Optimization of NP-hard Scheduling Problems by Developing Timing Algorithms and Parallelization

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften der Carl von Ossietzky Universität Oldenburg zur Erlangung des Grades und Titels eines

Doktor der Naturwissenschaften
(**Dr. rer. nat.**)

angenommene Dissertation

von Herrn **Abhishek Awasthi**

geboren am 22.08.1985 in Unnao, Indien

Gutachter:

Jun.-Prof. Dr. habil. Oliver Kramer
Prof. Dr.-Ing. Jörg Lässig

Mitglieder der Prüfungskommission:

Prof. Dr. Annegret Habel (Vorsitz)
Dr. Ute Vogel

Tag der Disputation: **5. Dezember 2016**

... to my mother

Acknowledgements

I would like to take this opportunity to acknowledge and thank several people involved in helping and providing me encouragement, without which this work would have been a challenge akin to impossibility. First and foremost, I would like to express my gratitude to Prof. Jörg Lässig, under whom I have carried out all my research work. I thank him for his patience for providing me with adequate time before the beginning of this work. I also thank him for his constant guidance, as well providing me with good exposure by allowing me to be on several research trips. I, equally thank Prof. Oliver Kramer for accepting me as his PhD student and for helping and guiding me with several aspects of this work. I am grateful for his forthright and extremely helpful suggestions without any deliberations, through out the period of this work. Moreover, his encouragement for my work has been second to none, which only helped me to reach this stage. Additionally, I would also show my gratitude to Prof. Thomas Weise, who although was involved with the later part of my work, nonetheless has been highly helpful and collaborative in providing me with his timely and frequent suggestions whenever I asked for one. I also thank him for inviting me to his research group in Hefei, China. I would also like to thank Dr. Marco Grawunder for guiding me to finish my preliminary requirements for my enrollment at the University of Oldenburg. Next, and no less important, I would certainly like to thank Prof. Lluís Alsedà i Soler, whose extremely useful guidance and exposure led my interest in this field, and had been great motivator. Had it not been for him, I would not have started with this work in the first place. I thank my colleagues Andreas Schulz, Waheed Ghumman, Markus Ullrich, Adam Bartusiak and Jens Leuschner who have not only helped me on my work but have provided me with an excellent companionship at work. I thank them all for all the help and good times during these years. Lastly, I thank my childhood friend Abhas, who has been an exceptional friend for his constant encouragement and helping me in several ways. I also thank my sisters Nidhi and Anamika, my teacher Virendra Singh, my loved ones and ofcourse my parents, who have allowed me to achieve this, so far away from them.

Abhishek Awasthi

Erklärung zur Selbständigkeit und zu Hilfsmitteln

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Abhishek Awasthi

Zusammenfassung

Diese Forschungsarbeit präsentiert Scheduling-Algorithmen für NP-schwere kombinatorische Optimierungsprobleme in der Fertigung und im Transportbereich. Scheduling spielt eine wichtige Rolle im Erfolg der meisten Logistiksysteme wie zum Beispiel Produktion, Materialumschlag, Verpackung, Warenbestand, Transport, Lagerhaltung etc. Scheduling-Aufgaben wurden seit den späten Fünfzigerjahren untersucht. Wegen der NP-schweren Eigenschaften der meisten dieser Aufgaben, wurden diese weitestgehend unter Verwendung metaheuristischer Algorithmen gelöst, bestehend aus Genetische Algorithmen, Simulated Annealing, Ant Colony Optimization, etc. Diese metaheuristischen Algorithmen weisen ein unermessliches Potential für fast alle NP-schweren Optimisierungsaufgaben auf. Trotzdem kann je nach Umfang der Aufgabe eine optimale Lösung nicht immer gefunden werden.

Auer diesen Methoden können solche Aufgabenstellungen auch mit Integer Programming (IP) angegangen werden. Wegen der exponentiell steigenden Anzahl von Entscheidungsvariablen scheitern IP-Anwender jedoch, gröere Aufgaben mit herkömmlichen Computern zu lösen. Diese Arbeit basiert auf der Aufteilung des 0-1 Integer Programming in zwei Teile, hauptsächlich um die Gröe des Suchraumes zu verkleinern. Diese Teilung führt zu einem linearen Programm und einem Satz von Entscheidungsvariablen. Weil Lineare Programmierung (LP) polynomiell lösbar ist, kann es in den metaheuristischen Algorithmus integriert werden, um eine optimale bzw. nahezu optimale Lösungen zu erzielen. Die Nutzung von LP-Solvern mit iterativen metaheuristischen Algorithmen ist jedoch zeitaufwendig, da diese Solver nicht schnell genug sind. Folglich ist für eine effektive Nutzung dieses Ansatzes die Entwicklung von schnellen spezialisierten polynomiellen Algorithmen für das resultierende Lineare Programm notwendig. Die Entwicklung dieser exakten polynomiellen Algorithmen ist jedoch kompliziert - auer für triviale Fälle und erfordert zuerst eine theoretische Analyse des spezifischen Linearprogramms.

In der vorliegenden Arbeit wird dieser Ansatz für verschiedene NP-schwere Scheduling-Aufgaben verwendet, hauptsächlich in den Bereichen Transport und Fertigung. Um optimale bzw. nahezu optimale Lösungen zu erhalten,

werden neue spezialisierte Algorithmen für resultierende LPs entwickelt und diese mit den metaheuristischen Algorithmen kombiniert. Die Entwicklung des polynomiellen Algorithmus erfolgt durch eine intensive Analyse der sich ergebenden Linearen Programme für alle Scheduling-Probleme, die in dieser Arbeit berücksichtig wurden. Die Ergebnisse verschiedener Scheduling - Benchmarks beweist das Potential dieses Ansatzes. Ein weiterer Vorteil dieses Ansatzes ist dessen inhärente Parallelisierbarkeit, die später in dieser Arbeit mit Hilfe von GPGPU (General Purpose Computation on Graphics Processing Unit) gezeigt wird. Auerdem wird auch diskutiert, wie dieser allgemeine Ansatz für andere kombinatorische Optimierungsprobleme angepasst werden kann.

Abhishek Awasthi

Abstract

This research work presents scheduling algorithms for NP-hard combinatorial optimization problems in manufacturing and transportation. Scheduling plays an important role in the success of most of the logistic systems such as production, material handling, packaging, inventory, transportation, warehousing, *etc.*. Scheduling problems have been investigated since the late fifties. Due to the NP-hard nature of most of these problems, they have predominantly been solved utilizing the metaheuristic algorithms, consisting of Genetic Algorithm, Simulated Annealing, Ant Colony Optimization, *etc.*. These metaheuristic algorithms have proven to be of immense potential for almost all the NP-hard optimization problems. Nonetheless, a solution to optimality can be hard to come by, depending on the instance size of the problem.

Apart from these methodologies, these problems can also be tackled with Integer Programming (IP). However, due to the exponentially increasing number of decision variables, IP solvers fail to solve large sized problem instances on conventional computing devices. This work is based on splitting the 0-1 Integer Programming in two parts to basically reduce the size of the search space. This split leads to a linear program and a set of decision variables. Since Linear Programming (LP) is polynomially solvable, they can be integrated with the metaheuristic algorithms to obtain optimal/near-optimal solutions. However, using LP solvers with an iterative metaheuristic algorithm is time consuming as these solvers are not fast enough. Hence, an effective utilization of this approach requires the development of some fast specialized polynomial algorithms for the resulting LP. However, developing these exact polynomial algorithms is not straight forward except for trivial cases, and requires theoretical analyses of the specific linear program at hand.

In this work, we utilize this approach over several NP-hard scheduling problems mainly in the field of transportation and manufacturing. We develop novel specialized algorithms for the resulting LPs to exploit them in conjunction with the metaheuristic algorithms to provide optimal/near-optimal solutions. The development of efficient polynomial algorithms is carried out by in-depth theoretical studies of the resulting LPs of all the scheduling prob-

lems considered in this work. Our results over several scheduling problems prove the potential of this approach. Another benefit of this approach is its inherent parallel structure which is demonstrated later in this work with the help of Graphics Processing Unit (GPU) computing. Moreover, we also discuss how this generalized approach can be extended to other combinatorial optimization problems, apart from scheduling.

Abhishek Awasthi

Contents

1 Introduction

Scheduling is a decision-making process that is used on a regular basis in many manufacturing and service industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. The resources and tasks in an organization can take many different forms. The resources may be machines in a workshop, runways at an airport, crews at a construction site, processing units in a computing environment and so on. The tasks may be operations in a production process, arrivals and departures at an airport, stages in a construction project, executions of computer programs, and so on. Each task may have a certain priority level, for instance, an earliest possible starting time and a due date along with a processing time. The objectives can also take many different forms. One objective may be the minimization of the completion time of the last task and another may be the minimization of the number of tasks completed after their respective due dates.

Scheduling problems have been investigated since the late fifties [16, 142, 137]. Two types of applications have mainly motivated research in this area: project planning and machine scheduling [64, 89]. While in machine scheduling a large number of specific scheduling situations depending on the machine environment and the job characteristics have been considered [91], the early work in project planning investigated scheduling situations with precedence constraints between activities assuming that sufficient resources are available to perform the activities [81]. On the other hand, also in machine scheduling more general and complex problems have been investigated. Due to these developments, today both areas are much closer to each other. Furthermore, applications like timetabling, industrial scheduling and several other logistics are connected to both areas [29]. Scheduling plays an important role in the success of most of the logistic systems such as; material handling, production, packaging, inventory, transportation, warehousing *etc.* [119]. One of the first works related to scheduling was presented by Henry Gantt, who gave the idea of Gantt Charts, a graphical representation of tasks in general production planning [59]. However, it took many years for the first scheduling publications to appear in the industrial engineering and operations research literature. Some of the first publications appeared in Naval Research Logistics Quarterly in the early fifties and contained results by W.E. Smith, S.M. Johnson and J.R. Jackson [130, 75, 71]. During the sixties a significant amount of work was done on branch-and-bound and integer programming formulations of scheduling problems [98, 2]. Integer Program (IP) is a mathematical model of an optimization problem, comprising of all the constraints and the objective function of the problem, and the conventional Branch-and-Bound algorithm works by relaxing the integer decision variables of an IP.

1.1 Motivation

An IP formulation of a scheduling problem is a combination of linear constraints and a set of integer decision variables. IP solvers are a strong tool to solve several NP-hard problems to optimality. However, since an IP itself is NP-hard, it ceases to solve problems with high number of integer decision variables. Hence, one of the best tools available today to produce optimal/near-optimal solutions to these problems are the metaheuristic algorithms.

A metaheuristic algorithm in general can be viewed as a problem-oriented intelligently randomized algorithm, generally based on a naturally occurring phenomenon, for any optimization problem, as opposed to mathematical optimization algorithms such as the linear programming algorithms. A lot of problems in scheduling are NP-hard and require a high runtime complexity even with the present state-of-the-art [29, 119, 79]. In recent years the main focus has been on developing metaheuristic or hybrid metaheuristic algorithms to solve the NP-hard scheduling problems in general [50, 54, 53, 110, 99, 1, 52, 13, 57]. Some of the benchmark problems such as the relatively small instances of job shop problem with 10 jobs and 10 machines, proposed by Muth and Thompson (1963) [111] remained unsolved for more than a quarter of a century and these problems were solved later with the advancement of the metaheuristic algorithms [146]. These mainly include Evolutionary Algorithms [63], Local Search [107], Tabu Search [62] *etc.*. Another related area which has been proven to be very efficient for these problems is hybridization of different approaches, where the strategies of two or more metaheuristics are combined together in a single algorithm [23]. Many of these algorithms have been implemented on CPUs and have turned out to be extremely useful for a large class of combinatorial problems in general [50, 54, 53, 110, 99, 1, 52, 13, 57].

In this work we focus on solving an NP-hard problem utilizing the Integer Programming (IP) formulation and the metaheuristic algorithms. The interesting aspect of an IP formulation which is utilized in this research work is that discarding the integer constraint on the decision variables renders us a linear programming formulation, which is polynomially solvable [78]. The idea is to fix the decision variables to a feasible set of values with the help of metaheuristic algorithms and utilize the resulting linear programming formulation.

However, even though a linear programming formulation is polynomially solvable to optimality, they are considerably slower when used with any iterative heuristic algorithm.

Due to the slowness of the LP solvers, faster specialized polynomial algorithms known as the timing algorithms are required to effectively utilize the benefit of the discussed approach. Hence, in this work we make extensive theoretical studies on several NP-hard scheduling problems and develop fast specialized polynomial algorithms for the resulting linear program, and utilize them with the metaheuristic algorithms. We demonstrate the advantage of this two-layered approach with the help of our computational results as well as its apparent straight forward benefit for multi-core processing.

1.2 Summary of the Book

We now present a short introduction and summary of each chapter of this book. Chapter 2 describes the generalized two-layered approach adopted in this work to optimize NP-hard scheduling problems. A general framework in the 0-1 mixed integer programming formulation of a combinatorial scheduling problem is utilized. This strategy helps us in the rest of this work to optimize the studied scheduling problems by combining the specialized polynomial algorithms with metaheuristic algorithms. We also present the inherent parallel structure in this approach for NP-hard scheduling problems. Related work for this strategy is also discussed.

We then study four different NP-hard scheduling problems utilizing the two-layered approach. In Chapter 3 we study the Aircraft Landing Problem (ALP), which involves landing of a certain number of planes at an airport on single or multiple runways with special constraint on the safety distances. The Aircraft Landing Problem is studied since the 90s till recently, by several authors [11, 12, 13, 52, 124, 126, 144]. In this work we propose a polynomial algorithm to optimize the landing times of the aircraft on runway(s) for a special case of the safety constraint. This algorithm is developed taking into account the integer programming formulation of the ALP. This formulation can be divided in two parts, where one part is solved with our polynomial algorithm, and the second part is dealt with by a metaheuristic algorithm. Our algorithm also accounts for the general case of the safety constraint in the sense that we provide high quality feasible solutions, which is evident from our experimental studies on the benchmark instances and its comparison with the sate-of-the-art results in this field.

Chapter 4 discusses another scheduling problem which is in the field of Just-in-Time philosophy, known as the Common Due-Date problem (CDD). The objective of this NP-hard combinatorial optimization problem is to schedule and sequence a set of jobs on single or parallel machines against a common due-date, to minimize the total weighted earliness/tardiness cost. This scheduling problem is of practical relevance to several mass producing manufacturing industries. Once again we utilize the two-layered approach and develop fast specialized deterministic algorithms to optimize the schedule of jobs in any given sequence. We present two different algorithms to solve the resulting linear program of the fixed job sequence. First algorithm is derived by reducing the CDD to ALP and the second algorithm is developed utilizing some intrinsic properties of this scheduling problem. Moreover, a simple yet highly effective heuristic is also proposed to locally improve any jobs sequence. We then club our algorithm with the Simulated Annealing metaheuristic and carry out experimental analysis on the benchmark instances. Our results are compared with the best known results found in the literature and the benefit of our strategy is justified. We then go on to study an extension of the common due-date problem known as the Common Due-Window (CDW) problem in Chapter 5. This scheduling problem is quite similar to the CDD except that in CDW the jobs are scheduled against a due-window bounded by two due-dates. Yet again, we break the integer programming formulation for this problem and solve the resulting linear program with a linear algorithm. This algorithm is developed by making a theoretical study of the problem and extending an important property of the CDD to the CDW problem. Similar to the CDD, we also propose a simplified heuristic which utilizes the V-shaped property. Henceforth, we present our results on the benchmark instances which are far superior to the best known solutions.

Next, we study the Un-restricted Common Due-Date problem with Controllable Processing times (UCDDCP), in Chapter 6. The idea behind the solution for this problem is based on the same principle. We make an extensive theoretical study and propose novel properties for this scheduling problem. These properties are then exploited to develop a linear timing algorithm to optimize any job sequence. We also provide new benchmark instances and combine simulated annealing and threshold accepting metaheuristics with our $O(n)$ algorithm to provide optimal/near-optimal solutions to all the benchmark instances. In Chapter 7 we show the utilization of the inherent parallel structure of the two-layered approach. We provide GPGPU based parallel metaheuristic algorithms to optimize CDD and UCDDCP. The tailor-made ease and efficiency of our approach for parallel processing is well described and justified with our experimental analyses. We conclude our work with Chapter 8, where the proposed strategy and this research work are summarized along with the discussions on its benefit and extension of the proposed approach to other NP-hard combinatorial optimization problems.

2

The Generalized Two-layered Approach

In this chapter we provide a short introduction to Linear Programming (LP) and Integer Programming (IP), for scheduling problems. We then explain the two-layered approach of solving an NP-hard scheduling problem by breaking-up the mixed integer programming formulation in two parts. One part is solved optimally by specialized polynomial algorithms, while the second part is solved utilizing a metaheuristic algorithm. We explain the benefits of the specialized polynomial algorithms over the LP solvers. Additionally, we also discuss the apparent advantage of the two-layered approach for multi-core processing, predominantly for parallel GPGPU computing.

2.1 Linear and Integer Programming

A mathematical model of a real world optimization problem helps us to formulate the exact requirements and constraints for better analysis. Linear Programming is one such methodology to meet the requirements of real world problems of different nature, such as allocation of resources, transportation, manufacturing *etc.* [69]. Linear Programming is a strong and useful numerical optimization tool, especially in Operations Research. In general, LP refers to a mathematical program of a min/max optimization problem, where the objective function and the constraints are all linear in terms of the decision variables. Linear programming formulation in general can be expressed as

$$\begin{aligned} \text{minimize} \quad & c^T \bar{\boldsymbol{x}} \\ \text{subject to} \quad & A\bar{\boldsymbol{x}} \leq b \\ & A_{eq}\bar{\boldsymbol{x}} = b_{eq} \\ & \bar{\boldsymbol{x}} \geq 0. \end{aligned}$$

In the above formulation, c, b and b_{eq} are the vectors of known parameters, A, A_{eq} are the matrices of known parameters and $\bar{\boldsymbol{x}}$ is the vector of decision variables. On the other hand, an Integer Program is a special case of a linear program which possesses an additional constraint that all the decision variables must be integers, *i.e.*, $\bar{\boldsymbol{x}} \in \mathbb{Z}^n$. Evidently, a *Mixed* Integer Programming (MIP) forces only some of the decision variables to take integer values, while the remaining variables are only upper/lower bounded [29, 119].

Although Simplex method is a highly effective and works well in practice to solve a linear program, in the worst case it can take exponential runtime [48, 84]. However, LP is polynomially solvable using the Ellipsoid method and Interior point methods like the Karmarkar's algorithm [78]. A 0-1 integer programming on the other hand is an NP-complete optimization problem [79] and the best methods to solve an IP are branch-and-bound and heuristic algorithms. Several NP-hard combinatorial optimization problems such as the scheduling problems can be expressed as a 0-1 mixed integer programming formulation, where certain decision variables are restricted to 0 or 1.

2.2 The Two-Layered Approach

We now present the intuition and the exact strategy for the two-layered approach. As a simple exemplary case, let us consider the 0-1 mixed integer programming formulation of an NP-hard scheduling problem [66]. The problem consists of scheduling and sequencing a certain number of jobs (n) against a common due-date (d) on a single machine. The processing time and the completion time for any job i is denoted by P_i and C_i, respectively. The objective of this problem is to minimize the total absolute deviation of the completion times of the jobs with the due-date.

The problem is NP-hard and its mixed integer formulation is shown below in Equation (2.1). The idea behind the approach is to break the integer programming formulation of an NP-hard scheduling problem in two parts, *i.e.*, (i) finding the optimal/near-optimal processing sequence and (ii) finding the optimal values of the completion times C_i for all the jobs in this processing sequence. The job sequences are optimized by using a metaheuristic algorithm and for each candidate job sequence, the metaheuristic solves the sub-problem (ii) as a linear program by applying specialized deterministic algorithm. To better understand this strategy, we need to look at the integer programming formulations for one such problem mentioned above.

$$\text{minimize} \sum_{i=1}^{n} |C_i - d| \tag{2.1}$$

subject to

$$C_1 \geq P_1,$$

$$C_i \geq P_i + C_j - G \cdot \delta_{ij}, \qquad i = 1, \ldots, n-1, j = i+1, \ldots, n,$$

$$C_j \geq P_j + C_i - G \cdot (1 - \delta_{ij}), \; i = 1, \ldots, n-1, j = i+1, \ldots, n,$$

$$\boldsymbol{\delta_{ij} \in \{0, 1\}} \qquad i = 1, \ldots, n-1, j = i+1, \ldots, n.$$

In the above formulation, G is some very large positive number and δ_{ij} is the decision variable with $\delta_{ij} \in \{0, 1\}$, $i = 1, 2, \ldots, n-1, j = i+1, \ldots, n$. We have $\delta_{ij} = 1$ if job i precedes job j in the sequence (not necessarily right before it) and vice-versa. It is interesting to observe that the sole purpose of this binary decision variable in the above formulation is to find the optimal job sequence. Additionally, any feasible set of δ_{ij} values also fetches us a feasible job sequence, although not necessarily optimal. For example, let us take one feasible set of values for δ_{ij} for a job sequence size of $n = 4$, where $\delta_{12} = 0$, $\delta_{13} = 0$, $\delta_{14} = 1$, $\delta_{23} = 0$, $\delta_{24} = 1$ and $\delta_{34} = 1$. With this assignment of δ_{ij}, we get a job sequence $\{3, 2, 1, 4\}$. Likewise, there are several possible feasible sets of values for δ_{ij}. Hence, if we take any such feasible set of values of δ_{ij}, we actually have a feasible job sequence at hand, and substituting those fixed δ_{ij} values in Equation (2.1) converts the above MIP formulation to a linear programming formulation.

Apparently, that linear program basically solves for the optimal completion times of the jobs for that particular job sequence. We know that linear programming problem is polynomially solvable and that is how we utilize the above strategy to break our NP-hard problems in two parts. One part deals in finding the completion times (C_i) of the jobs for any given job sequence. The second part, utilizes a metaheuristic algorithm to efficiently search for the optimal/best job sequence. Even though LP is polynomially solvable, the LP solvers are quite slow when run iteratively on some heuristic algorithm. Hence, to gain from the above mentioned strategy, some faster specialized polynomial algorithm for the resulting linear program still needs to be found.

There have been a few research works on optimizing the completion times of a fixed job sequence against distinct due-dates for minimizing the earliness-tardiness costs on a single machine. Szwarc and Mukopadhyay proposed an $O(n^2)$ timing algorithm for minimizing the total weighted earliness/tardiness for a fixed job sequence, with each job possessing a distinct due-date [133]. In 2001, Chrétinne provided an $O(n^3 \log n)$ for the general case of asymmetric and task-dependent earliness/tardiness costs [40]. Chrétinne and Sourd then improved the complexity of this algorithm and proposed an algorithm which ran in $O(n^2)$ [41]. In 2006, Bauman and Józefowska presented an $O(n \log n)$ polynomial algorithm for minimizing earliness/tardiness costs problem, for any given job sequence [10]. In 2007, Hendel and Sourd present an $O(n \log n)$

algorithm for the weighted earliness/tardiness problem with convex piecewise linear cost function for any job [68]. They extend this algorithm for the permutation flow shop scheduling problem. However, there is hardly any research work which utilizes this strategy to solve an NP-hard problem completely providing optimal or near-optimal solutions and prove its efficiency.

Although, this two-layered approach usually calls for the development of specialized polynomial algorithms for the resulting LP, evidently there are also some NP-hard problems, where the resulting LP is *trivial*. One example for such a case is the famous Travelling Salesman Problem (TSP). Consider a network of n cities with each city connected by a direct path. The objective of the TSP is for the salesman to start from its source, visit all the cities and return back to the source, with minimum possible distance covered. Let the source point of the salesman be represented as $i = 0$ and x_{ij} be a binary decision variable such that x_{ij} is equal to 1 if the salesman traverses the direct path between cities i and j, and 0 otherwise. Let c_{ij} be the distance between cities i and j. We can then formulate the integer programming for the TSP as

$$\text{minimize} \sum_{i=0}^{n} \sum_{i \neq j, j=0}^{n} c_{ij} \cdot x_{ij}$$

subject to

$$\sum_{i=0}^{n} x_{ij} = 1, \qquad \forall j \in \{0, 1, 2, \ldots, n\},\ i \neq j,$$

$$\sum_{j=0}^{n} x_{ij} = 1, \qquad \forall i \in \{0, 1, 2, \ldots, n\},\ i \neq j,$$

$$\boldsymbol{x_{ij} \in \{0, 1\}} \qquad \forall (i, j) \in \{0, 1, 2, \ldots, n\}^2,\ i \neq j.$$

Clearly, we have single decision variable in the above formulation and fixing x_{ij} $\forall i, j$ with a feasible set of values provides us with a feasible complete path for the travelling salesman, and the resulting LP is trivial in the sense that we only need to calculate the objective function value for those x_{ij} values. However, there are several NP-hard problems where the resulting LP is not trivial as in the case of above formulation mentioned in Equation (2.1), and thus development of a specialized polynomial algorithm becomes an important aspect of this approach. We study several such NP-hard scheduling problems and develop novel algorithms to solve the LP in polynomial time and prove their effectiveness with experiments on benchmark instances.

Not only does this approach help to reduce the overall complexity of the optimization problem, it also possesses an inherent parallel structure. Any population based metaheuristic can be easily parallelized if the evaluation for each instance of the population is the same. Besides, increasing the population size of a metaheuristic increases its probability to achieve an optimal or near optimal solution. However, if this increase in size is carried out on a single core processing unit, it increases the runtime of the computation, proportionately. With the help of our two-layered approach we can optimize any feasible problem instance in polynomial time and hence utilizing a multi-core processing unit is extremely efficient. For the above example in Equation (2.1), since any job sequence can be solved optimally in polynomial time, we can easily carry out parallel computations for several job sequences utilizing multi core processors. In this research work, we also exploit this parallel structure and combine our polynomial algorithms with parallel metaheuristic algorithms for some of the studied problems for GPGPU computations.

3

Efficient Polynomial Algorithm to Optimize a Given Landing Sequence of the Aircraft Landing Problem

We present a polynomial algorithm for optimizing the overall penalty cost incurred by the earliness/tardiness of the aircraft in a given landing sequence at one or more runways, against their target landing times. Scheduling against due-dates is a general problem in the production and transportation industry. In this work, we investigate the Aircraft Landing Problem (ALP) as an exemplary case and show how this problem is related to the general scheduling problem of weighted earliness-tardiness machine scheduling problems. Henceforth, we present our strategy of breaking down the problem in two parts, solving one part with a number of steps polynomial in the problem scale and the other by using a modified Simulated Annealing (SA) algorithm. Our polynomial algorithms optimize a given landing sequence for the ALP, while the SA is implemented to calculate the optimal processing sequence of the planes. Thereafter, we show the effectiveness of our approach by presenting extensive results for benchmark instances from the OR-library for both the problems and conduct a comparison with other recent work in the literature.

3.1 Introduction

Air Traffic Control (ATC) at any airport has the task to manage the incoming and outgoing flights at the airport. The ATC gives instructions to the aircraft regarding to the choice of runway, speed, and altitude in order to align it with the allocated runway and maintain the safety distance with its preceding aircraft. The first and foremost priority of the ATC is to guide the aircraft in its jurisdiction such that the safety distance between any two aircraft is maintained. The other priorities include the commercial aspect of the business, *i.e.*, to land the aircraft as close as possible to their scheduled landing times to avoid the capital losses. However, during peak hours this job becomes increasingly complicated as the controllers must handle safe and effective landings of a continuous flow of aircraft entering the radar range onto the assigned runway(s). The capacity of runways is highly constrained and this makes the scheduling of landings a difficult task. Increasing the capacity of an airport by building new runways is an expensive and difficult affair. Hence, the air traffic controllers face the problem of allocating a landing sequence and landing times to the aircraft in the radar range. Additionally, in case of airports with multiple runways, they have to make a decision on the runway allotment too, *i.e.*, which aircraft lands on which runway. These decisions are made based on certain information about the aircraft in the radar range [11, 52, 121]. A target landing time is defined as the time at which an aircraft can land if it flies straight to the runway at its cruise speed (most economical). This target landing time is bounded by an earliest landing time and a latest landing time commonly referred to as the time window. The earliest landing time is determined as the time at which an aircraft can land if it flies straight to the runway at its fastest speed with no holding, while the latest landing time is determined as the time at which an aircraft can land after it is held for its maximal allowable time before landing. All the aircraft have to land within their time window and there are asymmetric penalties associated with each aircraft for landing earlier or later than its target landing time. Besides, there is the constraint of the safety distance that has to be maintained by any aircraft with its preceding aircraft. This separation is necessary as every aircraft creates a wake vortices at its rear and can cause a serious aerodynamic instability to a closely following aircraft. There are several types of planes which land on a runway and the safety distance between any two aircraft depends on their types. This safety distance between any two aircraft can be easily converted to a safety time by considering the required separation and their relative speed. If several runways are available for landing, the application of this constraint for aircraft landing on different runways usually depends upon the relative positions of the runways [11, 52, 121]. A formal definition of the ALP is given in Section 3.3.

The objective of the ALP is to minimize the total penalty incurred due to the deviation of the scheduled landing times of all the aircraft with their target landing times. Hence, the air traffic controllers not only have to find suitable landing times for all the aircraft but also a landing sequence so as to minimize the total penalty. This work considers the static case of the aircraft landing problem where the set of aircraft that are waiting to land is already known. For a special but practically very common case of the safety constraint, we present a polynomially bound exact algorithm for optimizing any given feasible landing sequence for the single runway case and an effective strategy for the multiple runway case. In the later part of the chapter we present our results for all the benchmark instances provided by [14] and compare the results with previous works on this problem, as in [3].

3.2 Related Work

The aircraft landing problem described in this chapter was first introduced by [14] and since then, it has been studied by several researchers using different metaheuristics, linear programming, variants of exact branch and bound algorithms *etc.,* for both the static and dynamic cases of the problem. In 1997, Ciesielski *et al.* developed a real time algorithm for the aircraft landings using a genetic algorithm and performed experiments on landing data for the Sydney airport on the busiest day of the year [42]. Beasley *et al.* presented a mixed-integer zero-one formulation of the problem for the single runway case and later extended it to the multiple runway case [11]. The ALP was studied for up to 50 aircraft with multiple runways using linear programming based tree search and an effective heuristic algorithm for the problem. Ernst *et al.* presented a specialized simplex algorithm for the linear program which evaluates the landing times based on some partial ordering information. This method was used in a problem space search heuristic as well as a branch-and-bound method for both, the single and multiple runway case, for again up to 50 aircraft [52]. Beasley *et al.* adopted similar methodologies and presented extended results [11]. In 2001, Beasley *et al.* developed a population heuristic and implemented it on actual operational data related to aircraft landings at the London Heathrow airport [13].

The dynamic case of the ALP was studied again by Beasley *et al.* by expressing it as a displacement problem and using heuristics and linear programming [12]. In 2006, Pinol and Beasley presented two heuristic techniques, Scatter Search and the Bionomic Algorithm and published results for the available test problems involving up to 500 aircraft and 5 runways [121]. The dynamic case of the problem for the single-runway case was again studied by [109]. They used extremal optimization along with a deterministic algorithm to optimize a landing sequence. In 2008, Tang *et al.* implemented a multi-objective evolutionary approach to simultaneously minimize the total scheduled time of arrival and the total cost incurred [135]. In 2009, Bencheikh *et al.* approached the ALP using hybrid methods combining genetic algorithms and ant colony optimization by formulating the problem as a job shop scheduling problem [18]. The same authors presented an ant colony algorithm in conjunction with a heuristic (non-optimal) to adjust the landing times of the aircraft in a given landing sequence in order to minimize the total penalty cost, in 2011 [17].

In 2012, a hybrid meta-heuristic algorithm was suggested by Salehipour *et al.* using simulated annealing with variable neighbourhood search and variable neighbourhood descent [126]. Xie *et al.* presented a hybrid metaheuristic based on BAT algorithm to optimize the ALP. They incorporate four different approaches for the initial landing times of the aircraft [144]. Hancerliogullari *et al.* study the arrival-departure problem for multiple runways and propose greedy and metaheuristic algorithms. They test their algorithms on their own problem instances up to 25 aircraft 5 runways [67]. Faye proposes an approach to solve ALP using an approximation of the separation time required between any two aircraft, to calculate a lower bound of the problem [56]. They then incorporate a constraint generation algorithm to solve the problem. However, there results are not so promising compared to other works mentioned in the literature. Ma *et al.* proposed an approximation algorithm for the aircraft arrival for optimizing the makespan of the landing sequence of the aircraft [105]. Moghaddam *et al.* proposed a fuzzy programming approach for aircraft landing on single runway and present their results only up to 20 aircraft [108]. Lieder *et al.* study the aircraft landing problem with aircraft classes on multiple runways and propose a dynamic programming approach for the problem [93]. Lider and Stolletz study the similar problem but with both landings and take-offs and propose again a dynamic programming approach [94]. Sabar and Kendall implemented an Iterated Local Search algorithm along with local search phase [124]. In this work, we compare our approach with the four most recent works and algorithms proposed by [121, 144, 124, 126].

3.3 Problem Formulation

In this section, we present the mathematical formulation of the static aircraft landing problem based on Pinol and Beasley [121] and Beasley *et al.* [11]. We also define some new parameters which are later used in our algorithm, presented in the next sections.

3.3.1 Notation

Let,
N = total number of aircraft,
R = total number of runways,
ET_i = earliest landing time for aircraft i, $i = 1, 2, \dots, N$,
LT_i = latest landing time for aircraft i, $i = 1, 2, \dots, N$,
TT_i = target landing time for aircraft i, $i = 1, 2, \dots, N$, such that $ET_i < TT_i < LT_i$,
ST_i = scheduled landing time for aircraft i,
S_{ij} = required separation time between planes i and j, where plane i lands before plane j on the same runway, where $(i, j) \in \{1, 2, \dots, N\}^2$ and $i \neq j$,
s_{ij} = required separation time between planes i and j, where plane i lands before plane j on different runways, $(i, j) \in \{1, 2, \dots, N\}^2$ and $i \neq j$,
g_i = earliness (time) of plane i from TT_i,
h_i = tardiness (time) of plane i from TT_i,
α_i = penalty cost per time unit associated with aircraft i for landing before its target landing time TT_i,
β_i = penalty cost per time unit associated with aircraft i for landing after its target landing time TT_i.

Mathematically, the earliness and tardiness of any plane i from its target landing time can be expressed as

$$\left.\begin{aligned} g_i &= \max\{0, TT_i - ST_i\} \\ h_i &= \max\{0, ST_i - TT_i\} \end{aligned}\right\} \qquad i = 1, 2, \ldots, N\ .$$

The total penalty corresponding to any aircraft i is then expressed as $g_i\alpha_i + h_i\beta_i$. If aircraft i lands at its target landing time then both g_i and h_i are equal to zero and the cost incurred by its landing is equal to zero. However, if aircraft i does not land at TT_i, either g_i or h_i is non-zero and there is a strictly positive cost incurred. The objective function of the problem can now be defined as

$$\min \sum_{i=1}^{N} (g_i \cdot \alpha_i + h_i \cdot \beta_i). \tag{3.1}$$

We now discuss the 0-1 mixed integer programming formulation for the ALP with single runway and later in the chapter deal with the multiple runway case. The decision variables for the Aircraft Landing Problem with single runway are ST_i, g_i, h_i and δ_{ij}, where,

$$\delta_{ij} = \begin{cases} 1, & \text{if aircraft } i \text{ lands before } j \text{ and } i \neq j, \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

Note that in any landing sequence, either aircraft i will land before j, or vice-versa, hence we have $\delta_{ij} + \delta_{ji} = 1$.

3.3.2 Constraints

The first constraint on any aircraft landing at the airport is its time-window constraint. The landing time of any plane has to be within its time window, *i.e.*, $ET_i \leq ST_i \leq LT_i$. Another constraint which is forced over the aircraft landings is the safety distance constraints between the aircraft. This constraint is mathematically expressed as $ST_j \geq ST_i + S_{ij} - G \cdot \delta_{ij}$, where $(i, j) \in \{1, 2, \ldots, N\}^2$ and $i \neq j$. Pinol and Beasley use this formulation with $G \gg 0$ being a relatively large positive integer, to ensure that the above equation becomes redundant if aircraft j lands before i, *i.e.*, $\delta_{ji} = 1$ [121].

We can now express the complete 0-1 mixed integer programming formulation as

$$\text{Minimize} \sum_{i=1}^{N} (\alpha_i \cdot g_i + \beta_i \cdot h_i) \tag{3.3}$$

subject to

$$\begin{array}{ll}
ET_i \leq ST_i \leq LT_i, & \forall i \in \{1,2,\ldots,N\}, \\
ST_j \geq ST_i + S_{ij} - G \cdot \delta_{ij}, & \forall (i,j) \in \{1,2,\ldots,N\}^2,\ i \neq j, \\
ST_i = TT_i - g_i + h_i, & \forall i \in \{1,2,\ldots,N\}, \\
0 \leq g_i \leq TT_i - ET_i, & \forall i \in \{1,2,\ldots,N\}, \\
g_i \geq TT_i - ST_i, & \forall i \in \{1,2,\ldots,N\}, \\
0 \leq h_i \leq LT_i - TT_i, & \forall i \in \{1,2,\ldots,N\}, \\
h_i \geq ST_i - TT_i, & \forall i \in \{1,2,\ldots,N\}, \\
\delta_{ij} + \delta_{ji} = 1, & \forall (i,j) \in \{1,2,\ldots,N\}^2,\ i \neq j, \\
\boldsymbol{\delta_{ij} \in \{0,1\},} & \forall (i,j) \in \{1,2,\ldots,N\}^2,\ i \neq j.
\end{array}$$

As we can observe in Equation (3.3), δ_{ij} is the sole decision variable with integer constraint. Besides, this decision variable is also responsible for determining the optimal landing sequence. As discussed in the previous chapter, in this work we solve the resulting linear program by fixing δ_{ij} with a feasible set of values, by developing an efficient polynomial algorithm. This algorithm essentially optimizes the landing times of the aircraft in any given feasible sequence, where the safety constraint is forced only between the planes landing consecutively. Later we extend our polynomial algorithm for the general case of the safety constraint and compare our results with several other approaches in the literature.

3.4 The Exact Algorithm

In this section we present our exact polynomial algorithm for the aircraft landing problem with a special case of the safety constraint for the single runway case. The algorithm takes a feasible landing sequence and computes the optimal landing times to minimize Equation (3.1).

Once we are given a feasible landing sequence, we initialize the landing times of the aircraft by vector ST where any element ST_i is computed as

$$ST_i = \begin{cases} LT_i & \text{if } i = N \\ \min\{PS_i, LT_i\} & \text{if } 1 \leq i \leq N-1, \end{cases} \tag{3.4}$$

where,

$$PS_i = \min_{j=i+1,i+2,\ldots,N} \{(ST_j - S_{ij})\}\ . \tag{3.5}$$

Lemma 3.1. *If the initial assignment of the landing times of all the aircraft in any feasible landing sequence for a single runway is done according to Equation* (3.4) *and* (3.5)*, then the optimal scheduled landing times ST_i for this landing sequence can be obtained only by reducing the landing times of the aircraft while respecting the constraints or leaving the landing times unchanged.*

Proof. Equation (3.4) schedules the landing times of the aircraft in the reverse order starting from the last plane to the first plane in the landing sequence. The last plane is assigned a landing at its latest landing time LT_N and any of the preceding planes are assigned as late as possible from their target landing time, while maintaining the safety distance constraint. This is ensured by $\min\{PS_i, LT_i\}$, where LT_i is the latest landing time of aircraft i and PS_i is the closest landing time possible for aircraft i to all its following aircraft. We define PS_i according to Equation (3.5), where any plane i maintains the safety distance with all its following planes. Since the landing times are assigned as close as possible to their latest landing times, increasing the landing time of any aircraft will cause infeasibility as the last aircraft is landing at its latest landing time and all the preceding planes are scheduled as close as possible. Hence, the optimal solution can be obtained only by decreasing the landing times or leaving them unchanged if there is no reduction possible. □

It is important to note here that with the above initialization, if any of the airplanes in the given sequence lands outside its time window, it shows that this landing sequence is infeasible. In such a case, that particular sequence of aircraft cannot be landed and hence, our algorithm rejects it straight away.

We now present some new parameters, definitions and lemmas which are useful for the explanation of our algorithm. We first define vector D such that element i of this vector is represented by D_i as the algebraic deviation of the scheduled landing time of plane i from its target landing time, $D_i = ST_i - TT_i$, $i = 1, 2, \ldots, N$. We also define vector ES, such that any element ES_i of ES is the minimum of extra separation times maintained between plane i and all its preceding planes, and the deviation from its earliest landing time, for $i > 1$. For $i = 1$, we define ES_i as the deviation of its scheduled landing time with its earliest landing time, as there are no planes landing before the first plane. Mathematically, ES_i can be written as

$$ES_i = \begin{cases} ST_i - ET_i, & \text{if } i = 1, \\ ST_i - \max\{SP_i, ET_i\}, & \text{if } 2 \le i \le N, \end{cases} \tag{3.6}$$

where,

$$SP_i = \max_{j=1,2,\ldots,i-1} (ST_j + S_{ji}) \, . \tag{3.7}$$

Here, SP_i is the time at which an aircraft i can land maintaining the safety constraint with all its preceding planes. Let P be a given landing sequence of the planes where the ith plane in this sequence is denoted by i, $i = 1, 2, \ldots, N$. Note that without loss of any generality we can assume this, since the planes can be ranked in any sequence according to their order of landing.

Given this initialization, it is possible to reduce the landing times straight away to improve the solution. Algorithm 1 presents this improvement of the initial landing times for the single runway case. We would like to point out that Algorithm 1 will not necessarily return the optimal solution but only fetch an

Algorithm 1: Improvement of Individual Landing Times

Initialization: Equation (3.4)
Compute $D_i, ES_i \ \forall i$
for $i = 1$ *to* N **do**
  if $(D_i > 0)$ **then**
    $ST_i \leftarrow ST_i - \min\{D_i, ES_i\}$
  Update $D_i, ES_i \ \forall i$
  $i \leftarrow i + 1$
return ST, ES, D

improvement to the initial assignment of the landing times. The explanation of this improvement algorithm goes as follows. Let the initial landing times of the aircraft be assigned according to Equation (3.4) for any given feasible landing sequence. If any aircraft i with $i = 1, 2, \ldots, N$, has a positive deviation D_i from its target landing time and maintains a positive extra safety separation ES_i, then we can decrease the landing time ST_i by $\min\{D_i, ES_i\}$. The reason behind it is the fact that this reduction of the landing time is independent of other aircraft as we do not disturb the safety constraint and reduce the landing time of i only to bring it closer to its target landing time, which is the requirement of Equation (3.1). If $D_i > ES_i$ we reduce the landing time by ES_i to maintain the safety constraint and if $D_i < ES_i$, we reduce the landing time to its target landing time. Note that $ES_i \geq 0 \ \forall \ i$, since the safety distance constraint is always maintained and $ST_i \geq ET_i$ is a feasible solution.

However, if after initializing the landing times as per Equation (3.4), the value of $D_i \leq 0$ for all the aircraft, then there is no improvement possible and Equation (3.4) determines the optimal assignment for this landing sequence with respect to Equation (3.1).

Lemma 3.2. *When run on a setup according to Equation* (3.4)*, Algorithm 1 will yield either one of the below mentioned cases for any aircraft* $i, i = 1, 2, 3, \ldots, N$*:*

$$\begin{aligned} &a)\ D_i > 0, ES_i = 0, \quad &&b)\ D_i = 0, ES_i > 0, \\ &c)\ D_i = 0, ES_i = 0, \quad &&d)\ D_i < 0, ES_i = 0, \\ &e)\ D_i < 0, ES_i > 0. \end{aligned} \tag{3.8}$$

Proof. The initialization of the landing times using Equation (3.4) can assign the landing time to any aircraft i anywhere in its time window, if the landing sequence is feasible. Hence, we have for each aircraft one of the following five cases after running Algorithm 3.2:

Case 1: $ST_i = ET_i$.

