_{MILP1}$ show the number of instances which are solved by MILP1 with dominance property $i$ and MILP1, respectively. In each certain group, there are 20 instances. Hence, to compare the performance of dominance properties, MILP1 with each dominance property must be compared with MILP1 for each group of instances. Therefore, $n_i = n_{MILP1} = 20$. The critical region is in the range $(-\infty, t_{0.05,19})$. $t_{0.05,19}$ is equal to 0.519678 which means that if $t_{obs}$ is less than 0.519678, hypothesis $H_0$ is accepted and so, adding dominance property $i$ to MILP1 improves the average CPU time with the probability of 95%. Here, we investigate all cases when one or more dominance properties are added to MILP1:

- **MILP1+DP1 and MILP1+DP2:** $t_{obs}$ values of both MILP1+DP1 and MILP1+DP2 are less than 0.519678 for all groups of instances and both sizes of 75- and 100-order, based on Tables 3.11 and 3.12. According to that, we can claim that adding the first and the second dominance properties improves CPU times with the probability of 95% for all groups in large instances.

- **MILP1+DP3 and MILP1+DP2+DP3:** According to Tables 3.11, and 3.12, $t_{obs}$ values of both MILP1+DP3 and MILP1+DP2+DP3 are less than 0.519678 for all groups of 75-order and 100-order instances except the first group of instances in 100-order problems. Hence, we can claim that adding only the third dominance property or both the second and the third dominance properties to MILP1 improves the CPU times of groups 2, 3, 4, 5, and 6, with the probability of 95% and they may improve the CPU times of instances in group 1 with the probability of 5%.

### *3.4.4 Conclusions on Computational Experiments*

In this section, we performed computational experiments to compare the results obtained by solving MILP1 with and without dominance properties using CPLEX

12.5.1.0 and the results of previous studies.

We first explained the random data generation procedure used in creating our test instances. Next, we solved these instances using MILP1 with and without dominance properties by CPLEX 12.5.1.0 with one hour CPU time limit using the generated data.

Comparing MILP1 with the previous studies for 10-order and 20-order instances, we observed that MILP1 can be solved in average time of less than one second to solve all instances to optimality.

Then, we compared two previous methods with MILP1 for 75-order and 100-order instances for each certain group of instances. We arrived at the conclusion that Myopic algorithms is solved faster than all other methods, but its percentage deviation is considerably larger than that of others. Genetic algorithm works better than Myopic in aspect of optimality gap, but its running time is always larger than that of Myopic. MILP1 dominate all other methods in the first group of instances. MILP1 can be used most efficiently in the second and the sixth groups of instances. Adding the second dominance property is most helpful when instances of the third or the fifth group are solved. MILP1 with the third dominance property is most forceful for solving the fourth or the fifth groups of instances. Finally, MILP1 with the second and the third dominance properties are mostly suggested for solving the third and fourth groups of instances.

Figure 3.3 shows how different methods work for different groups of instances. In the figure, the optimality gap is calculated by taking the average of all optimality gaps for all 75-order and 100-order instances in each different group. MILP1 with and without dominance properties work the same to each other compared to Myopic and Genetic algorithms in aspect of their solutions quality. To observe the difference more precisely among MILP1 in absence or presence of dominance properties, Figure 3.4 is added to the end of this chapter. In this figure, the performance of MILP1 and MILP1 with different dominance properties are compared. The figure shows that all methods solve all instances of the first and the second group to optimality. Considering the

third and the fourth groups of instances, MILP1 with the second dominance property gives solutions with less optimality gap compared to others. MILP1 with the third dominance property works better than others for the fifth and the sixth groups of instances in aspect of solution quality.

Table 3.1: Summary of computational study for 10-order problems

| Method | Average dev. from opt. (%) | Worst dev. from opt. (%) | Percentage equal to opt. | Avg. time (sec) |
|---|---|---|---|---|
| MILP1 | 0 | 0 | 100 | 0.56566 |
| Optimal B&B [40] | 0 | 0 | 0 | 6154.12 |
| Beam/RBAS [40] | 0.27 | 6.78 | 80 | 341.01 |
| B&B/RBVA [40] | 0.06 | 3.19 | 95 | 58.74 |
| Beam/RBVA [40] | 0.64 | 9.54 | 66 | 59.79 |
| RHAS [40] | 1.33 | 15.56 | 50 | 0.45 |
| RHVA [40] | 17.31 | 119.21 | 17 | 0.03 |

Table 3.2: Summary of computational study for 20-order problems

| Method | Average dev. from opt. (%) | Worst dev. from opt. (%) | Percentage equal to opt. | Avg. time (sec) |
|---|---|---|---|---|
| MILP1 | 0 | 0 | 100 | 0.83851 |
| Beam/RBAS [40] | $\geq 0$ | $\geq 0$ | 0 | 8474 |
| Beam/RBVA [40] | $\geq 12.21$ | $\geq 80.06$ | $\leq 6$ | 2553 |
| RHAS [40] | $\geq 0$ | $\geq 0$ | $\leq 100$ | 3.9 |
| RHVA [40] | $\geq 119.15$ | $\geq 483.08$ | $\leq 5$ | 1.41 |

Table 3.3: Optimality gap comparison for 75-order problems

| Group num. | Average dev. from opt. (%) | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic [42] | Myopic [42] | MILP1 | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | 0 | 0.267025 | 0 | 0 | 0 | 0 | 0 |
| Group 2 | 0.020078 | 0.185981 | 0 | 0 | 0 | 0 | 0 |
| Group 3 | 0.325926 | 0.990148 | 0.04591 | 0.04196 | 0.03563 | 0.03584 | 0.02424 |
| Group 4 | 0.14621 | 1.142436 | 0.01011 | 0.01211 | 0.02059 | 0.00959 | 0.01599 |
| Group 5 | 0.212944 | 1.116169 | 0.01056 | 0.01533 | 0.01116 | 0.00247 | 0.02084 |
| Group 6 | 0.288231 | 2.041464 | 0 | 0 | 0 | 0 | 0 |

Table 3.4: Time comparison for 75-order problems

| Group num. | CPU time (sec) | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic [42] | Myopic [42] | MILP1 | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | 24.51 | 8.14 | 48.12 | 38.19 | 35.19 | 40.58 | 31.93 |
| Group 2 | 20.67 | 16.58 | 15.17 | 10.07 | 12.69 | 18.36 | 16.35 |
| Group 3 | 67.68 | 9.81 | 344.54 | 339.62 | 364.49 | 333.47 | 357.40 |
| Group 4 | 63.55 | 13.94 | 220.67 | 243.44 | 216.12 | 222.94 | 241.41 |
| Group 5 | 52.32 | 9.3 | 253.49 | 246.81 | 265.01 | 248.14 | 260.23 |
| Group 6 | 56.29 | 14.9 | 138.60 | 156.70 | 146.22 | 123.75 | 141.93 |

Table 3.5: Comparison of number of optimal solutions for 75-order problems

| Group num. | Number of solutions equal to UB (out of 20) | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic [42] | Myopic [42] | MILP1 | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | 20 | 2 | 20 | 20 | 20 | 20 | 20 |
| Group 2 | 19 | 7 | 20 | 20 | 20 | 20 | 20 |
| Group 3 | 1 | 0 | 14 | 14 | 14 | 15 | 15 |
| Group 4 | 2 | 0 | 18 | 16 | 17 | 17 | 16 |
| Group 5 | 5 | 0 | 18 | 17 | 17 | 19 | 17 |
| Group 6 | 3 | 0 | 19 | 19 | 19 | 20 | 19 |

Table 3.6: Maximum and minimum optimality gap comparison for 75-order problems

| Group Num. | Max & Min average dev. from UB (%) | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Genetic [42] | | Myopic [42] | | MILP1 | | MILP1 + DP1 | | MILP1 + DP2 | | MILP1 + DP3 | | MILP1 + DP2 + DP3 | |
| | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min |
| Group 1 | 0 | 0 | 1.061 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Group 2 | 0.402 | 0 | 1.167 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Group 3 | 1.136 | 0 | 6.271 | 0.070 | 0.335 | 0 | 0.347 | 0 | 0.211 | 0 | 0.413 | 0 | 0.191 | 0 |
| Group 4 | 0.446 | 0 | 3.010 | 0.074 | 0.107 | 0 | 0.115 | 0 | 0.291 | 0 | 0.097 | 0 | 0.194 | 0 |
| Group 5 | 0.666 | 0 | 5.434 | 0.017 | 0.140 | 0 | 0.147 | 0 | 0.102 | 0 | 0.049 | 0 | 0.290 | 0 |
| Group 6 | 1.291 | 0 | 8.946 | 0.049 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 3.7: Optimality gap comparison for 100-order problems

| Group num. | Average dev. from opt. (%) | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic [42] | Myopic [42] | MILP1 | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | 0.070696 | 0.325046 | 0 | 0 | 0 | 0 | 0 |
| Group 2 | 0.008249 | 0.181059 | 0 | 0 | 0 | 0 | 0 |
| Group 3 | 0.441785 | 0.811384 | 0.24027 | 0.15877 | 0.13600 | 0.17373 | 0.18668 |
| Group 4 | 0.265643 | 1.767784 | 0.05932 | 0.06619 | 0.05004 | 0.05707 | 0.04880 |
| Group 5 | 0.577417 | 1.430944 | 0.03746 | 0.03871 | 0.02953 | 0.03566 | 0.04899 |
| Group 6 | 0.352202 | 1.993508 | 0.01846 | 0.02284 | 0.01614 | 0.02251 | 0.02584 |

Table 3.8: Time comparison for 100-order problems

| Group num. | CPU time (sec) | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic [42] | Myopic [42] | MILP1 | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | 55.37 | 19.85 | 47.81 | 48.56 | 53.90 | 72.96 | 79.08 |
| Group 2 | 46.57 | 38.74 | 67.65 | 68.99 | 51.05 | 79.72 | 60.99 |
| Group 3 | 119.21 | 22.22 | 573.69 | 578.38 | 603.61 | 536.40 | 552.47 |
| Group 4 | 132.43 | 33.87 | 294.03 | 288.72 | 299.68 | 286.44 | 296.96 |
| Group 5 | 137.16 | 23.9 | 415.01 | 372.96 | 407.54 | 397.62 | 426.74 |
| Group 6 | 116.54 | 33.41 | 314.45 | 323.07 | 317.19 | 320.01 | 323.92 |

Table 3.9: Comparison of number of optimal solutions for 100-order problems

| Group num. | Number of solutions equal to UB (out of 20) | | | | | | |
|---|---|---|---|---|---|---|---|
| | Genetic [42] | Myopic [42] | MILP1 | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | 15 | 1 | 20 | 20 | 20 | 20 | 20 |
| Group 2 | 18 | 6 | 20 | 20 | 20 | 20 | 20 |
| Group 3 | 0 | 1 | 11 | 11 | 12 | 13 | 12 |
| Group 4 | 0 | 0 | 14 | 14 | 14 | 15 | 14 |
| Group 5 | 0 | 0 | 14 | 13 | 13 | 13 | 13 |
| Group 6 | 2 | 0 | 14 | 13 | 14 | 14 | 13 |

Table 3.10: Maximum and minimum optimality gap comparison for 100-order problems

| Group Num. | Max & Min average dev. from UB (%) | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Genetic [42] | | Myopic [42] | | MILP1 | | MILP1 + DP1 | | MILP1 + DP2 | | MILP1 + DP3 | | MILP1 + DP2 + DP3 | |
| | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min |
| Group 1 | 0.421 | 0 | 1.137 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Group 2 | 0.127 | 0 | 0.906 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Group 3 | 1.082 | 0.074 | 2.474 | 0 | 1.345 | 0 | 1.345 | 0 | 0.876 | 0 | 0.907 | 0 | 1.095 | 0 |
| Group 4 | 1.197 | 0.004 | 8.018 | 0.112 | 0.382 | 0 | 0.725 | 0 | 0.588 | 0 | 0.545 | 0 | 0.339 | 0 |
| Group 5 | 1.892 | 0.001 | 3.935 | 0.106 | 0.302 | 0 | 0.398 | 0 | 0.202 | 0 | 0.334 | 0 | 0.334 | 0 |
| Group 6 | 1.341 | 0 | 7.027 | 0.085 | 0.135 | 0 | 0.108 | 0 | 0.096 | 0 | 0.124 | 0 | 0.149 | 0 |

Table 3.11: $t_{obs}$ values of MILP1 and MILP1 with dominance properties for 75-order problems

| Group num. | $t_{obs}$ values (comparison with MILP1) | | | |
|---|---|---|---|---|
| | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | -0.33 | -0.45 | -0.26 | -0.64 |
| Group 2 | -0.92 | -0.43 | 0.38 | 0.19 |
| Group 3 | -0.04 | 0.16 | -0.09 | 0.10 |
| Group 4 | 0.21 | -0.04 | 0.02 | 0.19 |
| Group 5 | -0.07 | 0.11 | -0.05 | 0.07 |
| Group 6 | 0.24 | 0.11 | -0.23 | 0.05 |

Table 3.12: $t_{obs}$ values of MILP1 and MILP1 with dominance properties for 100-order problems

| Group num. | $t_{obs}$ values (comparison with MILP1) | | | |
|---|---|---|---|---|
| | MILP1 + DP1 | MILP1 + DP2 | MILP1 + DP3 | MILP1 + DP2 + DP3 |
| Group 1 | 0.02 | 0.22 | 0.60 | 0.75 |
| Group 2 | 0.02 | -0.35 | 0.21 | -0.13 |
| Group 3 | 0.04 | 0.29 | -0.35 | -0.19 |
| Group 4 | -0.04 | 0.04 | -0.06 | 0.02 |
| Group 5 | -0.34 | -0.06 | -0.14 | 0.10 |
| Group 6 | 0.06 | 0.02 | 0.04 | 0.07 |

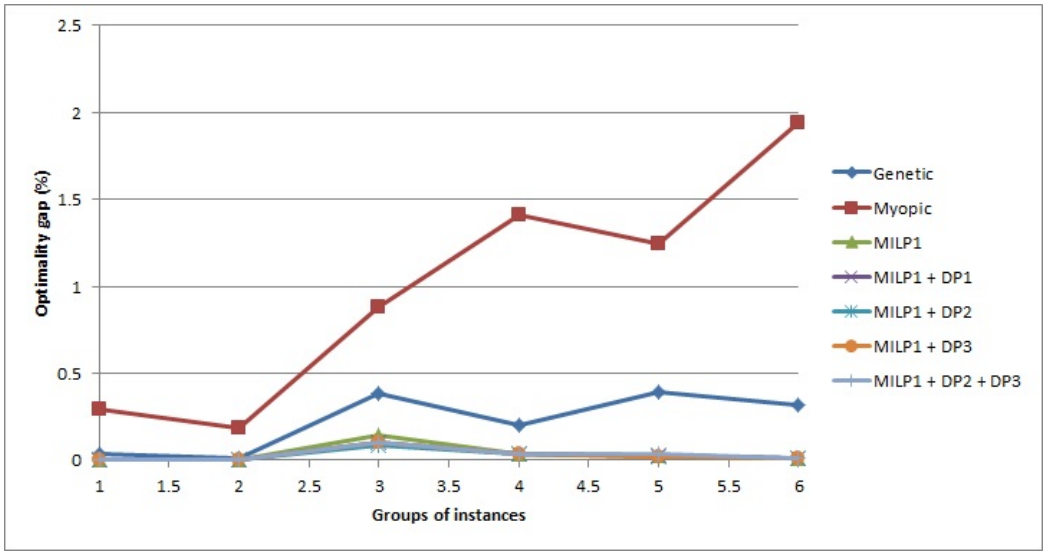


Figure 3.3: Comparison of different methods with respect to optimality gap for 75-order and 100-order instances

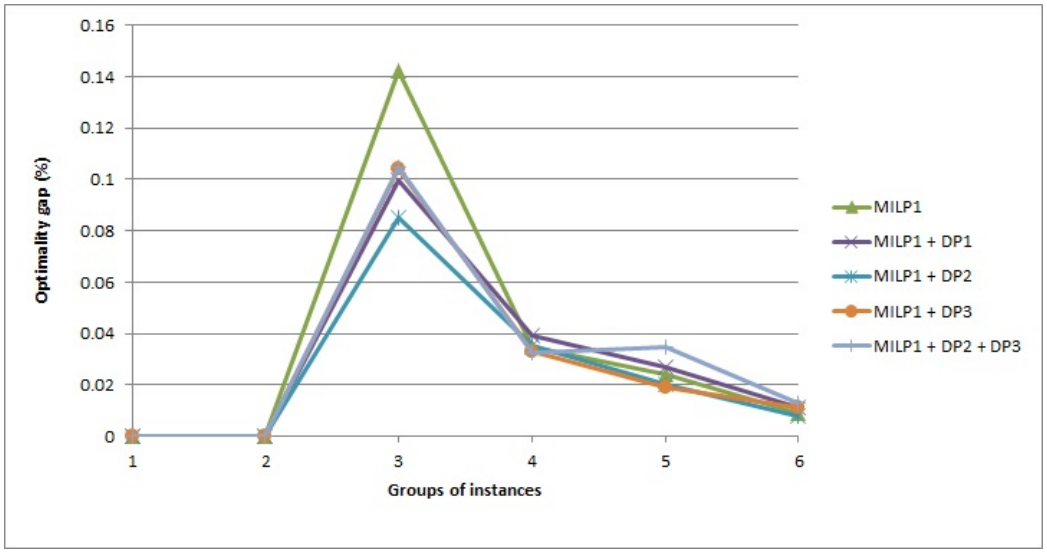


Figure 3.4: Comparison of MILP1 with and without dominance properties with respect to optimality gap for 75-order and 100-order instances

# Chapter 4

# OAS PROBLEM $1 \mid r_j, d_j, e_j, w_j, s_{ij} \mid \sum R_j$

In this chapter, the general version of the order acceptance and scheduling problem is considered in which release dates for each order and setup times for two consecutive order are also given.

The organization of this chapter is as follows: In Section 4.1, we give a definition for the $1 \mid r_j, d_j, e_j, w_j, s_{ij} \mid \sum R_j$ problem. In Section 4.2, we introduce a mixed integer linear programming model through a time-indexed formulation (MILP2). Then, in Section 4.3, we present Lagrangian relaxation method as well as different LP relaxations which we applied to MILP2 to obtain upper bounds. Afterwards, in Section 4.4, we give a simple heuristic as well as a revised mathematical model constructed for this problem. Section 4.5 describes a different mathematical formulation for this problem in which the variables' definitions are changed. Finally, in Section 4.6, we discuss the computational experiments in detail.

## *4.1 Problem Definition*

In this chapter, considering a single machine environment, the order acceptance and scheduling problem in which the order $i$ has the same parameters with the problem discussed in Chapter 3 as well as a release date and a setup time $s_{ij}$ between each two consecutive orders $i$ and $j$. Release date refers to the date when the order comes to us, so we are not able to start its processing before that date. Setup time refers to the time which should be spent between processing of two consecutive orders. Since setup time depends on both machine and orders, setup time can be started only after the order is released to us.

We break the time into $H + 2$ equal time periods. We use two dummy orders 0 and

$n+1$ which are set as the beginning and the end of each feasible schedule. Therefore, before solving the problem, we assign the dummy time period 0 to order $i = 0$, and dummy time period $H+1$ to order $i = n+1$. After constructing the feasible schedule, its total revenue can be obtained by subtracting tardiness penalties from the maximum revenue of each order, and then, adding up all gained revenues. To calculate tardiness cost of order $i$, we need to calculate its tardiness, $T_i$. Before modeling the problem, we also need to determine the total number of time periods, $H+2$. We can consider any big number for total number of time periods, but to reduce the number of decision variables as well as constraint, we find the minimum possible total number of time periods. $T_i$ and $H$ are calculated as belows:

$$t = 0, ..., H+1$$
$$T_i = max\{0, \sum_{t=1}^{H} t x_{it} + p_i - 1 - d_i\}$$
$$H = \max_{i=1,...,n} \bar{d}_i$$

### *4.2 Mathematical Model (MILP2)*

Using mixed integer programming, we modeled the problem as the following:

## Indices:

$i, j$ : orders.

$t$ : time periods.

## Sets:

$O$ : set of all original orders.

$S$ : set of selected orders.

$\sigma_S$ : processing sequence of selected order set $S$.

## Parameters:

$H$ : number of main time periods.

$n$ : number of original orders.

$r_i$ : release date of order $i$, $i \in O$.

$p_i$ : processing time of order $i$, $i \in O$.

$d_i$ : due date of order $i$, $i \in O$.

$\bar{d}_i$ : deadline of order $i$, $i \in O$.

$e_i$ : maximum revenue of order $i$, $i \in O$.

$w_i$ : importance weight of order $i$, $i \in O$.

$s_{ij}$ : sequence dependent setup time between two tandem orders $i$ and $j$, $i, j \in O$.

## Decision Variables:

$T_i$ : total tardiness of order $i$, $i \in S$.

$R_i$ : revenue gained from order $i$, $i \in S$.

$$x_{it} = \begin{cases} 1 & \text{If processing of order } i \text{ starts at time period } t, \\ 0 & \text{Otherwise.} \end{cases}$$

$$I_i = \begin{cases} 1 & \text{If order } i \text{ is accepted,} \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{If order } j \text{ is processed immediately after order } i, \\ 0 & \text{Otherwise.} \end{cases}$$

# MILP2:

$$max \sum_{i=1}^{n} R_i$$

s.t.

$$\sum_{i=1}^{n} x_{it} \leq 1 \qquad t = 0, .., H+1 \tag{4.1}$$

$$\sum_{t=r_i+1}^{\bar{d}_i - p_i} x_{it} = I_i \qquad i \in O \tag{4.2}$$

$$\sum_{t=0}^{H+1} t.x_{jt} \geq \sum_{t=0}^{H+1} t.x_{it} + p_i + s_{ij}y_{ij} + \bar{d}_i(y_{ij} - 1) \qquad i = 0, ..., n; j = 1, ..., n+1; i \neq j \tag{4.3}$$

$$\sum_{t=0}^{H+1} t.x_{jt} \geq I_j(r_j + 1) + s_{ij}y_{ij} \qquad i = 0, ..., n; j = 1, ..., n+1; i \neq j \tag{4.4}$$

$$T_i \geq \sum_{t=1}^{H} t.x_{it} + p_i I_i - d_i - 1 \qquad i \in O \tag{4.5}$$

$$T_i \geq 0 \qquad i \in O \tag{4.6}$$

$$\sum_{i=1, i\neq j}^{n+1} y_{ji} = I_j \qquad j = 0, ..., n \tag{4.7}$$

$$\sum_{i=0, i\neq j}^{n} y_{ij} = I_j \qquad j = 1, ..., n+1 \tag{4.8}$$

$$R_i = e_i I_i - w_i T_i \qquad i \in O \tag{4.9}$$

$$I_0 = 1, \;\; I_{n+1} = 1 \tag{4.10}$$

$$x_{00} = 1, \;\; x_{n+1,H+1} = 1 \tag{4.11}$$

$$x_{it} = \{0, 1\}, I_i = \{0, 1\}, y_{ij} = \{0, 1\} \qquad i, j \in O; \; t = 0, .., H+1 \tag{4.12}$$

Constraint set 4.1 prohibits assigning more than one order to each certain time

period. Constraint set 4.2 implies that if a certain order is accepted, one time period is needed to be allocated to this order, and this time period can be between its release date and the last period in which the order can start to be processed regarding its deadline. In case of rejection, no time period must be assigned to the order. Constraint set 4.3 ensures that if an order $j$ starts exactly after order $i$, it can start only after a setup time $s_{ij}$ is passed. Constraint set 4.4 ensures that if order $j$ is supposed to start after order $i$, and $r_j$ is later than finishing time of order $i$, order $j$ must wait until its release date, and then setup time is started. Constraint sets 4.5 and 4.6 calculate tardiness time for each order so that if the order is finished before its due date, its tardiness would be greater than a negative value, according to constraint set 4.5, and since all tardiness values should be nonnegative, according to constraint set 4.6, its tardiness would be zero. On the other hand, if a certain order is completed after its deadline, constraint set 4.5 calculates its tardiness time, which is a positive value, and constraint set 4.6 does not play any role in this case. Constraints sets 4.7 and 4.8 imply that only one order can be after and before a certain order which is selected to be processed. By constraint set 4.9, the model can calculate the revenue for each order, by subtracting the tardiness penalty, if there is any, from maximum revenue of the order. Constraints set 4.10 set the acceptance of dummy orders. Constraints set 4.11 assign the first and the last dummy time periods to dummy orders. Constraint set 4.12 defines the domain of the variables.

## *4.3 Upper Bounds*

### *4.3.1 Lagrangian Relaxation*

Lee and Sherali [59] and Sherali et al. [60] employed this technique to derive good lower and upper bounds for a scheduling problem with parallel unrelated machines subject to time windows and machine downtime constraints.

An observation on MILP2 is that the constraint set 4.3 includes the complicating constraints that undertake the most difficult order: subtour elimination. When we remove the constraint set 4.3, the model can be solved easily.

We applied Lagrangian relaxation based on subgradient optimization method for updating Lagrangian multipliers. We relaxed constraint set 4.3, and incorporated it into the objection function with a penalty coefficient. Then, we use subgradient optimization to solve the problem. However, the resulting value of the objective function is simply the summation of all maximum revenues which is a very obvious and loose upper bound. The reason for obtaining such a solution is that by relaxing constraint set 4.3, we let the problem to produce some subtours in the solution while constraint set 4.3 eliminated the subtours when it was used as a constraint. Our Lagrangian relaxation algorithm based on Subgradient Optimization method is given in Algorithm 1.

In the Lagrangian relaxation algorithm, no_improvement() is true if $z_{min}$ is not improved in a specified number of recent iterations (we set this value as 30). The function terminate() is true if the optimum is found (i.e. $z_{lb} = z^*_{LUBP}$), or if $z_{min}$ is not improved in a specified number of recent restarts (line 25). The function add_constraints() adjusts S, b, and sets $\lambda_k = 0$ for all new constraints. For $\pi_{min}$, we used the value 0.005.