If $i = 1$, then $D_i < 0$ and $ES_i = 0$ from Equation (3.6). If $i > 1$ then $D_i < 0$

but we need to check for the value of ES_i. According to Equation (3.6), we have $ES_i \leftarrow ST_i - \max\{SP_i, ET_i\}$. Note that $ST_i \geq SP_i$, $i = 2, 3, \ldots, N$ since the safety separation is always maintained between any two aircraft landing consecutively. This implies that we can write $\max\{SP_i, ET_i\} = ST_i$ due to the case constraint, *i.e.* $ET_i = ST_i$. Hence, we have $ES_i = 0$ from its definition. Since a reduction in the landing time is possible only if $D_i > 0$, the values of D_i and ES_i will remain unchanged by the implementation of Algorithm 1, satisfying Case *d*.

Case 2: $ET_i < ST_i < TT_i$.
We have $D_i < 0$ for any i, which means that the landing time for aircraft i will remain unchanged. If $i = 1$, then $ES_i > 0$ from Equation (3.6). If $i > 1$ then again from Equation (3.6) we can deduce that $ES_i \geq 0$ because $ST_i \geq SP_i$ (safety constraint) and $ST_i > ET_i$ (case constraint). This proves that if the initial landing time for any aircraft lies between ET_i and TT_i then Algorithm 1 will not fetch any reduction, hence satisfying Case *d* or *e* of Lemma 3.2.

Case 3: $ST_i = TT_i$.
We have $D_i = 0$ for any i since the landing occurs at the target landing time. And $ES_i \geq 0$ for any i, by the same reasons as in Case *2*. In this case as well there will be no reduction and Case *b* or *c* of Lemma 3.2 is satisfied.

Case 4: $TT_i < ST_i < LT_i$.
If the initial landing time for any aircraft i lies between TT_i and LT_i, then $D_i > 0$ by definition and $ES_i \geq 0$ because $ST_i > ET_i$ and $ST_i \geq SP_i$. Hence, Algorithm 1 will reduce the landing time of plane i by $\min\{D_i, ES_i\}$. If $\min\{D_i, ES_i\} = D_i$, then the reduction in the landing time will fetch $D_i = 0$ and $ES_i > 0$, satisfying Case *b*. If $\min\{D_i, ES_i\} = ES_i$ then the reduction in the landing time will fetch $D_i > 0$ and $ES_i = 0$, satisfying Case *a*. However, if after the initialization the values of D_i and ES_i are equal, then the implementation of Algorithm 1 will fetch $D_i = 0$ and $ES_i = 0$, satisfying Case *c*. Finally, if $ES_i = 0$ then there will be effectively no reduction because $\min\{D_i, ES_i\}$ will be equal to zero and Case *a* of Lemma 3.2 will be satisfied.

Case 5: $ST_i = LT_i$.
We have $D_i > 0$ and $ES_i > 0$ after the initialization and yet again the Algorithm 1 will fetch either one of Case *a*, *b* or *c*, with the same arguments as explained in Case 4. □

3.5 Illustration of the Improvement Algorithm

We now illustrate Algorithm 1 with a small example and explain why it provides an improvement to the initialization of the landing times, and not necessarily an optimal schedule. Let's say we have 3 aircraft to be landed on a single runway as per the data mentioned in Table 3.1.

i	ET_i	TT_i	LT_i	α_i	β_i
1	1	9	12	3	4
2	2	10	15	4	5
3	7	15	20	4	2

Table 3.1. Data set for an example with 3 aircraft.

$S_{i,j}$		j	
	-	4	6
i	2	-	2
	5	6	-

Table 3.2. Safety distance S_{ij} for all i and j.

We test this problem instance for a landing sequence of $1, 2$ and 3, and illustrate that Algorithm 1 only returns an improvement of the landing times and not the optimal solution. Initialization of the landing times of the aircraft as per Equation (3.4) leads to

$$ST_3 = LT_3 = 20,$$
$$ST_2 = \min\{PS_2, LT_2\} = \min\{(ST_3 - S_{23}), 15\} = \min\{(20-2), 15\} = 15,$$
$$ST_1 = \min\{PS_1, LT_1\} = \min\{\min\{(ST_2 - S_{12}), (ST_3 - S_{13})\}, LT_1\} = \min\{\min\{(15-4), (20-6)\}, 12\} = 11.$$

This schedule of landing times is feasible since all the three planes land within their permitted time window. Besides, referring to Table 3.1 it turns out that they are landing past their target landing times. Hence, the total penalty incurred with this schedule is $(0 \cdot 3 + 2 \cdot 4) + (0 \cdot 4 + 5 \cdot 5) + (0 \cdot 4 + 5 \cdot 2) = 43$.

We now apply Algorithm 1. For $i = 1$, we first calculate D_1 and ES_1. Recall the definition of D_i and ES_i from Section 3.4. Hence, we have $D_1 = ST_1 - TT_1 = 2$ and $ES_1 = ST_1 - ET_1 = 10$. Since, $D_1 > 0$ we set $ST_1 = ST_1 - \min\{D_1, ES_1\} = 9$. For $i = 2$, $D_2 = ST_2 - TT_2 = 5$ and $ES_2 = ST_2 - \max\{SP_2, ET_2\} = 15 - \max\{(ST_1 + S_{12})\} = 2$. Again $D_2 > 0$, $ST_2 = ST_2 - \min\{D_2, ES_2\} = 15 - 2 = 13$. Following the same procedure for $i = 3$, $D_3 = ST_3 - TT_3 = 5$, and $ES_3 = ST_3 - \max\{SP_3, ET_3\} = 20 - \max\{\max\{(ST_1 + S_{13}), (ST_2 + S_{23})\}, ET_3\} = 5$. $D_3 > 0$ leads to a reduction in the landing time of aircraft 3, with $ST_3 = ST_3 - \min\{D_3, ES_3\} = 15$.

The vectors D and ES are updated again, as per the improved landing times ST_i, as shown in Table 3.3.

i	TT_i	ST_i	D_i	ES_i	α_i	β_i
1	9	9	0	8	3	4
2	10	13	3	0	4	5
3	15	15	0	0	4	2

Table 3.3. Improved landing times with a total penalty of 15, as per Algorithm 1.

As is clear from Table 3.3, this landing scheduling offers only a tardiness penalty to aircraft 2, while airplanes 1 and 3 land at their designated target landing times and offer no penalties. Hence, the objective function value for this schedule is $(3 \cdot 5) = 15$, which corresponds to the tardiness penalty of plane 2. Clearly, we have obtained an improvement from the initial landing schedule which offered a total penalty of 43, but the new schedule with an overall penalty of 15 is not optimal for this landing sequence. This can be proved very easily. Suppose we reduce the landing times of aircraft 1 and 2 by time units of 3, then airplanes 2 and 3 land at their target landing time but aircraft 1 gets early by 3 time units, as shown in Table 3.4.

i	TT_i	ST_i	D_i	ES_i	α_i	β_i
1	9	6	-3	5	3	4
2	10	10	0	0	4	5
3	15	15	0	3	4	2

Table 3.4. Adjusted feasible landing times, after the improvement. The total penalty of this schedule is 9.

Note that this schedule is also feasible, as all the aircraft land within their time-windows and any pair of airplanes maintain the required safety distance. Besides, the new overall penalty of this schedule is $(3 \cdot 3) = 9$. Needless to say, the schedule from Algorithm 1 is definitely not optimal but it only provides an improvement to the initial landing times, which is what we are trying to obtain at this point. More specifically, Algorithm 1 is a non greedy improvement of the initialised landing times, in the sense, that after this improvement none of the lands before its target landing time while maintaining the safety constraint. In the next sections, we go on to explain the procedure of obtaining the optimal schedule once an improvement is obtained using Algorithm 1.

We now give some additional definitions necessary for the understanding of our main algorithm.

Definition 3.3. *PL is a vector of length N and any element of PL (PL_i) is the net penalty possessed by any aircraft i, $i = 1, 2, \ldots, N$. We define PL_i, $i = 1, 2, \ldots, N$, as*

$$PL_i = \begin{cases} -\alpha_i, & \text{if } D_i \leq 0 \\ \beta_i, & \text{if } D_i > 0\ . \end{cases} \tag{3.9}$$

With the above definition we can now express the objective function stated in Equation (3.1) in a compact form as

$$\min \sum_{i=1}^{N} (D_i \cdot PL_i)\ . \tag{3.10}$$

Definition 3.4. *Let i be any aircraft landing at ST_i then we define $\sigma(i)$ as the algebraic deviation of the landing time of aircraft i from its earliest landing time ET_i. Mathematically, $\sigma(i) = ST_i - ET_i$, $i = 1, 2, \ldots, N$.*

The value $\sigma(i)$ for any aircraft i can be interpreted as the maximum feasible reduction in the landing time of aircraft i within its time window. In other words, any reduction in the landing time of an aircraft i can not exceed its $\sigma(i)$ value.

Definition 3.5. *Let aircraft $(i, i+1, \ldots, j)$ be the aircraft in any given sequence which land consecutively in this order on the same runway, we define μ such that μ is the last plane in $(i, i+1, \ldots, j)$ with $D_\mu \leq 0$. Formally, $\mu = \underset{m=i,i+1,\ldots,j}{\operatorname{argmax}} (D_m \leq 0)$.*

Definition 3.6. *Let $\gamma = \{i, i+1, \ldots, j\}$ or in shorthand $\gamma = (i : j)$, be a set of aircraft landing consecutively in that order, such that $ES_i > 0$, $ES_m = 0$, for $m = i+1, \ldots, j$, $\sum_{m=i}^{j} PL_m > 0$ and $\sigma(m) > 0$, for $m = i, \ldots, j$. Needless to say, there may be more than one γ set in a landing sequence, γ_1, γ_2 and so on, such that they are pairwise disjoint collection of sets, i.e., $\gamma_u \cap \gamma_v = \emptyset$, $\forall u \neq v$.*

Example: The above definition of γ can be well understood with the help of a small example presented here. Let the values of $\sigma(i), ES, D$ and PL be as given in Table 3.5.

$\boldsymbol{i}$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\boldsymbol{\sigma(i)}$	4	7	8	7	0	7	9	0
$\boldsymbol{ES}$	4	3	0	0	0	2	0	0
$\boldsymbol{D}$	0	0	4	-3	-5	0	3	-7
$\boldsymbol{PL}$	-2	-1	6	-3	-1	-2	4	-2

Table 3.5. A small example to explain the γ sets for a given landing sequence.

Using Equation (3.6), we see that for aircraft $\{2, 3, 4, 5\}$, the first three properties are satisfied, but for aircraft 6, $\sigma(5) = 0$. Hence the first γ set for the above example is $\gamma_1 = \{2, 3, 4\}$. Likewise, we also have $\gamma_2 = \{6, 7\}$, since $\sigma(10) = 0$.

Definition 3.7. *Define $\Gamma = \{i, i+1, \ldots, j\} \subseteq \gamma$, such that, $\sum_{\rho=\mu}^{j} PL_\rho > 0$ if μ exists for γ. However, if $\sum_{\rho=\mu}^{j} PL_\rho \leq 0$, then $\Gamma = \{i, i+1, \ldots, \mu - 1\}$. Let c be the number of such sets of Γ.*

Using the same example as for γ, we have $\gamma_1 = \{2, 3, 4\}$. For calculation of Γ we need to check if μ exists for γ. As is clear, we have $\mu = \underset{m=2,3,4}{\operatorname{argmax}}(D_m \leq 0) = 4$. Since $\sum_{\rho=4}^{4} PL_\rho < 0$, we end up with $\Gamma(1) = \{2, 3\}$. Likewise, the other set $\Gamma(2) = \{6, 7\}$, since $\nexists\ \mu$ for γ_2. And the number of sets of Γ is two, *i.e.*, $c = 2$.

Definition 3.8. *We define $\Psi(X_{i:j})$ as the smallest strictly positive number in vector X from elements X_i to X_j, $(i < j)$.*

With the above concepts and definitions we present our main algorithm (Algorithm 2) for optimizing a given landing sequence P on a single runway for the special case of the ALP when the safety constraint for any aircraft is to be maintained only with its preceding plane. In other words, when $SP_i = ST_{i-1} + S_{i-1,i}$ and $PS_i = ST_{i+1} - S_{i,i+1}$. For the general case of the safety constraint the algorithm still returns a feasible solution but not necessarily optimal. As said before, we provide optimal schedules for the case when the safety constraint is considered only between consecutively landing planes. Later we show with our results that this special case of the safety constraint holds for several instances and we obtain optimum results for many of them. Moreover we also obtain better results than the best known solutions for several instances.

Our algorithm is based on an iterative shifting of blocks of consecutive airplanes (in the landing sequence). The shift for any block of aircraft occurs if the block all-together can improve the overall penalty of the schedule. Hence, at every stage of the iterative shift of the landing times, we need to calculate Γ sets in the sequence. Each set of Γ is on such block of airplanes, whose landing times are shifted by the same amount, depending on the penalties offered by them. Once we reach a stage when there are no block of aircraft that satisfies the condition of Γ, then we have our optimal schedule for the given landing sequence. We explain our algorithm with the help of an illustrative example.

3.6 Illustration of the Algorithm

We consider the '*airland1*' benchmark instance provided in the OR-library [14], with 10 aircraft and illustrate Algorithm 2 for a landing sequence which is ordered with respect to the target landing times of the aircraft, *i.e.* $P = \{3, 4, 5, 6, 7, 8, 9, 1, 10, 2\}$. As explained before, without loss of generality we can rank the aircraft as per their landing sequence. Let i denote the ith aircraft in the landing sequence then S_i for i denotes the safety separation required between aircraft $i-1$ and i such that $i-1$ lands before aircraft i. For the first aircraft we take its value to be equal to zero ($S_1 = 0$), as there is no safety constraint for aircraft 1. The notations used in this section are the same as throughout this chapter. Table 3.6 shows the initialization of the landing times using Equation (3.4). Aircraft 2 which is scheduled to land at the end is allocated a landing time equal to its latest landing time. All the preceding aircraft are scheduled in such a manner that they are as close as possible to

Algorithm 2: Main Algorithm: Single Runway

Apply Algorithm 1
Calculate PL, Γ, c, σ
while $\Gamma \neq \varnothing$ **do**
  for $k = 1$ ***to*** c **do**
    $(i_k, j_k) \leftarrow \Gamma(k)$
    $\Phi = \min_{\rho=i_k,\ldots,j_k} \sigma(\rho)$
    $pos = \min(\Psi(D_{\Gamma(k)}), ES_{i_k}, \Phi)$
    for $p = i_k$ ***to*** j_k **do**
      $ST_p \leftarrow ST_p - pos$
      $D_p \leftarrow D_p - pos$
    $ES_{i_k} \leftarrow ES_{i_k} - pos$
    if $j_k < N$ **then**
      $ES_{j_k+1} \leftarrow ES_{j_k+1} + pos$
  Calculate Γ, c
$Sol \leftarrow \sum_{i=1}^{N} (D_i \cdot PL_i)$
return Sol

Table 3.6. Initial landing times of all aircraft.

i	S_i	ET_i	TT_i	ST_i	LT_i	D_i	ES_i
1	0	89	98	496	510	398	407
2	8	96	106	504	521	398	0
3	8	110	123	512	555	389	0
4	8	120	135	520	576	385	0
5	8	124	138	528	577	390	0
6	8	126	140	536	573	396	0
7	8	135	150	544	591	394	0
8	15	129	155	559	559	404	0
9	15	160	180	657	657	477	0
10	15	195	258	744	744	486	0

their latest landing time while maintaining the safety constraint. With this initialization, all the aircraft have a positive deviation form their target landing time. The value of ES_i is equal to zero for all the aircraft except aircraft 1, for which ES is defined as its deviation from its earliest landing time, as shown in Equation (3.6). Thus, $ES_1 = ST_1 - ET_1 = 407$ and the penalty cost for this initialization is equal to 105710, where all the aircraft are scheduled to land as late as possible.

Following the initialization, we implement Algorithm 1, where all the landing times are reduced by maximum possible time units such that all the aircraft are either late or land at their target landing times. With the implementation of Algorithm 2 we obtain updated values for D_i and ES_i. Note that the value of PL for all the aircraft which land at their target landing times

Table 3.7. Improvement of the initial landing times using Algorithm 1.

i	ET_i	TT_i	ST_i	LT_i	D_i	ES_i	PL_i	PN_i
1	89	98	98	510	0	9	-30	0
2	96	106	106	521	0	0	-30	0
3	110	123	123	555	0	9	-30	0
4	120	135	135	576	0	4	-30	0
5	124	138	143	577	5	0	30	150
6	126	140	151	573	11	0	30	330
7	135	150	159	591	9	0	30	270
8	129	155	174	559	19	0	10	190
9	160	180	189	657	9	0	30	270
10	195	258	258	744	0	54	-10	0
								1210

Table 3.8. First iteration of the while loop in line 4 of Algorithm 2.

i	ET_i	TT_i	ST_i	LT_i	D_i	ES_i	PL_i	PN_i
1	89	98	98	555	0	9	-30	0
2	96	106	106	576	0	0	-30	0
3	110	123	123	577	0	9	-30	0
4	120	135	131	573	-4	0	-30	120
5	124	138	139	591	1	0	30	30
6	126	140	147	559	7	0	30	210
7	135	150	155	657	5	0	30	150
8	129	155	170	510	15	0	10	150
9	160	180	185	744	5	0	30	150
10	195	258	258	521	0	58	-10	0
								810

is negative by Definition 3.3. The values of PN_i in Table 3.7 are defines as the net weighted penalty incurred by any aircraft, where $PN_i = D_i \cdot PL_i$. Summation of PN_i values for all the aircraft is the value of Sol in line 3 of Algorithm 2, which is equal to 1210.

In the next step we calculate all the sets of Γ. Any set of Γ should hold all the properties mentioned in Definition 3.7. In this example, Γ has only one set of aircraft 4 to 9 which possess all the required properties and the value of c is equal to 1. Thus we have $k = 1, i_k = 4$ and $j_k = 9$, since there is only one set in Γ we have $c = 1$ which implies that the *for* loop at line 9 in Algorithm 2 will run only once with $k = 1$. We then need to calculate the value of *pos*. The value of $\Psi(D_{\Gamma(1)})$ is equal to 5 for aircraft 5 , $ES_{i_k} = 4$ and $\Phi = 15$, which implies that $pos = 4$. So the scheduled landing times ST of the aircraft and the deviation in the landing times D in the set $\Gamma(1)$ are reduced by 4. The value of ES_{j_k+1} is also increase by 4 as per line 15 in Algorithm 2 since $j_k < N$, along with an update to the PL. The new values of D, ES and PL are presented in Table 3.8. After this first iteration of the *while* loop we get $Sol = 810$.

Table 3.9. Second iteration of the *while* loop.

i	ET_i	TT_i	ST_i	LT_i	D_i	ES_i	PL_i	PN_i
1	89	98	98	555	0	9	-30	0
2	96	106	106	576	0	0	-30	0
3	110	123	122	577	-1	8	-30	30
4	120	135	130	573	-5	0	-30	150
5	124	138	138	591	0	0	-30	0
6	126	140	146	559	6	0	30	180
7	135	150	154	657	4	0	30	120
8	129	155	169	510	14	0	10	140
9	160	180	184	744	4	0	30	120
10	195	258	258	521	0	59	-10	0
								740

Table 3.10. Final iteration of the *while* loop.

i	ET_i	TT_i	ST_i	LT_i	D_i	ES_i	PL_i	PN_i
1	89	98	98	555	0	9	-30	0
2	96	106	106	576	0	0	-30	0
3	110	123	118	577	-5	4	-30	150
4	120	135	126	573	-9	0	-30	270
5	124	138	134	591	-4	0	-30	120
6	126	140	142	559	2	0	30	60
7	135	150	150	657	0	0	-30	0
8	129	155	165	510	10	0	10	100
9	160	180	180	744	0	0	-30	0
10	195	258	258	521	0	63	-10	0
								700

Following the same procedure, Γ again consists of a single set 3 to 9. The value of *pos* in this case is 1 and the update in the values of all the parameters for the aircraft in the new set of Γ will fetch us a solution value of 740 as shown in Table 3.9. The next iteration leads to another new set of Γ and this time it will again comprise of aircraft from 3 to 9. The value of *pos* is equal to 4 and the updates to all the parameters will now lead to the final solution of $Sol = 700$, as shown in Table 3.10. There is no further improvement possible since the set $\Gamma = \varnothing$. Hence, this value of $Sol = 700$ is also the optimum value for the '*airland1*' benchmark instance.

3.7 Proof of Optimality

In this section we explain and prove that Algorithm 2 gives the optimal value to the objective function for a special case of the ALP. We first prove a lemma which is later used in the proof of optimality of Algorithm 2.

Lemma 3.9. *If $\Gamma(k) \neq \varnothing$ then pos exists and $pos > 0$.*

Proof. From Algorithm 2, $pos = \min(\Psi(D_{\Gamma(k)}), ES_{i_k}, \Phi)$ holds. So pos will exist with a positive value only if $\Psi(D_{\Gamma(k)}) > 0$, $ES_{i_k} > 0$ and $\Phi > 0$. Clearly, $ES_{i_k} > 0$ from the definition of $\Gamma(k)$. Besides, $ES_m = 0$ for $m = i_k + 1, \ldots, j_k$ and $\sum_{m=i_k}^{j_k} PL_m > 0$ again from Definition 3.7. Note that we proved in Lemma 3.2 that if $ES_i = 0$ for any $i = 1, 2, 3, \ldots, N$ then $D_i \leq 0$. Moreover, $\sum_{m=i_k}^{j_k} PL_m > 0$ shows that for at least one aircraft m in the $\Gamma(k)$ has $PL_m > 0$. Recall from Equation (3.3) that for any aircraft m, $PL_m > 0$ only if $D_m > 0$. Thus, we have $ES_{i_k} > 0$ and $D_m > 0$ at least for one aircraft m, where $m = i_k, i_k + 1, \ldots, j_k$. Furthermore, if $\Gamma \neq \varnothing$ then obviously $\Phi > 0$ since the $\Phi = \min\limits_{\rho=i_k,\ldots,j_k} \sigma(\rho)$ and $\sigma(\rho) > 0$ for all the aircraft in the set $\Gamma(k)$ from Definition 3.7. Hence, this proves that pos will exist and will be greater than zero if $\Gamma(k) \neq \varnothing$. □

Theorem 3.10. *Algorithm 2 returns the optimal value for Equation* (3.10) *for any given feasible landing sequence on a single runway when* $SP_i = ST_{i-1} + S_{i-1,i}$ *for* $i = 2, 3, \ldots, N$ *and* $PS_i = ST_{i+1} - S_{i,i+1}$ *for* $i = 1, 2, \ldots, N-1$.

Proof. The initialization of the landing times for any sequence is done according to Lemma 3.1. It allocates the landing times as late as possible, hence the solution can be improved only by reducing the landing times. Thereafter, we show that we can reduce the landing time of any aircraft i straight away, independent of other aircraft, if $D_i > 0$ and $ES_i > 0$. The reason is, if there is an extra safety separation between $i-1$ and i as well as a positive deviation from the target landing time, then the reduction of ST_i by $\min\{D_i, ES_i\}$ will bring aircraft i closer to TT_i and hence yield an overall reduction in the total weighted tardiness thereby improving the overall solution. Note that this reduction will neither cause any aircraft to land earlier than its target landing time nor will it disrupt the safety separation. The implementation of Algorithm 1 will fetch one of the five possibilities to all the aircraft, mentioned and proved in Lemma 3.2.

The next step is to prove that a further improvement to the solution is possible iff $\Gamma \neq \varnothing$. If $\Gamma \neq \varnothing$, then we have $ES_{i_k} > 0$, $ES_m = 0, (m = i_k + 1, \ldots, j_k)$, $\sum_{\rho=i_k}^{j_k} PL_\rho > 0$, $\sigma(\rho) > 0$ where ρ are all the planes in the set $\Gamma(k)$ and $\sum_{\rho=\mu}^{j_k} PL_\rho > 0$, if μ exists. We have $ES_{i_k} > 0$ and $ES_m = 0, (m = i_k + 1, \ldots, j_k)$. Reducing the landing time of any aircraft in m will cause infeasibility as it will disrupt the safety constraint since $ES_m = 0$. But, reducing the landing times of all the aircraft from i_k to j_k by $pos = \min(\Psi(D_{\Gamma(k)}), ES_{i_k}, \Phi)$ will not cause any infeasibility for two reasons. First, the definition of $\Gamma(k)$ ensures that all the planes have a positive deviation from their earliest landing times since $\sigma(\rho) > 0$ and the reduction of the landing times by pos will not cause any infeasibility since all the aircraft in set $\Gamma(k)$ will be allocated a landing time within their time window since $pos \leq \Phi$. Second, we would reduce all the landing times by the same amount and not more than ES_{i_k}. This will maintain the safety separation between all the aircraft in $\Gamma(k)$ and also the required separation between aircraft $i_k - 1$ and i_k. Notice that PL_ρ is the net penalty of aircraft ρ as stated in Definition 3.3. Hence, a positive value for the summation of the net penalties of aircraft i_k to j_k landing consecutively means, that the total tardiness penalty is higher than the total earliness penalty and an increase in the landing times of all the aircraft in $\Gamma(k)$ by the same amount is only going to worsen the solution. As for μ, let's say there exists a μ for the set γ_k such that $\sum_{\rho=\mu}^{j_k} PL_\rho < 0$. This shows that aircraft μ to j_k already possess a net earliness penalty and further reducing their landing times will fetch an increase in the overall penalty. However, $\sum_{\rho=i_k}^{j_k} PL_\rho > 0$ means that $\sum_{\rho=i_k}^{\mu-1} PL_\rho > 0$ which implies that aircraft i_k to $\mu - 1$ possess a net positive tardiness penalty. Thus, a reduction in landing times by $\min(\Psi(D_{i_k:\mu-1}), ES_{i_k}, \Phi)$ will reduce the total weighted tardiness as well as ensure that the increase in the earliness penalty (if any) of aircraft i_k to $\mu - 1$ does not exceed the reduction in the net tardiness penalty and thereby reducing the overall penalty. In such a case $\Gamma(k)$ will become $(i_k : \mu - 1)$.

Conversely, if $\Gamma = \varnothing$, then either one of the cases will not hold in Definition 3.6 and 3.7. We prove this by contradiction for all these cases:

Case 1: $ES_{i_k} > 0$.
If $ES_{i_k} = 0$ and all the other conditions hold then there is no scope of reduction and an increase in the landing times will only worsen the solution. Note that ES_{i_k} will never be negative, for any i_k, $i_k = 1, 2, \ldots, N - 1$.

Case 2: $ES_m = 0, m = i_k + 1, \ldots, j_k$.
If $ES_m \neq 0, (m = i_k + 1, \ldots, j_k)$ then we have two cases. One, if for some m, $ES_m < 0$, then the solution is infeasible. Second, if for some m, $ES_m > 0$ then it contradicts the definition of Γ.

Case 3: $\sigma(\rho) > 0, \rho = i_k, \ldots, j_k$.
If the value of $\sigma(\rho) = 0$, then a reduction of the landing times for all the planes in the set $\Gamma(k)$ by any positive value will make the solution infeasible since the aircraft ρ is already landing at its earliest landing time. Note that the value of $\sigma(\rho)$ cannot be negative for any aircraft ρ at any stage.

Case 4: $\sum_{\rho=i_k}^{j_k} PL_\rho > 0$.
If $\sum_{\rho=i_k}^{j_k} PL_\rho = 0$ for any plane ρ, then any change to the landing times of all the aircraft in $\Gamma(k)$ will only worsen the solution by increasing the total lateness penalty or the total earliness penalty. If $\sum_{\rho=i_k}^{j_k} PL_\rho < 0$, then the reduction of landing times is again going to worsen the solution as the total earliness penalty is already higher than the total lateness penalty. Moreover, an increase in the landing time is not good either, because it will only take us back to an earlier step where $\sum_{\rho=i_k}^{j_k} PL_\rho > 0$. Hence, if $\Gamma = \varnothing$, then there is no more improvement possible to the schedule.

From Lemma 3.9 it is clear that *pos* exists and it is greater than positive if $\Gamma(k) \neq \varnothing$. This makes it clear, that if $\Gamma(k) \neq \varnothing$, then there needs to be reduction in the landing times of the planes in set $\Gamma(k)$. The reduction of the landing times is done by $pos = \min(\Psi(D_{\Gamma(k)}), ES_{i_k}, \Phi)$, because this will neither disrupt the safety constraint nor cause infeasibility. Besides, this will not alter the number of planes arriving early ($D_m < 0$). If we reduce the landing times by a greater quantity, we will certainly reduce the lateness penalty but we might as well end up increasing the earliness penalty by a greater amount. Hence we do not want to change the number of planes arriving early. Notice that a reduction in the landing time for aircraft j_k by pos means that it will increase the extra safety separation between j_k and $j_k + 1$, which is why we have line 14 in Algorithm 2. Hence, to summarize, Algorithm 2 initializes the latest possible landing times to all the aircraft and then makes improvements to the solution at every step until there is no improvement possible. □

3.8 Multiple Runways

In this section we propose an effective approach for allocating the runways to all the aircraft in a given landing sequence. We do not prove the optimality of this approach but our results show that it is an effective strategy and performs better than other approaches mentioned in the literature. In the multiple runway case, the only difference is the initial assignment of the runways to all the aircraft in a given sequence. We propose an initialization algorithm for the multiple runway case which again takes the input as a landing sequence of planes waiting to land and the number of runways R at the airport. We make an assumption as in Pinol and Beasley [121], that if aircraft i and j land on different runways, then the required safety distance (s_{ij}) between them is zero. Proposition (3.11) assigns the appropriate runway to all the aircraft and the landing sequence on each runway.

Proposition 3.11. *Assign the first R air planes $1, 2, \ldots, R$, one on each runway at their respective target landing times. For any following aircraft $i, i = R+1, R+2, \ldots, N$ assign the same runway as $i-1$ at a landing time of TT_i if TT_i is greater than or equal to the allowed landing time for plane i by maintaining the safety distance constraint with all the preceding aircraft on the same runway. Otherwise, assign a runway r at TT_i which offers zero deviation from TT_i. If none of the above two conditions hold, then select a runway which gives the least feasible positive deviation to plane i from its target landing time.*

Here we make an obvious assumption that the number of air planes waiting to land is more than the number of runways present at the airport. The landing sequence in this proposition is maintained in the sense that any aircraft i does not land before $i - 1$ lands. Once we have this assignment of aircraft to runways, each runway has a fixed number of planes landing in a known sequence. Using this to our benefit, we can now apply Algorithm 2 to each runway separately.

3.9 Algorithm Runtime Complexity

In this section we study and prove the runtime complexity of Algorithm 2.

Lemma 3.12. *The runtime complexity of Algorithm 2 is $O(N^3)$ for the general case of the safety constraint ($S_{i,i+2} > S_{i,i+1} + S_{i+1,i+2}$), and $O(N^2)$ for the special case of the safety constraint, when $S_{i,i+2} \leq S_{i,i+1} + S_{i+1,i+2}$, $\forall i = 1, 2, \ldots, N-2$.*

Proof. For the general case of the safety constraint, it is straight forward to observe that the runtime required to calculate ST and ES is $O(N^2)$. Calculating Γ requires finding all the sets of planes landing consecutively, such that they hold certain properties as mentioned in Definition 3.6 and 3.7. The worst case scenario for the calculation of Γ will occur when every aircraft lies in one of the sets of Γ. Let any set $\Gamma(k)$ have x_k aircraft where $k = 1, 2, \ldots, c$, then we have, $\sum_{k=1}^{c} x_k = N$, since all the sets of Γ are disjoint. The runtime for calculating a γ set from Definition 3.6 is $O(x_k)$. However, the computation of μ and checking $\sum_{\rho=\mu}^{j_k} PL_\rho > 0$ requires a computation of all the prior properties, if μ exists. In the worst case the value of j_k will drop down to $i_k + 1$ and this would require a total runtime of $O(x_k)$ where x_k is the number of aircraft in the set $\Gamma(k)$ obtained initially by the computation of the first four properties in Definition 3.7. Let T be the runtime of the computation of all the sets of Γ. Since all the properties are calculated in a sequential manner, we have, $T = \sum_{k=1}^{c} O(x_k)$. Now using $\sum_{k=1}^{c} x_k = N$ we get $T = O(N)$. The computation of PL and Sol in Algorithm 2 are both in $O(N)$ each. The *while* loop in line 3 involves several iterations so we first study the runtime of a single iteration of the *while* loop. The *for* loop in line 4 is run for the number of sets in Γ. Hence, the total runtime of the *for* loop is $\sum_{k=1}^{c} O(x_k)$, which is again equal to $O(N)$. The next steps inside the *while* loop involve the computation of Sol with a runtime of $O(N)$ and all the sets of Γ which requires $O(N)$ runtime each at every iteration. Since the computation of ES and Γ is carried out sequentially, the total runtime complexity of the algorithm is basically equal to $O(\lambda N^2)$, where λ is the number of times the *while* loop is iterated. Clearly, the maximum value of λ can be equal to the maximum number of aircraft in any set $\Gamma(k)$, which is equal to the total number of aircraft N. Hence the runtime complexity of Algorithm 2 is $O(N^3)$ for the general case of the safety constraint.

However, for the special case of the safety constraint when $S_{i,i+2} \leq S_{i,i+1} + S_{i+1,i+2}$, the computations of ES and ST along with Γ sets can be carried out in $O(N)$ runtime and thus reducing the overall complexity of Algorithm 2 to $O(N^2)$, for this case. □

3.10 Results and Discussion

We now present our results for the aircraft landing problem with single runways for the benchmark instances provided by Beasley in the OR-library [14].

We implement the algorithm as described above to find the optimal solution for the special case of the ALP in conjunction with Simulated Annealing (SA). In this work, the simulated annealing algorithm is used only to evolve the landing sequence of the given set of planes. While the landing times of the aircraft in a given landing sequence are computed by using our polynomial algorithm. Hence, at each iteration of the SA, a new landing sequence is generated via perturbation and the fitness function value of the perturbed sequence is calculated by Algorithm 2. As we have mentioned before, Algorithm 2 optimizes any given landing sequence such that the safety constraint is considered only between any aircraft and its preceding plane, not all the preceding aircraft. However, to avoid infeasibility, we incorporate a check of feasibility for the general case after any fitness function evaluation. If the solution obtained by Algorithm 2 is infeasible, it is discarded. However, in our tests we observe that this scenario occurs only for one instance with 50 aircraft. The rest of the benchmark instances have been solved feasibly by our polynomial algorithm.

The ensemble size for SA is taken to be 20 for all the instances. The initial temperature T_0 is kept as twice the standard deviation of the energy at infinite temperature, hence $T_0 = 2 \cdot \sqrt{\langle E^2 \rangle_{T=\infty} - \langle E \rangle^2_{T=\infty}}$. We estimate this quantity by randomly sampling the configuration space [125]. An exponential schedule for cooling is adopted with a cooling rate of 0.999. One of the modifications from the standard SA is in the acceptance criterion. We implement two acceptance criteria: the Metropolis acceptance probability, $\min\{1, \exp((-\triangle E)/T)\}$ [125] and a constant acceptance probability of $c_1 \cdot 10^{-2}$, c_1 being a constant with $c_1 < 10$. A solution is accepted with this constant probability if it is rejected by the Metropolis criterion. This concept of a constant probability is useful when the SA is run for many iterations and the metropolis acceptance probability is almost zero, since the temperature would become infinitesimally small. Apart from this, we also incorporate elitism in our modified SA. Elitism has been successfully adopted in evolutionary algorithms for several complex optimization problems [60, 82]. Lässig and Sudholt made theoretical studies analysing speed-ups in parallel evolutionary algorithms combinatorial optimization problems [87, 88]. As for the perturbation rule, we first randomly select a certain number of aircraft in any given landing sequence and permute them randomly to create a new sequence. The number of planes selected for this permutation is taken as $c_2 + \lfloor \sqrt{N/c_3} \rfloor$, such that N is the number of aircraft; c_2 and c_3 are positive constants. With our experimental analysis, we concluded that $c_2 = 3$ and $c_3 = 10$, works best for our algorithm. For large instances the size of this permutation is quite small but we have observed that it works well with our modified simulated annealing algorithm. We take the initial landing sequence for our algorithm as the sequence as per the order of their target landing times. As per the stopping criterion is concerned, Sabar and Kendall [124] propose a maximum of 150 iterations for their Iterative Local Search algorithm. For our PSA algorithm, we adopt a maximum number of 1500 Simulated Annealing iterations, given the fact that our polynomial algorithm runs fast and provides superior results to the state-of-the-art on this problem. Our Simulated Annealing algorithm is replicated 100 times for all the 49 instances, ranging from 10 to 500 aircraft. All the computations were carried out in MATLAB utilizing C++ mex-functions on a 1.73 GHz machine with 2 GB RAM. To better explain and compare our results we first define some new parameters used in Table 3.12 and 3.13. Most of these parameters are derived from [121] with slight changes as explained below.

Let,

SC = Results obtained by the Scatter Search Algorithm [121],
BA = Results obtained by the Bionomic Algorithm [121],
HBA = Results obtained by the Hybrid Bat Algorithm [144],
SA-VND = Results obtained by the Hybridized Simulated Annealing and Variable Neighbourhood Descent [126],
ILS = Results obtained by the Iterated Local Search Algorithm, proposed in [124],
PSA = Results obtained by the approach explained in this work,
Z_{opt} = Optimal solution value,
Z_{best} = Best known solutions for ALP provided in [121],
T_{run} = Average runtime in seconds,
G_{best} = Percentage deviation between the best obtained results and Z_{opt} if the optimal solution known and Z_{best} if the optimal solution is not known.

G_{best} is defined as $G_{\text{best}} = 100 \cdot (\text{best solution obtained} - Z_{\text{best}})/Z_{\text{best}}$; if the optimal solution is known then $Z_{\text{best}} = Z_{\text{opt}}$. However, if $Z_{\text{best}} = 0$ we follow the same notation as assumed in [121]. If $Z_{\text{best}} = 0$, then the value of $G_{\text{best}} = 0$ if the best solution obtained is also zero and n/d (not defined) if the best solution obtained is greater than zero. This definition of G_{best} is the same as explained by [121]. If for any instance the result obtained by us is better than the best known solution then G_{best} is negative. The values of Z_{best} are the best results obtained by [121] during the course of their work. The results shown in Table 3.12 and 3.13 are obtained by using Algorithm 2, Proposition 3.11 and simulated annealing depending on single or multiple runways. For the single runway case we use simulated annealing to generate the landing sequences and Algorithm 2 to optimize each sequence. For the multiple runway case we first generate a complete landing sequence of all the aircraft using simulated annealing, allocate the aircraft and their landing sequence to each runway using Proposition 3.11 and then apply Algorithm 2 to each runway separately for optimization. For brevity we call this approach *PSA*. Table 3.11 shows the configuration of the machines and the programming platform used by all the mentioned approaches. We follow this table from Sabar and Kendall [124], who present this data for fair comparison of the runtimes and as an indication of their algorithms efficiency.

Table 3.11. Computer hardware and programming platform for all the compared algorithms

No.	Algorithm	Language	Hardware
1	SC	C++	Intel 2 GHz Pentium, 512 MB RAM
2	BA	C++	Intel 2 GHz Pentium, 512 MB RAM
3	SA-VND	C++	Intel 2.4 GHz Pentium, 512 MB RAM
4	HBA	MATLAB	3 GHz AMD Athalon PC, 2 GB RAM
5	ILS	JAVA	Intel 2.66 GHz Pentium, 2 GB RAM
6	PSA	MATLAB	Intel 1.73 GHz, 2 GB RAM

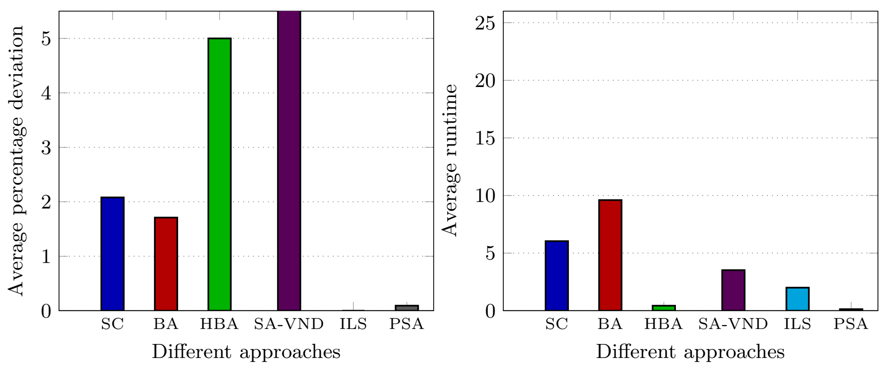
Before, we go on to present our result, we would like to emphasize here that the runtime provided by [121] in their paper for their algorithms, is the total time for 10 different replications of their algorithms (SC and BA), and not the average of 10 different runs. However, this fact has been misinterpreted as the latter, by some recent works. Hence, we in this work, present the runtimes of SC and BA, as 1/10th of the values mentioned in the results section of [121].

Table 3.12. Results for small benchmark instances and comparison of six different approaches.

			SC		**BA**		**HBA**		**SA-VND**		**ILS**		**PSA**	
N	**R**	$\mathbf{Z_{opt}}$	$\mathbf{G_{best}}$	$\mathbf{T_{run}}$	$\mathbf{G_{best}}$	$\mathbf{T_{run}}$	$\mathbf{G_{best}}$	$\mathbf{T_{run}}$	$\mathbf{G_{best}}$	$\mathbf{T_{run}}$	$\mathbf{G_{best}}$	$\mathbf{T_{run}}$	$\mathbf{G_{best}}$	$\mathbf{T_{run}}$
10	1	700	0	0.40	0	6.00	NA	NA	0	0.00	0	0.00	0	0.00
	2	90	0	2.40	0	4.50	0	0.08	0	0.00	0	0.00	0	0.00
	3	0	0	3.90	0	3.40	0	0.11	0	0.00	0	0.00	0	0.00
15	1	1480	0	0.60	0	9.00	NA	NA	0	1.59	0	0.00	0	0.06
	2	210	0	4.50	0	4.90	0	0.09	0	1.66	0	0.00	0	0.00
	3	0	0	4.60	0	4.30	0	0.10	100	1.98	0	0.00	0	0.00
20	1	820	0	0.80	0	9.90	NA	NA	0	1.78	0	0.00	0	0.09
	2	60	0	4.80	0	5.80	0	0.09	16.66	3.12	0	0.80	0	0.00
	3	0	0	6.20	0	6.30	0	0.10	100	3.29	0	0.10	0	0.00
20	1	2520	0	0.80	0	9.50	NA	NA	0	1.98	0	1.70	0	0.00
	2	640	0	5.20	0	5.50	0	0.55	3.12	3.56	0	1.90	0	0.00
	3	130	0	4.60	0	5.70	0	0.14	23.07	3.74	0	2.00	0	0.00
	4	0	0	5.60	0	5.20	0	0.14	100	4.06	0	2.30	0	0.00
20	1	3100	0	0.90	0	10.00	NA	NA	0	1.85	0	1.30	0	0.08
	2	650	0	5.00	3.08	6.10	36.92	1.44	0	3.04	0	2.40	0	0.35
	3	170	0	5.40	0	4.30	0	0.16	0	4.11	0	3.70	0	0.00
	4	0	0	5.60	0	6.80	0	0.21	100	4.35	0	3.10	0	0.00
30	1	24442	0	15.80	0	27.40	NA	NA	0	2.12	0	1.70	0	0.00
	2	554	0	7.00	3.61	10.10	14.8	1.61	0	3.98	0	2.60	0	0.47
	3	0	0	5.40	0	8.70	0	0.30	0	4.41	0	2.50	0	0.00
44	1	1550	0	19.50	0	7.90	NA	NA	0	2.68	0	1.80	0	0.00
	2	0	0	11.80	0	12.40	0	0.09	0	2.83	0	1.60	0	0.00
50	1	1950	52.05	4.20	36.15	28.70	NA	NA	0	7.10	0	4.80	2.31	1.45
	2	135	0	12.10	0	19.60	33.33	2.01	0	10.73	0	6.20	0	0.93
	3	0	0	13.90	0	18.10	0	0.16	100	14.11	0	9.50	0	0.00
Average			2.08	6.04	1.71	9.60	5.0	0.43	21.71	3.52	0	2.00	0.09	0.14

NA: Results not available.