### *4.3.2 LP Relaxation*

In this thesis, we use time-indexed formulations for all mathematical models, since time-indexed formulations are working very well if they are relaxed. In this part, to obtain good upper bounds, we relax all binary variables at first, and then, to prevent dividing $y_{ij}$ values to very small numbers, we remove the constraint related to assigning last dummy time period to last dummy order (the second constraint in constraint set 4.11), and instead of that, we add two new valid inequalities to the problem. Then, it is observed that the relaxed $x_{jt}$ values are not dispersed too much, but they indicate time locations which may be too far away from each other for order $j$ which indicates not tight upper bounds. Therefore, we add the third valid inequality which divides $x_{jt}$ values to very small parts, but almost all of them are near to each other, and the solution is closer to optimal now. The valid inequalities are as the following:

**Input**: LUBP(); // Lagrangian upper bound program solver
**Input**: $z_{lb}$; // Lower bound value
**Input**: separate(); // Separate constraint 4.3
**Output**: upper bound value

initialization;
$\pi = \pi_{init}$; // Subgradient agility, $\pi_{init} = 2$
S,b; // S: the multiplier matrix $(((n+1)^2 - n) * ((n+2)(T+2))$ of variables, b: right hand side vector $(((n+1)^2 - n) * 1)$ of constraints 4.3
$\lambda$; // Lagrangian multiplier, set them to $\lambda = 0$
$z_{min} = +\infty$; // Best upper bound so far
**repeat**
  $x^*_{LUBP} = LUBP(\Lambda)$; // Solve the LUBP to optimality
  $z^*_{LUBP} = t.x^*_{LUBP} + \lambda(b - S.x^*_{LUBP})$; //Corresp. objective value
  $\delta = b - S.x^*_{LUBP}$; // Compute subgradients $\delta$
  $\Delta = \frac{\pi(z^*_{LUBP} - z_{LB})}{\sum_{i=1}^{(n+1)^2-n} \delta_i^2}$; // Compute step size $\Delta$
  $\lambda_i = max(0, \lambda_i - \Delta.\delta_i) \quad \forall 1 \leq i \leq (n+1)^2 - n$; // Update Lagrangian mult.
  **if** $z^*_{LUBP} > z_{min}$ **then**
    $z_{min} = z^*_{LUBP}$; // Remember best upper bound
  **end**
  **if** *no_improvement()* **then**
    $\pi = \frac{\pi}{2}$; // Reduce agility
  **end**
  **if** $\frac{\pi}{2} \leq \pi_{min}$ **then**
    separate(); // Separate new constraints
  **end**
  **if** $\pi \leq \pi_{min}$ **then**
    add_constraint(); // Add recently separated constraints
    $\pi = \pi_{init}$ ; // Reset agility, i.e. restart SG
  **end**
**until** *until terminate()*;

**Algorithm 1:** Lagrangian relaxation algorithm

1. $\sum_{t=0}^{H+1} x_{n+1,t} \geq \min_{i=1,...,n} r_i + \sum_{i=1}^{n+1}(p_i I_i + \sum_{j=0}^{n} s_{ji} y_{ji}) + 1.$

   This valid inequality ensures that the time period allocated to last dummy order should be later than the release date of the order which is released earlier than all others plus the processing times of all accepted orders plus all setup times which should be between all consecutive orders.

2. $\sum_{t=0}^{H+1} x_{it} \geq (r_i + 1) I_i + \sum_{j=0}^{n} s_{ji} y_{ji}, \qquad \forall i = 1, ..., n.$

   This inequality implies that no order can be processed earlier than its release time plus the setup time between its preceding order and it.

3. $\sum_{i=0}^{n+1} \sum_{s=max(t-p_i+1,1)}^{t} x_{is} \leq 1, \qquad t = 1, .., H.$

   This inequality is an easier case of the constraint set 4.3. With this inequality, the problem needs to obey just processing times, which is a very obvious requirement. However, this inequality prevents $x_{it}$ values splitting into time periods that are distant, because unlike constraint set 4.3, this inequality works with time periods one by one, not with the summation of $t.x_{it}$ values.

We present the improvements achieved by employing the proposed valid inequalities in Tables 4.1-4.8, and analyze the improved cases in detail in Section 4.6.3, where we describe the computational experiments.

## *4.4 Lower Bounds*

At first, we coded MILP2, and tried to solve it as a Mixed Integer Linear Programming problem. However, it is observed that CPLEX is capable of solving only small size OAS problem instances via main model in reasonable time limits. For industrial-size problems, we need to develop other methods that efficiently solve the problem requiring considerably small CPU times.

### *4.4.1 Heuristic based on LP relaxation (HLP)*

Below we present simple heuristic developed for the OAS problem. Our heuristic algorithm is as belows:

**Input**: parameters of an instance
**Output**: lower bound value for that instance
1 initialization;
2 Relax all $x_{jt}$ variables.
3 Relax all $I_j$ variables.
4 Solve MILP2.
5 Take all $y_{ij}$ optimal values.
6 With those $y_{ij}$ values, construct a new schedule.

**Algorithm 2:** Heuristic based on LP relaxation (HLP)

This simple heuristic works very well, and almost all the solutions we obtain by it are optimal solutions. However, although we relax all $x_{jt}$ and $I_j$ variables, we keep all $y_{ij}$ variable as binary, and it still takes too much time to solve large instances.

### *4.4.2 A Revised Mathematical Model (MILP3)*

In the problem $1 \,|\, d_j, e_j, w_j \,|\, \sum R_j$, we do not use any variable $y_{ij}$ to care about two consecutive orders. For the problem $1 \,|\, r_j, d_j, \bar{d}_j, e_j, w_j, s_{ij} \,|\, \sum R_j$, we have to change MILP2 so that no $y_{ij}$ is used. In such a revised model, all constraints related to $y_{ij}$ variables would be removed accordingly. Therefore, the constraint sets 4.3, 4.4, 4.7, and 4.8 would be removed from MILP2, if we can write a constraint such that we do not violate setup times and release dates. In this case, we can remove $(n+2)^2$ variables, and a huge number of constraints. We develop the following constraint to do so:

$$x_{jt} + \sum_{u=t}^{\min(\max(t+p_j+s_{ji}-1, r_i+s_{ji}), H+1)} x_{iu} \leq 1 \qquad \forall i = 1, ..., n; j = 0, ..., n; i \neq j \qquad (4.13)$$

Therefore, we can write a revised mathematical model (MILP3) for the $1 \,|\, r_j, d_j, \bar{d}_j, e_j, w_j, s_{ij} \,|\, \sum R_j$

problem as follows:

$$max \sum_{i=1}^{n} R_i$$

s.t.

$$x_{jt} + \sum_{u=t}^{\min(\max(t+p_j+s_{ji}-1, r_i+s_{ji}), H+1)} x_{iu} \leq 1 \qquad \forall i = 1, ..., n; j = 0, ..., n; i \neq j \tag{4.13}$$

Constraints 4.1, 4.2, 4.5, 4.6, 4.9, 4.10, 4.11, and

$$x_{it} = \{0, 1\}, I_i = \{0, 1\} \qquad i \in O;\ t = 0, .., H + 1$$

The constraint set 4.13 ensures not to violate release dates and setup times, but it may cut some feasible solutions. Therefore, we need to find the cases in which this constraint does not cut any feasible solution, and this revised model can be used to solve the model to optimality. To find such cases, we need to discuss about two different cases related to the expression $\max(t + p_j + s_{ji} - 1, r_i + s_{ji})$.

*Case 1: if* $\max(t + p_j + s_{ji} - 1, r_i + s_{ji}) = t + p_j + s_{ji} - 1$*:* Assume that three orders $j$, $k$, and $i$ are in the optimal schedule in this order. Our new constraint may cause an unnecessary delay in processing of order $i$, if:

$$p_j + s_{ji} > p_j + s_{jk} + p_k + s_{ki}, \tag{4.14}$$

$$\implies \qquad s_{ji} - s_{jk} - s_{ki} > p_k. \tag{4.15}$$

To avoid making an unnecessary delay in processing of order $i$, we need to set our parameters so that

$$s_{ji} - s_{jk} - s_{ki} < p_k. \tag{4.16}$$

Also assume that four orders $j$, $k$, $m$, and $i$ are in the optimal schedule in this order. Our new constraint may again cause an unnecessary delay in processing of order $i$, if:

$$p_j + s_{ji} > p_j + s_{jk} + p_k + s_{km} + p_m + s_{mi}, \tag{4.17}$$

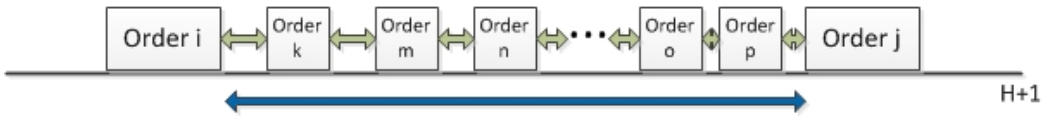


Figure 4.1: A simple instance

$$\Longrightarrow \qquad s_{ji} - s_{jk} - s_{mi} - s_{km} > p_k + p_m. \tag{4.18}$$

To avoid making an unnecessary delay in processing of order $i$, we need to set our parameters such that

$$s_{ji} - s_{jk} - s_{mi} - s_{km} < p_k + p_m. \tag{4.19}$$

Accordingly, if we want the new constraint not to cut any feasible solution, we need to consider all permutations of all possible subsets of all orders, and set our parameters such that the setup time between the first and the last order of each permutation is less than summation of all processing times and setup times between those two orders. We developed a program tool by which this constraint can be checked after data parameters of an instance are uploaded to the program. We illustrate this case in Figure 4.1, where orders $i$ and $j$ are placed in the beginning and the end of a permutation of a subset of all orders, respectively, and many orders are in between orders $i$ and $j$. All green two headed arrows show the setup times between two consecutive orders, and the length of each rectangular shows the processing time of each order. The blue two headed arrow shows summation of all processing times and setup times between those two orders $i$ and $j$.

*Case 2: if* $\max(t + p_j + s_{ji} - 1, r_i + s_{ji}) = r_i + s_{ji}$*:* Assume that three orders $j$, $k$, and $i$ are in the optimal schedule in this order. Let $ST_j$ denotes the starting time of order $j$. Our new constraint may cause a delay in processing of order $i$, if:

$$ST_j + r_i + s_{ji} > max\ (ST_k + s_{ki} + p_k - 1, r_i + s_{ki}). \tag{4.20}$$

To avoid this condition, we need to set our parameters so that:

$$ST_j + r_i + s_{ji} \leq max\ (ST_k + s_{ki} + p_k - 1, r_i + s_{ki}), \tag{4.21}$$

$$\Longrightarrow \qquad ST_j + r_i + s_{ji} \leq ST_k + s_{ki} + p_k - 1 \quad or \quad ST_j + r_i + s_{ji} \leq r_i + s_{ki}. \tag{4.22}$$

We cannot set anything to make sure about $ST_j + r_i + s_{ji} \leq r_i + s_{ki}$, so we have to force $ST_j + r_i + s_{ji} \leq ST_k + s_{ki} + p_k - 1$ to be true. Therefore:

$$ST_j + r_i + s_{ji} \leq ST_k + s_{ki} + p_k - 1), \tag{4.23}$$

$$\Longrightarrow \qquad r_i < ST_k - ST_j + p_k + s_{ki} - s_{ji} - 1. \tag{4.24}$$

We do not know the starting time of order $k$ and order $j$, so we need to use the minimum of $ST_k - ST_j$. Therefore, the condition becomes:

$$r_i < p_j + s_{jk} + p_k + s_{ki} - s_{ji} - 1. \tag{4.25}$$

As a conclusion, if all parameters of an instance meet the constraint which is explained in Figure 4.1, as well as constraint 4.25, it can be solved to optimality by using MILP3.

We examined all instances to see if we can solve them to optimality using MILP3, but unfortunately we found out that just a few instances out of 250 instances meet those conditions.

Since MILP3 keeps the feasibility of the solutions, solving it, without regarding to the conditions, can give us a lower bound for the main problem. Therefore, we solved it for all small instances, and not surprisingly, we observed that the solution for most of the instances is near or exactly the optimal solution. MILP3 can be used to find good lower bounds for instances of 10 and 15-order, but for larger instances, it needs too much time to be solved, and the optimality gap is considerably increased.

## *4.5 Second Problem Definition for* $1 \,|\, r_j, d_j, \bar{d}_j, e_j, w_j, s_{ij} \,|\, \sum R_j$

In this part, we change our main variable $x_{it}$ so that we model the problem differently. In the previous model, $x_{it}$ shows the first time period in which we start to process order $i$. In the new model, $x'_{it}$ is defined as the time period in which we process order $i$, not just the first period. To illustrate, let's assume that order 3 is supposed to be processed from time period 5 to time period 9. In the previous mathematical model, $x'_{35} = 1$ and $x'_{3t} = 0 \;\; \forall t | t \neq 5$. However, in the new model, $x'_{3t} = 1 \;\; \forall t = 3, ..., 9$ and $x_{3t} = 0 \;\; \forall t = 0, ..., 4 \quad \text{and} \quad t = 10, ..., H + 1$.

### *4.5.1 Mathematical Model (MILP4)*

Using mixed integer programming, we modeled the problem as the following:

# Indices:

$i, j$ : orders.

$t$ : time periods.

# Sets:

$O$ : set of all original orders.

$S$ : set of selected orders.

$\sigma_S$ : processing sequence of selected order set $S$.

# Parameters:

$H$ : number of main time periods.

$n$ : number of original orders.

$r_i$ : release date of order $i \quad i \in O$.

$p_i$ : processing time of order $i \quad i \in O$.

$d_i$ : due date of order $i \quad i \in O$.

$\bar{d}_i$ : deadline of order $i \quad i \in O$.

$e_i$ : maximum revenue of order $i \quad i \in O$.

$w_i$ : importance weight of order $i$ $i \in O$.

$s_{ij}$ : sequence dependent setup time between two tandem orders $i$ and $j$ $i, j \in O$.

## Decision Variables:

$T_i$ : total tardiness of order $i$, $i \in S$.

$R_i$ : revenue gained from order $i$, $i \in S$.

$$x'_{it} = \begin{cases} 1 & \text{If order } i \text{ is processed at time period } t, \\ 0 & \text{Otherwise.} \end{cases}$$

$$I_i = \begin{cases} 1 & \text{If order } i \text{ is accepted,} \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{If order } j \text{ is processed immediately after order } i, \\ 0 & \text{Otherwise.} \end{cases}$$

$$F_{ijt} = \begin{cases} 1 & \text{If order } i \text{ is processed at time period } t, \text{ and order } j \text{ is processed immediately after } i, \\ 0 & \text{Otherwise.} \end{cases}$$

## MILP4:

$$max \sum_{i=1}^{n} R_i$$

s.t.

$$\sum_{t=r_j+1}^{\bar{d}_j} x'_{jt} = p_j I_j \qquad j \in O \tag{4.26}$$

$$\sum_{j=1}^{n} x'_{jt} \leq 1 \qquad t = 0, ..., H+1 \tag{4.27}$$

$$\sum_{i=1, i\neq j}^{n+1} y_{ji} = I_j \qquad j = 0, ..., n \tag{4.28}$$

$$\sum_{i=0, i\neq j}^{n} y_{ij} = I_j \qquad j = 1, ..., n+1 \tag{4.29}$$

$$y_{ij} + x'_{it} \leq 1 + F_{ijt} \qquad i = 0, ..., n; j = 1, ..., n+1; \quad i \neq j; t = 0, ..., H+1 \tag{4.30}$$

$$x'_{j(t+s_{ij})} \leq (s_{ij}+1)(1 - F_{ijt}) \qquad i = 0, ..., n; j = 1, ..., n+1; \quad i \neq j; t = 0, ..., H+1-s_{ij} \tag{4.31}$$

$$\sum_{b=t-p_j+1}^{t+p_j-1} x'_{jb} \geq p_j . \sum_{i=1}^{n} F_{jit} \qquad j \in O; t = p_j - 1, ..., H - p_j + 2 \tag{4.32}$$

$$T_j \geq t x'_{jt} - d_j \qquad j \in O; t = 1, ..., H \tag{4.33}$$

$$I_0 = 1, \;\; I_{n+1} = 1 \tag{4.34}$$

$$x'_{00} = 1, \;\; x'_{n+1,H+1} = 1 \tag{4.35}$$

$$x'_{jt} \leq 1 - y_{ij} \qquad i = 0, ..., n; j = 1, ..., n+1; \quad i \neq j; t = 0, ..., r_j + s_{ij} \tag{4.36}$$

$$R_i = e_j I_j - w_j T_j \qquad j \in O \tag{4.37}$$

$$\frac{\sum_{t=0}^{H+1} t x'_{it}}{p_i} \leq \frac{\sum_{t=0}^{H+1} t x'_{jt}}{p_j} + M(1 - y_{ij}) \qquad i = 0, ..., n; j = 1, ..., n+1; i \neq j \tag{4.38}$$

$$x'_{it} = \{0,1\}, I_i = \{0,1\}, y_{ij} = \{0,1\}, \quad F_{ijt} = \{0,1\}, T_i \geq 0 \qquad i, j \in O; \; t = 0, .., H+1 \tag{4.39}$$

Constraint set 4.26 implies that the number of time period assigned to a certain accepted order should be between its release date and deadline, and they must be equal to its processing time. Constraint set 4.27 prohibits assigning more than one order to a certain time period. Constraints sets 4.28 and 4.29 imply that only one order can be after or before a certain order which is selected to be processed. By constraints sets 4.30 and 4.31, we want to show that if orders $i$ and $j$ are supposed to be processed in tandem, and if time period $t$ is assigned to order $i$, no time period between $t$ and $t + s_{ij}$ can be assigned to order $j$. Constraint set 4.32 ensures that if order $j$ is accepted to the schedule, all time periods assigned to this order, should

be in tandem. Constraint set 4.33 calculates tardiness time for each order so that if the order is finished before its due date, its tardiness would be greater than a negative value, according to constraint set 4.5, and since all tardiness values should be nonnegative, according to its tardiness would be zero. Constraints 4.34 set the acceptance of dummy orders. Constraints 4.35 assign the first and the last dummy time periods to dummy orders. Constraint set 4.36 implies that if order $j$ is assumed to be processed immediately after order $i$, it cannot be started earlier than its release date plus the setup time between orders $i$ and $j$. By constraint set 4.37, the model can calculate the revenue for each order, by subtracting the tardiness penalty, if there is any, from maximum revenue of the order. Constraint set 4.38 eliminates all subtours by implying that if $y_{ij}$ is equal to 1, the time periods assigned to order $j$ must be later than those assigned to order $i$. Constraint set 4.39 defines the domain of the variables.

Due to large number of variables and constraints in MILP4, it cannot be solved to optimality in a reasonable time even for small instances, but there may be future studies on MILP4 to improve it.

## *4.6 Computational Study*

We performed computational experiments to analyze the performance of the proposed mathematical models and valid inequalities as well as our simple heuristic. In this section, we first describe the random data generation phase of the experiments. For this problem, we use the benchmark instances generated by Oguz et al. [43] for 10, 15, 20, 25, 50 and 100-order instances.

### *4.6.1 Data Generation (Oguz et al. [43])*

The following problem parameters are integer numbers which were generated randomly from a uniform distribution in the following intervals; release dates $r_i : [0, p_T \times \tau]$ where $p_T$ is the total processing time of all orders and $\tau$ is the tardiness factor, processing times $p_i : [1, 20]$, sequence dependent setup times $s_{ij} : [1, 10]$, and revenues $e_i : [1, 20]$. The due dates are generated so that for each order $i$, due date $d_i$ is in the interval of

$(p_T + \max_j s_{ij} + r_i) \times [1 - \tau - \frac{R}{2}, 1 - \tau + \frac{R}{2}]$ and integer where $R$ represents the due date range respectively. The deadlines are generated from the formula $\bar{d_i} = d_i + R \times p_i$. The weight $w_i$ is calculated by the formula $w_i = \frac{e_i}{(\bar{d_i} - d_i)}$. In this setting, revenue gained from an order $i$ becomes 0 at its deadline $\bar{d_i}$. The values used for $\tau$ and $R$ are $0.1, 0.3, 0.5, 0.7$ and $0.9$. For each possible combination of $\tau$ and $R$, 10 instances were generated and the average results are reported. In this study, $\tau = 0.1, 0.3, 0.5$ have been found compatible with $R = 0.1, 0.3, 0.5, 0.7, 0.9$, and a total of $3 \times 5 \times 10 = 150$ test instances are generated for these parameter combinations for each $n$. Similarly, $\tau = 0.7$ is compatible with $R = 0.5, 0.7, 0.9$, and hence $3 \times 10 = 30$ instances are generated for each $n$. Finally, $\tau = 0.9$ is compatible only with $R = 0.9$, and therefore 10 instances are generated for each $n$ for $\tau = 0.9, R = 0.9$ combination. Resultantly, the number of the test instances for each $n$ is 190, which sums up to 1140 for 6 different $n$ values where $n = 10, 15, 20, 25, 50, 100$. The results of the computational experiment are presented in next sections.

### *4.6.2 Computational Platform*

All of the runs throughout these computational experiments are performed on a laptop with an Intel Xeon processor, 2.70 Ghz speed, and 8 GB of RAM. In MILP and MILP with valid inequalities runs, we have used ILOG CPLEX 12.5.1.0. Time limit is 3600 seconds for total CPU time for CPLEX runs. All of the solution methods developed within the scope of this thesis are coded in CPLEX.

### *4.6.3 Upper Bounds*

In Section 4.3, we proposed two methods for finding upper bounds, but as we explained, the objective function values obtained by Lagrangian Relaxation method are simply summation of all revenues which are useless and loose upper bounds. Therefore, we use the following methods to find upper bounds and compare them with previous studies:

1. $UB_{HLP}$: All variables $x_{jt}$ and $I_j$ are relaxed, but $y_{ij}$ variables are remained

as binary yet. Then, MILP2, explained in Section 4.2, is solved by branch-and-bound algorithm. After CPU time of 3600 seconds, the running process is stopped, and the upper bound is considered as $UB_{HLP}$. This is exactly what we showed in the first three steps of our heuristic based on LP relaxation (HLP), explained in Section 4.4.1.

2. $UB_{LPVI}$: All integer and binary variables, in MILP2, explained in Section 4.2, are relaxed. Then the second constraint in the constraint set 4.11 is removed, and three valid inequalities, explained in Section 4.3.2, are added to the problem. Then, it is solved to optimality, and the objective function value is considered as $UB_{LPVI}$.

To compare the quality of each upper bound, we use percentage improvement (API) which is calculated as belows:

$$\%\ improvement = \frac{UB - LB_{BEST}}{LB_{BEST}} \times 100. \tag{4.40}$$

In the equation 4.40, $LB_{BEST}$ shows the maximum lower bounds found so far for each instance. Many methods are constructed to solve OAS problem up to now, and we will discuss them in Section 4.6.4.

For each $(n, \tau, R)$ parameter combination, we solved 10 instances of the OAS problem, generated randomly as explained in Section 4.6.1. In Tables 4.1 and 4.3, we compare the upper bounds obtained by relaxing $x_{jt}$ and $I_j$ variables in MILP ($UB_{HLP}$) and LP relaxation bound strengthened with valid inequalities ($UB_{LPVI}$) for 10- and 15-order instances. In 10-order instances, according to Table 4.1, the objective function value of all heuristic solutions are equal to $UB_{HLP}$, which means that all of them are solved to optimality in CPU time of less than one hour so that $UB_{HLP}$ is equal to the optimal solution of MILP. It is also observed that as $\tau$ value grows, the percentage improvement of the LP model with valid inequalities increases. According to Table 4.3, HLP gives the optimal solutions for $\tau = 0.7$ and 0.9. For $\tau = 0.1$, 0.3, and 0.5, as $\tau$ increases, the quality of HLP solution decreases. $UB_{LPVI}$

has less potential as $\tau$ increases. All in all, for $\tau = 0.1$, 0.3, and 0.5, HLP and the LP model strengthened with valid inequalities work the same in aspect of solution quality, but for $\tau = 0.7$ and 0.9, HLP gives the optimal solution while LPVI gives very loose bounds with gap percentages of 20 to 30.

For instances larger than 15 orders, the third step of HLP cannot be solved to optimality in a reasonable time, and accordingly, the solution obtained by HLP is not optimal. So we have to stop the running process after CPU time of one hour which gives very bad upper bounds ($UB_{HLP}$). Therefore, for instances with $n \geq 15$, HLP and also $UB_{HLP}$ are not used any more, and we use LP bound to which valid inequalities are added.

Tables 4.2, 4.4, 4.5, 4.6, 4.7, and 4.8 show the comparison of the upper bounds found in this thesis and the previous upper bounds obtained by Ceyda Oguz et al. [43]. For 10- and 15-order instances, $UB_{NEW}$ is the minimum of $UB_{HLP}$ and $UB_{LPVI}$. For 20, 25, 50 and 100-order instances, $UB_{NEW}$ is equal to $UB_{LPVI}$. In all those tables, $UB_{PREVIOUS}$ shows the upper bound calculated by Ceyda Oguz et al. [43].

As Table 4.2 shows, $UB_{NEW}$ gives the optimal solution in all $\tau$ and $R$ values. $UB_{PREVIOUS}$ is also equal to the optimal solution in instances with $\tau = 0.9$. In other $\tau$ values, it doesn't always give the optimal solution, but its gap is a very small value.

According to Tables 4.2, 4.4, 4.5, 4.6, 4.7, and 4.8 for $\tau = 0.1$, 0.3, and 0.5, in smaller $R$ values, our LP model with the valid inequalities becomes more different than previous upper bounds. As $R$ value grows, the quality of $UB_{NEW}$ becomes more close to $UB_{PREVIOUS}$. For $\tau = 0.7$, and 0.9, both LPVI model and previous studies give nearly equal solutions. In average, the percentage improvement of $UB_{NEW}$ over $LB_{BEST}$ is better than $UB_{PREVIOUS}$ for all sizes of instances.

Figure 4.2 shows the difference between our upper bounds with previous ones. It is observed that as the size of instances grows, our LPVI model works better than previous methods to obtain upper bounds. The reason for such improvement is related to very good approximations of time-indexed LP problems, as discussed in J.M. van den Akker et al [6], as well as the potency of valid inequalities added to the LP

problem.

Figure 4.3 shows in how many instances our upper bound is less than, equal to, or greater than previous one. In all 10-order instances, $UB_{NEW} = UB_{LPS}$, because LPS can solve all 10-order instances to in CPU time of less than one hour, and its solution is optimal for all of them so that in 150 instances, it can defeat previous upper bounds. After that, number of improvements is decreased, because obtaining optimal solution by MILP2 is not possible any more. From 15-order instances, as the size of instances grows, the number of better upper bounds increases on average.