Table 3.12 shows our results and its comparison with five other approaches for the small instances till 50 aircraft. Our approach is much faster and finds the optimal solution for all benchmark instances except for one. The reason that the optimum is found for all other instances is that the optimal sequences for all those instances hold the special case of the safety constraint, *i.e.*, the safety constraint for any aircraft depends only on its preceding plane. However, for the instance '*airland8*' with 50 aircraft and a single runway, our algorithm does not return the optimal solution as the optimal landing sequence does not satisfy the special case of the safety constraint. Figure 3.1(a) and (b) show the bar plots of the average percentage deviation and the average runtimes, respectively, for all the six approaches mentioned in Table 3.12. The closest algorithms to our approach are the HBA ba Xie *et al.* [144] and ILS by Sabar and Kendall [124] in terms of the runtime and the average percentage deviation. However, we would like to point out that HBA has only been applied to multiple runway problems and not on the single runway cases. Hence, the average runtime for HBA, plotted in Figure 3.1(b) is the runtime for multiple runways only, unlike other algorithms, which have been applied to all instances.



(a) Average percentage deviation for small instances

(b) Average runtime for small instances

Fig. 3.1. The comparison of six different approaches in the terms of the percentage deviations and the runtimes for the small instances till 50 aircraft.

Comparing the runtime of HBA exactly with our approach, (*i.e.* comparing the runtimes only for multiple runway cases) suggests that HBA takes 0.434 seconds as opposed to 0.10 seconds with our approach, which shows that our approach performs better than HBA in terms of both the solution quality and runtime. The ILS algorithm performs perfectly for small instances and finds optimal solution for all the 25 instances in Table 3.12. However, when compared for the runtime, it is 14 times slower than PSA. Considering the machines used by the two approaches, this speed-up owes to the two-layered approach of PSA. The average runtimes for Scatter Search, Bionomic Algorithm and the SA with Variable Descent are 6.04, 9.604 and 3.52 seconds, respectively, which makes our algorithm 43, 68 and 25 times faster than these algorithms, respectively, on the same benchmark instances. Moreover, considering the percentage deviation with the best known results reveals that our algorithm has a deviation of only 0.09 percentage over all the instances and finds optimal values for 24 out of 25 instances. Hence, for small instances, PSA is just as superior to other approaches when compared for both the solution quality and the required runtime. However, the real benefit of our two-layered approach is evident from our experiments on large instances with 100 to 500 planes.

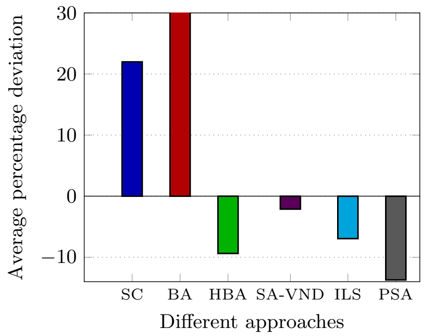
Table 3.13. Results for large benchmark instances and comparison of six different approaches.

		SC		BA		HBA		SA-VND		ILS		PSA	
N	**R**	G_{best}	T_{run}	G_{best}	T_{run}	G_{best}	T_{run}	G_{best}	T_{run}	G_{best}	T_{run}	G_{best}	T_{run}
100	1	30.06	11.9	14.51	55.4	NA	NA	8.55	11.6	0	7.6	0.00	5.67
	2	5.67	34.2	54.73	48.7	10.28	16.0	-0.58	13.8	-1.74	11.4	-1.95	3.88
	3	0	39.0	87.46	46.6	1.69	16.5	0	18.0	-2.31	10.9	0.00	0.36
	4	0	33.6	n/d	43.9	0	16.6	0	19.7	0	13.7	0.00	0.00
150	1	44.96	22.7	33.9	92.5	NA	NA	0	20.1	-0.06	14.3	2.20	11.96
	2	7.87	60.8	25.95	84.5	9.21	18.6	-5.39	21.3	-1.37	15.6	-10.83	8.83
	3	8.88	66.8	195.88	80.3	1.51	21.0	-6.49	27.6	-9.41	17.3	-7.06	3.24
	4	16.74	64.7	292.4	78.8	0	20.6	3.09	30.1	-6.16	22.7	0.00	1.12
	5	0	60.7	n/d	76.2	0	23.1	100	39.9	0	34.3	0.00	0.00
200	1	17.95	25.6	16.67	141.7	NA	NA	0	24.2	-0.05	18.4	1.67	18.54
	2	9.19	95.9	38.54	128.7	8.64	23.2	-8.04	29.1	-8.49	21.7	-12.38	13.77
	3	21.59	102.1	290.09	120.3	-0.06	26.1	-2.81	41.2	-3.46	34.2	-9.88	7.62
	4	2.77	99.3	474.47	116.8	0	27.4	0	42.4	-6.47	37.1	0.00	0.00
	5	0	95.6	n/d	115.8	0	27.2	0	66.2	0	54.8	0.00	0.00
250	1	22.15	38.1	23.58	201.1	NA	NA	0	219.0	0	197.7	3.75	32.39
	2	18.8	126.6	50.18	183.5	26.56	28.3	0	362.6	0	310.4	-13.38	22.22
	3	17.48	145.4	198.01	171.0	-15.95	31.5	-3.56	412.7	-6.21	401.5	-23.47	15.19
	4	271.63	144.5	13216.91	168.8	-30.09	33.3	0	410.3	-2.57	398.1	-30.09	0.24
	5	0	138.6	n/d	166.2	0	34.6	0	394.6	0	357.6	0.00	0.00
500	1	3.24	123.7	1.03	585.2	NA	NA	-7.54	566.8	-7.7	486.4	-10.57	153.21
	2	3.72	383.6	37.47	537.9	-5.78	58.0	-0.47	1047.9	-0.79	1011.2	-25.58	109.12
	3	1.98	456.0	182.69	515.8	-31.88	60.7	-32.79	1241.0	0	1123.4	-39.21	83.14
	4	22.98	441.3	1186.81	497.7	-52.23	63.7	-46.62	1201.8	-50.71	1181.2	-52.27	44.94
	5	0	442.1	22308.44	488.7	-100	65.9	-48.16	1203.9	-59.18	1152.4	-100.00	0.00
Average		21.9	135.5	1936.5	197.8	-9.4	32.3	-2.1	311.1	-6.945	288.91	-13.71	22.31

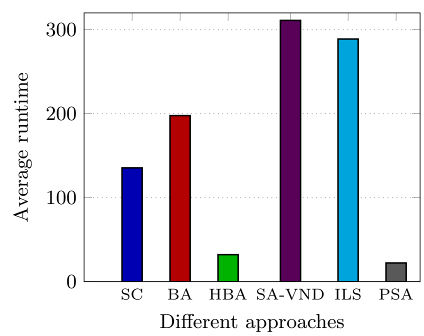
NA: Results not available.
n/d: Not defined.

Table 3.14. The best known solution values for large instances with single and multiple runways, as provided by Pinol and Beasley [121].

		100	**150**	**200**	**250**	**500**
R	1	5611.7	12329.31	12418.32	16209.78	44832.38
	2	452.92	1288.73	1540.84	1961.39	5501.96
	3	75.75	220.79	280.82	290.04	1108.51
	4	0	34.22	54.53	3.49	188.46
	5	0	0	0	0	7.35



(a) Average percentage deviation for large instances



(b) Average runtime for large instances

Fig. 3.2. The comparison of six different approaches in terms of the percentage deviations and the runtimes for large instances.

Table 3.13 presents the results for large instances for the six approaches. The optimal solutions of these instances are unknown and hence we compare our results with the best known solutions provided by [121], shown in Table 3.14. The average percentage deviation of PSA is −13.71 percent, which means that on average we achieve results that are 13.71 percent better than the best known results. The average runtime for PSA is 22.31 seconds which is almost 13 times faster than the recent works of Sabar and Kendall [124] and Salehipour *et al.* [126]. Yet again, we cannot compare our single runway results to HBA on the average basis, since Xie *et al.* [144] present their results for the multiple runway case only. However, for the multiple runway case, the average runtime for PSA is 16.51 seconds with a percentage gap of −17.16, whereas HBA requires 32.3 seconds to achieve a percentage deviation of −9.4 percent. The graphical comparison of the percentage deviations and runtimes can also be found in Figure 3.2. It is clear from the bar plots that on average, our approach is better than the state-of-the-art in both the runtimes and the percentage deviations.

Figure 3.3 shows the average percentage deviation of all the six approaches for every fleet size, irrespective of the number of runways. It can be seen that ILS is the closest to our approach, however, for large instances of 100 to 200, PSA performs just as well as ILS. Besides, for even larger fleet size of 250 and 500 aircraft, we outperform ILS and SA-VND by a good margin, and we obtain results that are on average superior than that of SA-VND and ILS. We do plot HBA in the figure, but the comparison is not fair, as the results for single runway case are not provided by HBA. Figure 3.4 provides the average runtime for each fleet size and clearly PSA is the best among all the instances, consistently.

Hence, we show that the use of our polynomial algorithm fetches faster and better results than previous approaches. We would like to mention here

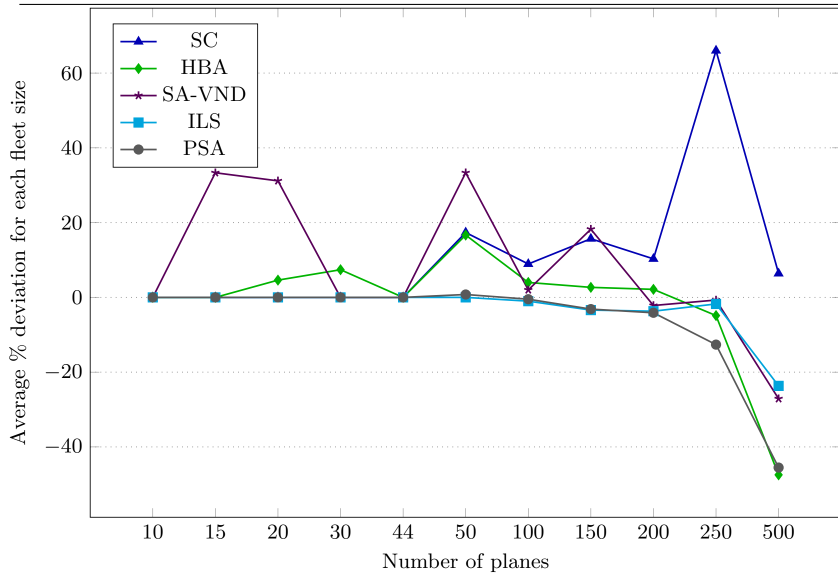


Fig. 3.3. Plot of average percentage deviation for each fleet size, comparing six different approaches.

that although we do not prove that Proposition (3.11) returns optimal results, nevertheless we obtain optimal solutions for all the small instances in much less time. For the large instances, the results again show that it is an effective approach and yields better results faster for all the instances. Interestingly, the runtime for multiple runways decreases significantly as the number of runways increases, as shown in Table 2. The reason for this observation is due to our approach. Since we implement our polynomial algorithm with the SA, we need $O(N^3)$ for each fitness function evaluation, for the single runway case. However, when the aircraft are divided on to R different runways (R being the number of runways), the total runtime required to optimize each runway is $O(N^2) + R \cdot O(N^3/R^3)$. Here, the first term corresponds to runway assignment, done in $O(N^2)$, in the worst case, and the second term corresponds to the runtime of optimizing the landing times on each runway. The runtime of $O(N^2 + N^3/R^2)$ is faster than $O(N^3)$ (runtime for single runway case) if $\frac{R^2}{R^2-1} < N$. In practice and in all the tested benchmark instances, this inequality always holds, since $1 < \frac{R^2}{R^2-1} \leq \frac{4}{3}$ for any $R \geq 2$.

In addition to these comparison of results with other recent approaches, we also present some measures of central tendencies, of our multiple runs of the Simulated Annealing algorithm on all the instances. As mentioned before, we carry out 100 different replications for all the benchmark instances. Hence, in Table 3.15 we present some measures of central tendency along with the

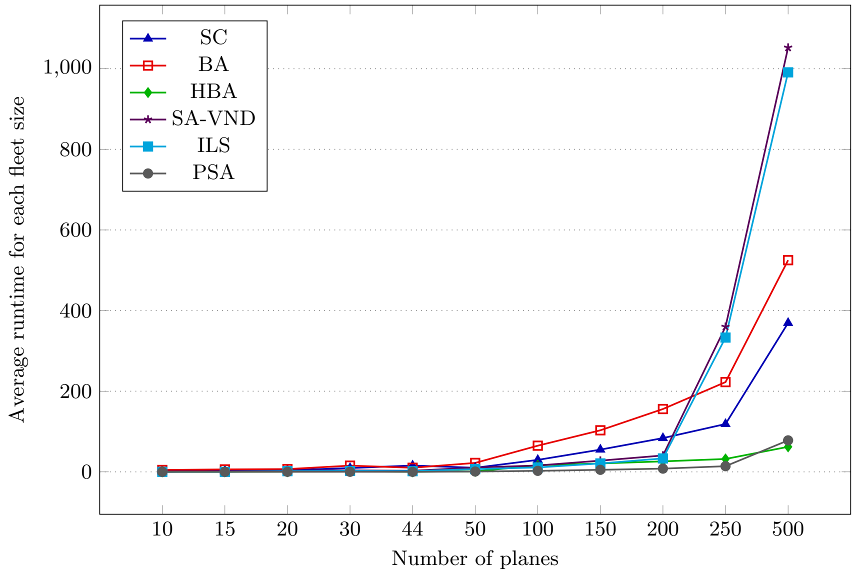


Fig. 3.4. Plot of the average runtimes for each fleet size, comparing six different approaches mentioned in the literature.

standard deviation and the number of fitness function evaluations required to obtain the results presented above. Table 3.15 shows the minimum, maximum, mean, median and mode of the percentage deviation for the results of Pinol and Beasley [121]. It also shows the standard deviation of the percentage deviation for the all the instances with different number of runways. As can be seen our standard deviation for any instance is less than or equal to 4.91 percent. Given, the complexity of the problem incorporating several parameters, our approach is quite robust and consistent over several benchmark instances. The fitness function evaluations are the average number of fitness functions which is Algorithm 2 in this case.

In Figure 3.5 We also present a graphical representation of the percentage deviation of the solution values obtained and the number of fitness function evaluations, along with their standard deviation, for problem instances with 50 aircraft and higher. Clearly, our algorithm produces results of higher quality as the problem size increases, owing to the exact methodology for optimizing any given landing sequence deterministically.

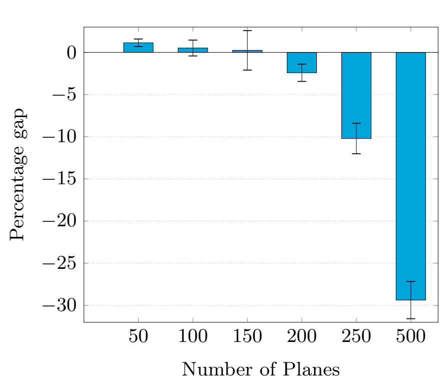
3.11 Summary

The Aircraft landing problem has mostly been approached using linear programming, metaheuristic approaches or branch and bound algorithms in the

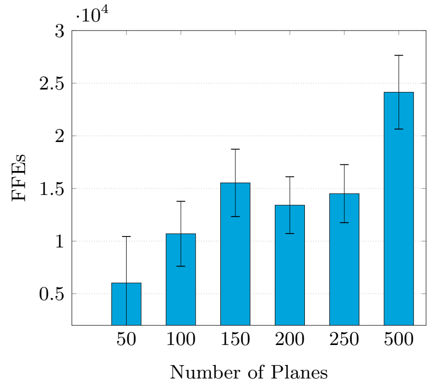
Table 3.15. Measures of central tendency and Standard Deviation of the obtained results and the total number of fitness function evaluations on average for all the instances.

N	R	Minimum	Maximum	Mean	Median	Mode	Std.	FFEs
10	1	0.00	0.00	0.00	0.00	0.00	0.00	20
	2	0.00	0.00	0.00	0.00	0.00	0.00	20
	3	0.00	0.00	0.00	0.00	0.00	0.00	20
15	1	0.00	0.00	0.00	0.00	0.00	0.00	2309
	2	0.00	0.00	0.00	0.00	0.00	0.00	20
	3	0.00	0.00	0.00	0.00	0.00	0.00	20
20	1	0.00	0.00	0.00	0.00	0.00	0.00	2483
	2	0.00	0.00	0.00	0.00	0.00	0.00	20
	3	0.00	0.00	0.00	0.00	0.00	0.00	20
20	1	0.00	0.00	0.00	0.00	0.00	0.00	20
	2	0.00	0.00	0.00	0.00	0.00	0.00	20
	3	0.00	0.00	0.00	0.00	0.00	0.00	20
	4	0.00	0.00	0.00	0.00	0.00	0.00	20
20	1	0.00	0.00	0.00	0.00	0.00	0.00	2276
	2	0.00	9.23	1.49	0.00	0.00	2.46	3588
	3	0.00	0.00	0.00	0.00	0.00	0.00	20
	4	0.00	0.00	0.00	0.00	0.00	0.00	20
30	1	0.00	0.00	0.00	0.00	0.00	0.00	20
	2	0.00	0.00	0.00	0.00	0.00	0.00	7688
	3	0.00	0.00	0.00	0.00	0.00	0.00	20
44	1	0.00	0.00	0.00	0.00	0.00	0.00	20
	2	0.00	0.00	0.00	0.00	0.00	0.00	20
50	1	2.31	8.46	3.42	3.85	3.85	1.34	11435
	2	0.00	0.00	0.00	0.00	0.00	0.00	6619
	3	0.00	0.00	0.00	0.00	0.00	0.00	20
100	1	0.00	9.90	2.84	2.46	0.00	2.08	25198
	2	-1.95	8.29	-0.77	-1.47	-1.95	1.67	15915
	3	0.00	0.00	0.00	0.00	0.00	0.00	1661
	4	0.00	0.00	0.00	0.00	0.00	0.00	20
150	1	2.20	14.84	9.10	9.24	10.34	3.16	28362
	2	-10.83	2.50	-6.32	-7.00	-10.83	2.83	24051
	3	-7.06	15.04	-1.57	-5.29	-5.29	5.7	21596
	4	0.00	0.00	0.00	0.00	0.00	0.00	3620
	5	0.00	0.00	0.00	0.00	0.00	0.00	20
200	1	1.67	10.52	5.86	6.03	6.44	1.75	27960
	2	-12.38	-0.96	-8.44	-8.78	-9.19	2.07	24210
	3	-9.88	-2.19	-9.50	-9.88	-9.88	1.29	14874
	4	0.00	0.00	0.00	0.00	0.00	0.00	20
	5	0.00	0.00	0.00	0.00	0.00	0.00	20
250	1	3.75	11.45	6.77	6.63	6.34	1.69	28331
	2	-13.38	-1.86	-9.41	-9.98	-12.53	2.43	24381
	3	-23.47	-3.20	-18.34	-20.20	-21.26	4.91	19443
	4	-30.09	-30.09	-30.09	-30.09	-30.09	0.00	372
	5	0.00	0.00	0.00	0.00	0.00	0.00	20
500	1	-10.57	-3.06	-7.08	-7.20	-5.65	1.87	29190
	2	-25.58	-19.77	-23.25	-23.28	-25.58	1.21	27972
	3	-39.21	-30.87	-36.64	-37.12	-37.77	1.92	24757
	4	-52.27	-27.47	-50.51	-52.27	-52.27	3.83	14656
	5	-100.00	-100.00	-100.00	-100.00	-100.00	0.00	20

last two decades [11, 52, 13, 144, 126, 124]. In this work, we use a two-layered approach to solve the ALP. We optimize any given feasible landing sequence with a polynomial algorithm and implement a modified Simulated Annealing to evolve the landing sequences. This approach is not a new one and has been utilized by a few researchers for the earliness/tardiness scheduling problem [49, 10, 132]. However, this work is the first attempt to schedule the



(a) Average percentage deviation and its standard deviation for fleet size of 50 and higher.



(b) Average number of fitness function evaluations and its standard deviation.

Fig. 3.5. The average percentage gap and standard deviation of PSA over any number of runways and fleet size of 50 and higher.

landings of the aircraft for a given feasible landing sequence using a polynomially bound algorithm. The benefit of this approach lies in the fact that the search space for any metaheuristic reduces considerably, and reduces to only finding a processing sequence. In the general sense, this idea fits with any NP-hard problem where the IP-formulation consists of a single decision variable. We demonstrate a specialized algorithm for the ALP along with appropriate illustrations. Specially for the Aircraft Landing Problem, it is sometimes inevitable to change the sequence of the aircraft landings at an airport due to weather conditions. In such a case, one needs to calculate the landing times of the aircraft for the known first-come-first-serve landing. This work provides an optimal algorithm for the special case of the safety constraint and a sub-optimal feasible solution for the general case of the safety constraint. As a matter of fact, our results show that we find better solutions for most of the benchmark instances than any other previous and recent research works. We demonstrate our results along with the hardware configuration used by other approaches, and the runtime and solution quality comparisons with prominent and recent works. Our algorithm for the special case of the safety constraint is also applicable for the general weighted earliness/tardiness scheduling problem with distinct release dates and deadlines, with no job pre-emption. The parameters involved in the ALP correspond directly to the parameters involved in the earliness/tardiness (E/T) job scheduling problem. The earliest landing time is the counterpart of the release date for a job, the safety distance between two consecutive landing aircraft corresponds to the sum of the processing time and set-up time of the job, the target landing time of any aircraft in the ALP is the due-date of any job in the E/T job scheduling problem and the latest landing time for an aircraft corresponds to the deadline of a job processing. Hence, our algorithm is just as well application to optimize any job sequence of the general E/T scheduling problem on a single machine and runs in $O(N^2)$ time, N being the number of jobs to be processed. In the next chapter we study the Common Due-Date problem. We present two polynomial algorithms to optimize a given job sequence.

4

Common Due-Date Problem: Exact Polynomial Algorithms for a Given Job sequence

In this chapter we present an extensive work on the Common due-date (CDD) scheduling problem on single and parallel machines. The CDD problem consists of scheduling jobs against a common due-date with an objective to minimize the weighted sum of the earliness and tardiness penalties. This scheduling problem has been proven to be NP-hard and we present two algorithms to optimize any job sequence for the CDD on a single machine. The first algorithm runs in time complexity of $O(n^2)$, n being the number of jobs. This algorithm is derived by reducing the common due-date problem to the aircraft landing problem discussed in the previous chapter. Thereafter, we make a theoretical study of the CDD and develop a faster algorithm which runs in $O(n)$ time. Additionally a simple heuristic based on the V-shaped property to improve a job sequence has also been proposed. Henceforth, the linear algorithm and the heuristic are used with a Simulated Annealing algorithm to obtain the optimal or best job sequence. Besides, it has also been proven that the linear algorithm is well suited for the dynamic case of the CDD. Furthermore, we also show that our approach for the parallel machine case is also equipped for non-identical parallel machines. Our solution approach is well tested on the benchmark instances along with the comparison with the best results in the literature and we find that our methodology can update 23 best-known solutions and significantly outperforms the state-of-the-art algorithms on several problem instances.

4.1 Introduction

The manufacturing industry often applies certain strategy to improve the efficiency of their productions, thereby avoiding both over and under production of their goods. Just-In-Time (JIT) is one such strategy which aims at maintaining the process efficiency while reducing any excess production. Scheduling against due-dates is based on the JIT philosophy, which aims to produce any good just at the *right-time*, neither too soon nor too late. For instance, any job at a machine shop has to be processed at a certain time only, to avoid any inventory cost if produced early, as well as minimizing customer dissatisfaction, if the completion of the job occurs later than the shipment deadline. In this work we deal with the problem of scheduling a given number of jobs on a single machine, against a common due-date.

An occurrence of the common due-date problem in an industry can be explained in the following manner. Suppose a small auto-mobile manufacturing industry is required to produce some 5000 units of cars of different models, on a certain date. The aim of the industry becomes not only to ship the requirements by the deadline, but also to reduce the inventory costs for the cars which are manufactured well ahead of the deadline. Obviously, this inventory cost can not be avoided completely, as all the cars can not be produced in a single day. On the other hand, if the units are not delivered on time, it leads to customer dissatisfaction and the industry has to bear some penalty due to late delivery of the unit(s). We can quantify the inventory cost and the cost of late delivery of the units as the earliness and tardiness penalties, respectively. Hence, the objective of the industry boils down to manufacturing the cars in a way that the total penalty involved due to earliness/tardiness is minimized so as to earn the highest possible profit from its manufactured cars. In practice, a Common due-date (CDD) problem occurs in almost any manufacturing industry adopting the JIT philosophy.

The Common due-date scheduling problem can be viewed as sequencing and scheduling of a certain number of jobs over a single machine against a common due-date (d). Each of these jobs possesses a required processing time along with the earliness/tardiness penalties per unit time in case the job is completed before or after the due-date. When scheduling on a single machine against a common due-date, at most only one job can be completed exactly at the due-date. Hence, some jobs will be processed earlier than the due-date while the others will finish later. Generally speaking, the CDD problem is categorized in two classes, each of which have proven to be NP-hard [66, 70]. A CDD problem is said to be *restrictive* when the optimal value of the objective function depends on the due-date of the problem instance. In other words, changing the due-date of the problem changes the optimal solution as well. However, in the *non-restrictive* case a change in the value of the due-date for the problem instance does not affect the solution value. It can be easily proved that in the restrictive case, the sum of the processing times of all the jobs is strictly greater than the due-date and in the non-restrictive case the sum of the processing times is less than or equal to the common due-date.

In this work we solve the CDD problem by breaking up the 0-1 integer programming in two-layers. One layer is solved using a specialized deterministic polynomial algorithm and the other layer is solved using metaheuristic algorithms. We propose two polynomial algorithms for the problem. One algorithm is developed by reducing the CDD to ALP while the second algorithm results from some intrinsic properties of the CDD problem. We make extensive theoretical analysis of the CDD problem and present important properties which are derived from the work of [35]. We also present an improvement heuristic to any given job sequence based on the V-shaped property. Henceforth, we utilize our linear algorithms and the heuristic for the CDD) in conjunction with Simulated Annealing to obtain the optimal/near-optimal solution to the studied NP-hard scheduling problems. The effectiveness and efficiency of our approach is presented via comparisons with previous results. The algorithms in the literature and this work are implemented and tested on the benchmark instances of the CDD provided in the OR-library [14].

4.2 Related Work

The common due-date problem has been studied extensively during the last 30 years, along with its several simplifications and variants. In 1981, Kanet presented an $O(n \log n)$ algorithm, n being the total number of jobs, for a simplified penalty function which only minimizes the total absolute deviation of the completion times of jobs but not the weighted earliness/tardiness [77]. Panwalkar *et al.* considered the problem of the common due-date assignment to minimize the total penalty for one machine. However, they again simplified the problem by considering constant earliness/tardiness penalties for all the jobs and developed an $O(n \log n)$ algorithm for the simplified case [128, 118]. Garey *et al.* also proposed an $O(n \log n)$ algorithm to solve the fixed-sequence problem with symmetric earliness/tardiness penalties [65]. Cheng again considered an easy variant of the CDD where the earliness and tardiness penalties were the same for each job. This simplification again led to a polynomial solution and a linear programming formulation was presented [35]. It should be noted here that all these works considered special simplified variants of the CDD problem which were no longer NP-Complete in nature. In this work, we propose a novel strategy where we present a heuristic solution to the NP-hard CDD problem which requires minimization of the weighted earliness/tardiness penalties with different asymmetric earliness/penalties, by solving one part of the problem polynomially and the other part by using the Simulated Annealing (SA) algorithm. More precisely, each job sequence is optimized in $O(n)$ time, while the generation of the best job sequence is carried out by the SA. Cheng and Kahlbachar [38] and Hall *et al.* [66] studied the CDD problem extensively, presenting some useful properties for the general case. These two properties have been of vital importance and have been exploited by many researchers over the years to device strategies for optimizing the CDD. This work also utilizes these properties to develop the $O(n)$ algorithm for a given job sequence, in addition to another property discussed later on.

Property 4.1. The optimal solution of the CDD has no machine idle time between any two jobs.

Proof. Refer to [38]. □

Property 4.2. Let t^* be the starting time for the first job and C_i be the completion time of job i, then for every instance of the CDD, there exists an optimal schedule with at least one of the following properties:
a) an optimal schedule with $t^* = 0$
b) an optimal schedule with $C_i = d$.

Proof. Refer to Hoogeveen and van de Velde [70]. □

A pseudo-polynomial algorithm with a runtime complexity of $O(n^2 d)$ was presented by Hoogeveen and van de Velde for the restrictive case with one machine when the earliness and tardiness penalty weights are symmetric for all the jobs [70]. Hall *et al.* studied the un-weighted earliness and tardiness problem and presented a dynamic programming algorithm [66]. Besides these earlier works, there have been some research on heuristic algorithms for the general common due-date problem with asymmetric penalty costs. James presented a tabu search algorithm for the general case of the problem [72]. In 2003, Feldmann and Biskup approached the problem using metaheuristic algorithms namely Simulated Annealing (SA) and threshold accepting (TA) and presented the results for the benchmark instances up to 1000 jobs on a single machine [20, 57]. Sourd and Sidhoum present a branch and bound algorithm for minimizing the earliness/tardiness of the jobs with the distinct due-dates and release dates for all the jobs [132].

In 2006, Pan *et al.* [115] proposed a discrete particle swarm optimization along with a heuristic algorithm based on the V-shaped property of the CDD, mentioned below.

Property 4.3. In the optimal schedule of the solution to the CDD problem, the jobs that are completed at or before the due date are sequenced in non-increasing order of the ratio P_i/α_i . On the other hand, jobs whose processing starts at or after the due date are sequenced in non-decreasing order of the ratio P_i/β_i. This property is also known as the V-shaped property of the CDD problem. [15]

A variable neighborhood search hybridized with Tabu Search was proposed by Liao and Cheng in 2007 [92]. In the same year, Tasgetiren *et al.* presented a Discrete Differential Evolution Algorithm for the CDD problem [136]. In 2008, Nearchou proposed a Differential Evolution approach [113]. Ronconi and Kawamura made a theoretical study on the lower bound of the CDD problem and proposed a branch and bound algorithm in 2010 for the general case of the CDD and gave optimal results for small benchmark instances till 20 jobs [123]. Another variant of the problem was studied by Toksari and Guner, where they considered the common due-date problem on parallel machines under the effects of time dependence and deterioration [138].

Kacem provides a polynomial time approximation to the total weighted tardiness problem against a common due-date [76]. In 2012, Rebai *et al.* proposed metaheuristic and exact approaches for the common due-date problem to schedule preventive maintenance tasks [122]. In 2013, Banisadr *et al.*

studied the single-machine scheduling problem for the case that each job is considered to have linear earliness and quadratic tardiness penalties with no machine idle time. They proposed a hybrid approach for the problem based upon evolutionary algorithm concepts [9]. The best and detailed results till date for the CDD problem have been proposed by Liu and Zhou in 2013, where the authors proposed a Population-based Harmony Search hybridized with Variable Neighborhood Search [97]. The authors provided detailed analysis of their results along with the exact solution values for all the benchmark instances. Xu *et al.* study the CDD problem on parallel machine with the objective to minimize the total weighted tardiness, depending on the start time of the jobs, and propose metaheuristic algorithms to solve the problem [145].

CDD has been extensively studied by many researchers and some useful properties have been proven. Cheng and Kahlbacher proved that in the optimal solution of the CDD machine has no idle time between any two jobs [38]. Hall *et al.* showed that if t^* is the starting time for the first job, then in the optimal schedule of every instance of the CDD, either $t^* = 0$ or $C_i = d$ for some i [66].

4.3 Problem Formulation

In this section we give the mathematical notation of the common due-date problem based on [20]. We also define some new parameters which are later used in the presented algorithm in the next section. Let,
n = number of jobs,
m = total number of machines,
n_j = number of jobs processed by machine j $(j = 1, 2, \dots, m)$,
d = common due-date,
P_i = actual processing time for job i, $\forall i = 1, 2, \dots, n$,
M_j = time at which machine j finished its previous job,
W_j^k = kth job processed by machine j,
C_i = completion time of job i,
g_i = earliness of job i, where $g_i = \max\{0, d - C_i\}$,
h_i = tardiness of job i, where $h_i = \max\{0, C_i - d\}$,
α_i = earliness penalty per time unit for any job i,
β_i = tardiness penalty per time unit for any job i.
The objective functions for the CDD problem can then be expressed as

$$\min \sum_{i=1}^{n} (\alpha_i \cdot g_i + \beta_i \cdot h_i) \ . \tag{4.1}$$

4.4 Motivation and Strategy for the Algorithms

In this section we present the intuition and the exact strategy for the developed algorithm for the sub-problem of CDD. As discussed in Chapter 2, the idea behind our approach is to break the integer programming formulation of these NP-hard problems in two parts, *i.e.*, (i) finding a good (near optimal) job sequence and (ii) finding the optimal values of the completion times C_i for all the jobs in this job sequence. Using the above parameters mentioned in Section 4.3, the mixed 0-1 integer programming (IP) formulation of the CDD can be presented as follows:

$$\text{Minimize} \sum_{i=1}^{n} (\alpha_i \cdot g_i + \beta_i \cdot h_i) \tag{4.2}$$

subject to

$$
\begin{aligned}
& C_1 \geq P_1, && \\
& C_i \geq P_i + C_j - G \cdot \delta_{ij}, && i = 1, \ldots, n-1, j = i+1, \ldots, n, \\
& C_j \geq P_j + C_i - G \cdot (1 - \delta_{ij}), && i = 1, \ldots, n-1, j = i+1, \ldots, n, \\
& g_i \geq d - C_i, && i = 1, \ldots, n, \\
& h_i \geq C_i - d, && i = 1, \ldots, n, \\
& g_i, h_i \geq 0, && i = 1, \ldots, n, \\
& \boldsymbol{\delta_{ij} \in \{0, 1\}} && i = 1, \ldots, n-1, j = i+1, \ldots, n.
\end{aligned}
$$

The variables have the same meaning as explained in Section 4.3, except for G and δ_{ij}. G is some very large positive number and δ_{ij} is the decision variable with $\delta_{ij} \in \{0, 1\}$, $i = 1, 2, \ldots, n-1, j = i+1, \ldots, n$. We have $\delta_{ij} = 1$ if job i precedes job j in the sequence (not necessarily right before it) and vice-versa. Hence any feasible set of values of δ_{ij} offers a feasible job sequence and we obtain a resultant linear program. In the course of this chapter, we make some theoretical analysis of the problem and develop two polynomial algorithms to the sub-problem of optimizing any given job sequence.

4.5 Polynomial Algorithm for CDD Job Sequence

We now present the ideas and the algorithm for solving the single machine case for a given job sequence, which is obtained by reducing the common due-date problem to the aircraft landing problem. We assume that there are n jobs to be processed by a machine and all the parameters stated in Section 4.3 represent the same meaning.

Lemma 4.4. *If the initial assignment of the completion times of the jobs, for a given sequence J is done according to C_i where,*

$$C_i = \begin{cases} \max\{P_1, d\} & \textit{if } i = 1 \\ C_{i-1} + P_i & \textit{if } 2 \leq i \leq n \ , \end{cases} \tag{4.3}$$

then the optimal solution for this sequence can be obtained only by reducing the completion times of all the jobs or leaving them unchanged.

Proof. We prove the above lemma by considering two cases of Equation (4.3).
Case 1: $d > P_1$
In this case Equation (4.3) will ensure that the first job is completed at the due-date and the following jobs are processed consecutively without any idle time. Moreover, with this assignment all the jobs will be tardy except for the first job which will be completed at the due-date. The total penalty (say, PN) will be $\sum_{i=1}^{n}(\beta_i \cdot h_i)$, where $h_i = C_i - d,\ i = 1, 2, \ldots, n$. Now if we increase the completion time of the first job by x units then the new completion times C_i' for the jobs will be $C_i + x \ \forall\ i, (i = 1, 2, \ldots, n)$ and the new total penalty PN' will be $\sum_{i=1}^{n}(\beta_i \cdot h_i^{'})$, where $h_i^{'} = h_i + x$ $(i = 1, 2, \ldots, n)$. Clearly, we have $PN' > PN$ which proves that an increase in the completion times cannot fetch optimality which in turn proves that optimality can be achieved only by reducing the completion times or leaving them unchanged from Equation (4.3).
Case 2: $d \leq P_1$
If the processing time of the first job in any given sequence is more than the due-date then all the jobs will be tardy including the first job as $P_1 > D$. Since all the jobs are already tardy, a right shift (*i.e.* increasing the completion times) of the jobs will only increase the total penalty hence worsening the solution. Moreover, a left shift (*i.e.* reducing the completion times) of the jobs is not possible either, because $C_1 = P_1$, which means that the first job will start at time 0. Hence, in such a case Equation (4.3) is the optimal solution. In the rest of the chapter we avoid this simple case and assume that for any given sequence the processing time of the first job is less than the due-date.
□

Before stating the algorithm we first introduce some new parameters, definitions and theorems which are useful for the description of the algorithm. We first define $DT_i = C_i - d,\ i = 1, 2, \ldots, n$ and $G_{start} = C_1 - P_1$. It is clear that DT_i is the algebraic deviation of the completion time of job i from the due-date and G_{start} is the maximum possible shift (reduction of completion time) for the first job.

Lemma 4.5. *Once C_i for each job in a sequence is assigned according to Lemma 4.4, a reduction of the completion times is possible only if $G_{start} > 0$.*

Proof. Lemma 4.4 proves that only a reduction of the completion times can improve the solution once the initialization is made as per Equation (4.3). Besides there is no idle time between any jobs, hence an improvement can be achieved only if $G_{start} > 0$, in which case all the jobs will be shifted left by equal amount. □

Definition 4.6. *PL is a vector of length n and any element of PL (PL_i) is the penalty possessed by job i. We define PL, as*

$$PL_i = \begin{cases} -\alpha_i, & \text{if } DT_i \leq 0 \\ \beta_i, & \text{if } DT_i > 0 \; . \end{cases} \tag{4.4}$$

With the above definition we can now express the objective function stated by Equation (4.1) as $\min(Sol)$, where Sol:

$$Sol = \sum_{i=1}^{n} (DT_i \cdot PL_i) \; . \tag{4.5}$$

Algorithm 3: Exact Algorithm for Single Machine

Initialize $C_i \; \forall \; i$ (Equation 4.3)
Compute PL, DT, G_{start}
$Sol \leftarrow \sum_{i=1}^{n} (DT_i \cdot PL_i)$
$j \leftarrow 2$
while $(j < n+1)$ **do**
    $C_i \leftarrow C_i - \min\{G_{start}, DT_j\}, \forall \; i$
    Update PL, DT, G_{start}
    $V_j \leftarrow \sum_{i=1}^{n} (DT_i \cdot PL_i)$
    if $(V_j < Sol)$ **then**
        $Sol \leftarrow V_j$
    else
        break
    $j \leftarrow j + 1$
return Sol

4.6 Proof of Optimality

Theorem 4.7. *Algorithm 3 finds the optimal solution for a single machine common due-date problem, for a given job sequence.*

Proof. The initialization of the completion times for a sequence P is done according to Lemma 4.4. It is evident from Equation (4.3) that the deviation from the due-date (DT_i) is zero for the first job and greater than zero for all the following jobs. Besides, $DT_i < DT_{i+1}$ for $i = 1, 2, 3, \ldots, n-1$, since $C_i < C_{i+1}$ from Equation (4.3) and DT_i is defined as $DT_i = C_i - d$. By Lemma 4.4 the optimal solution for this sequence can be achieved only by reducing the completion times of all the jobs simultaneously or leaving the completion times unchanged.

The total penalty after the initialization is $PN = \sum_{i=1}^{n}(\beta_i \cdot T_i)$ since none of the jobs are completed before the due-date. According to Algorithm 3 the completion times of all the jobs is reduced by $\min\{G_{start}, DT_j\}$ at any iteration. Since $DT_1 = 0$, there will be no loss or gain for $j = 1$. After any iteration of the *while* loop in line 5, we decrease the total weighted tardiness but gain some weighted earliness penalty for some jobs. A reduction of the completion times by $\min\{G_{start}, DT_j\}$ is the best non-greedy reduction. Let $\min\{G_{start}, DT_j\} > 0$ and t be a number between 0 and $\min\{G_{start}, DT_j\}$. Then reducing the completion times by t will increase the number of early jobs by one and reduce the number of tardy jobs by one. With this operation; if there is an improvement to the overall solution then a reduction by $\min\{G_{start}, DT_j\}$ will fetch a much better solution (V_j) because reducing the completion times by t will lead to a situation where none of the jobs either start at time 0 (because $G_{start} > 0$) nor any of the jobs finish at the due-date since the jobs $1, 2, 3, \ldots, j-1$ are early, jobs $j, j+1, \ldots, n$ are tardy and the new completion time of job j is $C_j^{'} = C_j - t$.

Since after this reduction $DT_j > 0$ and $DT_j < DT_{j+1}$ for $j = 1, 2, 3, \ldots, n-1$, none of the jobs will finish at the due-date after a reduction by t units. Moreover, it was proved by Cheng *et al.* [38] that in an optimal schedule for the restrictive common due-date, either one of the jobs should start at time 0 or one of the jobs should end at the due-date. This case can occur only if we reduce the completion times by $\min\{G_{start}, DT_j\}$. If $G_{start} < DT_j$ the first job will start at time 0 and if $DT_j < G_{start}$ then one of the jobs will end at the due-date. In the next iterations we continue the reductions as long as we get an improvement in the solution and once the new solution is not better than the previous best then we do not need to check any further and we have our optimal solution. This can be proved by considering the values of the objective function at two iterations indices; j and $j+1$. Let V_j and V_{j+1} be the value of the objective function at these two indexes then we can prove that the solution cannot be improved any further if $V_{j+1} > V_j$ by Lemma 4.8. □

Lemma 4.8. *Once the value of the solution at any iteration j is less than the value at iteration $j+1$, then the solution cannot be improved any further.*

Proof. If $V_{j+1} > V_j$ then it means that further left shift of the jobs does not fetch a better solution. Note that the objective function has two parts, penalty due to earliness and penalty due to tardiness. Let us consider the earliness and tardiness of the jobs after the j^{th} iterations are g_i^j and h_i^j for $i = 1, 2, \ldots, n$. Then we have $V_j = \sum_{i=1}^{n}(\alpha_i g_i^j + \beta_i h_i^j)$ and $V^{j+1} = \sum_{i=1}^{n}(\alpha_i g_i^{j+1} + \beta_i h_i^{j+1})$. Besides, after every iteration of the *while* loop in Algorithm 3, the completion times are reduced or in other words the jobs are shifted left. This leads to an increase in the earliness and a decrease in the tardiness of the jobs. Let's say, the difference in the reduction between V^{j+1} and V^j is x. Then we have $g^{j+1} = g^j + x$ and $h_{j+1} = h_j - x$. Since $V^{j+1} > V^j$, we have: $\sum_{i=1}^{n}(\alpha_i g_i^{j+1} +$

$\beta_i h_i^{j+1}) > \sum_{i=1}^{n}(\alpha_i g_i^j + \beta_i h_i^j)$. By substituting the values of g^{j+1} and h^{j+1} we get, $\sum_{i=1}^{j+1} \alpha_i x > \sum_{i=j+2}^{n} \beta_i x$. Hence, at the $(j+1)^{th}$ iteration the total penalty due to earliness exceeds the total penalty due to tardiness. This proves that for any further reduction there can not be an improvement in the solution because a decrease in the tardiness penalty will always be less than the increase in the earliness penalty. □

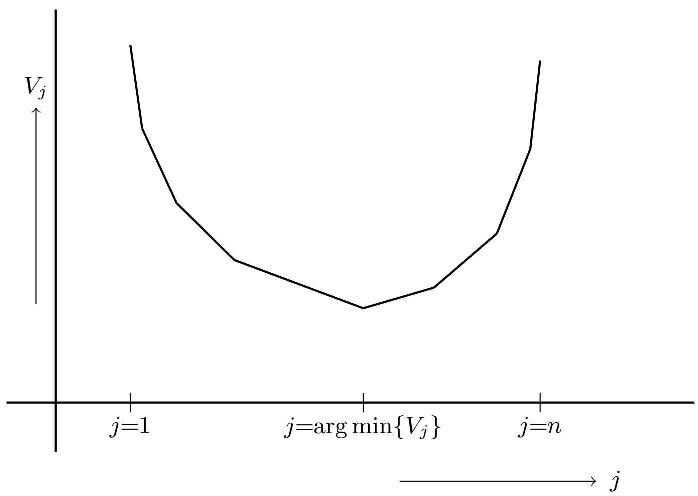


Fig. 4.1. The trend of the solution value against each iteration of Algorithm 3, for a job sequence. The value of the solution does not improve any further after a certain number of reductions.

4.7 Algorithm Run-Time Complexity

In this section we study and prove the runtime complexity of the Algorithm 3.

Lemma 4.9. *The runtime complexity of Algorithm 3 is $O(n^2)$ where n is the total number of jobs.*

Proof. For Algorithm 3 the calculations involved in the initialization step and evaluation of PL, DT, G_{start}, Sol are all of $O(n)$ complexity and their evaluation is irrespective of the any conditions unlike inside the *while* loop. The *while* loop again evaluates and updates these parameters at every step of its iteration and returns the output once their is no improvement possible. The worst case will occur when the *while* loop is iterated over all the values of j, $j = 2, 3, \ldots, n$. Hence the complexity of Algorithm 3 is $O(n^2)$ with n being the number of jobs processed by the machine. □

4.8 Exponential Search: An Efficient Implementation of Algorithm 3

Algorithm 3 shifts the jobs to the left by reducing the completion times of all the jobs by $\min\{G_{start}, DT_j\}$ on every iteration of the *while* loop. The run-time complexity of the algorithm can be improved form $O(n^2)$ to $O(n \log n)$ by implementing an exponential search instead of a step by step reduction, as in Algorithm 3. To explain this we first need to understand the slope of the objective function values for each iteration. In the proof of optimality of Algorithm 3, we proved that there is only one minimum present in V^j $\forall j$. Besides, the value of DT_j increases for every j as it depends on the completion times. Also note that the reduction in the completion times is made by $\min\{G_{start}, DT_j\}$. Hence, if for any j, $G_{start} \leq DT_j$ then every iteration after j will fetch the same objective function value, V^j. Hence the trend of the solution values after each iteration will have trend as shown in Figure 4.1.

With such a slope of the solution values (V_j), we can use the exponential search as opposed to a step by step search, which will in turn improve the runtime complexity of Algorithm 3. This can be achieved by increasing or decreasing the step size of the *while* loop by orders of 2 (*i.e.* $2, 2^2, 2^3, \ldots, n$) while keeping track of the slope of the solution. The index of the next iteration should be increased if the slope is negative and decreased if the slope is non-negative. At each step we need to keep track of the previous two indices and once the difference between the indices is less than the minimum of the two, then we need to perform binary search on the same lines. The optimum will be reached if both the adjacent solutions are greater than the current value. In this methodology we do not need to search for all values of j but in steps of 2^j. Hence the runtime complexity with exponential search will be $O(n \log n)$ for the single machine case.

In the next section we present some useful properties for the CDD which later help us to develop an algorithm with $O(n)$ complexity for finding the optimal completion times of the jobs in any given job sequence of CDD problem. We first present an extension of the property proved by [35] for the CDD with *symmetric* earliness/tardiness penalties. These properties are extended for the general CDD problem with asymmetric penalties.

4.9 Property for the CDD Job Sequence

We now present and prove a property for the CDD problem. This property is an extension to the one presented by Cheng [35], where the authors provide a theorem for the due-date assignments when there is a constant waiting allowance associated with each job. We prove a property for both the possible cases (restrictive and un-restrictive due-date) of the optimal schedule for the CDD problem. Furthermore, later in the chapter we describe how we combine the properties proved by Cheng and Kahlbachar [38] and Hall *et al.* [66]

with Theorem 4.10 for the general case of the CDD and present a linear algorithm for optimizing any given job sequence for both the restrictive and un-restrictive cases of the CDD.

Theorem 4.10. *If the optimal due-date position in any given job sequence of the CDD lies between C_{r-1} and C_r,* i.e.*, $C_{r-1} < d \leq C_r$, then the following relations hold for the two cases*
Case 1: If $C_{r-1} < d < C_r$

i) $\sum_{i=1}^{k-1} \alpha_i \leq \sum_{i=k}^{n} \beta_i, \; k = 1, 2, 3, \ldots, r$.

Case 2: If $C_r = d$

i) $\sum_{i=k+1}^{n} \beta_i \leq \sum_{i=1}^{k} \alpha_i, \; k = r, r+1, \ldots, n$ *and*
ii) $\sum_{i=1}^{k-1} \alpha_i \leq \sum_{i=k}^{n} \beta_i, \; k = 1, 2, 3, \ldots, r$.

Proof. We know from Property 4.2 that the optimal schedule of the CDD for any job sequence either has $t^* = 0$ or one of the job finishes at the due-date. Hence, we consider these two cases separately.
Case 1: optimal schedule with $C_{r-1} < d < C_r$
Let us first consider the case when the optimal schedule for any sequence lies

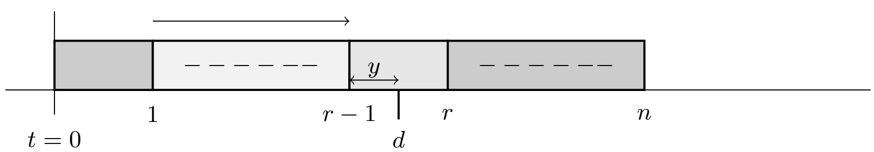


Fig. 4.2. Assume that the first job starts at time $t = 0$ and the due-date lies between the completion times of two consecutive jobs, with $y = d - C_r$.

strictly between C_{r-1} and C_r, *i.e.* $C_{r-1} < d < C_r$, as shown in Figure 4.2. We know from Property 4.2 that such a case can occur only when the first job starts at time $t = 0$ and all the following jobs are processed without any machine idle time. Let the difference between C_{r-1} and d be y such that $y = d - C_{r-1}$, as shown in Figure 4.2. Let g_i and h_i be the earliness and tardiness penalties of any job i, for this particular case, respectively. Hence, the solution value Sol_d for the schedule in Figure 4.2 can be written as

$$Sol_d = \sum_{i=1}^{r-1} g_i \cdot \alpha_i + \sum_{i=r}^{n} h_i \cdot \beta_i \; . \tag{4.6}$$

Now, the only possibility to get another schedule is to shift all the jobs to the right such that one of the jobs finishes at the due-date, as per Property 4.2.

Figure 4.3 shows the right shift of all the jobs by y units. It is clear that after this right shift of all the jobs, job $r-1$ offers no penalty. Hence, the earliness of the early jobs in Figure 4.3 will be $g_i - y$ for $i = 1, 2, \ldots, r-2$ and the tardiness of the tardy jobs will be $h_i + y$ for $i = r, r+1, \ldots, n$. We can now write the solution value for Figure 4.3 as Sol'_d where

$$Sol'_d = \sum_{i=1}^{r-2}(g_i - y)\cdot\alpha_i + \sum_{i=r}^{n}(h_i + y)\cdot\beta_i \ . \tag{4.7}$$

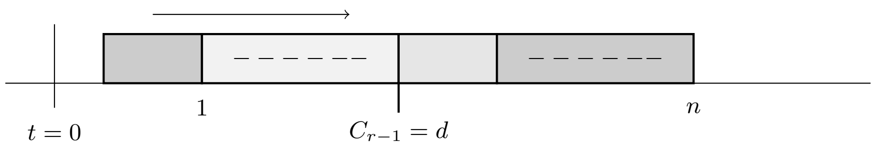


Fig. 4.3. Assume that the $(r-1)$th job finishes at the due-date d in the optimal schedule.

Since we already assumed that Figure 4.2 is the optimal schedule, we have

$$Sol_d \leq Sol'_d \ . \tag{4.8}$$

Note that in Figure 4.2, the earliness of job r is y. Hence Sol_d can be rewritten as

$$\begin{aligned} Sol_d &= \sum_{i=1}^{r-1} g_i\cdot\alpha_i + \sum_{i=r}^{n} h_i\cdot\beta_i \ , \\ &= \sum_{i=1}^{r-2} g_i\cdot\alpha_i + y\cdot\alpha_{r-1} + \sum_{i=r}^{n} h_i\cdot\beta_i \ . \end{aligned} \tag{4.9}$$

Likewise, the terms in Sol'_d can also be manipulated as

$$\begin{aligned} Sol'_d &= \sum_{i=1}^{r-2}(g_i - y)\cdot\alpha_i + \sum_{i=r}^{n}(h_i + y)\cdot\beta_i \ , \\ &= \sum_{i=1}^{r-2} g_i\cdot\alpha_i - \sum_{i=1}^{r-2} y\cdot\alpha_i + \sum_{i=r}^{n} h_i\cdot\beta_i + \sum_{i=r}^{n} y\cdot\beta_i \ . \end{aligned} \tag{4.10}$$

Substituting the value of Sol_d from Equation (4.9) and Sol'_d from Equation (4.10) in Equation (4.8), we get

$$\begin{aligned} y\cdot\alpha_{r-1} &\leq -\sum_{i=1}^{r-2} y\cdot\alpha_i + \sum_{i=r}^{n} y\cdot\beta_i \\ \sum_{i=1}^{r-1} y\cdot\alpha_i &\leq \sum_{i=r}^{n} y\cdot\beta_i \ . \end{aligned} \tag{4.11}$$

Since $y > 0$ due to the case constraint, Equation (4.11) fetches us

$$\sum_{i=1}^{r-1}\alpha_i \leq \sum_{i=r}^{n}\beta_i \ . \tag{4.12}$$

Clearly, if Equation (4.12) holds for any $k = r$, then it will also hold for any $k < r$, since α_i and β_i are positive for all i, $i = 1, 2, \dots, n$. This proves the first case of Theorem 4.10.

Case 2: optimal schedule at $C_r = d$

In this case we assume that the optimal solution lies at the completion time of some job r. Consider Figure 4.4, where the optimal schedule occurs with the due-date position at the completion time of job r, *i.e.* $C_r = d$. Let, g_i and

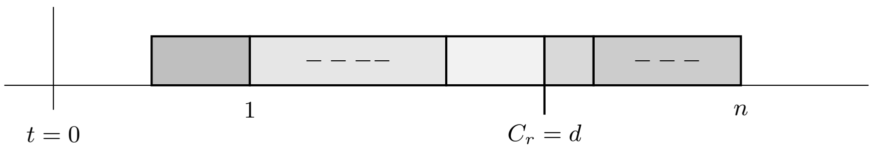


Fig. 4.4. Assume that the rth job finishes at the due-date d in the optimal schedule.

h_i be the earliness and tardiness of any job i, respectively, for this particular case (Figure 4.4) and the solution value for this case be Sol_r, then using Equation (4.1) we have

$$Sol_r = \sum_{i=1}^{r-1} g_i \cdot \alpha_i + \sum_{i=r+1}^{n} h_i \cdot \beta_i \; . \tag{4.13}$$

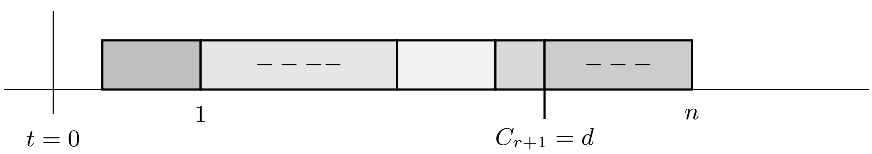


Fig. 4.5. Schedule with the completion time of job $r + 1$ lying at the due-date, $C_{r+1} = d$.

Let the solution value for the case when all the jobs are shifted to the left by P_{r+1}, *i.e.*, the $(r + 1)$th job ends at the due-date, be Sol_{r+1}, see Figure 4.5. Then the earliness of jobs 1 to $r - 1$ will increase by the processing time of job $r + 1$, compared to Figure 4.4, since the due-date position shifts to right by the same amount and job r will be early by P_{r+1}. Besides, job $r + 1$ offers no penalty and the tardiness of the all the jobs from $r + 2$ to n reduces by P_{r+1}. Hence, the objective function value when the due-date is situated at C_{r+1} becomes

$$Sol_{r+1} = \sum_{i=1}^{r-1} (g_i + P_{r+1}) \cdot \alpha_i + P_{r+1} \cdot \alpha_r + \sum_{i=r+2}^{n} (h_i - P_{r+1}) \cdot \beta_i \; . \tag{4.14}$$

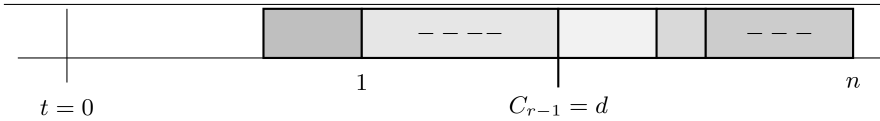


Fig. 4.6. Schedule with the completion time of job $r-1$ lying at the due-date, $C_{r-1} = d$.

Likewise, when all the jobs are shifted to the right such that the $(r-1)$th job finishes at the due-date, in comparison to Figure 4.4, then jobs 1 to $r-2$ will have their earliness reduced by P_r, job r will be tardy by P_r and the all the jobs from $r+1$ to n will have their tardiness increased by P_r. Let the solution value for Figure 4.6 where the $(r-1)$th job ends at the due-date be Sol_{r-1}, then

$$Sol_{r-1} = \sum_{i=1}^{r-2}(g_i - P_r)\cdot\alpha_i + P_r\cdot\beta_r + \sum_{i=r+1}^{n}(h_i + P_r)\cdot\beta_i \ . \tag{4.15}$$

Since we assume that Sol_r is the optimal value, we have,

$$Sol_r \quad \leq \quad Sol_{r+1}, \text{ and} \tag{4.16}$$

$$Sol_r \quad \leq \quad Sol_{r-1} \ . \tag{4.17}$$

Notice that in the first case, when $C_r = d$, the tardiness of job $r+1$ is P_{r+1} and the earliness of job $r-1$ is P_r. Hence, rearranging the terms in Sol_r we get,

$$\begin{aligned} Sol_r &= \sum_{i=1}^{r-1} g_i\cdot\alpha_i + \sum_{i=r+1}^{n} h_i\cdot\beta_i \\ &= \sum_{i=1}^{r-1} g_i\cdot\alpha_i + P_{r+1}\cdot\beta_{r+1} + \sum_{i=r+2}^{n} h_i\cdot\beta_i \ . \end{aligned} \tag{4.18}$$

Splitting the earliness penalty of job $r-1$, Sol_r can also be expressed as

$$\begin{aligned} Sol_r &= \sum_{i=1}^{r-1} g_i\cdot\alpha_i + \sum_{i=r+1}^{n} h_i\cdot\beta_i \\ &= \sum_{i=1}^{r-2} g_i\cdot\alpha_i + P_r\cdot\alpha_{r-1} + \sum_{i=r+1}^{n} h_i\cdot\beta_i \ . \end{aligned} \tag{4.19}$$

Substituting the values of Sol_r from Equation (4.18) and Sol_{r+1} from Equation (4.14) in Equation (4.16) we get

$$\begin{gathered} Sol_r \leq Sol_{r+1} \ , \\ \sum_{i=r+1}^{n} P_{r+1}\cdot\beta_i \leq \sum_{i=1}^{r} P_{r+1}\cdot\alpha_i, \text{ and} \\ \sum_{i=r+1}^{n} \beta_i \leq \sum_{i=1}^{r} \alpha_i \ . \end{gathered} \tag{4.20}$$

Likewise, substituting the values of Sol_r from Equation (4.19) and Sol_{r-1} from Equation (4.15) in Equation (4.17),

$$\begin{aligned} Sol_r &\leq Sol_{r-1} \ , \\ \sum_{i=1}^{r-1} P_r \cdot \alpha_i &\leq \sum_{i=r}^{n} P_r \cdot \beta_i, \text{ and} \\ \sum_{i=1}^{r-1} \alpha_i &\leq \sum_{i=r}^{n} \beta_i \ . \end{aligned} \tag{4.21}$$

Since α_i and β_i are positive for all i, Equation (4.20) also implies,

$$\sum_{i=k+1}^{n} \beta_i \leq \sum_{i=1}^{k} \alpha_i, \quad k = r, r+1, \ldots, n \ , \tag{4.22}$$

i.e., if the sum of the tardiness penalties for the jobs $(r+1)$ to n is less than the sum of the earliness penalties for the jobs from 1 to r, then the same inequality also holds for any $k \geq r$, since $\beta_i > 0$ and $\alpha_i > 0$ for $i = 1, 2, \ldots, n$. Likewise, Equation (4.21) implies that

$$\sum_{i=1}^{k-1} \alpha_i \leq \sum_{i=k}^{n} \beta_i, \quad k = 1, 2, \ldots, r \ , \tag{4.23}$$

i.e., if the sum of the earliness penalties for the jobs 1 to $(r-1)$ is less than the sum of the tardiness penalties for the jobs from r to n, then the same inequality also holds for any $k \leq r$, since $\beta_i > 0$ and $\alpha_i > 0$ for $i = 1, 2, \ldots, n$. Equation (4.22) and (4.23) prove that the difference of the sum of the earliness and the sum of the tardiness penalties changes sign before and after the optimal position of the due-date, provided the due-date position in the optimal solution lies at completion time of a job. □

4.10 Linear Algorithm for the CDD Job Sequence on a Single Machine

We now present the ideas and the algorithm for solving the single machine CDD problem for a given job sequence, mentioned in Algorithm 4. The intuition for our approach comes from the properties presented and proved by Cheng and Kahlbachar [38] (Property 4.1), Hall *et al.* [66] (Property 4.2) and Theorem 4.10. Cheng and Kahlbachar proved that the machine has no idle time between processing of any two jobs and Hall *et al.* proved that in the optimal schedule either the first job starts at time $t = 0$ or one of the jobs finishes processing at the common due-date. Besides, we proved in Theorem 4.10 that the difference in the sum of the tardiness and earliness penalties changes sign before and after the optimal due-date position, if the optimal solution has the due-date position at the completion time of a job. Now using Property 4.2

Algorithm 4: Linear algorithm to optimize a given job sequence of a CDD instance.

$C_i \leftarrow \sum_{k=1}^{i} P_k \;\forall\; i = 1, 2, \ldots, n$
$DT_i \leftarrow C_i - d \;\forall\; i$
$\tau \leftarrow \underset{i=1,2,\ldots,n}{\arg\max}\,(DT_i \leq 0)$
$l \leftarrow \tau$
if $(\tau \neq 0)$ **then**
  $pe \leftarrow \sum_{i=1}^{\tau} \alpha_i$
  $pl \leftarrow \sum_{i=\tau+1}^{n} \beta_i$
  $t \leftarrow 0$
  if $(DT_\tau < 0) \wedge (pl < pe)$ **then**
    $DT_i \leftarrow DT_i - DT_\tau \;\forall\; i$
  while $(\tau > 0) \wedge (pl < pe)$ **do**
    $pe \leftarrow pe - \alpha_\tau$
    $pl \leftarrow pl + \beta\tau$
    $t \leftarrow 1$
    $l \leftarrow \tau$
    $\tau \leftarrow \tau - 1$
  if $(t = 1)$ **then**
    $DT_i \leftarrow DT_i - DT_l \;\forall\; i$
$C_i \leftarrow DT_i + d,\; i = 1, 2, \ldots, n$
$g_i \leftarrow \max\{d - C_i, 0\},\; i = 1, 2, \ldots, n$
$h_i \leftarrow \max\{C_i - d, 0\},\; i = 1, 2, \ldots, n$
return $Sol \leftarrow \sum_{i=1}^{l} \alpha_i \cdot g_i + \sum_{i=l+1}^{n} \beta_i \cdot h_i$

and Theorem 4.10, it becomes clear that the optimal solution of any job sequence will either start at time $t^* = 0$ or possess the two properties proved in Theorem 4.10. Hence, it is evident that to achieve the optimal solution we must first start scheduling the jobs from time $t = 0$.

Let J be the input job sequence where J_i is the ith job in the sequence J. Note that without loss of any generality we can assume $J_i = i$, since we can rank the jobs for any sequence as per their order of their processing. Our algorithm first assigns the initial completion times to all the jobs such that the first job starts at time $t = 0$ and the rest of the jobs follow without any idle time, *i.e.* $C_i = \sum_{k=1}^{i} P_k$.

If the sum of the tardiness penalties is already greater than the sum of the earliness penalties then we know that this initialization is the optimal solution, as well. The reason is clear from Case 1 of Theorem 4.10, which basically states that the sum of the earliness penalties will be less than or equal to the sum of the tardiness penalties for the maximum value of k. Evidently, the jobs can not be shifted to the left any further and hence the maximum value of k will occur for the initial schedule with the first job starting at time $t = 0$. However, if the sum of the tardiness penalties is less than or equal to the sum of earliness penalties, then we shift all the jobs towards increasing completion times by placing the due-date position at the end of the completion times of jobs sequentially as long as the second property of Theorem 4.10 Case 2 holds. This procedure is continued until the sum of the tardiness penalties remains less than or equal to the sum of the earliness penalties.

We further explain Algorithm 4 with the help of an illustrative example consisting of $n = 5$ jobs. We optimize the given sequence of jobs J where $J_i = i, i = 1, 2, \ldots, 5$. The data for this example is given in Table 4.1. There are five jobs to be processed against a common due-date (d) of 16. The objective is to minimize Equation (4.1). We first initialize the completion times of all the

Table 4.1. The data for the exemplary case of the CDD problem. The parameters possess the same meaning as explained in Section 4.3.

$\boldsymbol{i}$	$\boldsymbol{P_i}$	$\boldsymbol{\alpha_i}$	$\boldsymbol{\beta_i}$
1	6	7	9
2	5	9	5
3	2	6	4
4	4	9	3
5	4	3	2

jobs (C_i, $i = 1, 2, , \ldots, n$), such that $C_i = \sum_{k=1}^{i} P_k$ as shown in Figure 4.7. Hence, we have $C_i = \{6, 11, 13, 17, 21\}$. The first job starts processing at time $t = 0$ and the following jobs are processed without any machine idle time. The due-date position lies in between the completion times of job 3 and 4. In the next step we compute the vector $DT_i = C_i - d$, which gives

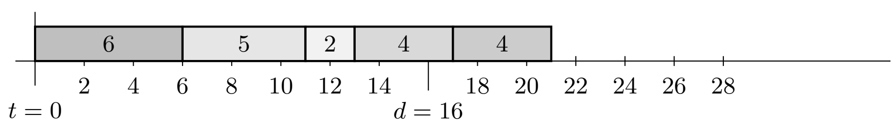


Fig. 4.7. Initialization of the schedule with the first job starting at time $t = 0$ and the remaining jobs following with no machine idle time.

us $DT_i = \{-10, -5, -3, 1, 5\}$. Notice that vector DT_i fetches us the earliness and tardiness values of the jobs with the negative values for the earliness and the positive values for the tardiness. We then calculate the maximum index τ of $DT_i \leq 0$ or in other words, maximum index of the job which is either early or finishes at the due-date. In this example we have $\tau = 3$ and $l = 3$. Since $\tau \neq 0$, we calculate the sum of the earliness and the tardiness of the jobs as indicated by line 6 and 7 of Algorithm 4. Hence, $pe = 22$ and $pl = 5$. In the next step we shift all the jobs by DT_τ to check if the property (ii) of Case 2 in Theorem 4.10 holds. After a right shift of 3 units, we have

$DT_i = \{-7, -2, 0, 4, 8\}$ and the 3rd job finishes at the due-date, as shown in Figure 4.8. Since this schedule still satisfies $pl < pe$, we again shift the jobs to

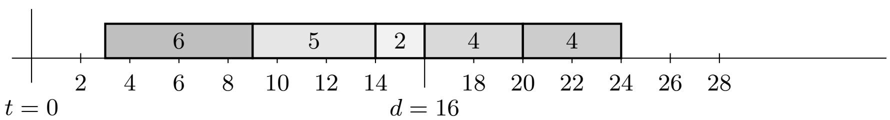


Fig. 4.8. Schedule with the completion time of job 3 lying at the due-date, after the right shift of all the jobs by 3 units.

the right, this time by the processing time of job 3. This also means that the 3rd job will now be tardy implying that the sum of the earliness penalty (pe) will reduce by $\alpha_\tau = 6$ and the sum of the tardiness penalties will increase by $\beta_\tau = 4$. Hence from lines 12 and 13 of the algorithm, we have $pe = 16$ and $pl = 9$. Figure 4.9 shows the schedule after the second right shift of the jobs with job 2 finishing at the due-date. After updating the values of t, l and τ, we have $t = 1$, $l = 3$ and $\tau = 2$. Yet again $pl < pe$, which calls for another right shift such that the first job now finishes at the due-date. We again update the values inside the *while* loop and we have $pe = 7$, $pl = 14$, $t = 1$, $l = 2$ and $\tau = 1$. Since, $pl > pe$, the while loop condition is not satisfied anymore and we update vector DT_i since $t = 1$. The value $t = 1$ simply implies that there was a right shift of the jobs and l signifies the job till which the shifts were made. In this case $l = 2$ and hence we end up with $DT_i = \{-5, 0, 2, 6, 10\}$, as is clear from Figure 4.9. Finally, we multiply the corresponding penalties with each vector element of DT_i and the sum of the resultant vector gives us the objective function value. Clearly we do not need to update the DT_i vector in

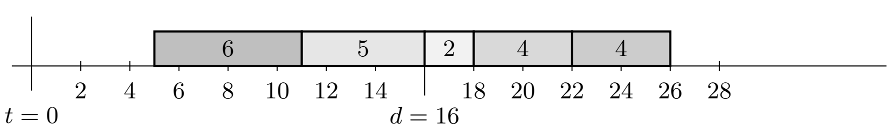


Fig. 4.9. Schedule with the completion time of job 2 lying at the due-date, after an additional right shift of all the jobs by 2 units.

every shift inside the *while* loop, but only update the sum of the penalties, because the right shift is always equal to the processing time of the closest early job to the due-date. It is clear from this illustration that due to the right shifting nature of the algorithm, we do not need to check for the first property in every step but only the second property, as is adopted in Algorithm 4. And once it is satisfied we have our optimal schedule.

4.11 Proof of optimality and Runtime Complexity of Algorithm 4

Theorem 4.11. *Algorithm 4 returns the optimal solution value for a given sequence of the Common due-date problem with linear time runtime complexity.*

Proof. Since there is only one way that the due-date position may be between the completion times of two consecutive jobs, we need to first calculate the sum of penalties before and after the due-date such that the first job starts at time zero and all the jobs follow without any machine idle time. The schedule with $t^* = 0$ will be optimal if the sum of the tardiness penalties is already greater than the sum of earliness penalties, due to Case 1 of Theorem 4.10, which states that the sum of the tardiness penalties will be less than or equal to the sum of the earliness penalties for maximum value of k. If that is not the case, we shift all the jobs towards right, as long as the sum of the tardiness penalties of jobs finishing after the due-date is less than or equal to the some of the earliness penalties of all the jobs which complete before the due-date, according to Theorem 4.10.

As for the runtime complexity, the calculations involved in the initialization step and the evaluation of DT are both of linear time. All the steps inside the *while* loop are of constant time. Evaluation of DT_i and Sol are again of complexity $O(n)$, but they are calculated only once. Hence the overall complexity of Algorithm 4 is $O(n)$. □

4.12 Parallel Machine Case

For the parallel machine case we first need to assign the jobs to each machine to get the number of jobs and their sequence in each machine. In addition to the parameters explained in Section 4.3, we define a new parameter λ, which is the machine assigned to each job, as mentioned in [4].

Definition 4.12. *We define λ as the machine which has the earliest scheduled completion time of the last job on that machine. Using the notation mentioned in Section 4.3, λ can be mathematically expressed as*

$$\lambda = \underset{j=1,2,\ldots,m}{\operatorname{argmin}} \; M_j \ .$$

Algorithm 5 assigns the first m jobs to each machine respectively such that they all finish processing after their processing time. For the remaining jobs, we assign a machine λ to job i since it offers the least completion time. Likewise each job is assigned at a specific machine such that the tardiness for all the jobs is the least for the given job sequence. The job sequence is

Algorithm 5: Exact Algorithm: Parallel Machine

$M_j \leftarrow 0 \; \forall j = 1, 2, \ldots, m$
$n_j \leftarrow 1 \; \forall j = 1, 2, \ldots, m$
$i \leftarrow 0$
for $j \leftarrow 1$ **to** m **do**
    $i \leftarrow i + 1$
    $W_j^1 \leftarrow i$
    $M_j \leftarrow P_i$
for $i \leftarrow m + 1$ **to** n **do**
    Compute λ
    $n_\lambda \leftarrow n_\lambda + 1$
    $W_\lambda^{n_\lambda} \leftarrow i$
    $M_\lambda \leftarrow M_\lambda + P_i$
for *each machine* **do**
    Algorithm 4

maintained in the sense that for any two jobs i and j such that job j follows i; the Algorithm 5 will either maintain this sequence or assign the same starting times at different machines to both the jobs. Finally, Algorithm 5 will give us the number of jobs (n_j) to be processed by any machine j and the sequence of jobs in each machine, W_j^k. This is the best assignment of jobs at machines for the given sequence. Note that the sequence of jobs is still maintained here, since Algorithm 5 ensures that any job i is not processed after a job $i+1$. Once we have the jobs assigned to each machine, the problem then converts to m single machine problems, since all the machines are independent.

For the non-identical parallel machine case we need a slight change in the definition of λ in Definition 4.12. Recall that M_j is the time at which machine j finished its latest scheduled job and λ is the machine which has the least completion time of jobs, among all the machines. In the non-identical machine case we need to make sure that the assigned machine not only has the least completion time but it is also feasible for the particular job(s). Hence, for the non-identical machines case, the definition of λ in Algorithm 5 will change to λ_i where

$$\lambda_i = \underset{j=1,2,\ldots,m}{\operatorname{argmin}} \; M_j \;, \text{ such that machine } j \text{ is feasible for job } i \;.$$

For the remaining part, the Algorithm 5 works in the same manner as for the identical parallel machines. Algorithm 5 can then be applied to the non-identical independent parallel machine case for the initial allocation of jobs to machines.

4.13 Illustration of the Parallel Machine Case

In the parallel machine case we consider two parallel machines and illustrate how we first assign the jobs in the same job sequence J to the machines and optimize them independently. The data used in this example is the same as in Table 4.1. The common due-date for the instance is also the same as earlier, $d = 16$.

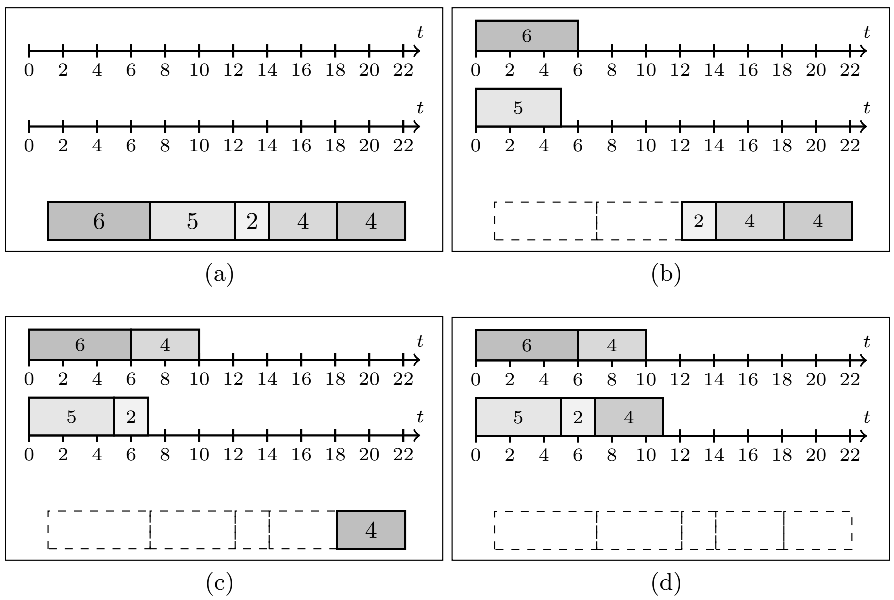


Fig. 4.10. Illustration of the assignment of jobs to machines. After the assignment, each machine has a certain number of jobs in the given sequence.

As shown in Figure 4.10(a), there are five jobs to be processed on two independent identical parallel machines, against a due-date of 16. Hence, we first assign the jobs to a machine. We start with the first two jobs in the sequence J and assign them to the machines separately at P_i, Figure 4.10(b). For the remaining jobs, we subsequently choose a machine which offers least completion time for each job. The third job in the sequence is assigned to the first machine (bottom machine) and the fourth job goes to the second machine on the same lines, as depicted in Figure 4.10(c). Finally, we have all the jobs assigned to a machine (Figure 4.10(d)) and each machine has a certain number of jobs to process in a given sequence. In this example, the first machine processes 3 jobs with the processing times of $5, 2$ and 4, while the second machine processes 2 jobs with processing times of 6 and 4, in that order. Once we have this assignment of jobs to machines, we can apply our single machine algorithm to both of them independently to optimize the overall earliness and tardiness penalty. Figure 4.11 (d) shows the best schedule for both the machines with an overall penalty of 32.

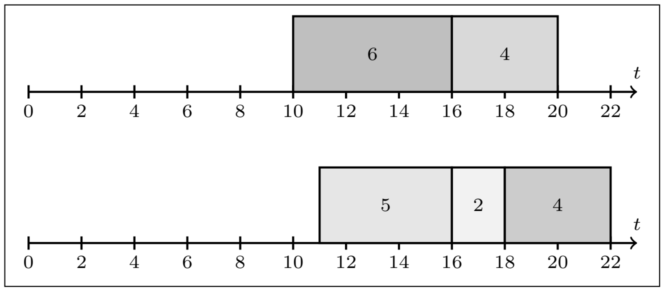


Fig. 4.11. Final optimal schedule for both the machines for the given sequence of jobs. The overall penalty of 32 is reached, which is the best solution value as per Algorithm 4 and 5.

4.14 A Dynamic Case of CDD

In this Section we discuss about a dynamic case of the common due-date problem for the single machine case at the planning stage, as discussed in [5]. Recall that in Algorithm 4 we shift all the jobs to the right as long as the property mention in Theorem 4.10 is satisfied. However, the same technique can also be implemented by initializing the jobs such that the first job starts at the due-date instead of time $t = 0$. In this case, all the jobs will be tardy and we would be required to shift the jobs to the left until the sum of the earliness penalties becomes greater than or equal to the some of the tardiness penalties. The benefit of left shifting the jobs as opposed to the right shifting proposed in Algorithm 4, lies in the fact that a dynamic case of the CDD problem can be dealt with easily. Consider the case when an optimal schedule has been calculated for a certain number jobs, and then an unknown number of jobs with unknown processing times arrive later. We assume that the original schedule is not disturbed and the new sequence of jobs can be processed after the first set of jobs. We show that in such a case the optimal schedule for the new extended job sequence can be achieved only by further reducing the completion times of all the jobs. We would like to emphasize here that we are considering the dynamic case at the planning stage when none of the jobs of the original known job sequence has gone to the processing stage.

Let us assume that at any given point of time there are a certain number of jobs (n) in a sequence J, for which the optimal schedule against a common due-date D on a machine has been already calculated using Algorithm 4. In such a case, if there are some additional jobs n' in a sequence J' to be processed against the same due-date and by the same machine without disturbing the sequence J, the optimum solution for the new sequence of $n + n'$ jobs in the extended sequence $J + J'$[1] can be found by further reducing the completion times of jobs in J and the same reduction in the completion times of jobs in J' using Algorithm 4. We prove it using Lemma 4.13.

Lemma 4.13. *Let, C_i $(i = 1, 2, \ldots, n)$ be the optimal completion times of jobs in sequence J and C'_j $(j = 1, 2, \ldots, n, n+1, \ldots, n+n'-1, n+n')$ be the optimal completion times of jobs in the extended job sequence $J + J'$ with $n + n'$ jobs. Then,*

i) $\exists\, \gamma \geq 0$ *s.t.* $C_i - C'_i = \gamma$ *for* $i = 1, 2, \ldots, n$
ii) $C'_k = C_n - \gamma + \sum_{\tau=n+1}^{k} P_\tau$, $(k = n+1, n+2, \ldots, n+n')$.

Proof. Let Sol_J denote the optimal solution for the job sequence J. This optimal value for sequence J is calculated using Algorithm 4 which is optimal according to Theorem 4.10. In the optimal solution let the individual penalties for earliness and tardiness be g_i and h_i, respectively, hence

$$Sol_J = \sum_{i=1}^{n} (\alpha_i g_i + \beta_i h_i) \ . \tag{4.24}$$

Clearly, the value of Sol_J cannot be improved by either reducing the completion times any further as explained in Theorem 4.10. Now, processing an additional job sequence J' starting from C_n (the completion time of the last job in J) means that for the new extended sequence $J + J'$ the tardiness penalty increases further by some value, say $P_{J'}$. Besides, the due-date remains the same, the sequence J is not disturbed and all the jobs in the sequence J' are tardy. Hence the new solution value (say $V_{J+J'}$) for the new sequence $J + J'$ will be

$$V_{J+J'} = Sol_J + P_{J'} \ . \tag{4.25}$$

For this new sequence we do not need to increase the completion times since it will only increase the tardiness penalty. This can be proved by contradiction. Let x be the increase in the completion times of all the jobs in $J + J'$ with $x > 0$. The earliness and tardiness for the jobs in J become $g_i - x$ and $h_i + x$, respectively and the new total penalty (V_J) for the job sequence J becomes

$$\begin{aligned} V_J &= \sum_{i=1}^{n} (\alpha_i \cdot (g_i - x) + \beta_i \cdot (h_i + x)) \\ &= \sum_{i=1}^{n} (\alpha_i \cdot g_i + \beta_i \cdot h_i) + \sum_{i=1}^{n} (\beta_i - \alpha_i) \cdot x \ . \end{aligned} \tag{4.26}$$

[1] J and J' are two disjoint sets of jobs, hence $J + J'$ is the union of two sets maintaining the job sequences in each set.

Equation (4.24) yields

$$V_J = Sol_J + \sum_{i=1}^{n} (\beta_i - \alpha_i) \cdot x \ . \tag{4.27}$$

Since Sol_J is optimal $Sol_J \leq V_J$, we have

$$\sum_{i=1}^{n} (\beta_i - \alpha_i) \cdot x \geq 0 \ . \tag{4.28}$$

Besides, the total tardiness penalty for the sequence J' will further increase by the same quantity, say δ, $\delta \geq 0$. With this shift, the new overall solution value $V'_{J+J'}$ will be

$$V'_{J+J'} = V_J + P_{J'} + \delta \ . \tag{4.29}$$

Substituting V_J from Equation (4.27) we have

$$V'_{J+J'} = Sol_J + \sum_{i=1}^{n} (\beta_i - \alpha_i) \cdot x + P_{J'} + \delta \ . \tag{4.30}$$

Using Equation (4.25) gives

$$V'_{J+J'} = V_{J+J'} + \sum_{i=1}^{n} (\beta_i - \alpha_i) \cdot x + \delta \ . \tag{4.31}$$

Using Equation (4.28) and $\delta \geq 0$ we have

$$V'_{J+J'} \geq V_{J+J'} \ . \tag{4.32}$$

This shows that only a reduction in the completion times of all the jobs can improve the solution. Thus, there exists a γ, $\gamma \geq 0$ by which the completion times are reduced to achieve the optimal solution for the new job sequence $J + J'$. Clearly, $C_i - C'_i = \gamma$ for $i = 1, 2, \ldots, n$ and $C'_k = C_n - \gamma + \sum_{\tau=n+1}^{k} P_\tau$, $(k = n+1, n+2, \ldots, n+n')$ since all the jobs are processed one after another without any idle time. □

4.15 Local Improvement of the Job sequence for the CDD

We now present a straight forward heuristic algorithm which is utilized to locally improve any CDD job sequence, by implementing the V-shaped property mentioned earlier in this chapter. We utilize Property 4.3 to develop an improvement heuristic to evolve any job sequence, optimized by Algorithm 4.

We use Algorithm 4 to obtain the '*breaking-point*', in other words, the position of the due-date in the optimal schedule for the job sequence J. Let l be the last job which is early or finishes at the due-date. It can happen that the optimal position of the due-date lies at the completion of job l or in between job l and $l+1$. We deal with these two cases separately.

We utilize the above property to develop an improvement heuristic to evolve any job sequence, optimized by Algorithm 4. We use Algorithm 4 to obtain the '*breaking-point*', in other words, the position of the due-date in the optimal schedule for the job sequence J. Let l be the last job which is early or finishes at the due-date. It can happen that the optimal position of the due-date lies at the completion of job l or in between job l and $l+1$. We deal with these two cases separately.

Case 1: If $C_l = d$
Since job l finishes at the due-date in the optimal schedule of the job sequence J, it is clear that changing the order of the early jobs will still keep the changed sequence optimal, with $C_l = d$. The reason is that the Case 2 of Theorem 4.10 still holds. Likewise, changing the order of the tardy jobs will also not change the position of the due-date for the new job sequence.

Hence, keeping this in mind we can arrange all the early jobs and job l which finishes at the due-date in the non-increasing order of the ratio P_i/α_i. In the same manner, the order of the tardy jobs can as well be arranged in the non-decreasing order of the ratio P_i/β_i.

Case 2: If $C_l < d < C_{l+1}$
If the position of the due-date appears in between two jobs, then changing the order of the early jobs will still retain the optimal schedule for the new job sequence. However, changing the order of the jobs which are tardy can change the position of the due-date, due to the straddling job $l+1$. Hence, for the straddling case we need to iteratively change the order of the early and tardy jobs with respect to the V-shape property, and optimize the new sequence as per Algorithm 4, as long as both the V-shaped property and the properties of Theorem 4.10 are consistent with each other. Although, in this work we do not apply this correction for the straddling case as our experimental results show that solutions of high quality can be obtained by applying the mentioned heuristic. Algorithm 6 shows this heuristic algorithm for improving a job sequence.