### *4.6.4 Lower Bounds*

In the literature, Oguz et al. [43] gave a mixed-integer linear programming formulation of the problem $1 \mid r_j, d_j, e_j, w_j, s_{ij} \mid \sum R_j$, which can solve instances with up to 15-order to optimal. In terms of approximate algorithms, Cesaret et al. [45] proposed a tabu search algorithm supported with a probabilistic local search procedure for this problem. Lin and Ying [46] presented an artificial bee colony algorithm combined with an iterated greedy heuristic for exactly the same problem. Chen et al. [58] proposed an improved genetic algorithm with local search (IGAL) for order acceptance and scheduling problem. Then in [47], they proposed a diversity controlling genetic algorithm (DCGA) in which a diversified population is maintained during the whole search process through survival selection considering both the fitness and the diversity of individuals. Among all previous studies explained, we selected Tabu Search ($TS$) presented by Cesaret et al. [45], and the diversity controlling genetic algorithm ($DCGA$) constructed by Chen et al. [47], and we compare our result with the results of these two methods, because they obtain best solutions so far.

#### *4.6.4.1 Heuristic based on LP relaxation (HLP)*

Using LP relaxation, a heuristic method is constructed to reduce solving time. The algorithm of this method is explained in Section 4.4.1. Although this method's solution is equal to the optimal solution most of the times, the time needed for solving the

problem is too much for problems larger than 10 orders. Therefore, for problems with $n > 10$, the running process must be stopped after CPU time of one hour, which results in a bad solution which is not comparable with the results of previous studies.

The first part of this computational study shows the performance of our heuristic for n = 10. The maximum, minimum, and average percentage deviation of the our heuristic objective function value from the best upper bound found so far ($UB_{BEST}$) in Table 4.9 and the corresponding CPU times for our heuristic algorithm are presented in Table 4.10.

The results of the experiments in Table 4.10 confirm that, as $\tau$ increases, the problem gets less time consuming. According to Table 4.9, HLP can solve all 10-order instances to optimality. However, its running time is too much, specially for instances with $\tau = 0.1$, and 0.3, according to Table 4.10. It is true because orders with $\tau = 0.1$, and 0.3 are unlikely to be tardy in the optimal schedule, so the rejection decision is not easy. In addition, when we keep $\tau$ constant, and increase R, we see that the problem becomes easier to be solved by our heuristic for $\tau = 0.1$, and 0.3, since as R grows, the due dates of the orders become looser, and this brings a scheduling flexibility to the system which makes the OAS problem easier. For instances with $\tau = 0.5$, the running time is about 70 seconds in average, which a good and reasonable time. When we go to instances with $\tau = 0.7$, and 0.9, all instances can be solved in average time of 2 seconds.

Overall we can say that our heuristic is an efficient solution method for the OAS problem only for small-size instances as its average percentage deviation from the best upper bound is zero in all cases of 10-order instances.

#### *4.6.4.2 Revised Mathematical Model (MILP3)*

As explained in Section 4.4.2, we made a revised mathematical model by changing our mathematical model (MILP2). In Section 4.4.2, we showed that MILP3 works properly only in some conditions, but not all. We found those conditions, and we explained that not many instances meet those conditions. However, since MILP3

does not add any infeasible solution to the feasible solution space, it can be used as a method to find lower bounds.

We ran all 10-order instances with the revised mathematical model, and we published the result in Tables 4.9 and 4.10. According to Table 4.9, as $\tau$ grows, the optimality gap of MILP3 over $UB_{BEST}$ also increases. This is true because as, $\tau$, the tardiness factor, increases, the time window for each order gets narrower, and hence the probability of being tardy increases for each order. Therefore, for $\tau = 0.5$, 0.7, and 0.9, the conditions, explained in Section 4.4.2, causes rejection of some orders and change the place of some order while it is not necessary in reality. It shows that those conditions are much more effective when $\tau$ increases. Table 4.10 shows that CPU time needed for MILP3 is 9.6 seconds in average, which is a very acceptable time. Also it is observed that as $\tau$ increases, CPU time is decreased, because as $\tau$ increases, the probability of being tardy is also increased, and so the running process can decide on scheduling and acceptance easier.

#### *4.6.4.3 Conclusions on Computational Experiments*

According to Tables 4.9 and 4.10, in $\tau = 0.1$, DCGA and HLP give the optimal solution. MILP3 has a small optimality gap value which is much better than Tabu Search algorithm. However, among all, Tabu Search solves much faster. DCGA and MILP3 have small running times, and HLP needs too much time to solve the problem. Generally, DCGA is the best one, and MILP3 be in the second place.

In $\tau = 0.3$, DCGA and HLP again give the optimal solution, but here, MILP3 and Tabu Search algorithm both give small and almost equal optimality gaps. In aspect of time, the situation is very similar to $\tau = 0.1$. Therefore, the best suggestion is using DCGA at the first place, and then, Tabu Search algorithm.

In $\tau = 0.5$, DCGA and HLP give the optimal solution as always. MILP3 has optimality gaps less than Tabu Search. In terms of CPU time, Tabu Search is the best as always, and all other three methods have the same situations. All in all, DCGA performs the best, and MILP3 has the second rank.

In $\tau = 0.7$, Tabu search performs better than MILP3, while DCGA and HLP solve all problems to optimality. Tabu Search is again the fastest method, and all others have almost the same running time. The general suggestion is using DCGA or HLP.

In $\tau = 0.9$, Tabu Search, HLP, and DCGA give the optimal solution, while MILP3 performs vary bad. In aspect of time, the situation is similar to $\tau = 0.7$. Therefore, the best choice is Tabu Search here.

Figure 4.4 shows how different methods work for different $\tau$ and $R$ combinations in aspect of optimality gaps. In the figure, the optimality gap is calculated by taking the average of all optimality gaps for all 10-order instances in each different $\tau$ and $R$. To observe the difference in aspect of CPU time, Figure 4.5 is added as well. It can be observed that all in all, DCGA performs very well and much better than others in both aspects of average optimality gaps and CPU times.

Table 4.1: The API of $UB_{HLP}$ and $UB_{LPVI}$ over $LB_{BEST}$ for 10-order instances.

| $\tau$ | R | $UB_{HLP}$ | | | $UB_{LPVI}$ | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 0 | 0 | 0 | 4 | 0 | 2 |
| | 0.3 | 0 | 0 | 0 | 8 | 0 | 3 |
| | 0.5 | 0 | 0 | 0 | 6 | 0 | 3 |
| | 0.7 | 0 | 0 | 0 | 12 | 0 | 3 |
| | 0.9 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0.3 | 0.1 | 0 | 0 | 0 | 7 | 3 | 5 |
| | 0.3 | 0 | 0 | 0 | 13 | 2 | 6 |
| | 0.5 | 0 | 0 | 0 | 9 | 2 | 5 |
| | 0.7 | 0 | 0 | 0 | 24 | 3 | 9 |
| | 0.9 | 0 | 0 | 0 | 19 | 0 | 7 |
| 0.5 | 0.1 | 0 | 0 | 0 | 13 | 0 | 7 |
| | 0.3 | 0 | 0 | 0 | 26 | 4 | 12 |
| | 0.5 | 0 | 0 | 0 | 17 | 0 | 9 |
| | 0.7 | 0 | 0 | 0 | 20 | 1 | 11 |
| | 0.9 | 0 | 0 | 0 | 33 | 2 | 12 |
| 0.7 | 0.1 | 0 | 0 | 0 | 20 | 7 | 13 |
| | 0.3 | 0 | 0 | 0 | 23 | 6 | 13 |
| | 0.5 | 0 | 0 | 0 | 31 | 7 | 17 |
| | 0.7 | 0 | 0 | 0 | 38 | 11 | 19 |
| | 0.9 | 0 | 0 | 0 | 36 | 3 | 15 |
| 0.9 | 0.1 | 0 | 0 | 0 | 39 | 6 | 24 |
| | 0.3 | 0 | 0 | 0 | 48 | 8 | 25 |
| | 0.5 | 0 | 0 | 0 | 47 | 4 | 25 |
| | 0.7 | 0 | 0 | 0 | 52 | 17 | 31 |
| | 0.9 | 0 | 0 | 0 | 37 | 14 | 25 |
| Avg | | 0.00 | 0.00 | 0.00 | 23.45 | 3.96 | 12.05 |

Table 4.2: The API of $UB_{NEW}$ and $UB_{PREVIOUS}$ over $LB_{BEST}$ for 10-order instances.

| $\tau$ | R | $UB_{NEW}$ | | | $UB_{PREVIOUS}$ (Oguz et al. [43]) | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 0.00 | 0.00 | 0.00 | 0.10 | 0.09 | 0.09 |
| | 0.3 | 0.00 | 0.00 | 0.00 | 0.10 | 0.09 | 0.10 |
| | 0.5 | 0.00 | 0.00 | 0.00 | 6.38 | 0.00 | 0.71 |
| | 0.7 | 0.00 | 0.00 | 0.00 | 0.10 | 0.00 | 0.06 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.09 | 0.00 | 0.02 |
| 0.3 | 0.1 | 0.00 | 0.00 | 0.00 | 0.10 | 0.08 | 0.09 |
| | 0.3 | 0.00 | 0.00 | 0.00 | 0.12 | 0.09 | 0.09 |
| | 0.5 | 0.00 | 0.00 | 0.00 | 0.10 | 0.03 | 0.08 |
| | 0.7 | 0.00 | 0.00 | 0.00 | 0.10 | 0.08 | 0.09 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.10 | 0.00 | 0.07 |
| 0.5 | 0.1 | 0.00 | 0.00 | 0.00 | 0.09 | 0.00 | 0.06 |
| | 0.3 | 0.00 | 0.00 | 0.00 | 0.10 | 0.01 | 0.07 |
| | 0.5 | 0.00 | 0.00 | 0.00 | 0.10 | 0.00 | 0.06 |
| | 0.7 | 0.00 | 0.00 | 0.00 | 0.10 | 0.00 | 0.07 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.10 | 0.00 | 0.08 |
| 0.7 | 0.1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 0.3 | 0.00 | 0.00 | 0.00 | 0.01 | 0.00 | 0.00 |
| | 0.5 | 0.00 | 0.00 | 0.00 | 12.86 | 0.00 | 1.29 |
| | 0.7 | 0.00 | 0.00 | 0.00 | 0.08 | 0.00 | 0.02 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.07 | 0.00 | 0.02 |
| 0.9 | 0.1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 0.3 | 0.00 | 0.00 | 0.00 | 0.02 | 0.00 | 0.00 |
| | 0.5 | 0.00 | 0.00 | 0.00 | 0.01 | 0.00 | 0.00 |
| | 0.7 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.01 | 0.00 | 0.00 |
| Avg | | 0.00 | 0.00 | 0.00 | 0.83 | 0.02 | 0.12 |

Table 4.3: The API of $UB_{HLP}$ and $UB_{LPVI}$ over $LB_{BEST}$ for 15-order instances.

| $\tau$ | R | $UB_{HLP}$ | | | $UB_{LPVI}$ | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 3 | 0 | 1 | 3 | 0 | 1 |
| | 0.3 | 5 | 0 | 2 | 5 | 0 | 2 |
| | 0.5 | 3 | 0 | 1 | 3 | 0 | 1 |
| | 0.7 | 4 | 0 | 1 | 4 | 0 | 1 |
| | 0.9 | 6 | 0 | 1 | 6 | 0 | 1 |
| 0.3 | 0.1 | 4 | 0 | 2 | 4 | 0 | 2 |
| | 0.3 | 11 | 0 | 4 | 11 | 0 | 4 |
| | 0.5 | 7 | 1 | 4 | 7 | 1 | 4 |
| | 0.7 | 8 | 1 | 3 | 8 | 1 | 3 |
| | 0.9 | 8 | 0 | 3 | 8 | 0 | 3 |
| 0.5 | 0.1 | 10 | 3 | 6 | 10 | 3 | 6 |
| | 0.3 | 16 | 3 | 7 | 16 | 3 | 7 |
| | 0.5 | 17 | 4 | 10 | 15 | 4 | 9 |
| | 0.7 | 11 | 0 | 6 | 11 | 4 | 7 |
| | 0.9 | 23 | 0 | 7 | 22 | 3 | 8 |
| 7 | 0.1 | 0 | 0 | 0 | 15 | 2 | 9 |
| | 0.3 | 0 | 0 | 0 | 23 | 6 | 11 |
| | 0.5 | 0 | 0 | 0 | 24 | 7 | 15 |
| | 0.7 | 9 | 0 | 1 | 29 | 5 | 16 |
| | 0.9 | 0 | 0 | 0 | 22 | 4 | 13 |
| 0.9 | 0.1 | 0 | 0 | 0 | 39 | 19 | 29 |
| | 0.3 | 0 | 0 | 0 | 49 | 18 | 28 |
| | 0.5 | 0 | 0 | 0 | 40 | 9 | 23 |
| | 0.7 | 0 | 0 | 0 | 29 | 11 | 19 |
| | 0.9 | 0 | 0 | 0 | 45 | 9 | 21 |
| Avg | | 5.81 | 0.46 | 2.36 | 17.93 | 4.31 | 9.79 |

Table 4.4: The API of $UB_{NEW}$ and $UB_{PREVIOUS}$ over $LB_{BEST}$ for 15-order instances.