4.16 Results for the CDD Problem

In this section we present our results for the CDD problem for single and parallel machines. The benchmark instances for the CDD problem have been provided by Biskup and Feldmann [20] in the OR-library [14]. We combine our linear algorithms with the Simulated Annealing algorithm. For the CDD, we also

Algorithm 6: Heuristic to evolve a job sequence.

$l \leftarrow$ Algorithm 4 till line 16
$J_E \leftarrow \{1, 2, \ldots, l\}$
$J_T \leftarrow \{l+1, l+2, \ldots, n\}$
$J'_E \leftarrow$ Jobs sorted in non-increasing order of $P_i/\alpha_i, \forall i \in J_E$
$J'_T \leftarrow$ Jobs sorted in non-decreasing order of $P_i/\beta_i, \forall i \in J_T$
$J' = \{J'_E, J'_T\}$
$Sol \leftarrow$ Algorithm 4 on J'

utilize the improvement heuristic mentioned in Section 4.15, to improve any job sequence with the help of the V-shaped property. Henceforth we present our results and compare them with the previous works of [115, 113, 136, 92] and [97], where the authors have implemented several metaheuristic algorithms for the CDD problem. All the computations in this work are carried out on MATLAB utilizing C++ mex functions. We implement our results on a 2 GB RAM PC with 1.73 GHz Intel dual core processor. As we state earlier, our polynomial algorithm optimizes any given job sequence of CDD. Nonetheless, finding the optimal (near-optimal) job sequence is still an open question. Hence, the job sequences for both the problems are evolved using the heuristic and Simulated Annealing.

4.16.1 Modified Simulated Annealing

We implement the modified Simulated Annealing explained in Chapter 3 to generate the job sequences. The parameters for the SA are deduced by experimental analysis on the CDD problem. For the CDD problem we take an ensemble size of 2 for any number of jobs, and the maximum number of SA iterations is taken as $500 \cdot n$, n being the problem size. An exponential schedule for cooling is adopted with a cooling rate of $1 - 10^{-4}$. As for the perturbation rule, we first randomly select two jobs in any job sequence and permute them randomly to create a new sequence. The job sequence obtained after the improvement heuristic is shuffled only once. Two jobs are selected randomly, one each from the set of tardy jobs (J'_T) and early set of jobs (J'_E). These two jobs are swapped, eventually belonging to different sets.

4.16.2 Experimental Results

We now present our results for the CDD problem using the improvement heuristic mentioned in Section 4.15 and the modified simulated annealing algorithm explained in the previous section. We call our algorithm LHSA for reference. The results are compared with all the available data provided by several recent and the best works on this problem. Let, F_{REF} denote the reference fitness function value reported by Feldmann and Biskup [57] and F_i

Table 4.2. Results obtained for the single machine case for the common due-date problem and comparison with five other approaches mentioned in the literature. The results shown for each h value for any job is the average over 10 different benchmark instances provided in OR-library by [20].

n	h	DDE	DE	DPSO	VNS/TS	PHVNS	LHSA
10	0.2	**0**	**0**	**0**	**0**	**0**	**0**
	0.4	**0**	**0**	**0**	**0**	**0**	**0**
	0.6	**0**	**0**	**0**	**0**	**0**	**0**
	0.8	**0**	**0**	**0**	**0**	**0**	**0**
20	0.2	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**	**-3.84**
	0.4	**-1.63**	**-1.63**	**-1.63**	**-1.63**	**-1.63**	**-1.63**
	0.6	**-0.72**	**-0.72**	**-0.72**	**-0.72**	**-0.72**	**-0.72**
	0.8	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**
50	0.2	-5.69	-5.69	-5.68	**-5.70**	**-5.70**	**-5.70**
	0.4	**-4.66**	**-4.66**	**-4.66**	**-4.66**	**-4.66**	**-4.66**
	0.6	**-0.34**	-0.32	**-0.34**	**-0.34**	**-0.34**	**-0.34**
	0.8	**-0.24**	**-0.24**	**-0.24**	**-0.24**	**-0.24**	**-0.24**
100	0.2	**-6.19**	-6.17	**-6.19**	**-6.19**	**-6.19**	**-6.19**
	0.4	**-4.94**	-4.89	**-4.94**	**-4.94**	**-4.94**	**-4.94**
	0.6	**-0.15**	-0.13	**-0.15**	**-0.15**	**-0.15**	**-0.15**
	0.8	**-0.18**	-0.17	**-0.18**	**-0.18**	**-0.18**	**-0.18**
200	0.2	-5.77	-5.77	**-5.78**	**-5.78**	**-5.78**	**-5.78**
	0.4	**-3.75**	-3.72	-3.74	**-3.75**	**-3.75**	**-3.75**
	0.6	**-0.15**	0.23	**-0.15**	**-0.15**	**-0.15**	**-0.15**
	0.8	**-0.15**	0.20	**-0.15**	**-0.15**	**-0.15**	**-0.15**
500	0.2	**-6.43**	**-6.43**	-6.42	-6.42	**-6.43**	**-6.43**
	0.4	-3.56	-3.57	-3.56	-3.56	**-3.58**	**-3.58**
	0.6	**-0.11**	1.72	**-0.11**	**-0.11**	**-0.11**	**-0.11**
	0.8	**-0.11**	1.01	**-0.11**	**-0.11**	**-0.11**	**-0.11**
1000	0.2	-6.76	-6.72	-6.76	-6.75	**-6.77**	**-6.77**
	0.4	-4.38	-4.38	-4.38	-4.37	**-4.40**	**-4.40**
	0.6	**-0.06**	1.29	**-0.06**	-0.05	**-0.06**	**-0.06**
	0.8	**-0.06**	2.79	**-0.06**	-0.05	**-0.06**	**-0.06**

be the solution values obtained by the algorithms reported in this work and the literature, then the value of the percentage deviation Δ for any approach is calculated as

$$\Delta = \frac{F_i - F_{REF}}{F_{REF}} \cdot 100 \, .$$

In Table 4.2 we present the comparison of our results in terms of this percentage deviation (Δ) with five other approaches mentioned in the literature. These approaches include the Discrete Particle Swarm Optimization (DPSO) by Pan *et al.* [115], Variable Neighborhood Search hybridized with Tabu Seach (VNS/TS) by Liao and Cheng [92], Discrete Differential Evolution (DDE) by Tasgetiren *et al.* [136], Differential Evolution (DE) by Nearchou [112] and the

Table 4.3. Best solution values for small benchmark instances till 100 jobs, obtained by PHVNS and LHSA.

n	k	PHVNS	LHSA	PHVNS	LHSA	PHVNS	LHSA	PHVNS	LHSA
10	1	1936	1936	1025	1025	841	841	818	818
	2	1042	1042	615	615	615	615	615	615
	3	1586	1586	917	917	793	793	793	793
	4	2139	2139	1230	1230	815	815	803	803
	5	1187	1187	630	630	521	521	521	521
	6	1521	1521	908	908	755	755	755	755
	7	2170	2170	1374	1374	1101	1101	1083	1083
	8	1720	1720	1020	1020	610	610	540	540
	9	1574	1574	876	876	582	582	554	554
	10	1869	1869	1136	1136	710	710	671	671
20	1	4394	4394	3066	3066	2986	2986	2986	2986
	2	8430	8430	4847	4847	3206	3206	2980	2980
	3	6210	6210	3838	3838	3583	3583	3583	3583
	4	9188	9188	5118	5118	3317	3317	3040	3040
	5	4215	4215	2495	2495	2173	2173	2173	2173
	6	6527	6527	3582	3582	3010	3010	3010	3010
	7	10455	10455	6238	6238	4126	4126	3878	3878
	8	3920	3920	2145	2145	1638	1638	1638	1638
	9	3465	3465	2096	2096	1965	1965	1965	1965
	10	4979	4979	2925	2925	2110	2110	1995	1995
50	1	40697	40697	23792	23792	17969	17969	17934	17934
	2	30613	30613	17907	17907	14050	14050	14040	14040
	3	34425	34425	20500	20500	16497	16497	16497	16497
	4	27755	27755	16657	16657	14080	14080	14080	14080
	5	32307	32307	18007	18007	14605	14605	14605	14605
	6	34969	34969	20385	20385	14251	14251	14066	14066
	7	43134	43134	23038	23038	17616	17616	17616	17616
	8	43839	43839	24888	24888	21329	21329	21329	21329
	9	34228	34228	19984	19984	14202	14202	13942	13942
	10	32958	32958	19167	19167	14366	14366	14363	14363
100	1	145516	145516	85884	85884	72017	72017	72017	72017
	2	124916	124916	72981	72981	59230	59230	59230	59230
	3	129800	129800	79598	79598	68537	68537	68537	68537
	4	129584	129584	79405	79405	68759	68759	68759	68759
	5	124351	124351	71275	71275	55286	55286	55103	55103
	6	139188	139188	77778	77778	62398	62398	62398	62398
	7	135026	135026	78244	78244	62197	62197	62197	62197
	8	160147	160147	94365	94365	80708	80708	80708	80708
	9	116522	116522	69457	69457	58727	58727	58727	58727
	10	118911	118911	71850	71850	61361	61361	61361	61361

best known results reported by Liu and Zhou [97], where the authors provide a Permutation-based Harmony Search hybridized with Variable Neighborhood Search (PHVNS). Our results of LHSA shown in Table 4.2 are the best results obtained in 100 different replications of LHSA over all the 280 benchmark instances.

Note that for each job size there are 10 different benchmark instances and hence the values presented in Table 4.2 are the average over those 10 instances. Additionally, there is a restrictive factor h, which determines the value of the due-date (d) for any instance as $d = \lfloor h \cdot \sum_{i=1}^{n} P_i \rfloor$. Consistent with the benchmarking of Biskup and Feldmann [20], we presented the results for 4 different h values ranging from 0.2 to 0.8. Table 4.2 shows the comparison of six different approaches in terms of Δ and it is clear that LHSA and PHVNS outperform all other approaches, since these two approaches achieve the best

Table 4.4. Best solution values for large benchmark instances till 1000 jobs and their comparison with PHVNS.

n	k	PHVNS	LHSA	PHVNS	LHSA	PHVNS	LHSA	PHVNS	LHSA
200	1	498653	498653	295684	295684	254259	254259	254259	254259
	2	541180	541180	319199	319199	266002	266002	266002	266002
	3	488665	488665	293886	293886	254476	254476	254476	254476
	4	586257	586257	353034	353034	297109	297109	297109	297109
	5	513217	513217	304662	304662	260280	**260278**	260278	260278
	6	478019	478019	279920	279920	235702	235702	235702	235702
	7	454757	454757	275017	275017	246307	246307	246307	246307
	8	494276	494276	279172	279172	225215	225215	225215	225215
	9	529275	529275	310400	310400	254637	254637	254637	254637
	10	538332	538332	323077	323077	268353	268353	268353	268353
500	1	2954852	2954852	1787693	1787693	1579031	1579031	1579031	1579031
	2	3365830	3365830	1994777	**1994771**	1712195	1712195	1712195	1712195
	3	3102561	3102561	1864365	1864365	1641438	1641438	1641438	1641438
	4	3221011	3221011	1887284	1887284	1640783	1640783	1640783	1640783
	5	3114756	3114756	1806978	1806978	1468232	**1468231**	1468231	1468231
	6	2792231	2792231	1610015	1610015	1411830	1411830	1411830	1411830
	7	3172398	3172398	1902624	**1902617**	1634330	1634330	1634330	1634330
	8	3122267	3122267	1819186	**1819185**	1540377	1540377	1540377	1540377
	9	3364310	3364310	1973635	1973635	1680188	**1680187**	1680188	**1680187**
	10	3120383	3120383	1837325	1837325	1519181	1519181	1519181	1519181
1000	1	14054917	14054917	8110907	**8110892**	6410875	6410875	6410875	6410875
	2	12295997	12295997	7271371	7271371	6110091	6110091	6110091	6110091
	3	11967282	11967282	6986822	**6986816**	5983303	5983303	5983303	5983303
	4	11796603	**11796594**	7024058	**7024050**	6085846	6085846	6085849	**6085846**
	5	12449586	12449586	7364795	7364795	6341477	6341477	6341477	6341477
	6	11644090	**11644085**	6927593	**6927584**	6078373	6078373	6078375	**6078373**
	7	13276996	13276996	7861297	7861297	6574306	**6574297**	6574306	**6574297**
	8	12274736	12274736	7222137	7222137	6067328	**6067312**	6067328	**6067312**
	9	11757063	11757063	7058786	**7058766**	6185321	6185321	6185321	6185321
	10	12427443	**12427441**	7275973	**7275935**	6145742	**6145737**	6145742	**6145737**

results consistently over all the benchmark instances. However, to make a more detailed comparison with the work of Liu and Zhou [97], who provide the state-of-the-art results for the CDD, we carry out an exact instance-by-instance comparison with the PHVNS algorithm along with the comparison of the robustness of our approach.

In Table 4.3 and 4.4 we present a detailed comparison of the exact solution values obtained by LHSA and PHVNS. Evidently, for small instances till 100 jobs both PHVNS and LHSA obtain the best known solution values for all the instances. However, the superiority of LHSA is proven for harder large instances of job size 200 to 1000. In Table 4.4 we show that LHSA obtains better solutions that the state-of-the-art results of PHVNS for a total of 24 different benchmark instances. Another interesting aspect of our approach is evident from the fact that LHSA improves the solution value of 1 instance with job size 200, 6 instances for 500 jobs and 17 instances with the job size of 1000. This shows that our algorithm is more and more adaptable for larger instances which are harder to solve in general.

Additionally, we also carry out statistical analysis of LHSA and compare the robustness of our approach with PHVNS. The results of LHSA and PHVNS are obtained over 40 different replications of the algorithms over all the instances. Hence, in Table 4.5 we show the comparison of the average rel-

Table 4.5. Experimental analysis of the robustness of LHSA and its comparison with the results of PHVNS.

n	h	$\%\bar{\Delta}_{\mathrm{PHVNS}}$	$\%\bar{\Delta}_{\mathrm{LHSA}}$	$\mathrm{sd}_{\mathrm{PHVNS}}$	$\mathrm{sd}_{\mathrm{LHSA}}$
10	0.2	0.0000	0.0000	0.0000	0.0000
	0.4	0.0000	0.0000	0.0000	0.0000
	0.6	0.0000	0.0000	0.0000	0.0000
	0.8	0.2378	**0.0000**	0.0000	0.0000
20	0.2	0.0000	0.0000	0.0000	0.0000
	0.4	0.0061	**0.0000**	0.0000	0.0000
	0.6	0.0034	**0.0000**	0.0000	0.0000
	0.8	0.0421	**0.0001**	**0.0000**	0.0003
50	0.2	**0.0000**	0.0095	**0.0000**	0.0111
	0.4	**0.0022**	0.0206	**0.0070**	0.0218
	0.6	0.0113	**0.0089**	0.0143	**0.0126**
	0.8	0.0227	**0.0040**	0.0295	**0.0074**
100	0.2	**0.0030**	0.0083	**0.0047**	0.0059
	0.4	**0.0036**	0.0154	**0.0056**	0.0128
	0.6	0.0085	**0.0007**	0.0096	**0.0013**
	0.8	0.0108	**0.0007**	0.0115	**0.0014**
200	0.2	**0.0020**	0.0024	0.0021	**0.0017**
	0.4	**0.0045**	0.0062	0.0045	**0.0030**
	0.6	0.0020	**0.0004**	0.0022	**0.0007**
	0.8	0.0017	**0.0003**	0.0018	**0.0005**
500	0.2	**0.0004**	0.0008	**0.0005**	0.0006
	0.4	**0.0011**	0.0023	**0.0006**	0.0009
	0.6	0.0011	**0.0001**	0.0010	**0.0001**
	0.8	0.0013	**0.0001**	0.0009	**0.0001**
1000	0.2	0.0005	**0.0004**	0.0005	**0.0002**
	0.4	0.0013	**0.0012**	0.0008	**0.0004**
	0.6	0.0008	**0.0000**	0.0010	**0.0000**
	0.8	0.0016	**0.0000**	0.0019	**0.0000**
Average		0.0132	**0.0029**	0.0036	**0.0030**

ative percentage deviation and the standard deviations of LHSA and PHVNS from their best obtained solution values over the 40 replications of the algorithms, for each benchmark instance. We first define the average relative percentage deviation, adopted by Liu and Zhou [97]. Let F^*_{LHSA} be the best value obtained for any instance over 10 different runs and $\bar{F}_{LHSA}$ denote the average objective value for these 10 replications, then the relative deviation of LHSA ($\%\Delta_{\mathrm{LHSA}}$) for any instance is calculated as

$$\%\Delta_{\mathrm{LHSA}} = \frac{\bar{F}_{\mathrm{LHSA}} - F^*_{\mathrm{LHSA}}}{F_{\mathrm{REF}}} \cdot 100 \,.$$

Henceforth, the average relative percentage deviation over ten different instances, denoted by $k = 1, 2, \ldots, 10$, is represented as $\%\bar{\Delta}_{\mathrm{PHVNS}}$, where

$$\%\bar{\Delta}_{\text{LHSA}} = \sum_{k=1}^{10} \%\Delta^{(k)}_{\text{LHSA}}/10 \ .$$

Table 4.5 shows the average relative percentage deviation values along the standard deviations (sd_{LHSA} and sd_{PHVNS}) of both the approaches for all the instances with distinct restrictive factor h. Since each job size consists of 10 different instances, the comparison of these two parameters show that LHSA obtains the best results consistently better than PHVNS for 16 out of 28 different benchmark instances on average. Likewise, the standard deviation values for LHSA are also better than PHVNS for a total of 14 different job sizes and due-date factors. With the help of our results, we show that not only do we obtain the best known solutions for several large and difficult instances, but we also edge ahead in terms of the robustness of our algorithm.

We now present the runtime analysis of LHSA and PHVNS in Table 4.6. Liu and Zhou implement their PHVNS algorithm on a PIV machine with 1.2 GHz processor [97]. Although, LHSA is implemented on a 1.73 GHz dual core machine, the runtime comparison clearly shows that LHSA algorithm is faster than PHVNS. For small instances till 100 jobs, LHSA runs faster than PHVNS, but the time required by both the approaches is quite low. However, for larger instances of 200, 500 and 1000, the speed-ups obtained by LHSA compared to PHVNS are of the order of 18, 15 and 12, respectively. These speed-up values prove that LHSA is indeed faster than PHVNS even if we consider the difference in the machines utilized for the computations.

Table 4.6. Average runtimes in seconds for the obtained solutions of PHVNS and LHSA. The runtime for any job is the average of all the 10 different instances for each restrictive factor h.

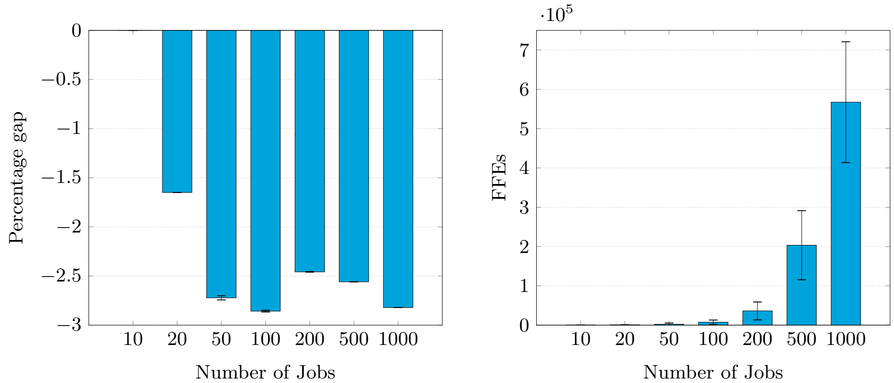
	h	**10**	**20**	**50**	**100**	**200**	**500**	**1000**
PHVNS	0.2	0.015	0.038	0.167	0.497	19.133	264.047	1283.282
	0.4	0.016	0.039	0.190	0.649	23.334	303.231	1617.400
	0.6	0.015	0.039	0.211	0.753	2.687	19.453	116.002
	0.8	0.017	0.051	0.283	0.904	2.856	22.753	127.032
Average		**0.016**	**0.042**	**0.213**	**0.701**	**12.003**	**152.371**	**785.929**
LHSA	0.2	0.000	0.001	0.009	0.072	0.747	10.626	65.933
	0.4	0.000	0.001	0.006	0.088	0.866	12.810	68.599
	0.6	0.000	0.001	0.004	0.044	0.509	8.283	55.702
	0.8	0.000	0.001	0.003	0.041	0.503	8.291	56.044
Average		**0.000**	**0.001**	**0.005**	**0.061**	**0.656**	**10.003**	**61.570**

Our results for the CDD problem show that LHSA obtains better results than the state-of-the-art for several instances. Moreover, we also prove that our algorithm is highly robust and better adapted to large instances which are harder to solve. To further prove the consistency and robustness of our approach, in Table 4.7 we present some measures of central tendency. Table 4.7 shows the minimum, mean, maximum, median, mode along with the standard deviation of the percentage deviation Δ, over 40 replication of LHSA on 280 different instances. The results obtained by LHSA in both the best and worst cases for any instances are quite consistent in terms of the solution quality. Additionally, this can be better understood be the fact the standard deviation of the percentage error is of the order of 10^{-3} for most of the instances. Even in the worst case, our standard deviation for 40 different replications, still falls at 0.041. This goes to show that our approach not only obtains good solutions but on a consistent basis, as well.

Table 4.7. Measures of central tendency with respect to the objective function values and the average number of fitness function evaluations, for 40 different replications of the instances.

n	h	Minimum	Mean	Maximum	Std.	Median	Mode	FFEs
10	0.2	0.000	0.000	0.000	0.000	0.000	0.000	48
	0.4	0.000	0.000	0.000	0.000	0.000	0.000	257
	0.6	0.000	0.000	0.000	0.000	0.000	0.000	23
	0.8	0.000	0.000	0.000	0.000	0.000	0.000	17
20	0.2	-3.842	-3.842	-3.842	0.000	-3.842	-3.842	313
	0.4	-1.630	-1.630	-1.630	0.000	-1.630	-1.630	625
	0.6	-0.721	-0.721	-0.721	0.000	-0.721	-0.721	820
	0.8	-0.409	-0.409	-0.405	0.001	-0.409	-0.409	656
50	0.2	-5.696	-5.686	-5.651	0.016	-5.689	-5.695	3772
	0.4	-4.658	-4.637	-4.490	0.041	-4.652	-4.652	2581
	0.6	-0.336	-0.327	-0.278	0.020	-0.335	-0.335	1678
	0.8	-0.242	-0.238	-0.216	0.009	-0.242	-0.242	1344
100	0.2	-6.194	-6.185	-6.146	0.011	-6.189	-6.189	8287
	0.4	-4.939	-4.923	-4.859	0.023	-4.931	-4.938	10794
	0.6	-0.145	-0.145	-0.141	0.001	-0.145	-0.145	5860
	0.8	-0.176	-0.175	-0.174	0.001	-0.176	-0.176	5399
200	0.2	-5.778	-5.776	-5.764	0.004	-5.777	-5.778	40772
	0.4	-3.754	-3.748	-3.724	0.007	-3.749	-3.752	46864
	0.6	-0.154	-0.154	-0.152	0.001	-0.154	-0.154	28878
	0.8	-0.154	-0.154	-0.152	0.001	-0.154	-0.154	28415
500	0.2	-6.433	-6.432	-6.428	0.001	-6.432	-6.432	209392
	0.4	-3.583	-3.581	-3.572	0.003	-3.582	-3.583	257597
	0.6	-0.112	-0.112	-0.112	0.000	-0.112	-0.112	173059
	0.8	-0.112	-0.112	-0.112	0.000	-0.112	-0.112	173201
1000	0.2	-6.773	-6.772	-6.771	0.000	-6.772	-6.772	600753
	0.4	-4.400	-4.398	-4.395	0.001	-4.399	-4.399	627233
	0.6	-0.057	-0.057	-0.057	0.000	-0.057	-0.057	518594
	0.8	-0.057	-0.057	-0.057	0.000	-0.057	-0.057	522636

Figure 4.12 shows the graphical representation of the percentage deviation averaged over all the due-date factors, along with the average fitness function evaluations and its standard deviation.



(a) Average percentage deviation and its standard deviation for all job sizes.

(b) Average number of fitness function evaluations and its standard deviation.

Fig. 4.12. The average percentage deviation and standard deviation of LHSA averaged over all the due-date positions for each job size.

4.17 Summary

In this work we predominantly make a theoretical study of scheduling problem against a common due-date. The intuition behind our approach for the problem is to break the integer programming formulation of these NP-hard problems in two parts, *i.e.*, (i) finding a good (near optimal) job sequence and (ii) finding the optimal values of the completion times C_i for all the jobs in this job sequence. The job sequences are optimized by using a modified Simulated Annealing (SA) algorithm explained later in the chapter. The SA solves the sub-problem (ii) as linear program by applying specialized deterministic algorithms. Most of the work in this chapter emphasizes on the second sub-problem of finding specialized algorithms for the linear program.

We first develop an $O(n^2)$ algorithm by reducing the common due-date problem to the aircraft landing problem. We then implement important properties and develop an improved linear algorithms to optimize any given job sequence. We prove the CDD property which states that for any job sequence, the position of the due-date depends only on the earliness/tardiness penalties, irrespective of the processing times. Using this property, we are able to develop an $O(n)$ algorithm to optimize any given job sequence. Additionally, we also present a straight forward heuristic to improve any job sequence. This heuristic is developed using the V-shaped property of the CDD problem, which basically states that the arrangement of the jobs pertaining to both their processing times and the earliness/tardiness property. This property has already been exploited by several previous works [115, 136, 97], but the overall approach in this work is more simplified and achieves better results than any other previous work on this problem. We improve solution values for several benchmark instances and prove that our algorithm is much accurate and robust than the present state-of-the-art [97].

Additionally, we also propose a heuristic for the parallel machine case of the CDD problem. This heuristic first allocates jobs to machines depending on the machine availability. We then optimize the produced job sequences on each machine by using our linear algorithm and the improvement heuristic. Henceforth, we explain how our approach is beneficial to a dynamic case of the CDD when the jobs arrive after the optimization process is started for any given sequence. Hence, once again we show that the development of these specialized algorithms for the result linear program are advantageous in many respect.

5

Common Due-Window Problem: Polynomial Algorithms for a Given Processing Sequence

This chapter discusses and presents an algorithm for a variant of the CDD problem known as the Common Due Window problem. The set-up of this scheduling problem is similar to that of CDD, except that the jobs are scheduled against a common due *window*, as opposed to a common due *date* in the CDD. This due-window is defined by the *left* and *right* common due-dates. Similar to the CDD problem, each job possesses different processing times but different and asymmetric earliness and tardiness penalties. The objective of the problem is to find the processing sequence of jobs, their completion times to minimize the total penalty incurred due to tardiness and earliness of the jobs. Jobs that finish before (after) the left (right) due-date are termed as early (tardy) jobs. This work presents an exact polynomial algorithm for optimizing any given job sequence for a single machine with the runtime complexities of $O(n)$, where n is the number of jobs. The algorithm takes a job sequence J_i consisting of all the jobs ($i = 1, 2, \ldots, n$) as input and returns the optimal completion times of the jobs, which offers the minimum possible total penalty for the given sequence. Furthermore, we also present a heuristic based on the V-shaped property to improve any job sequence. We then incorporate our polynomial algorithm with the heuristic, in conjunction with the Simulated Annealing (SA) algorithm to obtain the optimal/near-optimal solutions. The results of our approach are compared with the benchmark results provided by Biskup and Feldmann [21] for different due-window lengths.

5.1 Introduction

The Common Due-Window (CDW) scheduling problem involves sequencing and scheduling of jobs over machine(s) against a given common due-window. The objective is to find the position of the due-window of a given length and the job sequence to minimize the total tardiness and earliness penalties. Each job possesses a processing time and different penalties per unit time in case the job is completed before or later than the due-window. The jobs which are completed between or at the due-window are called straddle jobs and do not incur any penalty. Similar to the Common Due-Date (CDD) problem, the CDW also occurs in the supply chain management industry to reduce the earliness and tardiness of the goods produced.

Common due-date problems have been studied extensively during the last 30 years with several variants and special cases [128, 77, 118, 70, 35, 72]. CDW is an extension of the CDD with the presence of a common due-window instead of a common due-date. However, several important similar properties hold for both the problems. In 1994, Krämer and Lee studied the due-window scheduling for the parallel machine case and presented useful properties for the CDW [85], explained later in the chapter.

Krämer and Lee also showed that the CDW with unit weight case is also NP-complete and provided a dynamic programming algorithm for the two machine case [85]. Liman *et al.* considered the CDW with constant earliness/-tardiness penalties and proposed an $O(n \log n)$ algorithm to minimize the weighted sum of earliness, tardiness and due-window location [95]. The same authors also studied the CDW on a single machine with controllable processing times with constant penalties for earliness, tardiness and window location, and different penalties for compression of job processing times. They showed that the problem can be formulated as an assignment problem and can be solved using the well-known algorithms [96].

In 2002, Chen and Lee studied the CDW on parallel machines and solved the problem using a Branch and Bound algorithm and showed that the problem can be solved up to 40 jobs on any number of machines [34] in a reasonable time. In 2005, Biskup and Feldmann dealt with the general case of the CDW problem and approached it with three different metaheuristic algorithms, namely, evolutionary strategy, simulated annealing and threshold accepting. They also validated their approaches on 250 benchmark instances up to 200 jobs [21]. Wan studied the common due-window problem with controllable processing times with constant earliness/tardiness penalties and distinct compression costs, and discussed some properties of the optimal solution along with a polynomial algorithm for the solving the problem in 2007 [143]. Zhao *et al.* studied the CDW with constant earliness/tardiness penalties and window location penalty, and proposed polynomial time approximation schemes [153].

In 2010, Yeung *et al.* formulated a supply chain scheduling control problem involving single supplier and manufacturer and multiple retailers. They formulated the problem as a two machine CDW and presented a pseudo-polynomial algorithm to solve the problem optimally [148]. Cheng *et al.* considered the common due-window assignment problem with time-dependent deteriorating jobs and a deteriorating maintenance activity. They proposed a polynomial algorithm for the problem with linear deterioration penalties and its special cases [37]. Gerstl and Mosheiov studied the due-window assignment problem with unit-time jobs and proposed an $O(n^3)$ algorithm for solving the problem [61]. Yin *et al.* considered the batch delivery single-machine scheduling problem with assignable common due-window with constant penalties and proposed an $O(n^8)$ dynamic programming algorithm under an assumption on the relationship among the cost parameters [151]. In 2013, Janiak *et al.* presented a survey paper on the common due-window assignment scheduling problem and discussed more than 30 different variations of the problem [74]. Again in 2013, Janiak *et al.* studied the CDW assignment problem on parallel machines to minimize the earliness/tardiness penalties along with the penalties associated with the location and size of the due-window [73].

In this work, we consider the single machine case for the CDW problem with asymmetric penalties for the general case. We make a theoretical study of the CDW problem and present a linear exact algorithm to optimize any given job sequence on a single machine. Henceforth, we present a heuristic algorithm to evolve and improve any job sequence. This heuristic is similar to Algorithm 6 presented in the previous chapter for the CDD problem.

5.2 Problem Formulation

In this section, we give the mathematical notation of the common due-window problem based on [21]. We also define some new parameters which are later used in the presented algorithms in the next section.

Let
n = the total number of jobs,
d_l = the left common due-date,
d_r = the right common due-date,
P_i = the processing time of job i, $i = 1, 2, \ldots, n$,
C_i = the completion time of job i,
g_i = the earliness of job i, where $g_i = \max\{0, d_l - C_i\}$, $i = 1, 2, \ldots, n$,
h_i = the tardiness of job i, where $h_i = \max\{0, C_i - d_r\}$, $i = 1, 2, \ldots, n$,
α_i = the earliness penalty per unit time for job i,
β_i = the tardiness penalty per unit time for job i.

The objective of the problem is to schedule the jobs against the due-window to minimize the total weighted penalty incurred by the earliness and tardiness of all the jobs.

$$\min \sum_{i=1}^{n} \{\alpha_i \cdot g_i + \beta_i \cdot h_i\} \ . \tag{5.1}$$

We now present some important properties for the CDW problem and prove that the property proved in Theorem 4.10 of Chapter 4 is also valid for CDW problem.

Property 5.1. There exists an optimal schedule without machine idle time between the first and the last job [85, 90].

Property 5.2. In any optimal schedule, jobs completed before the left due-date are sequenced in the non-increasing order of the ratio P_i/α_i and the jobs that are tardy are sequenced in non-decreasing order of the ratio P_i/β_i [21].

Property 5.3. Let C_i be the completion time of job i, then there exists an optimal schedule where one of the jobs finishes at d_l or at d_r, *i.e.*,
a) $C_i = d_l$ for some i, or
b) $C_i = d_r$ for some i [85, 90].

5.3 Property for the Common Due-Window

In this section we show that the relationship between the sum of the earliness penalties and the tardiness penalties proved in Theorem 4.10 for the CDD problem also holds for the CDW problem. We know from Property 5.3 that the optimal schedule of the CDW will either start at time $t = 0$ or one of the due-dates will fall at the completion time of some job.

Theorem 5.4. *In the optimal schedule of a CDW instance, if the jobs* $1, 2, \ldots, r-1$ *are early, job* r *finishes at the left due-date and jobs* $k, k+1, \ldots, n$, $k > r$ *are tardy, then we have,* $\sum_{i=k}^{n} \beta_i \leq \sum_{i=1}^{r} \alpha_i$ *for minimum possible values of* k *and* r.

Proof. Let us assume without loss of any generality that in the optimal schedule the left due-date d_l lies at the completion time of a job r and the right due-date d_r lies between the completion times of two adjacent jobs $k-1$ and k, as shown in Figure 5.1. In this case the jobs which are finishing in between the due-window do not offer any penalty and hence do not participate to either tardiness/earliness penalty.

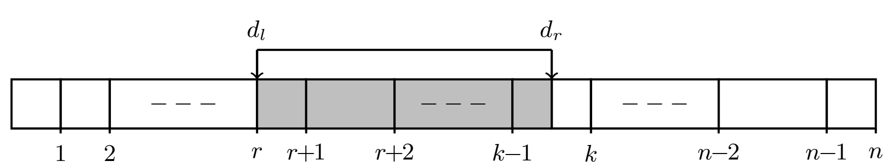


Fig. 5.1. Schedule for the due-window case, with the left due-date (d_l) situated at C_r and the right due-date (d_r) in between the completion times of jobs $k-1$ and k.

In other words, this schedule become equivalent to a schedule shown in Figure 5.2, wherein the jobs lying in the due-window are completely removed, and in the new schedule job k has a smaller schedule. If the schedule in Figure 5.1 is optimal, then the schedule in Figure 5.2 is also optimal as the other jobs offer no penalty.

The problem now converts to the CDD with d_l being the due-date. Hence, the properties of Theorem 4.10 will also hold for the CDW on the same lines as for the CDD. □

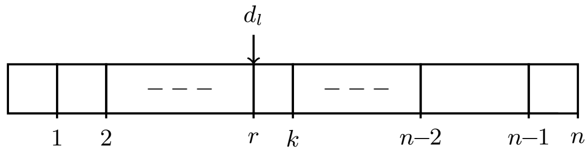


Fig. 5.2. Schedule for the CDW such that all the straddling jobs are removed. The problem now converts to the CDD with the due-date position at C_r.

5.4 The Exact Algorithm

Using Theorem 5.4, we now present our exact polynomial algorithm for the CDW problem to optimize any given job sequence. As mentioned above in Property 5.1, we know that the optimal schedule of the CDW has no idle time of the machine between C_1 and C_n. Hence, for our algorithm, the first job starting at time $t = 0$ and are shifted to the right by minimum deviation of the completion times from the right and the left due-dates. This way, every shift ensures that one of the jobs finishes at one of the due-dates (Property 5.3) and we do not skip over the optimal position of the due-dates. Once the property mentioned in Theorem 5.4 is satisfied, we have our optimal schedule and no more shifting is required. This approach of right-shifting-the-jobs is implemented in [7]. However, it requires the update of the completion times C_i of all the jobs and thus accounts for a runtime complexity of $O(n)$ for each shift. We present a much faster approach where the calculation of the completion times of all the jobs needs to be done only once, throughout the algorithm.

The idea behind our approach lies in the fact that the calculation of the objective function or checking the property mentioned in Theorem 5.4, only requires the relative deviation of the completion times of all the jobs with the left and right due-dates. Hence, shifting all the jobs and due-dates together with the same amount does not effect the objective function value, as well as the set of early and tardy jobs. For our algorithm, we initialize the completion times of the jobs such that $C_1 = P_1$ and the subsequent jobs are followed without any machine idle time. The C_i values remain fix for the whole algorithm. Thereafter, we find the optimal position of a movable due-window (d'_l, d'_r) of the same length as of the original due-window ($d_r - d_l$). This optimal position is calculated using the property mentioned Theorem 5.4, by shifting the movable due-window from extreme right to left as long as the sum of the tardiness penalties is less than the sum of the earliness penalties. If the optimal position of this new due-window lies to the left of the original due-window *i.e.*, $d'_l < d_l$ or $d'_r < d_r$, (note that both the inequalities will be satisfied simultaneously, since the due-windows are of same lengths, $d'_l < d_l \Rightarrow d'_r < d_r$), then we take d'_l and d'_r for calculating the final earliness/tardiness of the jobs. However, if the position of this movable due-window lies to the right of the original due-window, *i.e.*, $d'_l > d_l$ or $d'_r > d_r$, then we retain the original due-dates for calculating the final earliness/tardiness of the jobs. The reason for the above statements can be proved by considering the two cases separately.

Case 1: $d'_l < d_l$

If the optimal position of the movable due-window is such that $d'_l < d_l$, then it means that the property mentioned in Theorem 5.4 is satisfied for some value k but the original due-date falls at some job index i where $i > k$. Hence, practically we need to shift all the jobs to the right such that jobs $k, k+1, \ldots, n$ are tardy. Instead, we can just take the d'_l and d'_r as the due-dates to calculate the final earliness/tardiness of the jobs to obtain the objective function value, because of the fact that the earliness/tardiness are relative deviations with the due-dates.

Case 2: $d'_l > d_l$

If the optimal position of the movable due-window is such that $d'_l > d_l$, then it means that the property mentioned in Theorem 5.4 is satisfied for some value k but the original due-date falls at some job index i where $i < k$. In this case, we are actually required to practically shift the jobs to the left, which can not be done as the schedule of the jobs is already starting at time $t = 0$. Hence, for this case we need to take the original due-window (d_l, d_r) to calculate the final earliness/tardiness of the jobs. The case, $d'_l = d_l$ is apparent.

Our algorithm first assigns the right due-date (d'_r) of the movable due-window as C_n and $d'_l = d'_r - d_r + d_l$. This ensures that the right due-date falls at the completion time of the last job and the left due-date lies at some job $i < k$. In the next steps, this movable due-window is shifted to the left as long as we obtain the optimal position using the property mentioned in Theorem 5.4.

We now formalize some parameters which are essential for the understanding of the exact algorithm. Let η_l, φ_l, η_r, φ_r and be as defined in Equation 5.2.

$$\begin{aligned} \eta_l &= \underset{i=1,2,\ldots,n}{\arg\max}(C_i - d'_l < 0), \ \ \varphi_l = \underset{i=1,2,\ldots,n}{\arg\max}(C_i - d'_l \leq 0), \\ \eta_r &= \underset{i=1,2,\ldots,n}{\arg\max}(C_i - d'_r < 0), \ \ \varphi_r = \underset{i=1,2,\ldots,n}{\arg\max}(C_i - d'_r \leq 0) \ . \end{aligned} \tag{5.2}$$

In the above equation, η_l depicts the last job which finishes strictly before the left due-date d'_l, while φ_l is the last job which finishes at or before d'_l. Clearly, if for some schedule the completion time of a job i lies at the left due-date then $\varphi_l = i$ and $\eta_l = i - 1$. However, if $C_i < d'_l < C_{i+1}$, *i.e.* the left due-date falls in between the completion times of jobs i and $i+1$, then we get $\varphi_l = \eta_l = i$. η_r and φ_r can be understood on the same lines, with respect to the right due-date d'_r.

We also define Δ_l and Δ_r as the deviation of the completion time of the job which finishes right before the left and right due-date, respectively. Notice that η_l (or η_r) is the last job which finishes completion, strictly before the left (or right) due-date. Hence, one can write $\Delta_l = d'_l - C_{\eta_l}$ and $\Delta_r = d'_r - C_{\eta_r}$. Clearly, $\min\{\Delta_l, \Delta_r\}$ is the minimum possible left shift of the due-window required such that either one of the left/right due-dates falls at the completion time of

Algorithm 7: Linear algorithm for any CDW job sequence.

$C_i \leftarrow \sum_{k=1}^{i} P_k \ \forall \ i = 1, 2, \ldots, n$
$d'_r \leftarrow C_n$
$d'_l \leftarrow d'_r - d_r + d_l$
Compute $\eta_l, \varphi_l, \eta_r, \varphi_r$
$\Delta_l \leftarrow d'_l - C_{\eta_l}$
$\Delta_r \leftarrow d'_r - C_{\eta_r}$
$pe \leftarrow \sum_{i=1}^{\varphi_l} \alpha_i$
$pl \leftarrow 0$
while $pl < pe$ **do**
    $LeftDueDate \leftarrow d'_l$
    $RightDueDate \leftarrow d'_r$
    $\Delta \leftarrow \min\{\Delta_l, \Delta_r\}$
    $d'_l \leftarrow d'_l - \Delta$
    $d'_r \leftarrow d'_r - \Delta$
    if $(\eta_l < \varphi_l)$ **then**
        $pe \leftarrow pe - \alpha_{\varphi_l}$
        $\varphi_l \leftarrow \varphi_l - 1$
    if $(\eta_r < \varphi_r)$ **then**
        $pl \leftarrow pl + \beta_{\varphi_r}$
        $\varphi_r \leftarrow \varphi_r - 1$
    if $\Delta_l < \Delta_r$ **then**
        $\eta_l \leftarrow \eta_l - 1$
    else if $\Delta_r < \Delta_l$ **then**
        $\eta_r \leftarrow \eta_r - 1$
    else if $\Delta_r = \Delta_l$ **then**
        $\eta_l \leftarrow \eta_l - 1$
        $\eta_r \leftarrow \eta_r - 1$
    if $\eta_l > 0$ **then**
        $\Delta_l \leftarrow d'_l - C_{\eta_l}$
        $\Delta_r \leftarrow d'_r - C_{\eta_r}$
    else
        break
if $LeftDueDate \leq d_l$ **then**
    $d_l \leftarrow LeftDueDate$
    $d_r \leftarrow RightDueDate$
$g_i \leftarrow \max\{d_l - C_i, 0\}$, $\forall i$
$h_i \leftarrow \max\{0, C_i - d_r\}$, $\forall i$
$Sol \leftarrow \sum_{i=1}^{n} (g_i \cdot \alpha_i + h_i \cdot \beta_i)$
return Sol

a job. With the help of these parameters and the properties proved earlier, we now present our linear algorithm to optimize any given job sequence for the CDW problem, shown in Algorithm 7. We explain the steps of the algorithm in detail with the help of an illustrative example. The data for this example is given in Table 5.1.

Table 5.1. The data for the exemplary case of the CDW problem. The parameters possess the same meaning as explained in Section 5.2.

$\boldsymbol{i}$	$\boldsymbol{P_i}$	$\boldsymbol{\alpha_i}$	$\boldsymbol{\beta_i}$
1	2	9	6
2	6	7	6
3	8	1	4
4	6	2	5
5	10	8	4

We consider 5 jobs with the due-window defined by the left (d_l) and right (d_r) due-dates of 12 and 19, respectively. As explained above, we initialize the completion times of the jobs with the first job starting at time $t = 0$ as shown in Figure 5.3. Meanwhile, we consider a movable due-window of same length as the original due-window. The right due-date of this movable due-window is initialized as $d'_r = C_n = 32$ and the left due-date as $d'_l = d'_r - d_r + d_l = 25$, as shown in Figure 5.3. In the next steps we move this due-window to the left while keeping the completion times of the all the jobs, unchanged. Note that the gray boxes in Figure 5.3 represent the jobs, and the number inside any box depicts the processing time of that particular job.

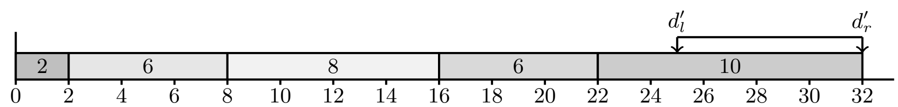


Fig. 5.3. Schedule for the initialization of the completion times of the jobs and the initial placement of the movable due-window.

After the initialization step, we first calculate the parameters mentioned in Equation (5.2). These parameters basically determine the jobs that are finishing just before or at the due-dates. For our initialization shown in Figure 5.3, we get $\eta_l = 4$, $\varphi_l = 4$, $\eta_r = 4$, $\varphi_r = 5$, $\Delta_l = 3$ and $\Delta_r = 10$. We also need to calculate the sum of the earliness penalties ($pe = \sum_{i=1}^{\varphi_l} \alpha_i$) of all the jobs that have their completion times with $C_i - d'_l \leq 0$. For the initialized schedule, we have $pe = 19$. Clearly, the sum of the tardiness penalties of the tardy jobs is $pl = 0$ since there are no tardy jobs. From Figure 5.3 we have $\eta_l = \varphi_l$ and $\eta_r < \varphi_r$, which tells us that left due-date lies somewhere in between the completion times of jobs η_l and $\eta_l + 1$, while the right due-date falls at C_{φ_r}. Additionally, we also have $\Delta_l = d'_l - C_{\eta_l} = 3$ and $\Delta_r = d'_r - C_{\eta_r} = 10$, which means that we need to shift the due-window by 3 time units, such that the left due-date d'_l falls at $C_4 = 22$ and d'_r is placed at 29.

Figure 5.4 shows the next step after the left shift of the due-window, with $d'_l = 22$ and $d'_r = 29$. As it is clear from the figure, we now have tardy job

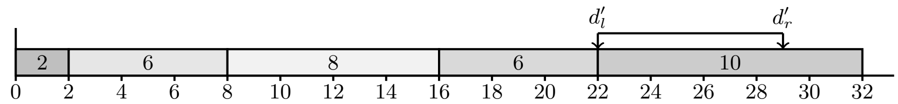


Fig. 5.4. Schedule with the first left shift of the movable due-window by the amount of $\Delta_l = 3$.

and hence the sum of the tardiness penalties is increased by $\beta_5 = 4$, which implies $pl = 4$. Note that for the property mentioned in Theorem 5.4, we need to add the early penalty of the job that completes at the left due-date. Thus with this shift, we do not need update the value of pe. Also note that since we shifted the due-window such that d'_l falls at C_4, we need to reduce η_l by 1 as the last early job for the left due-date is now job 3, hence the updated value of $\eta_l = 3$ while $\varphi_l = 4$ remains unchanged. Likewise, the last early job for the right due-date is still job 4 thus keeping $\eta_r = 4$. But the value of φ_r will reduce by 1 because the right due-date now lies in between the completion times of the job 4 and 5, fetching $\varphi_r = 4$.

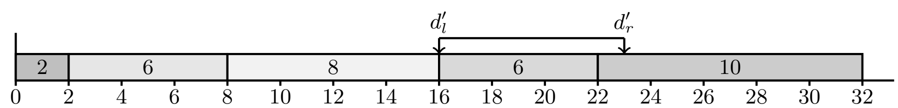


Fig. 5.5. Schedule with the second left shift of the movable due-window by the amount of $\Delta_l = 6$.

From the first left shift, $pl < pe$ since $pl = 4$ and $pe = 19$, hence need to shift the due-window further to the left to check if $pl < pe$ still holds, in accordance with Theorem 5.4. From Figure 5.4, we have $\Delta_l = 6$ and $\Delta_r = 7$, hence the amount of shifting required such that one of the due-dates falls at the completion time of some job is $\Delta_l = 6$, as shown in Figure 5.5. With this shift, since d'_l is shifted from the completion time of job 4 to its preceding job 3, hence we need to reduce both η_l and φ_l by 1, which gives us $\eta_l = 2$ and $\varphi_l = 3$. However, the values $\eta_r = 4$ and $\varphi_r = 4$ because the right due-date d'_r still lies between C_4 and C_5. Beside, the we also need to reduce pe by α_4 since job 4 is no longer early or falls at d'_l, implying $pe = 17$. The value of pl remains unchanged from the previous step since the tardy jobs remain the same, and this $pl = 4$. Yet again $pe < pl$ asks for the further left shift or the due-window to check if this property holds. Since the new values for Δ_l and Δ_r are $\Delta_l = d'_l - C_{\eta_l} = 8$ and $\Delta_r = d'_r - C_{\eta_r} = 1$, respectively, we shift the due-window by $\min\{\Delta_l, \Delta_r\} = 1$, as shown in Figure 5.6.

Figure 5.6 shows that the right due-date now falls at the completion time of job 4 while the left due-date is situated between C_2 and C_3. Since d'_l is moved from C_4 to a position between jobs 3 and 4, $\eta_l = 2$ remains unchanged, but φ_l is reduced by 1 to 2, by its definition. Likewise, the value of η_r is

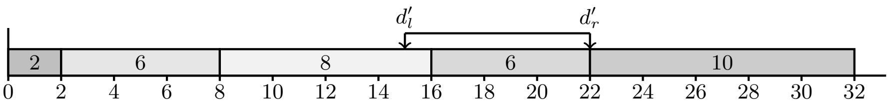


Fig. 5.6. Schedule with the third left shift of the movable due-window by the amount of $\Delta_r = 1$.

reduced by 1 to 3 but $\varphi_r = 4$ remains unchanged. Also note that pe will be reduce by β_3 as it is past the left due-date, thus $pe = 16$. However, $pl = 4$ remains the same as the tardy jobs are unchanged. Since, pe is still less than pl we need to shift the due-window further to the left.

The reason that in any case we reduce the values of η_l (η_r) or φ_l (φ_r) by only 1, lies in the fact that we shift the due-window by the minimum possible amount such that one of the due-dates fall at the completion time of any job. In doing so, we do not skip over any job and thus are required to reduce η_l (η_r) or φ_l (φ_r) by just 1.

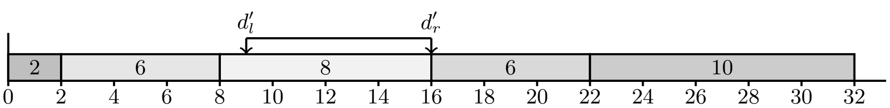


Fig. 5.7. Schedule with the fourth left shift of the movable due-window by the amount of $\Delta_r = 6$.

Figure 5.7 shows the fourth left shift of the due-window, with $d'_l = C_2$ and $C_2 < d'_r < C_3$. After this shift we obtain, $\eta_l = 1, \varphi_l = 2, \eta_r = 2, \varphi_r = 2$. The value of $pe = 16$ and pl becomes 9 since job 4 is now tardy.

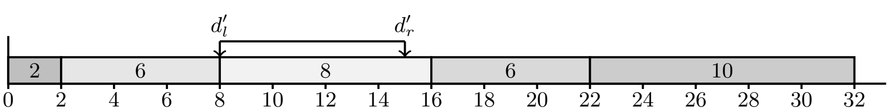


Fig. 5.8. Schedule with the fifth left shift of the movable due-window by the amount of $\Delta_l = 1$.

This procedure of left shift is continued as long as $pl < pe$. Figure 5.8 shows the optimal schedule for this example with $d'_l = 8$ and $d'_r = 15$. A further left shift (Figure 5.9) renders $pe = 9$ and $pl = 13$, thus violating Theorem 5.4. Hence we know that the schedule in Figure 5.8 is optimal for this movable due-window, with $d'_l = 8$ and $d'_r = 15$. Recall, that the original due-windows were defined by $d_l = 22$ and $d_r = 29$. Since, the new due-dates of the movable due-window are less than the original due-dates we can take d'_l and d'_r to calculate the final earliness/tardiness of the jobs. The reason again is the fact that we can as well shift all the jobs together to the right such that the second job finishes at $d_l = 22$. However, the objective function value would not change because the earliness/tardiness are relative to the position of the due-dates. Hence we can calculate the earliness/tardiness of the jobs against d'_l and d'_r, for the schedule where the first job starts at time $t = 0$, thus obtaining the optimal objective function value of 161 for the studied job sequence.

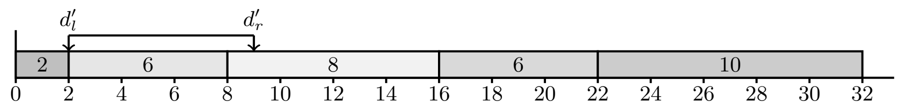


Fig. 5.9. Schedule with the sixth left shift of the movable due-window by the amount of $\Delta_l = 6$.

5.5 Proof of Optimality

In this section, we present the optimality of Algorithm 7 with respect to the objective function value of Equation (5.1), for any given job sequence.

Theorem 5.5. *Algorithm 7 is optimal for any given sequence of the CDW problem, with respect to the objective function value.*

Proof. We first schedule the given job sequence such that the processing of the first job starts at time $t = 0$ and the remaining jobs are processed without any machine idle time, maintaining the sequence of the jobs. We then place the movable due-window in a way such that $d'_r = C_n$ for some n and $d'_l = d'_r - d_r + d_l$ as shown in Figure 5.3, maintaining the length of the due-window same as the original. From this point on, we refer to this *movable* due-window as the due-window itself, unless mentioned otherwise.

Meanwhile, we also calculate $\eta_l, \varphi_l, \eta_r$ and φ_r to keep track of the jobs that are closest to the due-dates. The η_l (or η_r) values represent the last job processed strictly before the left (or right) due-date, while the combination of η_l, φ_l (or η_r, φ_r) values tells us if the left (or right) due-date falls at the completion time of some job or lies in between the completion times of two consecutive jobs. In other words, one can interpret the η_l (η_r) or φ_l (φ_r) values to indicate the position of the due-window. At each step of the iterative left shift, the due-window is shifted by the minimum possible amount such that one of them falls at the completion time of some job, while keeping track of the *pe* and *pl* values to find the optimal schedule. This shift is calculated as $\Delta = \min\{\Delta_l, \Delta_r\}$, where $\Delta_l = d'_l - C_{\eta_l}$ and $\Delta_r = d'_r - C_{\eta_r}$. Apparently, any left shift of $\min\{\Delta_l, \Delta_r\}$ ensures that we do not skip over any job while checking for the property mentioned in Theorem 5.4. However, depending on the position of the due-window (or the η_l, φ_l and η_r, φ_r values), we can have several different cases that can occur during the left shift.

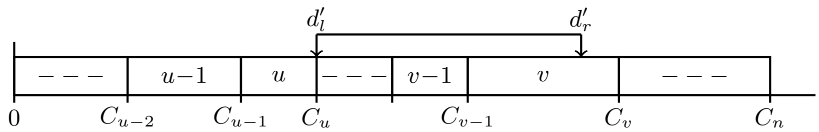


Fig. 5.10. Schedule representing the case when d'_l falls at the completion time of a job and d'_r lies in between the completion time of two jobs, along with $\Delta_l < \Delta_r$.

During the course of any left shift, if $\eta_l < \varphi_l$ then we are certain that d'_l falls at the completion time of some job, as shown in Figure 5.10. Recall that $\eta_l = \arg\max(C_i - d'_l < 0)$, and $\varphi_l = \arg\max(C_i - d'_l \leq 0)$. Hence, $\eta_l < \varphi_l$ necessarily implies that $\varphi_l = \arg\max(C_i - d'_l \leq 0) = u$ for some job u, such that $C_u = d'_l$ and $\eta_l = u - 1$. Moreover, at any instance of the algorithm, we never make a left shift that is greater than Δ_l. Hence, the value of pe will be reduced by α_{φ_l}, since for any shift which is greater than zero, job u will fall after the left due-date. Likewise, whatever the left shift, the value of φ_l will also be reduced by 1, for the same reason that u will no longer fall at d'_l, after the shift of the due-window. Also note that the only possible case such that $\eta_l = \varphi_l$, is the one when d'_l falls in between the completion time of two jobs, after the left shift, say $u - 1$ and u. Since, $\Delta = \min\{\Delta_l, \Delta_r\}$, the maximum possible shift will lead to $d'_l = C_u$. However, in this case we do not need to update pe and φ_l as per their definitions. The value of η_l will indeed get reduced by 1, if the shift is made by Δ_l, *i.e.*, the left due-date falls at the completion time of a job after the left shift of the due-window. Hence, to check if η_l needs to be updated or not, we only need to check if the shift if made by Δ_l, or in other words, $\Delta_l \leq \Delta_r$, as implemented in Algorithm 7.

On the same lines of argument, if $\eta_r < \varphi_r$, then we have a case when the right due-date d'_r falls at the completion time of some job, say v, such that $\eta_r = v - 1$ and $\varphi_r = v$. Hence, any left shift greater than zero will lead to job v being tardy, and so we are required to increase sum of the tardiness penalty by β_{φ_r}. As for the left due-date, since we never make a shift greater than Δ_r, the value of φ_r will get reduced by 1. Concerning the value of η_r, recall that the value of η_r changes only when the right due-date falls at the completion time of some job, after the left shift. Since, we shift the due-window by $\min\{\Delta_l, \Delta_r\}$, the only possible case when the right due-date will land at some C_i (for some i), when $\Delta_r \leq \Delta_l$, as shown in Algorithm 7.

After every left shift, we update the values for $\eta_l, \varphi_l, \eta_r, pe$ and pl depending on the position of the due-window and the amount of left shift. Henceforth, we update the values of Δ_l and Δ_r with the new values of η_l and η_r, respectively. Note that we need to check for a special case where the earliness penalty of the first job is higher than the sum of the tardiness penalties of the jobs which are completed after the right due-date. In this case, the optimal schedule will occur when the left due-date falls at C_1. It is for this case, that we need to check that $\eta_l > 0$ before making a left shift, as depicted in Algorithm 7 line 28. For the next iteration of the *while* loop in line 9 of Algorithm 7, we update the positions of the left and right due-date by $\Delta = \min\{\Delta_l, \Delta_r\}$ and repeat the same procedure as long as $pl < pe$, according to Theorem 5.4.

We then check if the optimal position of this *movable* due-window lies to the left or the right of the original due-window. If the optimal position of the left due-date of the *movable* due-window is situated to the left of the original left due-date then we take the positions of the *movable* due-window for calculating the final earliness/tardiness of the jobs. If this is not the case, then we retain the original due-window for the objective function calculation. The reason for this has been stated in Section 5.4.

One important fact about this iterative left shift is that we can end up with a maximum of $2 \cdot n$ different left shifts. It can be understood by the fact that after every left shift we can have a case where $d'_l = C_u$, $C_{v-1} < d'_r < C_v$, and vice-versa. Note that in our illustrative example we made 7 different left shifts, for a sequence of 5 jobs. □

5.6 Algorithm Runtime Complexity

In this section we study and prove the runtime complexity of Algorithm 7.

Theorem 5.6. *The runtime complexity of Algorithm 7 is $O(n)$ where n is the total number of jobs.*

Proof. It can easily observed that the complexity of the Algorithm 7 is $O(n)$, since all the initialization steps and the calculation of the parameters all require $O(n)$ runtime. As for the iterative *while* loop, all the computations inside the loop are of $O(1)$ as we update the values of any parameter by simple one step computation. However, as said before, this iterative left shift can take $2 \cdot n$ steps in the worst case, however, it does not affect the complexity of Algorithm 5.4. Hence, the complexity of Algorithm 7 is $O(n)$. □

5.7 Improvement Heuristic for CDW Job Sequence

We now present a straight forward heuristic algorithm which is utilized to locally improve any CDW job sequence, by implementing the V-shaped property given below. This heuristic is exactly the same as the one we present for the CDD problem in Chapter 4. As we mentioned in Property 5.2, the V-shaped property holds for the CDW problem in the similar fashion to the CDD problem. We utilize this property to develop an improvement heuristic to evolve any job sequence, optimized by Algorithm 7. Algorithm 7 is implemented to obtain the straddling jobs of the sequence, or in other words, the position of the due-window in the optimal schedule of any job sequence J. As we know from Algorithm 7 and Property 5.3, the optimal schedule for the CDW occurs when one of the due-dates for the due-window lies at the completion time of some job, or when the first job starts at times $t = 0$. Let, l be the last early job and m be the first tardy job. Our heuristic arranges the early jobs $J_E = \{1, 2, \ldots, l\}$ in the non-increasing order of their P_i/α_i values, where $i = 1, 2, \ldots, l$. Likewise, the tardy jobs $J_T = \{m, m+1, \ldots, n\}$ are arranged in the non-decreasing order of their P_i/β_i values. The new sequence is obtained by placing the straddling job $J_S = \{l+1, l+2, \ldots, m-1\}$ in between the sorted sequences of J_E and J_T. Algorithm 8 shows our improvement heuristic to locally improve and job sequence, as per the V-shaped property.

Algorithm 8: Heuristic to evolve any job sequence of the CDW problem.

Apply Algorithm 7
$l \leftarrow$ last early job
$m \leftarrow$ first tardy job
$J_E \leftarrow \{1, 2, \ldots, l\}$
$J_S \leftarrow \{l+1, l+2, \ldots, m-1\}$
$J_T \leftarrow \{m, m+1, \ldots, n\}$
$J'_E \leftarrow$ Jobs sorted in non-increasing order of $P_i/\alpha_i, \forall i \in J_E$
$J'_T \leftarrow$ Jobs sorted in non-decreasing order of $P_i/\beta_i, \forall i \in J_T$
$J' = \{J'_E, J_S, J'_T\}$
$Sol \leftarrow$ Algorithm 7 on J'

5.8 Computational Results

We now present our computational results for our approach discussed in this chapter. We use the CDW benchmark instances provided by Biskup and Feldmann in [21] and compare our results with theirs. All the computations were carried out on a 1.73 GHz PC with 2 GB RAM on MATLAB with C++ mex functions. As described in the previous chapters, we implement a modified Simulated Annealing algorithm to generate job sequences, while each job sequence is optimized with Algorithm 7 and further improved by the improvement heuristic explained in the previous section. The SA parameters and methodology to generate the job sequences is similar to the one we use for the CDD problem in Chapter 4. The only difference is in the perturbation rule due to the problem structure. As for the perturbation rule, we first randomly select a certain number of jobs in any job sequence and permute them randomly to create a new sequence. The number of jobs selected for this permutation is taken as $3 + \lfloor \sqrt{n/10} \rfloor$, where n is the number of jobs. In addition to this, we also incorporate swapping of one job each from the set J'_E and J'_T

with each other. This swapping is specially useful for the instances with large due-window size, as random perturbation might cause little to no effect on the job sequence if all the jobs that are perturbed belong to the straddling set of jobs J_S. The reason for this is that changing the sequence of straddling jobs does not change the objective function value.

We present our results for the CDW where the due-window size for any instance is calculated using the values of h_1 and h_2, known as the *due-window restriction factor*. A due-window has a left (d_l) and right (d_r) due-date, where $d_l = \lfloor h_1 \cdot \sum_{i=1}^{n} P_i \rfloor$ and $d_r = \lfloor h_2 \cdot \sum_{i=1}^{n} P_i \rfloor$, as described in [21]. For each due-window there exists 10 different benchmark instances for every problem size. Hence there a total of 50 different benchmark instances for each job size. We carry out 100 different replications of our Simulated Annealing algorithm on each of the 250 benchmark instances of different job sizes and 5 due-window factors given by $h1$ and $h2$. In Table 5.2 and 5.3 we present our results for the CDW where the due-window size for any instance is calculated using the values of $h1$ and $h2$. A due-window has a left (d_l) and right (d_r) due-date, where $d_l = \lfloor h1 \cdot \sum_{i=1}^{n} P_i \rfloor$ and $d_r = \lfloor h2 \cdot \sum_{i=1}^{n} P_i \rfloor$, as described in [21]. As can be seen in Table 5.2, for the first 50 instances with 10 jobs we obtain the optimal solution for all the instances. For the instance size of 20 we reach the benchmark results for 47 instances and achieve better results for 3 instances. The benefit our approach is highlighted with the results for jobs 50 or more. We achieve better solution for 42 instances out of 50 and equal results for 6 benchmark instances, for the instance size of 50 jobs. Furthermore, for the job size of 100 our approach offers better solution values for 49 out of 50 instances provided in [21]. With the job size of 200, we again achieve better results for a total of 49 instances out of the 50 instances with varying due-window sizes, as shown in Table 5.3.

Additionally, we also carry out statistical analysis of our results for the benchmark instances till 200 jobs, and provide the best, worst, mean, median, mode, standard deviation and the average number of fitness function evaluations, along with the average runtime, for each job size and due-window location defined by h_1 and h_2. These results are presented in Table 5.4 and are expressed in terms of the percentage error with respect to the benchmark results provided by Biskup and Feldmann [21]. Let F^k_{REF} be the benchmark solution for an instance k and F^k_i be the solution obtained with our approach for ith replication of SA, then the percentage error $\varDelta^k_i$ for that run of SA is calculated as

$$\varDelta^k_i = \frac{(F^k_i - F^k_{\text{REF}}) * 100}{F^k_{\text{REF}}} . \tag{5.3}$$

As mentioned before, there are 10 different benchmark instances for each job size for any given due-window, *i.e.*, $k = 1, 2, \ldots, 10$, we represent our results in terms of the average percentage error $\overline{\varDelta}_i$ for the ith replication of SA, where $\overline{\varDelta}_i = \left(\sum_{k=1}^{10} \varDelta^k_i\right)/10$. In Table 5.4, $\overline{\boldsymbol{\varDelta}}_{\mathbf{best}} = \min\limits_{i=1,2,\ldots,100} \{\overline{\varDelta}_i\}$ represents the best average percentage error with the benchmark results, for 100 different

Table 5.2. Results obtained for single machine common due-window problem till 50 jobs. For each job there are 10 different instances each with a value for k and for each k there are 5 different due-windows.

k	**h1 - h2**	**10***		**20**		**50**	
		BR	Algo 7+SA	BR	Algo 7+SA	BR	Algo 7+SA
1	0.1 - 0.2	1896	1896	4089	4089	39250	39461
	0.1 - 0.3	1330	1330	2713	2713	28225	28225
	0.2 - 0.5	540	540	1162	1162	12756	12754
	0.3 - 0.4	919	919	2294	2294	21137	21110
	0.3 - 0.5	587	587	1559	1559	14002	13971
2	0.1 - 0.2	947	947	8251	8251	29110	29043
	0.1 - 0.3	539	539	5950	5950	20133	20133
	0.2 - 0.5	191	191	2770	2770	8480	8470
	0.3 - 0.4	432	432	4482	4482	15166	15150
	0.3 - 0.5	265	265	2923	2923	9436	9428
3	0.1 - 0.2	1488	1488	5881	5881	33407	33180
	0.1 - 0.3	1012	1012	4067	4067	23027	23020
	0.2 - 0.5	398	398	1675	1675	9935	9969
	0.3 - 0.4	760	760	3035	3035	17640	17508
	0.3 - 0.5	462	462	1998	1998	11402	11389
4	0.1 - 0.2	2128	2128	8977	8977	25869	25856
	0.1 - 0.3	1576	1576	6609	6609	17568	17544
	0.2 - 0.5	712	712	3113	3113	7378	7373
	0.3 - 0.4	1162	1162	4832	4830	13633	13609
	0.3 - 0.5	740	740	3210	3210	8448	8418
5	0.1 - 0.2	1150	1150	4028	4028	31468	31456
	0.1 - 0.3	755	755	2850	2850	21693	21689
	0.2 - 0.5	284	284	1192	1192	8954	8947
	0.3 - 0.4	542	542	2112	2112	15767	15747
	0.3 - 0.5	339	339	1341	1341	9994	9956
6	0.1 - 0.2	1479	1479	6306	6306	33452	33452
	0.1 - 0.3	1023	1023	4247	4247	23267	23261
	0.2 - 0.5	439	439	1557	1557	10245	10221
	0.3 - 0.4	779	779	3042	3042	17400	17392
	0.3 - 0.5	500	500	1778	1778	11207	11178
7	0.1 - 0.2	2093	2093	10204	10204	42257	42234
	0.1 - 0.3	1521	1521	7492	7492	29277	29274
	0.2 - 0.5	717	717	3573	3573	12014	12000
	0.3 - 0.4	1190	1190	5722	5722	20718	20696
	0.3 - 0.5	809	809	3846	3846	12953	12935
8	0.1 - 0.2	1644	1644	3749	3742	42220	42218
	0.1 - 0.3	1287	1287	2519	2519	28411	28403
	0.2 - 0.5	670	670	991	990	11167	11154
	0.3 - 0.4	952	952	1801	1801	21014	20965
	0.3 - 0.5	680	680	1069	1069	12917	12913
9	0.1 - 0.2	1466	1466	3317	3317	33222	33222
	0.1 - 0.3	1121	1121	2342	2342	23848	23840
	0.2 - 0.5	492	492	1056	1056	10987	10977
	0.3 - 0.4	772	772	1767	1767	17999	17972
	0.3 - 0.5	513	513	1187	1187	11951	11935
10	0.1 - 0.2	1835	1835	4673	4673	31492	31492
	0.1 - 0.3	1384	1384	3266	3266	22056	22040
	0.2 - 0.5	691	691	1355	1355	9653	9653
	0.3 - 0.4	1047	1047	2419	2419	16538	16510
	0.3 - 0.5	717	717	1474	1474	10628	10597

Table 5.3. Results obtained for single machine common due-window problem for larger instances with 100 and 200 jobs.

k	**h1 - h2**	**100**		**200**	
		BR	Algo 7+SA	BR	Algo 7+SA
1	0.1 - 0.2	139595	139573	474756	474431
	0.1 - 0.3	95217	95219	324620	324499
	0.2 - 0.5	39553	39515	136994	136824
	0.3 - 0.4	72187	72121	246945	246672
	0.3 - 0.5	46020	45832	158530	158158
2	0.1 - 0.2	120511	120484	517562	517316
	0.1 - 0.3	82105	82031	353194	353020
	0.2 - 0.5	35399	35303	145664	145495
	0.3 - 0.4	62458	62423	267993	267706
	0.3 - 0.5	39813	39720	169796	169574
3	0.1 - 0.2	124365	124317	466715	466706
	0.1 - 0.3	86305	86241	319485	319373
	0.2 - 0.5	38195	38179	134770	134657
	0.3 - 0.4	67204	67054	245096	245023
	0.3 - 0.5	44184	44054	157146	157013
4	0.1 - 0.2	122983	122901	564907	564840
	0.1 - 0.3	84155	84117	396769	396636
	0.2 - 0.5	35640	35511	177161	176946
	0.3 - 0.4	65082	65062	304831	304646
	0.3 - 0.5	41770	41591	200878	200675
5	0.1 - 0.2	119151	119115	489018	488841
	0.1 - 0.3	82506	82398	333670	333486
	0.2 - 0.5	34956	34873	140147	139995
	0.3 - 0.4	61038	60965	254798	254525
	0.3 - 0.5	38976	38785	162410	162152
6	0.1 - 0.2	133656	133545	458252	458022
	0.1 - 0.3	89634	89567	311964	311882
	0.2 - 0.5	35341	35146	126460	126426
	0.3 - 0.4	65622	65474	233685	233382
	0.3 - 0.5	40987	40880	146369	146288
7	0.1 - 0.2	129866	129849	428986	428115
	0.1 - 0.3	91045	90963	285446	285332
	0.2 - 0.5	39449	39344	115633	115499
	0.3 - 0.4	67858	67797	220974	220294
	0.3 - 0.5	43870	43737	139361	139088
8	0.1 - 0.2	154029	153974	474702	474457
	0.1 - 0.3	106558	106525	320865	320704
	0.2 - 0.5	45070	44963	130149	129966
	0.3 - 0.4	80338	80272	235612	235490
	0.3 - 0.5	51540	51417	147181	146988
9	0.1 - 0.2	111521	111474	509383	508283
	0.1 - 0.3	75280	75159	350521	350375
	0.2 - 0.5	31417	31279	147224	147298
	0.3 - 0.4	57168	57102	263963	263255
	0.3 - 0.5	36386	36319	168249	168090
10	0.1 - 0.2	112942	112799	515112	514939
	0.1 - 0.3	78771	78670	358610	358495
	0.2 - 0.5	34152	34074	158191	157948
	0.3 - 0.4	60096	59922	274262	273839
	0.3 - 0.5	38775	38639	178679	178480

runs of SA on each instance. Likewise, $\overline{\boldsymbol{\Delta}}_{\mathbf{mean}}$ and $\overline{\boldsymbol{\Delta}}_{\mathbf{worst}}$ depict the mean and the worst average percentage error obtained on 100 different runs of SA, respectively. We also present the percentage standard deviation, depicted by $\overline{\boldsymbol{\Delta}}_{\boldsymbol{\sigma}}$, along with the average runtime in seconds as well as the total number of fitness function evaluations (**FFEs**) on average for all job sizes depending on the due-window location. A negative value for $\overline{\boldsymbol{\Delta}}_{\mathbf{best}}$, $\overline{\boldsymbol{\Delta}}_{\mathbf{mean}}$, $\overline{\boldsymbol{\Delta}}_{\mathbf{worst}}$, $\overline{\boldsymbol{\Delta}}_{\mathbf{median}}$ and $\overline{\boldsymbol{\Delta}}_{\mathbf{mode}}$ shows that the results obtained by our approach are better than the best known solution for this problem. Not only do we obtain better results in the best runs of SA but also on average of 100 runs. In the worst case as well, our results are within a percentage error of 1.1 percent.

Table 5.4. Average runtime in seconds and percentage gap of our solutions with the benchmark results of [21], for each due-window size. The values presented are the average over all 10 k-values.

n	$\mathbf{h_1}$**-**$\mathbf{h_2}$	$\overline{\boldsymbol{\Delta}}_{\mathbf{best}}$	$\overline{\boldsymbol{\Delta}}_{\mathbf{worst}}$	$\overline{\boldsymbol{\Delta}}_{\mathbf{mean}}$	$\overline{\boldsymbol{\Delta}}_{\mathbf{median}}$	$\overline{\boldsymbol{\Delta}}_{\mathbf{mode}}$	$\overline{\boldsymbol{\Delta}}_{\boldsymbol{\sigma}}$	**Runtime**	**FFEs**
10	0.1-0.2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	34
10	0.1-0.3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	44
10	0.2-0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	304
10	0.3-0.4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	88
10	0.3-0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.001	353
Average		**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**164**
20	0.1-0.2	-0.187	0.060	-0.018	-0.016	-0.019	0.002	0.014	4898
20	0.1-0.3	0.000	0.085	0.003	0.012	0.012	0.005	0.030	7029
20	0.2-0.5	-0.101	0.000	-0.010	-0.010	-0.010	0.000	0.020	7676
20	0.3-0.4	-0.041	0.000	-0.004	-0.004	-0.004	0.000	0.020	7623
20	0.3-0.5	0.000	0.000	0.000	0.000	0.000	0.000	0.021	7896
Average		**-0.066**	**0.029**	**-0.006**	**-0.004**	**-0.004**	**0.002**	**0.021**	**7024**
50	0.1-0.2	-0.679	0.555	-0.037	-0.039	-0.039	0.010	0.189	32252
50	0.1-0.3	-0.137	0.169	0.020	0.021	-0.007	0.032	0.191	32909
50	0.2-0.5	-0.234	1.132	0.219	0.197	0.069	0.177	0.138	23517
50	0.3-0.4	-0.748	0.171	-0.124	-0.124	-0.163	0.049	0.133	20229
50	0.3-0.5	-0.380	0.473	-0.021	-0.030	-0.094	0.120	0.103	18612
Average		**-0.436**	**0.500**	**0.011**	**0.005**	**-0.047**	**0.078**	**0.151**	**25504**
100	0.1-0.2	-0.127	0.049	-0.029	-0.030	-0.034	0.010	0.680	53430
100	0.1-0.3	-0.161	0.183	-0.016	-0.023	-0.051	0.040	0.804	65115
100	0.2-0.5	-0.552	0.443	-0.079	-0.087	-0.151	0.109	0.897	72869
100	0.3-0.4	-0.290	0.075	-0.100	-0.103	-0.111	0.020	0.846	71505
100	0.3-0.5	-0.490	0.105	-0.214	-0.221	-0.284	0.066	0.885	71371
Average		**-0.324**	**0.171**	**-0.088**	**-0.093**	**-0.126**	**0.049**	**0.822**	**66858**
200	0.1-0.2	-0.215	0.029	-0.062	-0.062	-0.061	0.005	3.675	155644
200	0.1-0.3	-0.055	0.168	0.001	-0.003	-0.017	0.026	4.044	173108
200	0.2-0.5	-0.160	0.641	0.073	0.061	0.017	0.106	4.392	184657
200	0.3-0.4	-0.309	0.013	-0.109	-0.109	-0.106	0.014	4.310	182128
200	0.3-0.5	-0.242	0.212	-0.054	-0.060	-0.081	0.046	4.298	180953
Average		**-0.196**	**0.213**	**-0.030**	**-0.035**	**-0.050**	**0.039**	**4.144**	**175298**

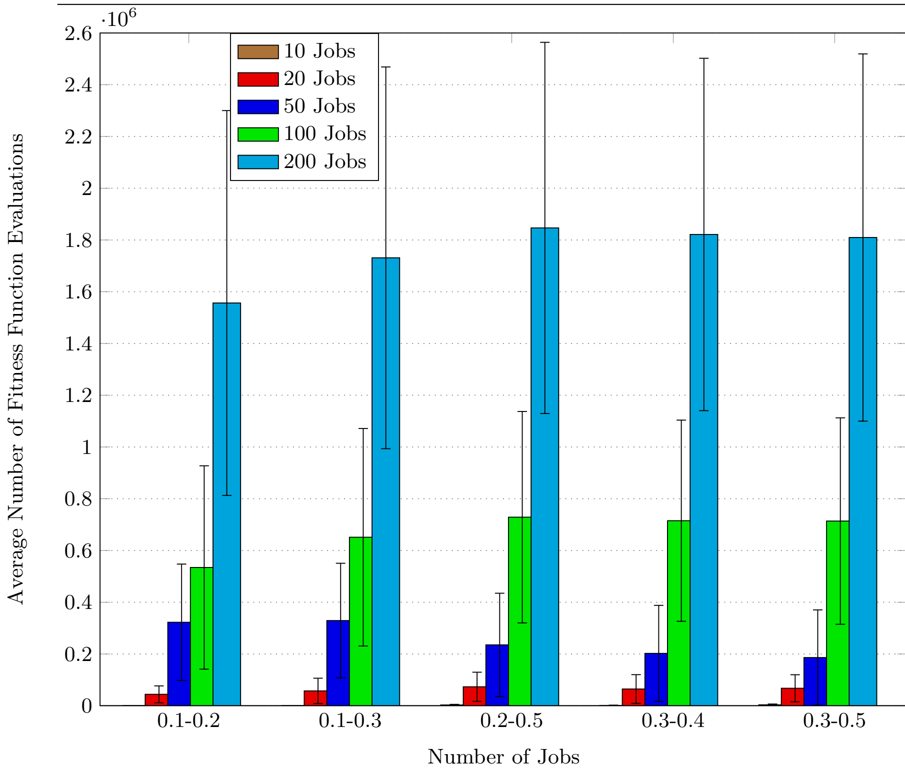


Fig. 5.11. Average Number of Fitness Function Evaluations and its standard deviation.

Figure 5.11 shows the average number of fitness function evaluations required for each job size over different due-window lengths and locations. As can be seen from the figure our approach performs consistently over all job sizes with relatively same standard deviation. We also present the graphical representation of the average percentage error obtained by our algorithm. The negative value indicates that we obtain better results than the benchmark solutions, in Figure 5.12. Evidently, the worst possible solution values by our approach are for the case when the due-window size is as big as the 30% of the total length of the schedule, *i.e.,* with the due-window restriction factor of 0.2 to 0.5. As explained before, the reason behind it is the fact that the perturbation causes the least change in the processing sequence of the jobs as several jobs belong to the straddling set. In such a case, swapping of jobs from J'_E to J'_T is highly useful. However, it must be noted that the percentage error values presented in Figure 5.12 are averaged over 100 different replications of SA and the worst value for $\overline{\Delta}_{\text{mean}}$ over all the instances is only 0.2 percent.

Regardless, over all the replications of SA, in the average case we still obtain solutions better than the best known values as is clear from Table 5.4.

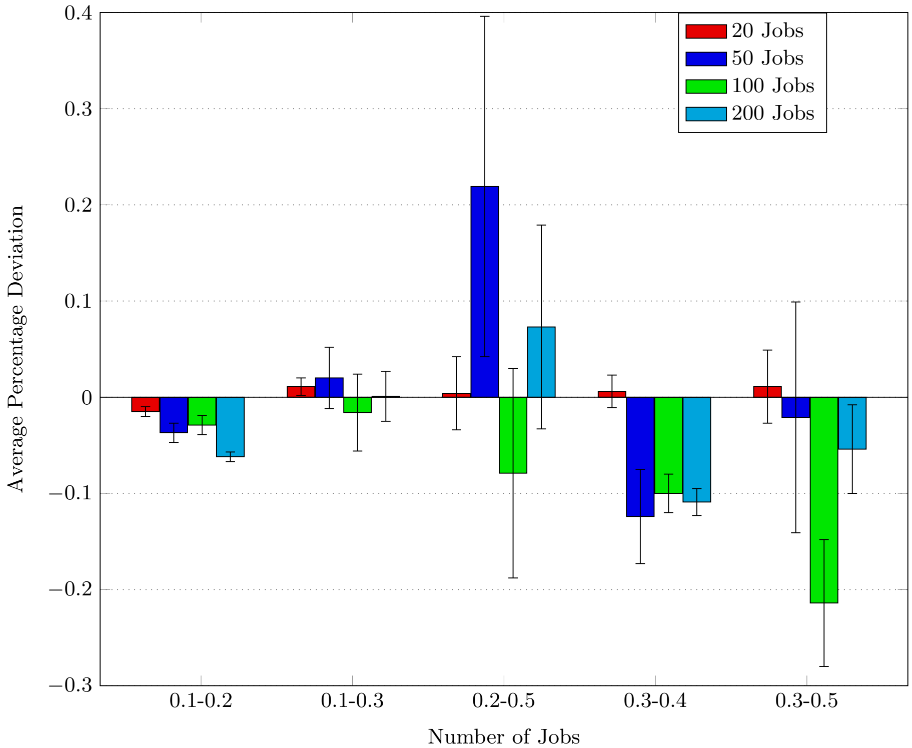


Fig. 5.12. Comparative average percentage deviation of CDW along with its standard deviation.

Furthermore, the robustness of our approach is highlighted by the small standard deviation values $\overline{\boldsymbol{\Delta}_{\boldsymbol{\sigma}}}$ over all the instances, as shown in Figure 5.12. Our algorithm consistently obtains good quality solutions with the worst possible standard deviation of just 0.177%. The average runtime over 100 different replications of our algorithm for job size of 20 and above is only $0.018, 0.151, 0.822$ and 4.144 seconds. A previous approach for this problem involved an $O(n^2)$ algorithm to optimize any job sequence [7] and the average runtimes for 10 different replications of SA were 0.173, 0.465 and 6.028 seconds for 10, 20 and 50 jobs, respectively, on the same machine. This comparison shows that the approach mentioned in this work offers a speedup of 25 and 40 for job sizes of 20 and 50, respectively. As far as the solution quality is concerned, not only our approach is robust over all the instances, we also improve several benchmark results provided in [21].

5.9 Summary

In this chapter we present a novel strategy for the Common Due-Window problem based on developing specialized linear algorithm for the linear program resulting from a given fixed job sequence. We prove the property for the CDW and prove its similarity to the CDD problem. Thereafter, we provide an $O(n)$ algorithm for a the general case of the CDW to optimize a given job sequence, proving its runtime complexity and its optimality with respect to the solution value. Additionally, we also propose an improvement heuristic similar to the CDD problem to locally improve any job sequence with the help of the V-shaped property. We applied our algorithms to the benchmark instances provided by Biskup and Feldmann [21] and obtain better results for 137 instances out of 150 benchmark instances for the job size of 50 and higher. The benefit of our approach is evident from the results and its adaptability to certain other problems that are a variant of CDW. One of the obvious case is one when the processing times of all the jobs are equal or of unit time length. Our approach for that problem would be same except for the fact that on Algorithm 7 each shift will be equal to the length of the processing time. The remaining procedure of improvement heuristics and the SA can be utilized as described in this chapter. Our approach also works for the Common Due Window Assignment (CDWA) problem with no penalty on the due-window assignment. Recall that for any job sequence, we first locate the best position of the *movable* due-window and then adjust it depending on the original due-window position. However, for the CDWA we do not need any adjustment as the position of the due-window is itself a decision variable. Hence, our approach can find the optimal (near optimal) due-window assignment in exactly the same manner. Similar to the parallel machine case of the CDD problem, the algorithm mentioned in this chapter is also suitable for the parallel machine case of the CDW problem. A brief version of this chapter has also been accepted for publication in [155].

6

Un-restricted Common Due-Date Problem with Controllable Processing Times: Polynomial Algorithm for a Given Job Sequence

This chapter considers another variant of the CDD problem known as the un-restricted case of the Common Due-Date problem with controllable processing times. The problem consists of scheduling jobs with controllable processing times on a single machine against a common due-date to minimize the overall earliness/tardiness and the compression penalties of the jobs. The objective of the problem is to find the processing sequence of jobs, the optimal reduction in the processing times of the jobs and their completion times. We first present and prove an essential property for the controllable processing time CDD problem for the un-restricted case along with an exact polynomial algorithm for optimizing a given job sequence for a single machine with a runtime complexity of $O(n)$, where n is the number of jobs. Henceforth, we implement our polynomial algorithm in conjunction with a modified Simulated Annealing algorithm and Threshold Accepting to obtain the optimal/best processing sequence while comparing the two heuristic approaches, as well. The implementation is carried out on appended CDD benchmark instances provided in the OR-library.

6.1 Introduction

When a production is made against a due-date, the manufacturer can have another modification to reduce the over penalty of the production. If a job can be processed in a shorter time than its actual processing time, or in other words, the processing time of the job is reduced, then the overall earliness/-tardiness penalties can be reduced further. However, reducing the processing time of a job essentially means that the job is processed by the machine faster than its usual processing time. In doing so, the machine needs to operate at rather extreme pace, consuming more resources such a fuel. Due to this reason, a penalty per unit time is associated with each job in case it is processed faster, in other words, when the processing time is reduced. This penalty is termed as the compression penalty of the job. This modification leads to another variation of the CDD, known as the Common Due-Date Problem with controllable processing time (CDDCP).