| $\tau$ | R | $UB_{NEW}$ | | | $UB_{PREVIOUS}$ (Oguz et al. [43]) | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 3 | 0 | 1 | 3 | 0 | 2 |
| | 0.3 | 5 | 0 | 2 | 7 | 0 | 3 |
| | 0.5 | 3 | 0 | 1 | 3 | 0 | 1 |
| | 0.7 | 4 | 0 | 1 | 4 | 0 | 1 |
| | 0.9 | 6 | 0 | 1 | 6 | 0 | 1 |
| 0.3 | 0.1 | 4 | 0 | 2 | 6 | 0 | 3 |
| | 0.3 | 11 | 0 | 4 | 13 | 0 | 4 |
| | 0.5 | 7 | 1 | 4 | 7 | 1 | 4 |
| | 0.7 | 8 | 1 | 3 | 8 | 1 | 3 |
| | 0.9 | 8 | 0 | 3 | 8 | 0 | 3 |
| 0.5 | 0.1 | 10 | 3 | 6 | 14 | 3 | 8 |
| | 0.3 | 16 | 3 | 7 | 16 | 3 | 8 |
| | 0.5 | 15 | 4 | 9 | 18 | 4 | 10 |
| | 0.7 | 11 | 0 | 6 | 11 | 0 | 6 |
| | 0.9 | 22 | 0 | 7 | 23 | 0 | 7 |
| 7 | 0.1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.3 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.5 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.7 | 9 | 0 | 1 | 9 | 0 | 1 |
| | 0.9 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.9 | 0.1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.3 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.5 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.7 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.9 | 0 | 0 | 0 | 0 | 0 | 0 |
| Avg | | 5.73 | 0.46 | 2.35 | 6.27 | 0.49 | 2.57 |

Table 4.5: The API of $UB_{NEW}$ and $UB_{PREVIOUS}$ over $LB_{BEST}$ for 20-order instances.

| $\tau$ | R | $UB_{NEW}$ | | | $UB_{PREVIOUS}$ (Oguz et al. [43]) | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 2.34 | 0.00 | 1.09 | 2.40 | 0.00 | 1.56 |
| | 0.3 | 2.39 | 0.00 | 1.44 | 2.39 | 0.82 | 1.69 |
| | 0.5 | 3.14 | 0.00 | 1.05 | 3.14 | 0.00 | 1.05 |
| | 0.7 | 1.30 | 0.00 | 0.30 | 1.30 | 0.00 | 0.30 |
| | 0.9 | 2.68 | 0.00 | 0.27 | 2.68 | 0.00 | 0.27 |
| 0.3 | 0.1 | 6.30 | 0.00 | 2.86 | 7.14 | 0.00 | 3.23 |
| | 0.3 | 5.63 | 1.46 | 3.11 | 5.63 | 1.46 | 3.25 |
| | 0.5 | 5.50 | 0.78 | 3.24 | 5.63 | 0.78 | 3.29 |
| | 0.7 | 7.48 | 0.00 | 2.33 | 7.48 | 0.00 | 2.33 |
| | 0.9 | 4.92 | 0.00 | 1.54 | 4.92 | 0.00 | 1.54 |
| 0.5 | 0.1 | 5.08 | 2.11 | 3.26 | 5.08 | 2.11 | 4.02 |
| | 0.3 | 8.28 | 1.44 | 5.28 | 8.28 | 1.44 | 5.38 |
| | 0.5 | 9.18 | 2.11 | 5.68 | 10.14 | 2.11 | 5.87 |
| | 0.7 | 11.88 | 1.17 | 4.63 | 11.88 | 1.17 | 4.63 |
| | 0.9 | 11.82 | 0.00 | 5.93 | 11.82 | 0.00 | 5.93 |
| 0.7 | 0.1 | 14.14 | 2.03 | 8.66 | 15.66 | 2.03 | 8.91 |
| | 0.3 | 14.76 | 0.00 | 7.93 | 13.33 | 6.32 | 9.01 |
| | 0.5 | 23.03 | 0.00 | 9.82 | 25.28 | 0.09 | 9.94 |
| | 0.7 | 12.80 | 0.00 | 6.40 | 13.86 | 0.09 | 7.17 |
| | 0.9 | 14.55 | 0.00 | 8.39 | 14.28 | 0.00 | 7.82 |
| 0.9 | 0.1 | 0.01 | 0.00 | 0.00 | 0.01 | 0.00 | 0.00 |
| | 0.3 | 0.09 | 0.00 | 0.04 | 0.09 | 0.00 | 0.04 |
| | 0.5 | 0.10 | 0.00 | 0.07 | 0.10 | 0.00 | 0.07 |
| | 0.7 | 0.10 | 0.00 | 0.08 | 0.10 | 0.00 | 0.08 |
| | 0.9 | 15.22 | 0.00 | 1.59 | 15.22 | 0.00 | 1.59 |
| Avg | | 7.31 | 0.44 | 3.40 | 7.51 | 0.74 | 3.56 |

Table 4.6: The API of $UB_{NEW}$ and $UB_{PREVIOUS}$ over $LB_{BEST}$ for 25-order instances.

| $\tau$ | R | $UB_{NEW}$ | | | $UB_{PREVIOUS}$ (Oguz et al. [43]) | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 2.76 | 0.73 | 1.47 | 2.76 | 0.73 | 1.71 |
| | 0.3 | 1.97 | 0.69 | 1.20 | 2.80 | 0.69 | 1.46 |
| | 0.5 | 1.66 | 0.00 | 0.72 | 1.66 | 0.00 | 0.72 |
| | 0.7 | 1.01 | 0.00 | 0.19 | 1.01 | 0.00 | 0.19 |
| | 0.9 | 1.06 | 0.00 | 0.17 | 1.06 | 0.00 | 0.17 |
| 0.3 | 0.1 | 3.28 | 0.97 | 2.13 | 3.69 | 0.69 | 2.26 |
| | 0.3 | 4.53 | 1.13 | 2.85 | 4.53 | 1.13 | 2.94 |
| | 0.5 | 3.62 | 0.67 | 1.60 | 3.62 | 0.67 | 1.60 |
| | 0.7 | 4.28 | 0.00 | 1.36 | 4.28 | 0.00 | 1.36 |
| | 0.9 | 2.42 | 0.00 | 0.82 | 2.42 | 0.00 | 0.82 |
| 0.5 | 0.1 | 5.36 | 1.63 | 3.44 | 5.95 | 1.63 | 3.74 |
| | 0.3 | 7.62 | 1.64 | 3.75 | 7.62 | 1.64 | 3.98 |
| | 0.5 | 6.40 | 0.98 | 4.04 | 6.40 | 0.98 | 4.26 |
| | 0.7 | 7.73 | 0.64 | 4.35 | 7.73 | 0.64 | 4.39 |
| | 0.9 | 7.14 | 0.69 | 2.99 | 7.14 | 0.69 | 3.06 |
| 0.7 | 0.1 | 14.29 | 0.93 | 7.29 | 17.46 | 0.93 | 8.09 |
| | 0.3 | 13.24 | 4.94 | 8.73 | 14.15 | 5.32 | 9.13 |
| | 0.5 | 14.58 | 5.24 | 10.07 | 16.32 | 5.68 | 11.19 |
| | 0.7 | 13.74 | 1.72 | 7.42 | 13.74 | 1.61 | 7.06 |
| | 0.9 | 15.35 | 0.73 | 9.03 | 15.79 | 0.01 | 8.99 |
| 0.9 | 0.1 | 5.18 | 0.00 | 0.52 | 5.18 | 0.00 | 0.52 |
| | 0.3 | 0.01 | 0.00 | 0.00 | 0.01 | 0.00 | 0.00 |
| | 0.5 | 14.03 | 0.00 | 2.73 | 14.03 | 0.00 | 2.73 |
| | 0.7 | 26.57 | 0.00 | 7.98 | 26.57 | 0.00 | 7.98 |
| | 0.9 | 23.61 | 0.00 | 6.87 | 23.61 | 0.00 | 6.87 |
| Avg | | 8.06 | 0.93 | 3.67 | 8.38 | 0.92 | 3.81 |

Table 4.7: The API of $UB_{NEW}$ and $UB_{PREVIOUS}$ over $LB_{BEST}$ for 50-order instances.

| $\tau$ | R | $UB_{NEW}$ | | | $UB_{PREVIOUS}$ (Oguz et al. [43]) | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 1.83 | 1.14 | 1.46 | 1.83 | 1.14 | 1.46 |
| | 0.3 | 1.20 | 0.71 | 0.96 | 1.20 | 0.71 | 0.96 |
| | 0.5 | 0.85 | 0.00 | 0.42 | 0.85 | 0.00 | 0.42 |
| | 0.7 | 0.00 | 0.00 | 0.00 | 19.60 | 0.00 | 1.96 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.3 | 0.1 | 2.58 | 1.23 | 2.03 | 2.58 | 1.23 | 2.05 |
| | 0.3 | 2.96 | 1.52 | 2.05 | 2.96 | 1.52 | 2.05 |
| | 0.5 | 3.73 | 0.38 | 1.42 | 3.73 | 0.38 | 1.46 |
| | 0.7 | 0.36 | 0.00 | 0.14 | 0.36 | 0.00 | 0.14 |
| | 0.9 | 1.03 | 0.00 | 0.14 | 1.03 | 0.00 | 0.14 |
| 0.5 | 0.1 | 2.81 | 1.12 | 2.07 | 2.81 | 1.12 | 2.07 |
| | 0.3 | 5.43 | 2.10 | 3.28 | 5.43 | 2.10 | 3.28 |
| | 0.5 | 6.02 | 1.04 | 2.88 | 6.02 | 1.04 | 2.88 |
| | 0.7 | 3.82 | 0.00 | 1.70 | 3.82 | 0.00 | 1.70 |
| | 0.9 | 4.21 | 0.00 | 1.53 | 4.21 | 0.00 | 1.53 |
| 0.7 | 0.1 | 4.66 | 2.48 | 3.73 | 4.66 | 2.48 | 3.84 |
| | 0.3 | 7.87 | 2.66 | 4.71 | 8.43 | 2.66 | 4.77 |
| | 0.5 | 9.78 | 4.77 | 6.43 | 12.18 | 4.77 | 6.71 |
| | 0.7 | 14.34 | 1.39 | 7.04 | 16.93 | 1.39 | 7.41 |
| | 0.9 | 13.02 | 4.63 | 8.10 | 14.94 | 2.85 | 8.09 |
| 0.9 | 0.1 | 13.95 | 7.01 | 10.89 | 20.16 | 7.01 | 12.32 |
| | 0.3 | 20.23 | 11.27 | 14.80 | 21.07 | 10.23 | 15.44 |
| | 0.5 | 19.11 | 5.61 | 12.75 | 19.79 | 2.70 | 14.15 |
| | 0.7 | 20.78 | 8.42 | 13.83 | 21.41 | 5.95 | 12.87 |
| | 0.9 | 20.39 | 9.05 | 14.17 | 18.90 | 10.22 | 14.06 |
| Avg | | 7.24 | 2.66 | 4.66 | 8.60 | 2.38 | 4.87 |

Table 4.8: The API of $UB_{NEW}$ and $UB_{PREVIOUS}$ over $LB_{BEST}$ for 100-order instances.

| $\tau$ | R | $UB_{NEW}$ | | | $UB_{PREVIOUS}$ (Oguz et al. [43]) | | |
|---|---|---|---|---|---|---|---|
| | | Max | Min | Average | Max | Min | Average |
| 0.1 | 0.1 | 1.24 | 0.60 | 0.85 | 1.24 | 0.60 | 0.85 |
| | 0.3 | 1.35 | 0.82 | 1.02 | 1.35 | 0.82 | 1.02 |
| | 0.5 | 0.38 | 0.00 | 0.14 | 0.38 | 0.00 | 0.14 |
| | 0.7 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.3 | 0.1 | 1.50 | 0.69 | 0.98 | 1.84 | 0.69 | 1.18 |
| | 0.3 | 1.28 | 0.66 | 0.97 | 2.61 | 0.66 | 1.33 |
| | 0.5 | 1.23 | 0.45 | 0.94 | 1.72 | 0.45 | 1.16 |
| | 0.7 | 0.58 | 0.00 | 0.20 | 0.58 | 0.00 | 0.20 |
| | 0.9 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.5 | 0.1 | 2.54 | 1.47 | 2.05 | 3.61 | 1.76 | 2.56 |
| | 0.3 | 1.97 | 1.27 | 1.58 | 3.23 | 1.52 | 2.37 |
| | 0.5 | 1.73 | 0.96 | 1.29 | 3.47 | 1.41 | 2.32 |
| | 0.7 | 1.37 | 0.17 | 0.90 | 2.11 | 0.17 | 1.13 |
| | 0.9 | 1.82 | 0.00 | 0.58 | 2.01 | 0.00 | 0.61 |
| 0.7 | 0.1 | 3.66 | 1.78 | 2.46 | 4.40 | 2.41 | 3.21 |
| | 0.3 | 2.96 | 1.35 | 2.10 | 6.67 | 1.88 | 4.08 |
| | 0.5 | 3.09 | 1.66 | 2.28 | 12.92 | 2.21 | 4.95 |
| | 0.7 | 5.53 | 2.06 | 3.75 | 8.07 | 2.41 | 5.34 |
| | 0.9 | 5.78 | 3.01 | 4.58 | 7.92 | 3.11 | 5.62 |
| 0.9 | 0.1 | 5.28 | 3.01 | 4.68 | 9.96 | 4.17 | 6.62 |
| | 0.3 | 10.71 | 5.04 | 7.36 | 13.77 | 5.60 | 10.51 |
| | 0.5 | 13.40 | 7.16 | 8.99 | 19.72 | 9.19 | 12.95 |
| | 0.7 | 12.09 | 6.57 | 8.92 | 15.41 | 8.10 | 12.59 |
| | 0.9 | 14.03 | 4.56 | 9.71 | 20.86 | 4.46 | 12.73 |
| Avg | | 3.74 | 1.73 | 2.65 | 5.75 | 2.06 | 3.74 |

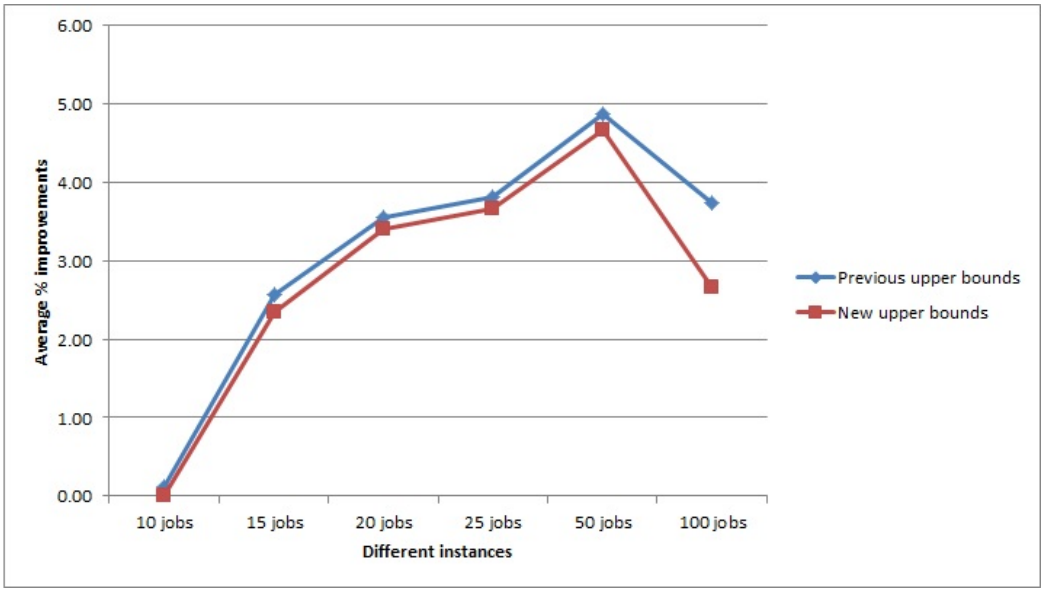


Figure 4.2: Comparison of new and previous upper bounds in aspect of percentage improvements for all sizes of instances

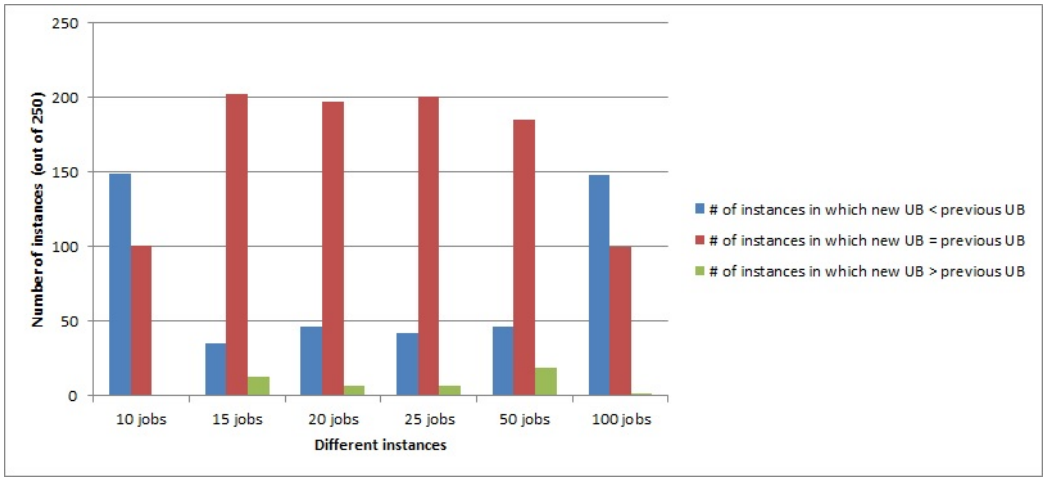


Figure 4.3: Comparison of new and previous upper bounds in aspect of number of improvements for all sizes of instances

Table 4.9: The maximum, the average and the minimum percentage improvements of lower bounds over $UB_{BEST}$ for 10-order instances.

| n=10 | | % Deviations from $UB_{BEST}$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\tau$ | R | DCGA (Chen et al. [47]) | | | TS (Cesaret et al. [45]) | | | *HLP* | | | MILP3 | | |
| | | Max (%) | Avg (%) | Min (%) | Max (%) | Avg (%) | Min (%) | Max (%) | Avg (%) | Min (%) | Max (%) | Avg (%) | Min (%) |
| 0.1 | 0.1 | 0 | 0 | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.3 | 0 | 0 | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.5 | 0 | 0 | 0 | 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.7 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| | 0.9 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.3 | 0.1 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.3 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| | 0.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0.7 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| | 0.9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 |
| 0.5 | 0.1 | 0 | 0 | 0 | 6 | 1 | 0 | 0 | 0 | 0 | 5 | 1 | 0 |
| | 0.3 | 0 | 0 | 0 | 5 | 1 | 0 | 0 | 0 | 0 | 5 | 1 | 0 |
| | 0.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | 0.7 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | 0.9 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |
| 0.7 | 0.1 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 5 | 1 | 0 |
| | 0.3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 2 | 0 |
| | 0.5 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 14 | 1 | 0 |
| | 0.7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 1 | 0 |
| | 0.9 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 6 | 1 | 0 |
| 0.9 | 0.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 4 | 0 |
| | 0.3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 17 | 6 | 0 |
| | 0.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 3 | 0 |
| | 0.7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 3 | 0 |
| | 0.9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 3 | 0 |
| Avg. | | 0 | 0 | 0 | 1.92 | 0.28 | 0 | 0 | 0 | 0 | 5 | 1 | 0 |

Table 4.10: Performance of DCGA, TS, our heuristic (HLP), and our revised mathematical model (MILP3) in aspect of CPU time for 10-order instances.

| $\tau$ | R | Average CPU time (sec) | | | |
|---|---|---|---|---|---|
| | | DCGA (Chen et al. [47]) | TS (Cesaret et al. [45]) | HLP | MILP3 |
| 0.1 | 0.1 | 1.7 | 0.00 | 1745.6 | 21.4 |
| | 0.3 | 1.8 | 0.00 | 1771.9 | 47.5 |
| | 0.5 | 1.8 | 0.00 | 1525 | 19.6 |
| | 0.7 | 1.8 | 0.01 | 1183.1 | 77 |
| | 0.9 | 1.8 | 0.00 | 474.6 | 17 |
| 0.3 | 0.1 | 1.7 | 0.00 | 1091.6 | 7.8 |
| | 0.3 | 1.7 | 0.00 | 964.8 | 6.7 |
| | 0.5 | 1.8 | 0.00 | 800.9 | 6.4 |
| | 0.7 | 1.8 | 0.00 | 957.7 | 11.5 |
| | 0.9 | 1.8 | 0.00 | 397.3 | 4.2 |
| 0.5 | 0.1 | 1.6 | 0.00 | 47.9 | 1.8 |
| | 0.3 | 1.6 | 0.00 | 115.3 | 2.8 |
| | 0.5 | 1.7 | 0.00 | 29 | 2.1 |
| | 0.7 | 1.8 | 0.00 | 66.3 | 2.1 |
| | 0.9 | 1.8 | 0.00 | 80.8 | 2.9 |
| 0.7 | 0.1 | 1.5 | 0.00 | 2.5 | 0.8 |
| | 0.3 | 1.6 | 0.00 | 3.3 | 0.9 |
| | 0.5 | 1.6 | 0.00 | 4.2 | 1 |
| | 0.7 | 1.7 | 0.00 | 3.6 | 0.9 |
| | 0.9 | 1.7 | 0.00 | 4.8 | 1.3 |
| 0.9 | 0.1 | 1.4 | 0.00 | 1.2 | 0.5 |
| | 0.3 | 1.5 | 0.00 | 1.4 | 0.7 |
| | 0.5 | 1.6 | 0.00 | 1.6 | 0.7 |
| | 0.7 | 1.6 | 0.00 | 1.9 | 0.9 |
| | 0.9 | 1.7 | 0.00 | 2.2 | 0.8 |
| Avg | | 1.7 | 0.00 | 519.9 | 9.6 |

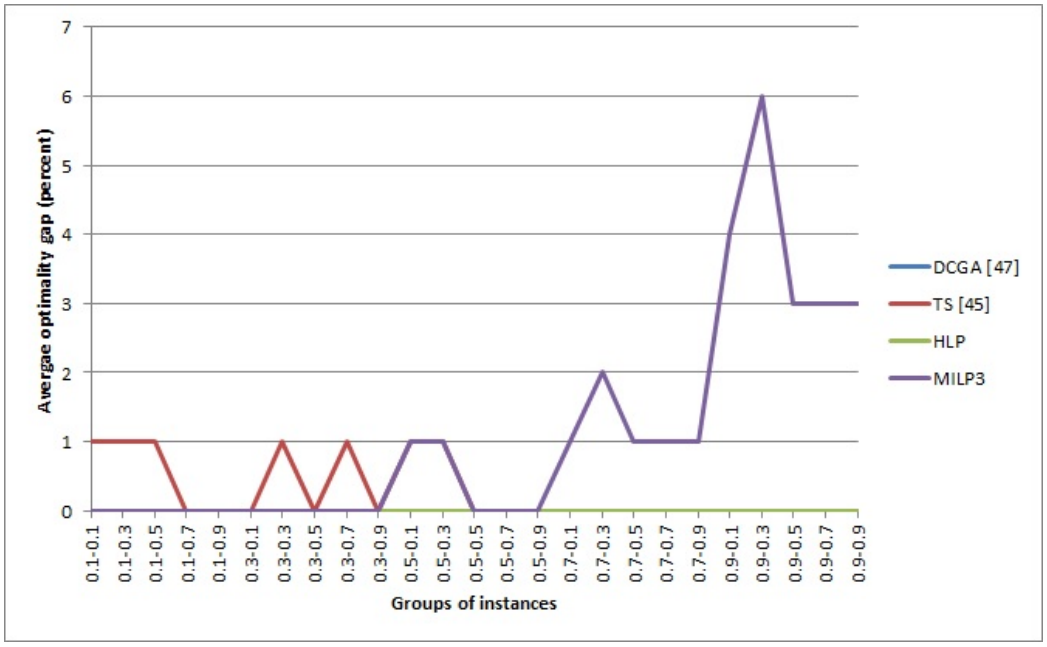


Figure 4.4: Performance of DCGA, TS, our heuristic (HLP), and our revised mathematical model (MILP3) in aspect of optimality gap for 10-order instances.

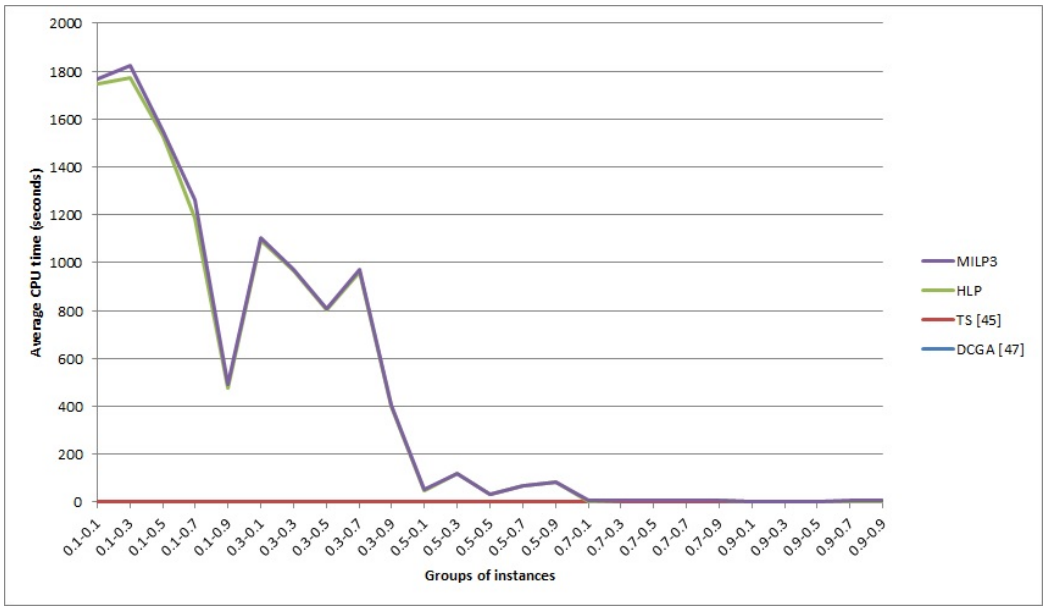


Figure 4.5: Performance of DCGA, TS, our heuristic (HLP), and our revised mathematical model (MILP3) in aspect of CPU time for 10-order instances.