As explained in the previous chapter, CDD involves minimization of the total weighted earliness/tardiness penalty against a common due-date. For the controllable processing time case, in addition to the CDD, the processing times of some or all the jobs can be reduced to a certain minimum value at a cost of some penalty per unit of reduction. This controlling of the processing times can help the jobs to reduce their earliness/tardiness penalties if the penalties incurred due to the compressions are relatively smaller than the earliness/tardiness penalties. The objective of solving the problem is to obtain the optimal job sequence, final processing times of the jobs and the completion times of all the jobs to minimize the total weighted penalty. Generally speaking, there are two classes of common due-date problem, which have proven to be NP-hard, namely the restrictive and the un-restrictive CDD problem. In this work, we consider the un-restrictive case of this problem, where the common due-date is greater than or equal to the sum of the processing times of all the jobs and each job possesses different penalties. The CDD has already been proven to be NP-hard, and clearly the controllable case is NP-hard as well [149, 20].

6.2 Related Work

Panwalkar and Rajagopalan studied the CDDCP problem with constant earliness/tardiness penalties and distinct compression penalties against a common due-date and presented a polynomial algorithm for the special case [117]. Cheng *et al.* considered the single machine scheduling with compressible processing times and assignable due-date with constant penalties for earliness/-tardiness and compression [36]. In 1999 Biskup and Cheng studied the controllable processing times common due-date problem with constant penalties for earliness/tardiness and distinct penalties for compression. They also considered the penalty for the completion time of the jobs and proved the similarity of the problem to the assignment problem [19]. In 2001 Biskup and Jahnke studied a slightly different version of the problem. They analyse the assignable due-date problem with controllable processing times but instead of arbitrary compression of the jobs, they consider the case where each job is reduced by a constant proportional amount. Besides, they consider the case where each job possesses constant penalties for earliness/tardiness and the compression of the jobs [22].

In 2007 Shabtay and Steiner made an extensive survey for scheduling with controllable processing times, covering research in this field from the last 25 years [129]. Wan studied the common due-window problem with controllable processing times with constant earliness/tardiness penalties and distinct compression costs and discussed some properties for the optimal solution along with a polynomial algorithm for solving the problem, in 2007 [143]. In 2009, Tseng *et al.* studied the general problem with compressible processing times with distinct due-dates and presented a heuristic algorithm to minimize the total tardiness and the compression penalties [139]. Nearchou studied a slightly different version of the problem in 2010, where the objective was to minimize the total weighted completion times and the compression costs and presented a population based metaheuristic algorithm, considering four different heuristic approaches namely, differential evolution, particle swarm optimization, genetic algorithms and evolution strategies [113]. Yin *et al.* considered the single machine batch delivery scheduling with assignable common due-date and controllable processing times with constant penalties and presented a $O(n^5)$ dynamic programming algorithm, in 2013 [149]. Again in 2013 Kayvanfar *et al.* studied the general case with distinct due-dates for all the jobs. Additionally, they also consider the case where the processing times of the jobs can be both compressed and expanded. They also study the parallel machine case with the additional constraint to the objective function which minimizes the makespan of the schedule, as well [80]. Yin *et al.* consider the problem of controllable processing times against a CDD, and study the case with constant earliness/tardiness penalties. Additionally, the common due-date is taken as a decision variable, in 2014 [150]. Again in the same year, Lu *et al.* study the due-date assignment problem for the case where the processing times of the jobs is dependent on the resource. However, they also consider the case with constant earliness/tardiness penalties [100]. In 2014, Yang *et al.* consider the problem multiple due-window assignments with controllable processing times, with constant penalties for earliness/tardiness, compression of the jobs and due-window position for any job [147].

We consider the single machine case for the un-restricted CDD problem with asymmetric penalties and controllable processing times with distinct linear costs. We make a theoretical study of the problem and first present an important property for this problem. We then present an $O(n)$ exact polynomial algorithm to optimize a given job sequence on a single machine.

6.3 Problem Formulation

In this Section, we present the mathematical notation of the common due-date problem with the controllable processing times. Let,

n = number of jobs,
d = common due-date,
P_i = actual processing time for job i, $\forall i = 1, 2, \ldots, n$,
mP_i = minimum processing time for job i, $\forall i = 1, 2, \ldots, n$, $mP_i \leq P_i \; \forall \; i$
C_i = completion time of job i,
g_i = earliness of job i, where $g_i = \max\{0, d - C_i\}$,
h_i = tardiness of job i, where $h_i = \max\{0, C_i - d\}$,
x_i = actual reduction in the processing time of job i,
α_i = earliness penalty per time unit for any job i,
β_i = tardiness penalty per time unit for any job i,
ν_i = compression penalty per time unit for any job i,
The objective functions for the studied problem can then be expressed as,

$$\min \sum_{i=1}^{n} (\alpha_i \cdot g_i + \beta_i \cdot h_i + \nu \cdot x_i) \; . \tag{6.1}$$

6.4 Important Properties for the UCDDCP

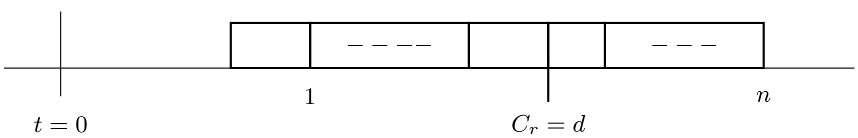


Fig. 6.1. Assume that the rth job finishes at the due-date d in the optimal solution.

In this section we prove some properties for the un-restricted CDD with controllable processing times. Let the solution value for the case when there is no compression of the processing times and the due-date lies at the completion time of job r, as shown in Figure 6.1, be Sol_r. Then we have

$$Sol_r = \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} P_j \alpha_i + \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} P_j \beta_i \; , \tag{6.2}$$

where
$\sum\limits_{i=1}^{r-1} \sum\limits_{j=i+1}^{r} P_j$ = the total earliness for any job i and
$\sum\limits_{i=r+1}^{n} \sum\limits_{j=r+1}^{i} P_j$ = the total tardiness for any job i.
Let us assume that the reductions in the processing times in the optimal schedule be x_i for all $i = 1, 2, \ldots, n$. Then the objective function value (Sol'_r) when the due-date position is at C_r will be

$$Sol'_r = \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} (P_j - x_j)\alpha_i + \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} (P_j - x_j)\beta_i + \sum_{j=1}^{n} x_j \nu_j \ . \tag{6.3}$$

We first present and prove an important property regarding the amount of compression of the processing times of the jobs.

Property 6.1. If controlling the processing times fetches a better solution, then the compression of the processing times should be to their minimum value.

Proof. If the compression of the processing times fetches a better solution, then we have $Sol'_r \leq Sol_r$. Using Equation (6.2) and (6.3), we obtain

$$\sum_{i=1}^{r-1} \sum_{j=i+1}^{r} x_j \alpha_i + \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} x_j \beta_i - \sum_{j=1}^{n} x_j \nu_j \geq 0 \ . \tag{6.4}$$

Let us assume that instead of reducing the processing times by x_j, we reduce them by y_j, where $y_j < x_j \ \forall j = 1, 2, \ldots,$. Let the solution value for this case be $Sol'_{r'}$ and $x_j - y_j = \epsilon_j$. If $Sol'_{r'} < Sol'_r$, then with some manipulation of the terms we get

$$\sum_{i=1}^{r-1} \sum_{j=i+1}^{r} \epsilon_j \alpha_i + \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} \epsilon_j \beta_i - \sum_{j=1}^{n} \epsilon_j \nu_j \leq 0 \ . \tag{6.5}$$

Since $x_j \geq 0 \ \forall j = 1, 2, \ldots, n$, Equation (6.4) should also hold for any $\epsilon_j > 0$. However, Equation (6.5) is a contradiction. Hence, our assumption that $Sol'_{r'} < Sol'_r$ is wrong. This proves that the solution value only improves if we reduce the processing times further, which in turn shows that the best solution value will be obtained for maximum possible compression of the processing times. □

We now present and prove a novel property for the UCDDCP problem. Recall from Chapter 4 that we presented a property for the common due-date problem, where the optimal schedule position was independent of the processing times of the jobs but depended only on the earliness/tardiness penalties of the jobs. Since we are dealing with only the un-restricted case of the CDD, Equation (6.6) and (6.7) depict this property if the optimal schedule occurs when the due-date d falls at the completion time of job r.

$$\sum_{i=k+1}^{n} \beta_i \leq \sum_{i=1}^{k} \alpha_i, \ k = r, r+1, \ldots, n \text{ and} \tag{6.6}$$

$$\sum_{i=1}^{k-1} \alpha_i \leq \sum_{i=k}^{n} \beta_i, \ k = 1, 2, 3, \ldots, r \ . \tag{6.7}$$

We now use this property of the CDD prove an essential property for the un-restricted case of the CDD with controllable processing times.

Theorem 6.2. *If the due-date position in the optimal schedule of the un-restricted case of the CDD falls at the completion time of some job r, then its position remains unchanged for the controllable case of the un-restricted CDD problem.*

Proof. We know from Theorem 4.10 that if the optimal CDD schedule has the due-date at the completion time of job r, then Equation (6.6) and (6.7) hold. Besides, we also proved in Property 6.1 that if a job is reduced then it has to be reduced to its minimum processing time, to gain from the compression of the jobs. Let us consider that the optimal reduction of the processing times of the jobs is given by $x_i \ \forall \ i$ to minimize Equation (6.1) for the given job sequence. Then, for this reduction the objective function value Sol'_r for the case when due-date position is at C_r can be written as Equation (6.3).
Rearranging the terms of Sol'_r from Equation (6.3), we have

$$\begin{aligned} Sol'_r &= \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} (P_j - x_j)\alpha_i + \sum_{j=1}^{n} x_j \nu_j + \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} (P_j - x_j)\beta_i \ , \\ Sol'_r &= \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} P_j \alpha_i - \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} x_j \alpha_i - \sum_{i=r+2}^{n} \sum_{j=r+2}^{i} x_j \beta_i + \sum_{j=1}^{n} x_j \nu_j \\ &+ \sum_{i=r+1}^{n} (P_{r+1} - x_{r+1})\beta_i + \sum_{i=r+2}^{n} \sum_{j=r+2}^{i} P_j \beta_i \ . \end{aligned} \tag{6.8}$$

With some manipulations of terms, Sol'_r can be also written as

$$\begin{aligned} Sol'_r &= \sum_{i=1}^{r-2} \sum_{j=i+1}^{r-1} P_j \alpha_i - \sum_{i=1}^{r-2} \sum_{j=i+1}^{r-1} x_j \alpha_i + \sum_{i=1}^{r-1} (P_r - x_r)\alpha_i + \sum_{j=1}^{n} x_j \nu_j \\ &+ \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} P_j \beta_i - \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} x_j \beta_i \ . \end{aligned} \tag{6.9}$$

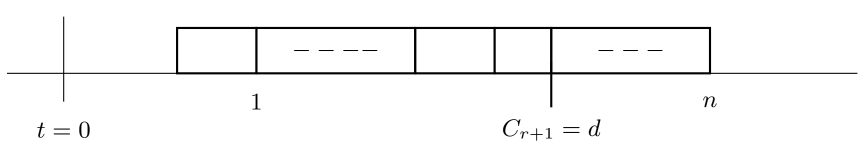


Fig. 6.2. Schedule with the completion time of job $r + 1$ lying at the due-date, $C_{r+1} = d$.

Likewise, if the jobs are shifted to the left such that the due-date now falls at the completion time of job $r+1$ (Figure 6.2), the objective function value for the optimal reduction x_i for this case can be written as Sol'_{r+1}, where

$$Sol'_{r+1} = \sum_{i=1}^{r}\left(\sum_{j=i+1}^{r+1}(P_j - x_j)\right)\alpha_i + \sum_{j=1}^{n} x_j\nu_j + \sum_{i=r+2}^{n}\left(\sum_{j=r+2}^{i}(P_j - x_j)\right)\beta_i \ . \quad (6.10)$$

As for Sol'_r, the terms of Sol'_{r+1} in Equation (6.10) can also be rearranged such that

$$\begin{aligned} Sol'_{r+1} &= \sum_{i=1}^{r}\sum_{j=i+1}^{r+1}(P_j - x_j)\alpha_i + \sum_{i=r+2}^{n}\sum_{j=r+2}^{i}(P_j - x_j)\beta_i + \sum_{j=1}^{n} x_j\nu_j \\ &= \sum_{i=1}^{r-1}\sum_{j=i+1}^{r} P_j\alpha_i - \sum_{i=1}^{r-1}\sum_{j=i+1}^{r} x_j\alpha_i + \sum_{i=r+2}^{n}\sum_{j=r+2}^{i} P_j\beta_i \\ &\quad + \sum_{i=1}^{r}(P_{r+1} - x_{r+1})\alpha_i - \sum_{i=r+2}^{n}\sum_{j=r+2}^{i} x_j\beta_i + \sum_{j=1}^{n} x_j\nu_j \ . \end{aligned} \quad (6.11)$$

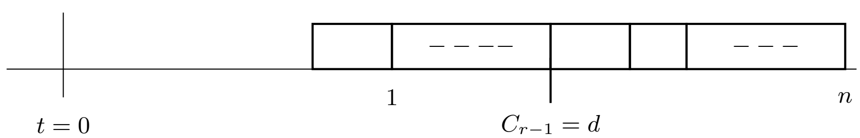


Fig. 6.3. Schedule with the completion time of job $r-1$ lying at the due-date, $C_{r-1} = d$.

Now, if the jobs from Figure 6.1 are shifted to the right such that the due-date now falls at the completion time of job $r-1$ (Figure 6.3), the objective function value for the optimal reduction x_i can be written as

$$Sol'_{r-1} = \sum_{i=1}^{r-2}\left(\sum_{j=i+1}^{r-1}(P_j - x_j)\right)\alpha_i + \sum_{i=r}^{n}\left(\sum_{j=r}^{i}(P_j - x_j)\right)\beta_i + \sum_{j=1}^{n} x_j\nu_j \ . \quad (6.12)$$

Likewise Sol'_{r-1} in Equation (6.12) can also be expressed as

$$
\begin{aligned}
Sol'_{r-1} =& \sum_{i=1}^{r-2} \sum_{j=i+1}^{r-1} (P_j - x_j)\alpha_i + \sum_{i=r}^{n} \sum_{j=r}^{i} (P_j - x_j)\beta_i + \sum_{j=1}^{n} x_j \nu_j \\
=& \sum_{i=1}^{r-2} \sum_{j=i+1}^{r-1} P_j \alpha_i - \sum_{i=1}^{r-2} \sum_{j=i+1}^{r-1} x_j \alpha_i + \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} P_j \beta_i \\
&+ \sum_{i=r}^{n} (P_r - x_r)\beta_i - \sum_{i=r+1}^{n} \sum_{j=r+1}^{i} x_j \beta_i + \sum_{j=1}^{n} x_j \nu_j \ .
\end{aligned} \tag{6.13}
$$

Now we prove by contradiction that the position of the due-date will not change even after the optimal reduction in the processing times. Let us assume that Sol'_r is not optimal, then we have

$$
Sol'_r > Sol'_{r+1} \text{ and} \tag{6.14}
$$

$$
Sol'_r > Sol'_{r-1} \ . \tag{6.15}
$$

Substituting values of Sol'_r from Equation (6.8) and Sol'_{r+1} from Equation (6.11) in Equation (6.14), we have

$$
\begin{aligned}
Sol'_r >& Sol'_{r+1} \ , \\
\sum_{i=r+1}^{n} (P_{r+1} - x_{r+1})\beta_i >& \sum_{i=1}^{r} (P_{r+1} - x_{r+1})\alpha_i \text{ and}
\end{aligned}
$$

$$
(P_{r+1} - x_{r+1}) \left(\sum_{i=r+1}^{n} \beta_i - \sum_{i=1}^{r} \alpha_i \right) > 0 \ . \tag{6.16}
$$

Now, using Equation (6.6), we obtain

$$
P_{r+1} < x_{r+1} \ . \tag{6.17}
$$

Likewise, substituting the values of Sol'_r from Equation (6.9) and Sol'_{r-1} from Equation (6.13) in Equation (6.15), we get

$$
\begin{aligned}
Sol'_r >& Sol'_{r-1} \ , \\
\sum_{i=1}^{r-1} (P_r - x_r)\alpha_i >& \sum_{i=r}^{n} (P_r - x_r)\beta_i \text{ and}
\end{aligned}
$$

$$(P_r - x_r)\left(\sum_{i=1}^{r-1}\alpha_i - \sum_{i=r}^{n}\beta_i\right) > 0 \; . \tag{6.18}$$

Equation (6.7) then fetches

$$P_r < x_r \; . \tag{6.19}$$

Equation (6.17) and (6.19) show that if the optimal solution for the uncompressed case occurs such when the due-date position is at C_r for some r, then for the compressed case, the position of the due-date will remain fixed at C_r as well, since a change in the position of the due-date will require a compression in the processing time which is more than the actual processing time itself. □

6.5 The Exact Algorithm

In the previous section we proved that if the due-date position for the general common due-date problem lies at the completion time of a job then its position remains unchanged for the controllable processing time case as well. We now present how to utilize this property to formulate an exact algorithm to optimize a given job sequence for the un-restricted case of the CDD with controllable processing times.

To optimize a given sequence for the un-restricted case, we first find the optimal position of the due-date without any compression of the jobs and then reduce the processing times of the jobs closest to the due-date moving outward. Consider Figure 6.1, the optimal position of the due-date is at C_r. In the next step, we first reduce the processing times of tardy jobs starting with job $r + 1$. Reducing its processing time such that C_{r+1} moves closer to d will not only reduce the tardiness of job $r + 1$ but of all the jobs which follow, provided the penalty incurred by compressing the processing time of job $r + 1$ is less than the reduction in the weighted tardiness penalties of the jobs $r + 1, r + 2, \ldots, n$. Thereafter, we compress job $r + 2$ and reduce its tardiness along with all the jobs following it. If a compression does not offer any reduction in the overall penalty then we move on to the next job without compressing the current job.

We perform the same operations in the sequential manner for the remaining jobs, starting with job r to job 2. However, in this case the reduction in the processing times leads the jobs to move towards right, *i.e.*, closer to the due-date. Notice that the reduction in the first job is never going to improve the penalty since the earliness of the first job will not be altered by its compression but will only offer more penalty due to compression. Algorithm 9 presents the pseudo code for optimizing a given sequence.

Algorithm 9: Exact polynomial algorithm with $O(n)$ runtime complexity to optimize any given sequence of the un-restricted CDD with controllable processing times.

$C_i \leftarrow$ **Algorithm 4**
$\tau \leftarrow \underset{i=1,2,\ldots,n}{\arg\min}\,(C_i > d)$
$pls \leftarrow \sum_{i=\tau}^{n} \beta_i$
$i \leftarrow \tau$
$lShift \leftarrow 0$
while $(i \leq n)$ **do**
  if $(\nu_i \leq pls) \wedge (P_i > mP_i)$ **then**
    $dec \leftarrow P_i - mP_i$
    $lShift \leftarrow lShift + dec$
  $pls \leftarrow pls - \beta_i$
  $C_i \leftarrow C_i - lShift$
  $i \leftarrow i + 1$
$\tau \leftarrow \tau - 1$
$ple \leftarrow \sum_{i=1}^{\tau-1} \alpha_i$
$i \leftarrow \tau$
$rShift \leftarrow 0$
while $(i > 1)$ **do**
  if $(\nu_i \leq ple) \wedge (P_i > mP_i)$ **then**
    $inc \leftarrow P_i - mP_i$
    $rShift \leftarrow rShift + inc$
  $ple \leftarrow ple - \alpha_{i-1}$
  $C_{i-1} \leftarrow C_{i-1} + rShift$
  $i \leftarrow i - 1$
$x_1 \leftarrow 0$
$x_i \leftarrow P_i - C_i + C_{i-1},\ i = 2, 3, \ldots, n$
Calculate $g_i, h_i\ \forall\ i$
return $\sum_{i=1}^{n}(\alpha_i \cdot g_i + \beta_i \cdot h_i + \nu_i x_i)$

We now illustrate Algorithm 9, to optimize a job sequence of UCDDCP. The data for the example is given in Table 6.1. Observe that the only additional details for this problem than the CDD are the minimum processing times and the compression penalties. Since we deal with the un-restricted case we take the due-date $d = 22$ $(\geq \sum_{i=1}^{n} P_i)$. The minimum processing time of a job is the time it takes to complete, if processed faster. The compression penalty is the penalty per unit time associated with each job when the processing time of the job is reduced.

The idea of the algorithm for the UCDDCP is to first optimize the sequence for the CDD problem and then compress the jobs towards the due-date. Figure 6.4 shows the optimal schedule for the CDD problem with the second job completing at the due-date. As Property 6.2 suggests, the position of the due-date will remain unchanged for the UCDDCP problem. This, in turn means

Table 6.1. The data for the exemplary case of the UCDDCP. The parameters possess the same meaning as explained in Section 6.3.

i	P_i	mP_i	α_i	β_i	ν_i
1	6	5	7	9	5
2	5	5	9	5	4
3	2	2	6	4	3
4	4	3	9	3	2
5	4	3	3	2	1

that if the compression of the jobs yield a better solution, they necessarily have to be compressed towards the due-date (indicated by the arrows in Figure 6.4.

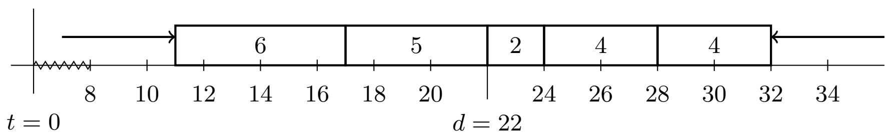


Fig. 6.4. Schedule with the completion time of job 2 lying at the due-date, after an additional right shift of all the jobs by 2 units.

Property 6.1 shows that compression of the jobs should be to their minimum value, if it fetches an improvement. Hence it gets clear that if required, the processing time of the jobs has to be reduced to their minimum possible value, indicated by mP_i in Table 6.1. Ultimately, we only need to determine which jobs should be reduced in terms of the processing times. To determine that, we start from the last job of the sequence. In the above example, we first consider job 5 which is tardy. The compression penalty of this job is 1, while the tardiness penalty is 2. Hence reduction of the processing time from $P_5 = 4$ to $mP_5 = 3$ will increase the compression penalty by $x_5 \cdot \nu_5$ (where, $x_5 = P_5 - mP_5$) but reduce the tardiness penalty by $x_5 \cdot \beta_5$ (since the job is compressed towards the due-date). Since $\beta_5 > \nu_5$, reducing the processing time of job 5 fetches us an overall improvement in the penalty by $x_5 \cdot (\beta_5 - \nu_5) = 1$. The schedule after this compression is shown in Figure 6.5.

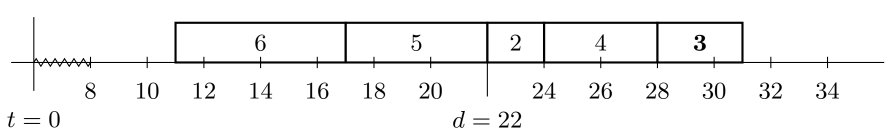


Fig. 6.5. Schedule with the reduction of job 5 to its minimum value of 3 time units.

In the next step, we move to the second last job, job 4 in this case. Job four is reducible by 1 time unit. Since there should not be any machine idle time in between jobs, observe that reducing job 4 towards the due-date will reduce the tardiness of job 4 as well as job 5. Hence, a reduction will offer an improvement by $x_4 \cdot (\beta_4 + \beta_5 - \nu_4)$. Since $\beta_4 + \beta_5 - \nu_4 = 3$, reduction of job 4 gives us a further improvement of 3 cost units, as shown in Figure 6.6. This procedure is iterated over all the jobs. The procedure for all the tardy

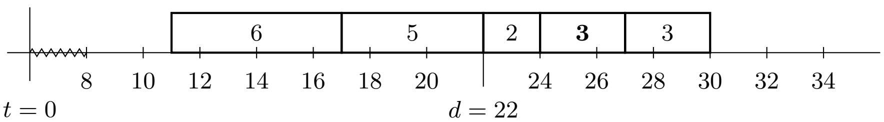


Fig. 6.6. Schedule with the reduction of job 4 to its minimum value of 3 time units.

jobs remains the same as just explained. The jobs which are early or the one which finishes at the due-date have to be dealt with in the opposite manner. A job is reduced if the compression penalty is less than the sum of the earliness penalties of all its preceding jobs. In our exemplary case, Figure 6.6 depicts the optimal schedule for the studied sequence with an overall penalty cost of 77, as any further reduction will not improve the objective function value.

6.6 Proof of Optimality and Runtime Complexity

We now provide the proof of the optimality of Algorithm 9 with respect to the solution value. Recall that we consider the un-restricted case of the CDD.

Theorem 6.3. *Algorithm 9 returns the optimal solution value to the un-restricted case of the CDD with controllable processing times for any given job sequence with a runtime complexity of $O(n)$.*

Proof. Since there is only one way that the due-date position may be between the completion times of two consecutive jobs, we need to first calculate the sum of penalties before and after the due-date such that the first job starts at time zero and all the jobs follow without any idle time. The schedule with $t^* = 0$ will be optimal if the sum of the tardiness penalties is already greater than the sum of earliness penalties. If that is not the case, we shift all the jobs towards right, as long as the sum of the tardiness penalties of jobs finishing after the due-date is less than or equal to the some of the earliness penalties of all the jobs which complete before the due-date, according to Equation (4.22) and (4.23).

Hence, we first optimize any given sequence for the general CDD problem and obtain the due-date position. We have already proved in Theorem 6.2 that the due-date position for the general CDD and the controllable processing times cases will be unaltered for the un-restricted case of the CDD. It is clear that the optimal solution will occur only if the jobs are brought closer to the due-date since the due-date position should not change and the best case would be the one when all the jobs finish at the due-date, which is impossible. Hence, we reduce the processing times of jobs starting from the most adjacent one to the due-date moving further away. The processing time of a job is reduced only if the penalty incurred due to compression is less than the penalty reduced by the reduction in the earliness (tardiness) of the jobs before (after) it.

As for the runtime complexity, the first part of Algorithm 9 is to optimize a given sequence for the un-restricted CDD problem to find the optimal position of the due-date. It can be easily observed that the complexity for this part is of linear runtime. The next expensive operations in terms of the runtime occur at the next two *while* loops and they are both of $O(n)$ in the worst case. The remaining steps are all linear and hence the overall complexity of Algorithm 9 is $O(n)$. □

6.7 Computational Experiments

Due to the unavailability of benchmark instances for this problem as per our knowledge, we first present our methodology to append the benchmark instances of the general CDD provided in the OR-library with the extra parameters for the controllable processing time case [14]. The instances provided in [14] provide the processing times, earliness/tardiness penalties and the due-date. Hence, we append the information about the minimum processing times and the cost of controlling the processing times per unit time. We take the minimum processing time of any job as $mP_i \sim DU(0.6P_i, P_i)$ and $\nu_i \sim DU(1, 5)$, where $\sim DU(a, b)$ is a discrete uniform random number between a and b. The rest of the parameters remain the same as in [14]. These benchmark instances for the UCDDCP problem are available at `http://eadgroup.org/research/benchmark-data.html`. In this work we implement our polynomial algorithm for any given job sequence in conjunction with the Simulated Annealing and Threshold Accepting, as explained below.

6.7.1 Modified Simulated Annealing

We use a modified Simulated Annealing (SA) algorithm explained in Chapter 3 to generate job sequences and Algorithm 9 to optimize each sequence to its minimum penalty. Our experiments over all the instances suggest that an ensemble size of $\approx n/10$ and the maximum number of iterations of $500n$, where n is the number of jobs, work best for the provided instances in general. As for the perturbation rule, we first randomly select a certain number of jobs in any job sequence and permute them randomly to create a new sequence, in the same manner as explained in Chapter 3. The number of jobs selected for this permutation is taken as $c + \lfloor\sqrt{n/10}\rfloor$, where n is the number of jobs and c is a constant.

6.7.2 Threshold Accepting

Threshold Accepting (TA) is another heuristic algorithm based on Simulated Annealing, proposed by Dueck and Scheuer [51]. The basic difference from the SA is the different acceptance rules. Unlike the standard SA where a worse solution is accepted as per the metropolis acceptance criterion, TA instead accepts *'every new configuration which is not much worse than the old one'* [51]. The exact details of the acceptance criterion are as follows.

The initial temperature (T_0) is kept the same as in the Simulated Annealing. As opposed to the exponential cooling schedule of SA, we adopt probabilistic arithmetic cooling scheduling in TA. Let, mE_j and mE_{j-1} be the mean of the energy (in this optimization problem, energy is the objective function values) of all the states in the current (j) and the previous iteration ($j-1$), respectively. Then, the temperature T_j is reduced as

$$T_j = \begin{cases} T_{j-1} - \theta, & \text{if } mE_j - mE_{j-1} \leq prob, \\ T_{j-1}, & \text{otherwise .} \end{cases} \tag{6.20}$$

In Equation (6.20), θ and $prob$ are defined as $\theta = \varrho \cdot T_0$ and $prob = \omega \cdot T_0/\sqrt{M}$,where $\varrho = c_1 \cdot 10^{-1}$ and $\omega = c_2 \cdot 10^{-4}$, c_1 and c_2 are integer constants less than 5 and M is the ensemble size. The acceptance criterion for Threshold Accepting as proposed by Dueck and Scheuer [51] is the current temperature T_j at any iteration j. The remaining parameters such as the ensemble size, perturbation size and the number of iterations are kept the same for both SA and TA, to exactly compare the two approaches.

Table 6.2. Results obtained for single machine common due-date problem with controllable processing times. For any given number of jobs, there are 10 different instances provided and each instance is designated a number k.

Jobs	10		20		50		100	
Approach	SA	TA	SA	TA	SA	TA	SA	TA
k=1	763	763	2576	2589	14681	14605	60107	60110
k=2	598	598	2555	2555	11955	11877	50359	50369
k=3	672	672	3127	3123	13776	13774	57551	57475
k=4	757	757	2761	2761	11859	11867	60689	60995
k=5	473	473	1936	1936	12408	12376	46003	45991
k=6	669	669	2767	2767	12201	12194	51989	52034
k=7	913	913	3124	3124	14848	14848	53724	53720
k=8	497	497	1492	1492	17599	17604	68079	68030
k=9	510	510	1760	1760	11848	11864	48744	48756
k=10	601	601	1824	1824	11850	11841	50989	50991

Table 6.3. Results obtained for single machine common due-date problem with controllable processing times. For any given number of jobs, there are 10 different instances provided and each instance is designated a number k.

Jobs	200		500		1000	
Approach	SA	TA	SA	TA	SA	TA
k=1	205083	205088	1320625	1321074	5324784	5325158
k=2	224091	224103	1421489	1423335	5092028	5092486
k=3	215488	215492	1366170	1365806	5022548	5023318
k=4	242867	242846	1367905	1367758	5094669	5097182
k=5	214950	214894	1209465	1209331	5244483	5242728
k=6	199051	199163	1185855	1185316	5039738	5041527
k=7	210797	210694	1353048	1353037	5480493	5481022
k=8	189845	189763	1282900	1283187	5067156	5067322
k=9	215633	215524	1393380	1394423	5165348	5164056
k=10	228101	228335	1260302	1260234	5158166	5157819

Table 6.4. Measures of central tendency, standard deviation, runtime and number of fitness function evaluations for Simulated Annealing and Threshold Accepting, obtained over 25 replications of both the algorithms, over all the benchmark instances.

Simulated Annealing								
Jobs	**Min.**	**Max.**	**Mean**	**Median**	**Mode**	**Std.**	**Runtime**	**FFEs**
10	0.000	0.196	0.022	0.000	0.000	0.055	0.064	7111
20	0.000	2.504	0.849	0.716	0.270	0.701	0.492	52243
50	0.019	1.474	0.546	0.503	0.374	0.358	1.767	156736
100	0.020	0.704	0.237	0.209	0.104	0.178	5.563	386254
200	0.014	0.323	0.129	0.117	0.055	0.075	16.567	789042
500	0.002	0.117	0.058	0.060	0.028	0.030	90.015	2204361
1000	0.002	0.080	0.044	0.044	0.002	0.021	353.931	4687141
Threshold Acceptance								
Jobs	**Min.**	**Max.**	**Mean**	**Median**	**Mode**	**Std.**	**Runtime**	**FFEs**
10	0.000	2.667	0.841	0.481	0.079	0.884	0.146	10423
20	0.763	10.582	3.576	3.101	1.094	2.486	0.537	36093
50	0.065	1.549	0.603	0.477	0.241	0.420	2.572	152054
100	0.027	0.724	0.257	0.221	0.171	0.169	7.110	355286
200	0.017	0.280	0.134	0.125	0.107	0.073	18.953	744273
500	0.015	0.131	0.074	0.073	0.039	0.030	97.483	2178340
1000	0.006	0.093	0.048	0.048	0.006	0.021	373.345	4670832

6.7.3 Comparison of Results

In Table 6.2 and 6.3 we present our results for the un-restricted common due-date problem with controllable processing times, where the due-date $d \geq \sum_{i=1}^{n} P_i$, for the benchmark instances. For the first 10 instances with 10 jobs, we reach the optimal solutions for all the instances, as it turns out by comparing our results with that of integer programming. However, for larger instances, integer programming is unable to solve the instances, hence we are not aware if our results are optimal or not.

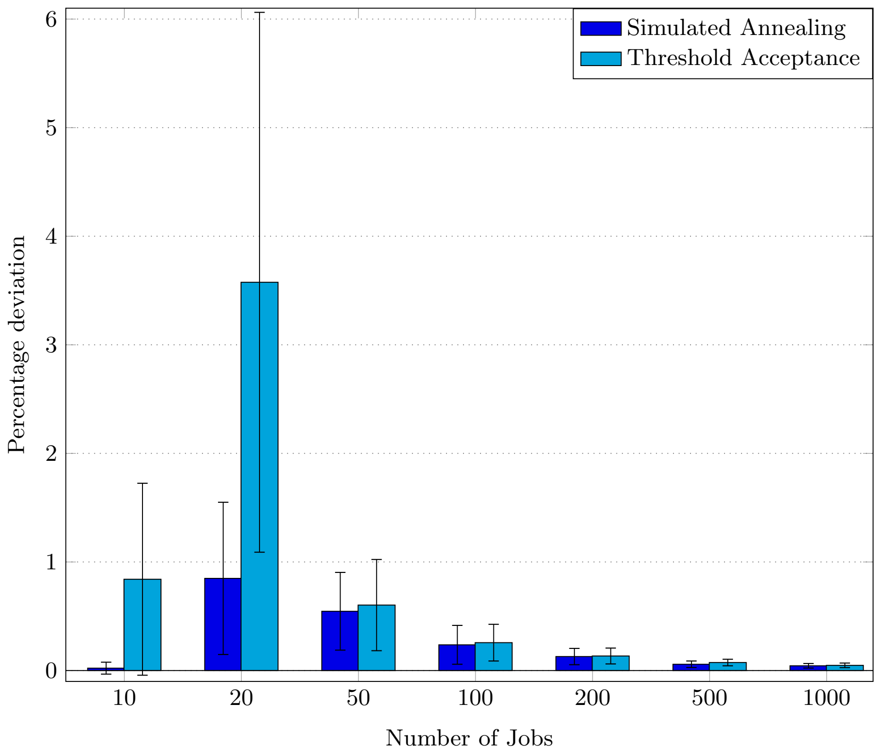


Fig. 6.7. Average percentage deviation and its standard deviation for all job sizes for Simulated Annealing and Threshold Accepting averaged over all the due-date positions for each job size.

The results for Simulated Annealing and Threshold Accepting as quite similar with respect to the solution values for the benchmark instances, for problem instance size of 20 and more. Simulated Annealing obtains better results than Threshold Accepting for 28 instances, while later performs better than SA for 25 instances, as shown in Table 6.2 and 6.3. Both the metaheuristic algorithms obtain equal results for 7 instances, for job sizes of 20 or more. Hence, as far as the quality of the solution is concerned, Threshold Accepting and Simulated Annealing offer almost the same results, with SA performing better for 3 more instances than TA.

For further analysis, we also carry out some statistical tests to compare the two algorithms in detail. Recall we carry out 25 different replications for each of the 250 benchmark instances for both SA and TA. In Table 6.4 we present some measures of central tendency for the metaheuristic algorithms. It can be seen that Simulated Annealing obtains better results in the best and average cases for all job sizes of 20 or more. However, the standard deviation of the two algorithms is better for Threshold Accepting for instances with 100 or more jobs. This leads to concluding that Threshold Accepting gets stagnant and is not able to come out of the local minima, considering the fact that SA obtains better results, although with a slightly higher value for its standard deviation. Interestingly, TA also offers a high standard deviation for small instances of 10, 20 and 50 jobs. The graphical representation of the percentage gap of the solutions obtained by SA and TA is provided in Figure 6.7. We also provide the Mann-Whitney U-test for comparing the results of SA and TA, and in Table 6.5, it can be seen that the p-test value for the two algorithms is quite high at 0.815 in terms of the objective function value. Note that the p-values mentioned in Table 6.5 are the minimum values for which the null-hypothesis of equal medians can be rejected, *i.e.*, H=1. A p-value of 0.815 indicate null hypothesis of equal medians can be rejected only with a significance level of 19%.

Considering the runtime and fitness function evaluations (FFEs), shown in Table 6.4, it can seen that SA requires only slightly higher number of FFEs than TA but SA is considerably faster in evaluations for large instances. The plot of the average FFEs and its standard deviation is presented in Figure 6.8. It is clear from the figure and p-test value for FFEs in Table 6.4, both the algorithms are quite similar. However, looking at the p-test results and the runtime values in Table 6.4 for the runtime, SA clearly performs better with the p-value of 0.01156 for the comparison. This suggests that the null-hypothesis of equal medians can be rejected with a significance level of 98%.

Table 6.5. Results of the Mann-Whitney U-test for solution value, number of fitness function evaluations and the runtime required by Simulated Annealing and Threshold Accepting.

Parameter	Objective Function	Fitness Function Evaluations	Runtime
p-value	0.81530	0.17541	0.01156

6.8 Summary

In this work, we once again implement the two-layered approach. For developing the specialized polynomial algorithm for the resulting linear program, we make extensive theoretical study of the problem and present a novel property for the problem of scheduling against a common due-date with controllable processing times for the un-restricted case. We show that the due-date position in the optimal schedule for the un-restricted case remains the same for both the CDD and for controllable processing time cases. This essential

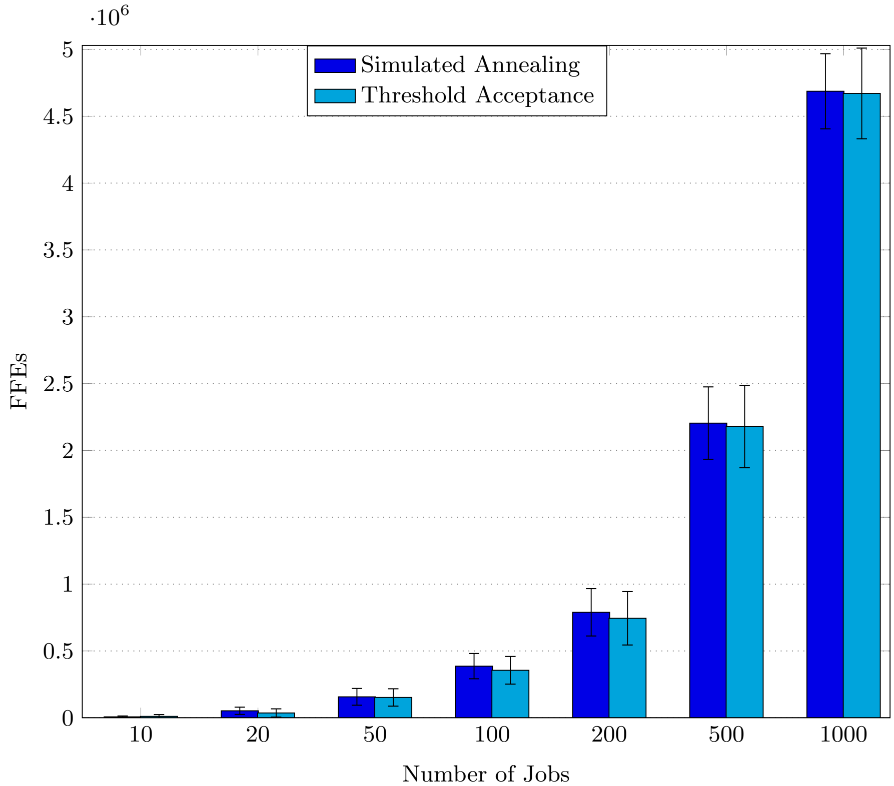


Fig. 6.8. Average number of fitness function evaluations and its standard deviation for Simulated Annealing and Threshold Accepting averaged over all the due-date positions for each job size.

and important property helps us to develop a specialized linear algorithm for the resulting linear program of scheduling jobs of any given sequence. We then present and explain our $O(n)$ algorithm for any given job sequence and prove the runtime complexity and its optimality with respect to the solution value. Due to the unavailability of any set of benchmark instances in the literature, for the problem studied, we offer a new set of benchmark instances. These instances are appended to the CDD benchmarks by Biskup and Feldmann [20]. Henceforth, we carry out experimental analysis of our approach with Simulated Annealing and Threshold Accepting metaheuristic algorithms, and notice that our modified SA performs better that TA.

We have studied a total four NP-hard scheduling problems which are solved using a common underlying approach mentioned in Chapter 2. We develop specialized polynomial algorithms for all these problems and club them with a metaheuristic algorithm. Our results show that this approach is certainly effective and possesses an intrinsic parallel structure. In the next chapter we utilize this inherent parallel structure and present GPGPU parallelized algorithm for the CDD and UCDDCP problems.

7

GPGPU-based Parallel Algorithms for Scheduling Against Due Dates

This work demonstrates an in-depth analysis and successful implementation of parallel programming on combinatorial optimization problems, namely, the Common Due Date (CDD) problem and the Un-restricted CDD with Controllable Processing Times (UCDDCP), examples of NP-hard problems. The CDD and UCDDCP consist of scheduling and sequencing a certain number of jobs with different processing times on a single machine against a common due-date to minimize the total weighted penalty incurred due to earliness or tardiness of the jobs and the penalty due to the compression of the processing times of the jobs. We present a parallel Simulated Annealing (SA) algorithm based on CUDA^{TM} programming and implement our parallel approach on SA and the Discrete Particle Swarm Optimization (DPSO) Algorithm. Optimization for these two problems is carried out in two parts, any given job sequence is first optimized using linear algorithms and the best job sequence is obtained by the parallel SA and DPSO, implemented over Nvidia® graphics processing unit. Our parallel approaches are tested upon the benchmark instances provided in the OR-library. The success of our parallel approach is evident from the quality of our results with respect to the solution value as well as the runtime.

7.1 Introduction

Given the possibility of massive parallelization it makes good sense to try and exploit the GPU computing for real world combinatorial optimization problems. Many of the NP-hard combinatorial optimization problems and the ones which we consider in this work do not possess any intrinsic parallel component. Hence, we combine the two-layered approach to exploit the massive parallelization of GPUs. GPUs have transitioned from graphics-only processing to become a general purpose parallel computing architecture. Today, it is possible to use GPUs on a PC or a compute cluster for high-performance scientific computing applications. One of the goals in High-Performance Computing (HPC) is to achieve the best possible performance from parallel computers.

Recent advances in consumer computer hardware makes parallel computing capabilities widely available to most users. Platforms like OpenCL™ and CUDA™ have made it easier to write the code for GPU programming. OpenCL™ was the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and hand-held/embedded devices. CUDA™ is a parallel computing platform and programming model that enables dramatic increases in computing performance by harnessing the power of the NVIDIA® graphics processing unit (GPU). Graphics Processing Units (GPUs) are very efficient for computer graphics computations, and their highly parallel structure makes them more effective than central processing units (CPUs) for a range of algorithms. GPU computing is to use a CPU and GPU together in an independent co-processing computing model. The sequential parts of the application run on the CPU and the computationally-intensive part can in the ideal case be accelerated by parallelization on GPUs. The basic difference between a CPU and GPU lies in their design architecture as shown in Figure 7.1.