# Chapter 5

# CONCLUSIONS AND FUTURE WORK

In this research, we focused on order acceptance and scheduling problems in a make-to-order system. We investigated two different types of OAS problem: the problem $1 \,|\, d_j, e_j, w_j \,|\, \sum R_j$, in which parameters include due dates, maximum revenues, and importance weights, and the problem $1 \,|\, r_j, d_j, e_j, w_j, s_{ij} \,|\, \sum R_j$, in which release dates and setup times are considered as well as all parameters of the first problem.

Firstly we introduced the general OAS problem, and mentioned which parts we are improving in this thesis. Then we reviewed the literature related to OAS problem after dividing all previous work into three main areas. Then, in Chapter 3, we discussed our first OAS problem. We modeled the problem and we showed that we are able to solve all benchmark instances of 10- and 20-order by MILP1 in average CPU time of less than one seconds while previous studies were not able to do so. After that, to solve larger benchmark instances of 75- and 100-order, we developed three dominance properties by considering simple characteristics of the problem. We showed that by adding these dominance properties one by one or as a combination to MILP1, we can improve the optimality gaps compared to the literature. Secondly, in Chapter 4, we consider our second problem which is the general form of OAS problem. We gave the first time-indexed MILP (MILP2) for the problem and then we find out that MILP2 cannot solve benchmark instances which are larger than 15 orders to optimality. In Section 4.3, we tried two different methods of Lagrangian and LP relaxation. We then showed that Lagrangian Relaxation create subtours, and that is why its objective function value is a very loose upper bound. We then focused on finding valid inequalities to be added to the LP relaxation version of the problem to produce tight upper bounds, and we showed that we could improve all previous upper

bounds with this method. In Section 4.4, we introduced two new heuristic methods to find good lower bounds, but, as the result showed, both methods perform well for small instances, and their CPU time is too large for instances with more than 15 orders.

The proposed methodologies for both of the OAS problems have some weaknesses mentioned in each section. To improve these weaknesses, research can be done with the purposes as listed below.

- To find good heuristics for solving both OAS problems in a reasonable time to optimality.

- To improve the methodology, discussed in Sections 3.2, and 3.3 for the first problem, and in Section 4.3 for the second problem, by finding better and more powerful dominance properties and valid inequalities to reduce the CPU time and produce better bounds.

Regarding the first problem, discussed in Chapter 3:

- To build dynamic representation of the system where either the acceptance decisions or some parameters are known in advance or stochastic changes are required to be included to the problem.

Regarding the second problem, discussed in Chapter 4:

- To find methodology that consider stochastic nature of the environment.

- To add some valid inequalities to the MILP2, discussed in Section 4.5, reduce the running time

# BIBLIOGRAPHY

[1] M. E. Dyer, and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Applied Mathematics* 26(2): 255–270, 1990.

[2] L-P. Bigras, M. Gamache, and G. Savard. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization* 5(4): 685–699, 2008.

[3] N. Bansal, H-L. Chan, R. Khandekar, K. Pruhs, B. Schicber, and C. Stein. Non-preemptive min-sum scheduling with resource augmentation. In Foundations of Computer Science, *48th Annual IEEE Symposium* : 614–624, 2007.

[4] M. Goemans. Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques. 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2001 and 5th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2001 Berkeley, CA,USA, August 18-20, 2001.

[5] P. Babu, L. Peridy, and E. Pinson. A branch and bound algorithm to minimize total weighted tardiness on a single processor. *Annals of Operations Research* 129(1-4): 33–46, 2004.

[6] J. M. Van den Akker, C. Hurkens, and M. W. P. Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing* 12(2): 111–124, 2000.

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

[8] Y. Pan, and L. Shi. 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, 2007.

[9] L-P. Bigras, M. Gamache, and G. Savard. Time-Indexed Formulations and the Total Weighted Tardiness Problem. *INFORMS Journal on Computing* 20(1): 133–142, 2008.

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

[11] T. Ibaraki. Enumerative approaches to combinatorial optimization. Baltzer, 1987.

[12] A. B. C. Altunc, and A. B. Keha. Interval-indexed formulation based heuristics for single machine total weighted tardiness problem. *Computers and Operations Research* 36(6): 2122-2131, 2009.

[13] P. Baptiste, and R. Sadykov. On scheduling a single machine to minimize a piecewise linear objective function: A compact MIP formulation. *Naval Research Logistics* 56(6): 487-502, 2009.

[14] F. Sourd, and S. Kedad-Sidhoum. An efficient algorithm for the earliness-tardiness scheduling problem. *Optimization* Online (1205), 2005.

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

[16] P. Baptiste, and R. Sadykov. Time-indexed formulations for scheduling chains on a single machine: An application to airborne radars. *European Journal of Operational Research* 203(2): 476–483, 2010.

[17] C. Phillips, R.N. Uma, and J. Wein. Off-line admission control for general scheduling problems. *Journal of Scheduling* 3: 365–381, 2000.

[18] M. W. P. Savelsbergh, R. N. Uma, and J. Wein. An experimental study of LP-based approximation algorithms for scheduling problems. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 453-462, 1998.

[19] R. N. Uma, and J. Wein. On the relationship between combinatorial and LP-based approaches to NP-hard scheduling problems. *Integer Programming and Combinatorial Optimization*, Springer Berlin Heidelberg, 394–408, 1998.

[20] J. M. Van den Akker, C. P. M. Van Hoesel, and M. W. P. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming* 85(3): 541–572, 1999.

[21] J. P. Sousa, and L. A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. *Mathematical programming* 54(1–3): 353–367, 1992.

[22] J. M. Van den Akker, C. Hurkens, and M. W. P. Savelsbergh. A Time-indexed formulation for single-machine scheduling problems: branch-and-cut. Center for Operations Research & Econometrics. Universit catholique de Louvain, 1996.

[23] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of operations research* 22(3): 513–544. 1997.

[24] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics* 15(2): 165–192, 2002.

[25] H. A. J. Crauwels, A. M. A. Hariri, C. N. Potts, and L. N. Van Wassenhove. Branch and bound algorithms for single-machine scheduling with batch set-up times to minimize total weighted completion time. *Annals of Operations Research* 8: 59-76, 1998.

[26] P. Avella, M. Boccia, and B. DAuria. Near-Optimal Solutions of Large-Scale Single-Machine Scheduling Problems. *INFORMS Journal on Computing* 17(2): 183–191, 2005.

[27] L. Peridy, E. Pinson, and D. Rivreau. Using short-term memory to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research* 148(3): 591–603. 2003.

[28] J. M. Van den Akker, C. P. M. Van Hoesel, and M. W. P. Savelsbergh. Facet inducing inequalities for single-machine scheduling problems. Eindhoven University of Technology, Department of Mathematics and Computing Science, 1993.

[29] M. Queyranne, and A. S. Schulz. Polyhedral approaches to machine scheduling. TU, Fachbereich 3, 1994.

[30] H. Waterer, E. L. Johnson, P. Nobili, and M. W. P. Savelsbergh. The relation of time indexed formulations of single machine scheduling problems to the node packing problem. *Mathematical Programming* 93(3): 477–494. 2002.

[31] H. I. Stern, and Z. Avivi. The selection and scheduling of textile orders with due dates. *European journal of Operational Research* 44(1): 11–16, 1990.

[32] S. K. Gupta, J. Kyparisis, and C-M. Ip. Project selection and sequencing to maximize net present value of the total return. *Management Science* 38(5): 751–75, 1992.

[33] B. Aspvall, S. D. Flam, and K. P. Villanger. Selecting among scheduled projects. *Operations research letters* 17(1): 37–40, 1995.

[34] G. J. Kyparisis, S. K. Gupta, and C-M. Ip. Project selection with discounted returns and multiple constraints. *European Journal of Operational Research* 94(1): 87–96, 1996.

[35] S. A. Slotnick, and T. E. Morton. Selecting jobs for a heavily loaded shop with lateness penalties. *Computers & Operations Research* 23(2): 131–140, 1996.

[36] J. B. Ghosh. Job selection in a heavily loaded shop. *Computers & Operations Research* 24(2): 141–145, 1997.

[37] H. F. Lewis, and S. A. Slotnick. Multi-period job selection: planning work loads to maximize profit. *Computers & Operations Research* 29(8): 1081–1098, 2002.

[38] K. Charnsirisakskul, P. M. Griffin, and P. Keskinocak. Order selection and scheduling with leadtime flexibility. *IIE transactions* 36(7): 697–707, 2004.

[39] K. Charnsirisakskul, P. M. Griffin, and P. Keskinocak. Pricing and scheduling decisions with leadtime flexibility. *European Journal of Operational Research* 171(1): 153–169, 2006.

[40] S. A. Slotnick, and T. E. Morton. Order acceptance with weighted tardiness. *Computers & Operations Research* 34(10): 3029–3042, 2007.

[41] B. Yang, and J. Geunes. A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. *Computers & Industrial Engineering* 53(3): 420–432, 2007.

[42] W. O. Rom, and S. A. Slotnick. Order acceptance using genetic algorithms. *Computers & Operations Research* 36(6): 1758–1767, 2009.

[43] C. Oguz, F. S. Salman, and Z. Bilginturk Yalcin. Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics* 125(1): 200–211. 2010.

[44] F. T. Nobibon, and R. Leus. Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. *Computers & Operations Research* 38(1): 367–378, 2011.

[45] B. Cesaret, C. Oguz, and F. S. Salman. A tabu search algorithm for order acceptance and scheduling. *Computers & Operations Research* 39(6): 1197–1205, 2012.

[46] S. W. Lin, and K. C. Ying. Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm. *Journal of the Operational Research Society* 64(2): 293–311, 2013.

[47] C. Chen, Z. Yang, Y. Tan, and R. He. Diversity Controlling Genetic Algorithm for Order Acceptance and Scheduling Problem. *Mathematical Problems in Engineering*, 2014.

[48] S. Nguyen, M. Zhang, and M. Johnston. Enhancing Branch-and-Bound Algorithms for Order Acceptance and Scheduling with Genetic Programming. *Genetic Programming*, Springer Berlin Heidelberg, 124–136, 2014.

[49] S. Thevenin, N. Zufferey, and M. Widmer. Metaheuristics for a scheduling problem with rejection and tardiness penalties. *Journal of Scheduling* 18(1): 89–105, 2014.

[50] K. Ono, and C. Jones. An heuristic approach to acceptance rules in integrated scheduling systems. *Journal of Operations Research Society of Japan* 16(1): 36–58, 1973.

[51] B. Pourbabai. A short term production planning and scheduling model. *Engineering costs and production economics* 18(2): 159–167, 1989.

[52] B. Pourbabai. Optimal selection of orders in a just-in-time manufacturing environment: A loading model for a computer integrated manufacturing system. *International Journal of Computer Integrated Manufacturing* 5(1): 38-44, 1992.

[53] R. Kolisch. Integrated production planning, order acceptance, and due date setting for make-to-order manufacturing. *Operations Research Proceedings*, Springer Berlin Heidelberg, 492–497, 1998.

[54] S. Mestry, P. Damodaran, and C-Sh. Chen. A branch and price solution approach for order acceptance and capacity planning in make-to-order operations. *European Journal of Operational Research* 211(3): 480–495, 2011.

[55] Y-Y. Xiao, R-Q. Zhang, Q-H. Zhao, and I. Kaku. Permutation flow shop scheduling with order acceptance and weighted tardiness. *Applied Mathematics and Computation* 218(15): 7911–7926, 2012.

[56] X. Wang, X. Xie, and T. C. E. Cheng. A modified artificial bee colony algorithm for order acceptance in two-machine flow shops. *International Journal of Production Economics* 141(1): 14–23, 2013.

[57] X. Wang, X. Xie, and T. C. E. Cheng. Order acceptance and scheduling in a two-machine flowshop. *International Journal of Production Economics* 141(1): 3660-376, 2013.

[58] C. Cheng, Z. Yang, L. Xing, and Y. Tan. An improved genetic algorithm with local search for order acceptance and scheduling problems. *Computational Intelligence In Production And Logistics Systems (CIPLS)*, IEEE Workshop on, 115–122. IEEE, 2013.

[59] Y. Lee, and H. D. Sherali. Unrelated machine scheduling with time-window and machine downtime constraints: An application to a naval battle-group problem. *Annals of Operations Research* 50(1): 339–365, 1994.

[60] H. D. Sherali, Y. Lee, and D. D. Boyer. Scheduling target illuminators in naval battlegroup antiair warfare. *Naval Research Logistics (NRL)* 42(5): 737–755, 1995.

[61] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics* 5: 287–326, 1979.

# VITA

Saeed Saffari was born in Sabzevar, Iran on June 5, 1991. He graduated from Nemooneh High School in 2009. He received his B.Sc. degree in Industrial Engineering from Sharif University of Technology, Tehran in 2013. Same year, he joined the M.Sc. program in Industrial Engineering at Koç University and from September 2013 to June 2015, he worked as a teaching and research assistant at Koç University, Turkey. He has presented part of his thesis at the ECCO (European Chapter on Combinatorial Optimization) XXVII Conference (Catania, Italy) in 2015 and submitted the result of his thesis to Computers and Operations Research. Next year, he will be a Ph.D. candidate of Industrial Engineering program at North Carolina State University.