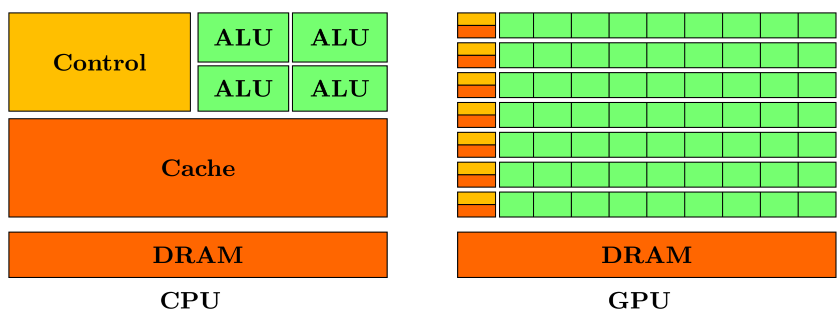


Fig. 7.1. Architecturally, the CPU is composed of a only few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously. *(Source: www.Nvidia.com)*

The design of a CPU is optimized for sequential code performance. It makes use of sophisticated control logic to allow instructions from a single thread of execution to execute in parallel or even out of their sequential order while maintaining the appearance of sequential execution. More importantly, large cache memories are provided to reduce the instruction and data access latencies of large complex applications. Neither control logic nor cache memories contribute to the peak calculation speed. As of 2009, the new general-purpose, multi-core microprocessors typically have several large processor cores designed to deliver strong sequential code performance [83].

Whereas, the GPU hardware takes advantage of a large number of execution threads to find work to do when some of them are waiting for long-latency memory accesses, thus minimizing the control logic required for each execution thread. Small cache memories are provided to help control the bandwidth requirements of these applications so multiple threads that access the same memory data do not need to all go to the DRAM. As a result, much more chip area is dedicated to the floating-point. GPUs are designed as numeric computing engines, and they will not perform well on some tasks (e.g. sequential calculations) on which CPUs are designed to perform well; therefore, one should expect that most applications will use both CPUs and GPUs, executing the sequential parts on the CPU and numerically intensive parts on the GPUs [83].

7.2 Related Work

GPUs have gained popularity as a flexible, accessible, and low cost option for parallel processing. Algorithms with a high degree of arithmetic intensity are well suited to processing on GPUs; a high ratio of arithmetic operations to memory operations indicates that a significant speed-up could be achieved when computed on a GPU architecture. As an example, applications handling operations such as computing several Fast Fourier Transforms in parallel, or mathematical operations, such as large matrix operations, map efficiently to GPUs. Many of the combinatorial optimization problems are NP-hard. These problems occur in several industrial contexts like planning, logistics, manufacturing, finance, telecommunications and many more. Recent years have witnessed an accelerated development and utilization in massive computational capability of the GPUs. In the nineties, GPUs were mainly designed to assist in the display and usability of graphical operating systems [127]. However, GPUs can now be used to perform complex scientific computations, too, which has led to the term GPGPU (*i.e.* General Purpose Computation on Graphic Processing Units). The highly parallel structure of a GPUs makes them more effective for a range of algorithms than CPUs. For combinatorial optimization problems, GPUs have been successfully utilized, yielding several folds of speed-ups and better solutions. Applications that make effective use of the GPUs have reported significant performance gains [101, 103, 141, 27].

Researchers have been trying to develop fast metaheuristic algorithms using GPU computing. In 2010, Luong *et al.* [101] to compute the best approximation of the Weierstrass-Mandelbrot functions [104] using parallel evolutionary algorithms. They showed many-fold speed-ups on GPUs compared to a CPU. The second implementation was the parallel island model of the genetic algorithm on the same problem.

Zhou and Tan developed a parallel Particle Swarm Optimization algorithm for the GPUs and obtained speeds-up of up to 11 times to that of a CPU implementation, on high dimensional functions [154]. A speed-up of 12 times was reported by Tsutsui and Fujimoto, who implemented a GPU parallelized evolutionary algorithm for the Quadratic Assignment Problem [140]. GPUs have also been successfully used to solve combinatorial optimization problems with metaheuristic algorithms. Some of these approaches have evaluated the solutions in parallel on the GPU, while the others outsource some computations to it or perform the full computation on the GPU. Choong *et al.* presented a parallel simulated annealing algorithm for the FPGA placement, which was about 10 times faster than the CPU version [39]. In 2010 Luong *et al.* also investigated the parallelization of large neighborhood local search algorithms and experimented on binary problems and achieved runtime speed-ups of up to 25 times [102]. Czapiński and Barnes proposed a GPU based parallel tabu search algorithm for the Permutation Flowshop Scheduling Problem and computed the solutions 89 times faster than the CPU [46].

In 2012 Melab *et al.* studied the problem using GPU and implemented the Branch and Bound algorithm on the GPU cores, fetching speed-ups of 10 to 50 times [106]. Same year they also presented their speed-ups for the same problem but on multi-GPUs [32]. Bożejko *et al.* studied the flow shop problem on GPUs by parallel evaluation of the fitness function on GPU cores and presented speed-ups on Taillard flow shop problem benchmark instances up to 10000 operations [26]. They also obtained a speed-up of up to 25 on GPU compared to a CPU for the Taillard instances of the job shop scheduling problem using the disjunctive graph [27, 134]. Again in 2012, they implemented a parallel tabu search metaheuristic on multi-GPU cluster [25]. Bożejko *et al.* also provide parallel tabu search algorithms for the job shop scheduling problem [28]. Coelho *et al.* proposed a hybrid CPU-GPU parallel local search algorithm for the unrelated parallel machine scheduling problem to minimize the total makespan [43]. A parallel variant of genetic algorithm was proposed by Pinel *et al.* for minimizing makespan for the batch scheduling of independent tasks on a fixed number of machines [120]. In 2013, Luong *et al.* implemented parallel local search metaheuristic algorithms using GPU computing on the Taillard instances of quadratic assignment problem and showed a speed up of up to 25 times compared to a CPU [103]. Bukata *et al.* developed parallel tabu search algorithms for the resource constrained project scheduling problem [31, 30].

Chakroun *et al.* proposed a branch and bound algorithm to solve the NP-hard combinatorial optimization problems on GPUs [33]. They performed experiments on the flow shop scheduling problem and the speed-ups which are achieved up to 160 compared to the corresponding CPU implementations. Luong *et al.* introduced a parallel local search algorithm using the GPU, in 2013 [103]. They performed computational studies on the Travelling Salesman Problem, the Quadratic Assignment Problem and the Permuted Perceptrons Problem while showing significant speed-ups in comparison to the serial CPU implementations. Somani and Singh present a parallel genetic algorithm for the job shop scheduling using topological sort [131]. Dali *et al.* proposed a parallel particle swarm optimization algorithm on GPUs for the maximal constrained satisfaction problem [47]. Zelanzy and Penpera presented a parallel tabu search algorithm for the multi-objective permutation flow shop problem [152].

In this research work, we investigate GPU parallelization using CUDA on two NP-hard scheduling problems, the Common Due Date problem (CDD) and the Un-restricted Common Due Date problem with Controllable Processing Times (UCDDCP). For both the problems, any given sequence is optimized to its minimum weighted penalties by polynomial algorithms and the best job sequence is obtained with parallel versions of Simulated Annealing (SA) and Discrete Particle Swarm Optimization (DPSO). The realization and development of these parallel metaheuristics on a GPU using CUDA is been explained in detail. Later on, we present the results obtained via these parallel approaches for the benchmark problem instances and compare the quality of the solutions and runtimes between CPU and GPU implementations.

7.3 CUDA Memory Model

Any computation on the GPU is carried out by threads of the blocks. Each block consists of a number of threads (1024 in the device used for this work), while the blocks are placed inside the grids of the device. CUDA accesses and processes the data on threads of the blocks with different levels of hierarchy and accessibility, depending on the type of memory. Each thread has a private local memory and registers associated with it. On the next level, each block of any grid contains a shared memory which is accessible by all the threads of the block. The global memory is the main memory of the GPU device with read and write permissions to all the threads of the device [114]. The lifetime of the shared memory lasts till the lifetime of the block, while the lifetime of the global memory spans from data allocation to de-allocation from host to device. The transfer of the data between the host and device requires a huge overhead, hence the best strategy for GPU computations, is to store all the data required in the GPU device till the end of the computation [55]. CUDA enabled GPGPUs contain *on-chip* and *on-board* memory. Farber describes on-chip streaming multiprocessor (SM) memory as the fastest and most scalable memory on the device measured in KB (Kilobyte), while the on-board global memory which can be accessed all the SMs is measured in GB (Gigabytes). The global on-board memory is the largest and most commonly used, however, it is the slowest memory of the GPU [55].

Figure 7.2 illustrates several types of device memories that can be used with CUDA. The constant, texture and global memory can be written on and read from the host. The threads in any block can read and write to the global memory, while the constant and texture memories offer read only accessibility. The shared memory and registers are readable and writable, only by the threads of the block. Each thread in a block consists of its own registers, which are not accessible by other threads of the same block. Table 7.1 shows the bandwidth and the latency values for different memory types [44]. The usage of registers provides the best bandwidth and lowest latency. The shared memory has a lower bandwidth and memory latency than registers. Texture, constant and global memory all have the same values, but are much lower than those of the shared memory and registers.

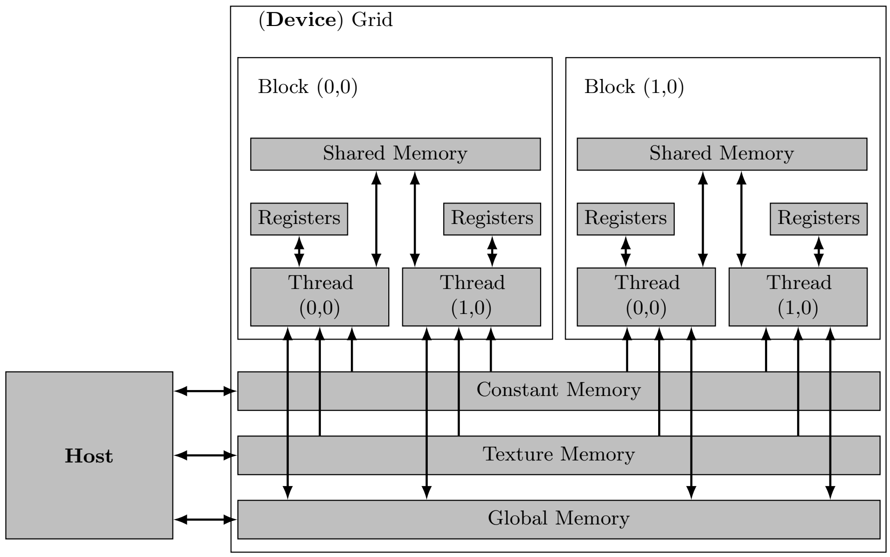


Fig. 7.2. CUDA memory model, representing the memory hierarchy of CUDA capable compute devices [83].

Table 7.1. Bandwidth and latency of the hierarchical CUDA memory types. Bandwidth and Latency decide the time it takes to transfer a given set of data. [44]

Storage Type	Registers	Shared Memory	Texture Memory	Constant Memory	Global Memory
Bandwidth	8TB/s	1.5TB/s	30GB/s	30GB/s	30GB/s
Latency	1 cycle	1 to 32 cycles	400 to 600 cycles	400 to 600 cycles	400 to 600 cycles

7.4 Parallel Approach

The idea behind our approach is to break the NP-hard problem in two parts, one part deals with finding the completion times of the jobs for any given job sequence using the algorithms presented in Chapter 4 and 6. And the second part, utilizes the GPGPU parallelized metaheuristic algorithms to efficiently search for the optimal/best job sequences. There are several strategies proposed by Ferreiro *et al.* to parallelize SA [58]. One strategy they proposed is an application dependent parallelization, where the operations of the objective function itself could be broken down and computed in parallel. This approach is not applicable here since the operands of our objective functions occur sequentially, *i.e.*, each operand needs to wait for its preceding operand to complete. Another technique for parallelization could be a domain decomposition strategy, where the search space is basically segregated in several sub-domains, with different processors carrying out the search of the best solutions in their respective sub-domain, while sharing their subsequent results with the other processors [58]. The drawback of this strategy is the enormous size of the search space itself, and it becomes ineffective for a job size of 50 or more. The last and the best SA parallelization strategy they propose is the utilization of multiple Markov chains. In this strategy, several Markov chains are executed asynchronously. After a certain period or at the end of the process, the processors communicate their results to each other. Depending on the number of communications, this strategy is classified by Ferreiro *et al.* [58] into *Asynchronous* and *Synchronous* simulated annealing.

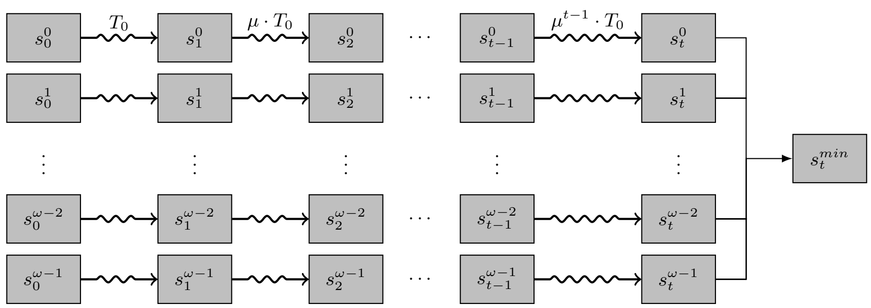


Fig. 7.3. Schematic representation of the asynchronous approach of parallel simulated annealing algorithm as suggested by Ferreiro *et al.* [58].

7.4.1 Asynchronous Simulated Annealing

The *asynchronous* SA basically performs several independent annealing processes in parallel on the available processors *i.e.*, each processor (CUDA thread) performs a separate SA asynchronously. When all these annealing processes are completed, the best result is selected among all the threads using a reduction operation. The configuration of SA for initialization on each thread can be same or different for all the Markov chains [58]. Figure 7.3 shows represents the schematic diagram of the asynchronous SA, where ω

processors (or threads) carry out independent SA simultaneously. The initial temperature T_0 and the cooling rate μ are kept the same for all the threads. Each chain carries out t iterations and at the end a reduction operation is used to select the best solution s_t^{min}.

7.4.2 Synchronous Simulated Annealing

The *synchronous* parallel version of SA starts in the similar manner as the asynchronous SA. However, after each temperature state, all the threads compute a Markov chain of some constant length and at the end report their current solution s_j^i, where $i = 0, 1 \ldots, \omega - 1$, and $j = 1, 2, \ldots, t$. A reduction operation is then carried out to obtain the best solution s_j^{min}, among all the threads. In the next step (or the next temperature state), each thread performs the same calculations all over again, with the exception that each thread starts its Markov chain with the best solution obtained in the previous temperature state, *i.e.* s_j^{min}. Evidently, Ferreiro *et al.* [58] claim that the exchange of the states and results can be very intensive in terms of the runtime. Figure 7.4 shows the synchronous approach with ω processors for t iterations of SA at each temperature level.

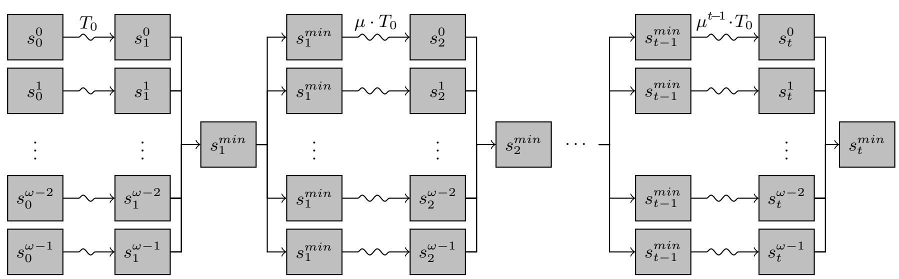


Fig. 7.4. Schematic representation of the synchronous approach of parallel simulated annealing algorithm as suggested by Ferreiro *et al.* [58].

7.5 GPU Based Simulated Annealing

We now explain our parallel implementation of the Asynchronous Simulated Annealing algorithm [58]. The reason for choosing the asynchronous version over the synchronous SA is due to the premature convergence of the latter approach, examined from our experimental analysis. SA implemented on each CUDA thread involves the standard metropolis acceptance criterion and the exponential cooling schedule, as shown in Algorithm 10. The initial temperature T_0 is taken as the standard deviation of fitness values of 5000 different job sequences, generated randomly. This value for the initial temperature has been suggested by [125]. The exponential cooling rate is adopted in this work and the neighborhood of any individual (*i.e.* job sequence) is generated by a perturbation mechanism, where a certain number of consecutively processed jobs are selected at random from the current sequence and shuffled using the *Fisher Yates* algorithm, provided in [45].

Algorithm 10: The core Simulated Annealing algorithm running in each CUDA thread.

$s \leftarrow s_0$
$T \leftarrow T_0$
$E \leftarrow Fitness(s)$
while $(i \leq \#Iterations)$ **do**
    $s_{new} \leftarrow Neighbour(s)$
    $E_{new} \leftarrow Fitness(s_{new})$
    if $\exp((E - E_{new})/T) \geq rand\,(0,1)$ **then**
        $s \leftarrow s_{new}$
        $E \leftarrow E_{new}$
    $T \leftarrow T \cdot \mu$
    $i \leftarrow i + 1$
Return s

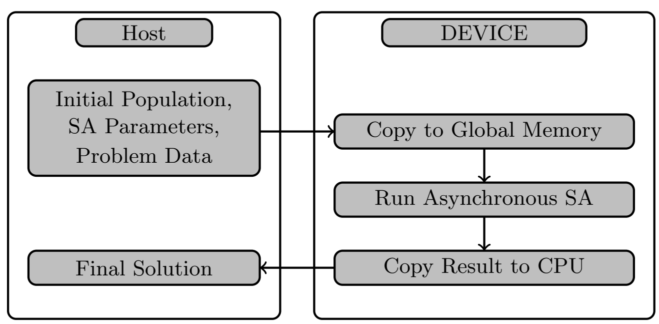


Fig. 7.5. Schematic representation of data transfer between the host and device. The data is transferred two times, back-and-forth, while the SA iterations are performed by the device.

The parallelization of the SA is initiated by allocating the number of threads and blocks on the GPU. CUDA offers three dimensional grids and blocks in (x, y, z) directions. The grid configuration G can be written as (g_x, g_y, g_z) and the block configuration B as (b_x, b_y, b_z). The grid configuration G implies that there are g_x, g_y and g_z number of blocks in x, y and z

directions, respectively. Likewise, B configuration for the blocks implies b_x, b_y, b_z threads in the three dimensions. Let N be the total ensemble size and N_B be the block size, then a grid size of $\lceil N/N_B \rceil$ is allocated in the device for the parallel runs of the algorithms. In our work, we consider linear configurations for both the grid and the blocks, with $G = (\lceil N/N_B \rceil, 1, 1)$ and $B = (N_B, 1, 1)$. Henceforth, the initial job sequences are copied to the GPU global memory, along with the earliness, tardiness penalties and the processing times of the jobs. The due-date d and the number of jobs n are transferred to the constant memory of the device to benefit from its broadcast mechanism. For the UCDDCP, the minimum processing times and the compression penalties are also copied to the GPU. Figure 7.5 shows the data transfer mechanism from the host to device and vice-versa. We then launch four different kernels, one after the other to calculate the *i*) *cuRand* initial states, *ii*) fitness function, *iii*) perturbation, and *iv*) SA acceptance. Figure 7.6 shows these kernels in CUDA standard double *bra-ket* notation. The CUDA threads are depicted as T1, T2, *etc.*, implying that each thread is running the same algorithm in parallel.

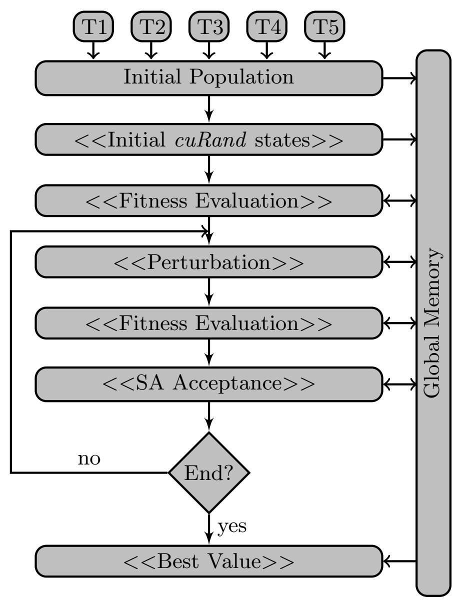


Fig. 7.6. Flow chart of the parallel Asynchronous Simulated Annealing.

7.5.1 Initialization of *cuRand* states

The first kernel (Initial *cuRand* states) that is launched is to initialize the *cuRand* states to generate the random numbers, required for perturbation and

SA acceptance. This kernel is launched only once, before the computations for SA begin. The random numbers are then created using the *cuRand* library on each thread [114].

7.5.2 Fitness Function Kernel

The fitness kernel first copies the processing times, earliness and tardiness cost per unit of time, inside the shared memory of a block, because this memory has shorter latency than global memory, as depicted in Table 7.1. For the UCDDCP problem, we also copy the minimum processing times and the compression penalty of the jobs to the shared memory. After transferring these constant parameters to the shared memory of each block, the kernel synchronizes the current block. This must be done because the warps within the block can be at different positions of the program depending on their scheduling. This synchronization ensures that all the write operations on the shared memory are finished before reading them. Otherwise, a thread would have the chance to read from an address where no thread has written. Next, the fitness value for the job sequence in each thread uses the linear algorithms illustrated in Chapter 4 and 6 for CDD and UCDDCP, respectively.

7.5.3 Perturbation Kernel

The neighborhood of any job sequence is calculated by applying the *Fisher Yates* algorithm on part of the parent job sequence. For CDD problem, a sub-sequence of size $Pert = 4$ is selected from the parent job sequence and then the Fisher Yates algorithm is implemented on this sub-sequence while retaining the position of other jobs in the sequence. Likewise, same methodology is adapted for UCDDCP with $Pert = c + \lfloor\sqrt{N/10}\rfloor$, where c is a constant less than 5 and N is the number of jobs. The random numbers required for the acceptance criterion are generated using the *cuRand* library of CUDA. Since *cuRand* provides only integer values, a normalization is carried out to obtain a floating point value in $[0, 1]$. After creating a new permutation for each thread, the fitness kernel is launched again to evaluate the solution for each newly created job sequence.

7.5.4 Acceptance Kernel

Henceforth, the acceptance kernel is launched, which simply checks if the given solution should be accepted or not, depending on the standard metropolis acceptance criterion of the SA algorithm. Additionally, if any thread contains the best solution obtained so far, it is not accepted. The random numbers in this kernel are again created using the *cuRand library*. In the same kernel, we also carry out the reduction operation to find the best solution obtained thus far.

After invoking all these kernels, there should be a synchronization of the device, because all kernel calls are asynchronous and inside a queue. Hence, the synchronization operation is performed by the CPU. Through this operation, the CPU waits until the GPU finishes processing. At the end of the number of iterations, the global best solution is copied back to the host. We now present a short explanation and the implementation of the core discrete particle swarm optimization algorithm. The parallel implementation of the DPSO algorithm on the GPU is carried out in the asynchronous manner, as explained for the SA. In the forthcoming section, we then present and compare our results for our GPGPU utilized parallelization of both SA and DPSO with the CPU implementations.

7.6 Discrete Particle Swarm Optimization (DPSO)

In this section, we explain the core part of the DPSO algorithm. Since, the traditional Particle Swarm Optimization [23] was developed to work on real valued positions, for the problems studied we required the discrete version of PSO. Pan *et al.* have previously used DPSO on the no-wait flow shop problem [116]. DPSO contains an adjusted method to update the particles position, based on discrete job permutations. The updated method includes particles' position ($p_i(t)$), its best position ($p_i^b(t)$) and the swarms best position ($g(t)$) [116]. The new position $p_i(t+1)$ of the particle is given by

$$p_i(t+1) = c_2 \oplus F_3 \left(c_1 \oplus F_2 \left(F_1 \left(p_i(t)\right), p_i^b(t)\right), g(t)\right) \ . \tag{7.1}$$

In the above equation, operator $\oplus$ in any clause $x' = c \oplus f(x)$ means, operate function f on x with a probability of c, *i.e.* $x' = f(x)$, if $rand(0,1) < c$ and $x' = x$, if $rand(0,1) > c$. The first component of the update mechanism in Equation (7.1) is the particles velocity given by $\lambda_i(t+1) = F_1(p_i(t))$, where F_1 represents a swap operator which selects two different jobs in the sequence ($p_i(t)$) randomly and swaps their position in the job sequence.

The second component is given by $\vartheta_i(t+1) = c_1 \oplus F_2(\lambda_i(t), p_i^b(t))$ and represents the particles cognition part, where F_2 is a one-point crossover operator with a probability of c_1, given by Equation (7.2).

$$\vartheta_i(t+1) = \begin{cases} F_2(\lambda_i(t), p_i^b(t)), & rand() \leq c_1 \\ \lambda_i(t), & rand() > c_1 \ . \end{cases} \tag{7.2}$$

The one-point crossover is implemented by generating an integer uniform random number within the job size and then the first part of the two job sequences are swapped with each other, preserving the precedence constraint for both the sequences. The one-point crossover carried out in this work runs in linear runtime complexity, with respect to the number of jobs.

The last component is the particles social part, representing the orientation on the swarm behaviour. This third component results in the new position of a particle and can be defined as $X_i(t+1) = c_2 \oplus F_3(\vartheta_i(t), g(t))$, according to Equation (7.3), where we retain F_3 as a one point crossover similar to F_2. F_3 is carried out every 5 iterations of the DPSO.

$$s_i(t+1) = \begin{cases} F_3(\vartheta_i(t), g(t)), & rand() \leq c_2 \\ \vartheta_i(t), & rand() > c_2 \ . \end{cases} \tag{7.3}$$

Apart from the core aspect of the algorithm, the parallelization approach remains the same as for SA. Algorithm 11 provides the pseudo code for the DPSO implemented in this work, based on Pan *et al.* [116].

Algorithm 11: The core Discrete Particle Swarm Optimization Algorithm implemented.

```
Initialize Population
Evaluate fitness-function
while (i ≤ #Iterations) do
    find particles' best
    find swarms best
    Update particles' position
    Evaluate fitness-function
    i ← i + 1
Return Best Particle
```

7.7 Results

In this section, we present our results for the SA and DPSO parallelization on the GPU for the CDD and UCDDCP problems. We compare the two algorithms for both the problems and present the results for the benchmark instances. The runtime of the presented GPU based metaheuristics is influenced by the number of generations and the number of GPU threads which perform the optimization. Figure 7.7 shows the influence of both parameters for the CDD.

It is evident that increasing the number of generations or threads is increases the runtime. However, to achieve a good quality solution in a considerably short time, one needs to keep a balance between these two parameters and avoid run unnecessary iterations as well not invoking several serial processing of the blocks. Increasing the number of threads also increases the runtime of the algorithm, since a SM is limited in the number of blocks and in the number of threads it can perform at the same time. This implies that loading several

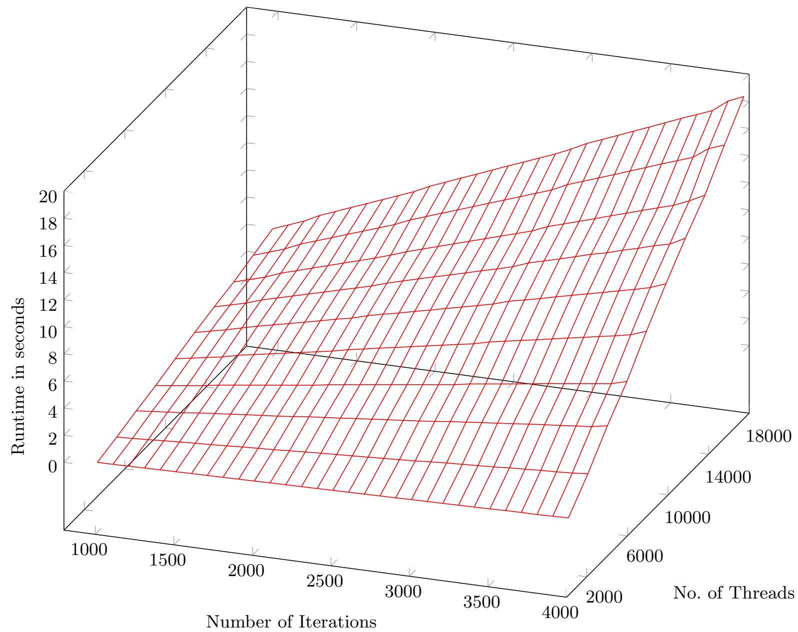


Fig. 7.7. Runtime in seconds for the parallel fitness function evaluations of the CDD problem, with respect to the number of threads (population size) and the number of generations.

threads within a block results in serial processing of the blocks through the SM. On the other hand, increasing the block size offers less registers which a thread can use.

The theoretical limit for the number threads in one block of the Kepler device we use is 1024. However, after several experimental evaluations we observed that the best results for both the problems were achieved with a block size of 192. Selecting the number of grids and the number of iterations, is a rather complex task and usually problem dependent, due to the above mentioned trade-off between the number of iterations and the number of threads. Having a high value for these parameters results in less performance but on the other hand it also fetches us better results. Hence, after testing our approach on several experimental values, we chose to present our results for two best configurations, which resulted in best speed-ups compared to the results provided by [86] and [6]. In both the cases the grid size was kept at a constant value of four. This was not a high value considering the GPU device we used, but the results obtained were of excellent quality with a high speed-up in comparison to the CPU runtimes. Hence, the total number of threads, which is also the population size was equal to 768. The number of generations for the first case was kept at 1000 and in the other case as 5000. The cooling factor for the Simulated Annealing was kept at 0.99 with an exponential cooling rate. The implementation of the parallel algorithm was carried out on a Nvidia GeForce GT 740M device, with 2 GB graphics card memory on a host CPU of 8 GB RAM with Intel i5 1.6 GHz processor. We replicate all the 280 instances of CDD and 70 instances of UCDDCP, 25 times each for SA and DPSO. Before presenting the results, we first explain some parameters used in the analysis of our results. Let,

SA_{1000} = SA algorithm with 1000 iterations,
SA_{5000} = SA algorithm with 5000 iterations,
$DPSO_{1000}$ = DPSO algorithm with 1000 iterations,
$DPSO_{5000}$ = DPSO algorithm with 5000 iterations,
Z = Solution value obtained with our parallel approach,
Z_{best} = Solution obtained with the CPU versions of [86], Chapter 4 for CDD and the best results for UCDDCP in Chapter 6,
$\%\Delta$ = Percentage deviation between Z and Z_{best}, where $\%\Delta = \frac{(Z - Z_{best})}{Z_{best}} \cdot 100$.

Table 7.2. Average percentage deviation for the best results of our approaches for each problem size for CDD, relative to the CPU implementation of LHSA mentioned in Chapter 4 and the results of Lässig *et al.* [86].

Jobs	SA_{1000}		SA_{5000}		$DPSO_{1000}$		$DPSO_{5000}$	
	[86]	LHSA	[86]	LHSA	[86]	LHSA	[86]	LHSA
10	0.000	0.000	0.000	0.000	0.104	0.104	0.000	0.000
20	-0.089	0.000	-0.089	0.000	1.298	1.393	1.231	1.327
50	-0.215	0.591	-0.798	0.000	1.898	2.739	1.794	2.634
100	-0.684	1.139	-1.633	0.151	0.738	2.621	0.710	2.592
200	-0.462	1.466	-1.352	0.539	0.463	2.433	0.449	2.418
500	-0.424	1.889	-1.147	1.127	0.260	2.611	0.247	2.598
1000	-0.157	2.446	-0.929	1.631	0.425	3.060	0.415	3.049

7.7.1 Results for the CDD

We now present our results for the Common Due Date problem obtained with our parallel approaches. Table 7.2 presents the average percentage deviation ($\%\Delta$) of the best results obtained over 25 replications for the CDD problem relative to the sequential implementation in Lässig *et al.* [86] and the results presented in Chapter 4. It should be mentioned here that the results in [86] are obtained by incorporating only Algorithm 4 with the Simulated Annealing algorithm, however the results in Chapter 4 incorporates both the Algorithm 4 and 6 in conjunction with Simulated Annealing. Moreover, the parallel implementation of SA and DPSO for CDD is carried out utilizing Algorithm 4 only without the improvement heuristic mentioned in Algorithm 6. This is important to notice as we show that not only our parallel implementation obtains better results from its exact sequential implementation as in [86], but it

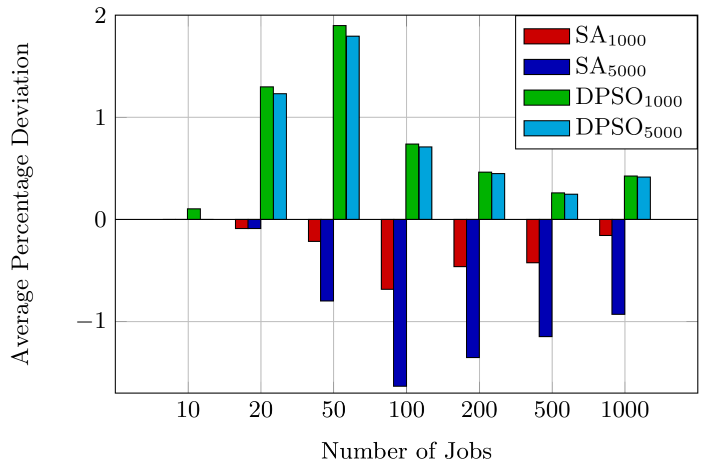


Fig. 7.8. Comparative average percentage deviation of our four parallel algorithms relative to the solutions of [86] for the CDD problem.

Table 7.3. Obtained speed-ups of the parallel algorithms for the CDD problem relative to [86] and CPU implementation of LHSA mentioned in Chapter 4.

Jobs	$\mathbf{SA_{1000}}$		$\mathbf{SA_{5000}}$		$\mathbf{DPSO_{1000}}$		$\mathbf{DPSO_{5000}}$	
	[86]	LHSA	[86]	LHSA	[86]	LHSA	[86]	LHSA
10	8.043	0.000	1.754	0.000	5.968	0.000	1.307	0.000
20	10.253	0.013	2.213	0.003	8.351	0.010	1.653	0.002
50	17.578	0.022	3.836	0.005	14.682	0.019	2.878	0.004
100	41.236	0.135	8.845	0.029	35.312	0.115	7.049	0.023
200	46.485	0.761	9.665	0.158	38.979	0.638	7.673	0.126
500	85.578	4.882	17.804	1.016	60.907	3.474	12.117	0.691
1000	95.460	15.493	19.711	3.199	61.495	9.981	12.351	2.005

furnishes good speed-ups compared to the CPU implementation of Chapter 4 where we incorporate an improvement heuristic along with a linear algorithm. The reason for not implementing the improvement heuristic for the GPU is to show that the even without the improvement heuristic (Algorithm 6) which offers better results for the CPU; the parallel implementation of the linear algorithm as well achieves good results within a percentage gap of only 1.6% and a maximum speed-up of 15 times. Additionally, we compare our parallel implementation with the exact CPU implementation and show that not only does the GPU offer high speed up of 95 times, but it also improves solution values for all benchmark instances higher than a job size of 10. The percentage deviation shown in the table is the average over 40 different instances for each job size. The graphical representation of these percentage deviations with the exact CPU implementation is shown in Figure 7.8 as a bar chart. As

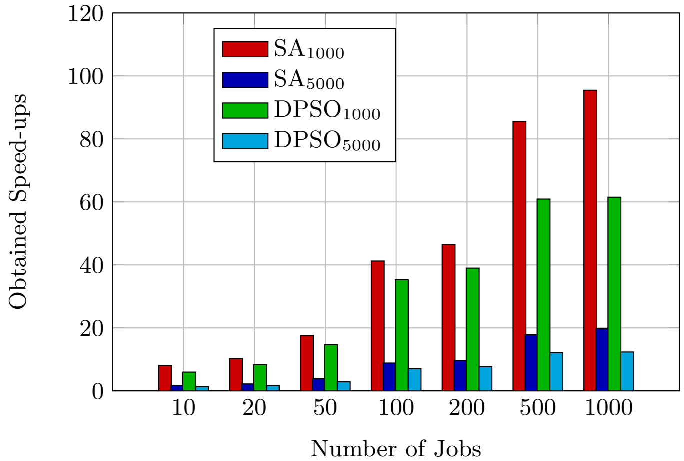


Fig. 7.9. Graphical representation of the obtained speed-ups of the parallel algorithms for the CDD problem relative to [86].

can be seen from the table and the bar chart, SA performs extremely well and obtains better results for all the benchmark instances except for job size 10, where we reach the optimal solutions. Comparison with the results of Chapter 4 shows that the parallel SA offers results which are within 1.63% of the CPU implementation of LHSA. However, the DPSO algorithm is not able to improve the CPU results of both LHSA and [86]. On average DPSO with 1000 iterations offers a percentage deviation of 0.74 with [86] and 2.137 with LHSA, additionally for 5000 iterations of DPSO the results do not improve as much as SA with 5000 iterations. Evidently, as the problem size increase the DPSO algorithm does not converge to better solutions than SA. Among all the four approaches, SA_{5000} performs the best and fetches us a superior solution quality with average percentage deviation within 1.63 percent of the efficient CPU implementation of CDD in Chapter 4, for any instance. Moreover, comparing the runtimes of the CPU implementations by Lässig *et al.* [86] and LHSA, we observe that the speed-ups obtained by our parallel algorithms certainly prove their worth, as shown in Table 7.3 and Figure 7.9. The speed-ups of all the four approaches increase several folds as the problem size increases, with SA_{1000} offering a 95 times faster runtime in comparison to [86] for 1000 jobs, as well as improving the benchmark results. $DPSO_{1000}$ is again not as fast and it is 1.5 times slower than Simulated Annealing. The comparison of the speed-ups with the results of LHSA in Chapter 4 are not as high as LHSA also incorporates a improvement heuristic. Nonetheless, SA_{1000} is 15 times faster and achieves solutions values which are within 2.5% of the LHSA. As it can be seen in Table 7.3 the speed-ups of DPSO are not as high as that of SA, moreover the results of DPSO with both 1000 and 5000 iterations are worse

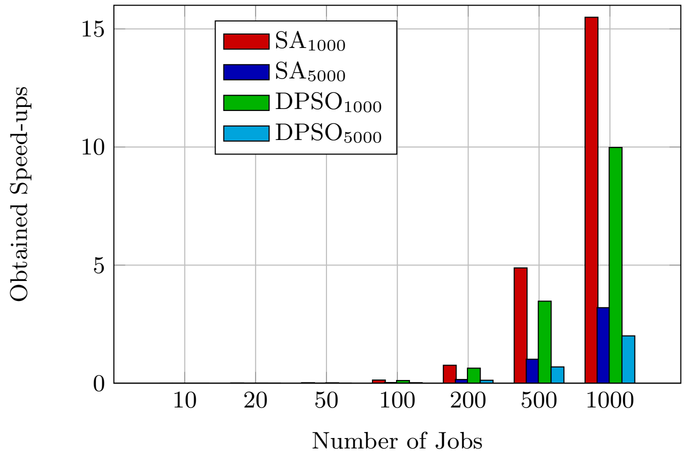


Fig. 7.10. Graphical representation of the obtained speed-ups of the parallel algorithms for the CDD problem relative to CPU implementation of LHSA mentioned in Chapter 4.

than the parallel SA algorithm. The graphical representation of the speed-ups of our four parallel algorithms compared to the CPU implementations are shown in Figure 7.9 and 7.10. Considering the level of percentage deviation, the speed-ups obtained and the fact that we same CUDA thread counts and the number of iterations for both SA and DPSO, we reckon that DPSO does not perform as efficient as the Simulated Annealing.

Additionally, we also present the measures of central tendency four our four parallel approaches, with minimum, maximum, mean, median, mode and standard deviation for all job sizes in average. It is clear that the standard deviations of all the approaches are very low, suggesting that the algorithms are robust and consistent in achieving the results mentioned above. We do not present the fitness function evaluations, since the number of iterations is fixed and calculating the iteration number at which the solution does not improve would requires unnecessary memory transfer between the host and device. Hence the total number of FFEs for all the approaches are fixed at number of iterations times the population size of 768, which corresponds to the total number of threads utilized for our GPU implementation.

7.7.2 Results for the UCDDCP

We now present our results for the Un-restricted Common Due Date Problem with Controllable Processing Times (UCDDCP) using the parallel SA and DPSO with 1000 and 5000 generations, each. The exact percentage deviations for all the jobs on average for the best results obtained can be found in Table 7.5. The negative values mean that the results obtained by these parallel

Table 7.4. Measures of central tendency with respect to the percentage error of the best solution obtained by GPU implementation of Simulated Annealing and Discrete Particle Swarm Optimization for 1000 and 5000 iterations, for the CDD problem.

Simulated Annealing 1000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.000	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000	0.000
50	0.591	0.961	0.760	0.108	0.760	0.697
100	1.139	1.818	1.488	0.207	1.487	1.371
200	1.466	2.029	1.759	0.152	1.772	1.629
500	1.889	2.201	2.066	0.077	2.076	1.894
1000	2.446	2.641	2.562	0.049	2.569	2.452
Simulated Annealing 5000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.000	0.000	0.000	0.000	0.000	0.000
20	0.000	0.000	0.000	0.000	0.000	0.000
50	0.000	0.000	0.000	0.000	0.000	0.000
100	0.151	0.268	0.213	0.031	0.213	0.211
200	0.539	0.723	0.643	0.048	0.646	0.557
500	1.127	1.308	1.232	0.045	1.238	1.135
1000	1.631	1.757	1.702	0.032	1.704	1.632
Discrete Particle Swarm Optimization 1000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.104	0.480	0.274	0.156	0.344	0.333
20	1.393	2.206	1.813	0.341	1.847	1.817
50	2.739	3.477	3.146	0.241	3.199	3.047
100	2.621	3.186	2.929	0.163	2.950	2.786
200	2.433	2.837	2.682	0.105	2.690	2.622
500	2.611	2.789	2.727	0.046	2.734	2.653
1000	3.060	3.172	3.129	0.030	3.136	3.075
Discrete Particle Swarm Optimization 5000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.000	0.178	0.053	0.073	0.020	0.010
20	1.327	2.466	1.807	0.363	1.664	1.606
50	2.634	3.556	3.145	0.261	3.185	3.094
100	2.592	3.205	2.923	0.178	2.945	2.782
200	2.418	2.838	2.674	0.115	2.690	2.595
500	2.598	2.793	2.724	0.049	2.732	2.639
1000	3.049	3.169	3.127	0.031	3.133	3.078

algorithms are better than the CPU results provided in Chapter 6. Figure 7.11 shows the graphical representation of these relative percentage deviation of the GPU results in comparison to the CPU based algorithm. Note that the results presented here are improved over our results in [8], for both SA and DPSO, however the superiority of SA over DPSO is nonetheless evident.

Table 7.5. Average percentage deviation for the best results of our approaches for each problem size for UCDDCP, relative to CPU implementation of Simulated Annealing mentioned in Chapter 6.

Jobs	SA_{1000}	SA_{5000}	$DPSO_{1000}$	$DPSO_{5000}$
10	0.000	0.000	0.000	0.00
20	-0.038	-0.038	-0.038	-0.038
50	0.084	-0.129	0.043	-0.122
100	0.094	-0.090	0.315	0.067
200	0.157	0.074	0.516	0.127
500	0.501	0.087	0.933	0.389
1000	0.799	0.185	1.048	0.717

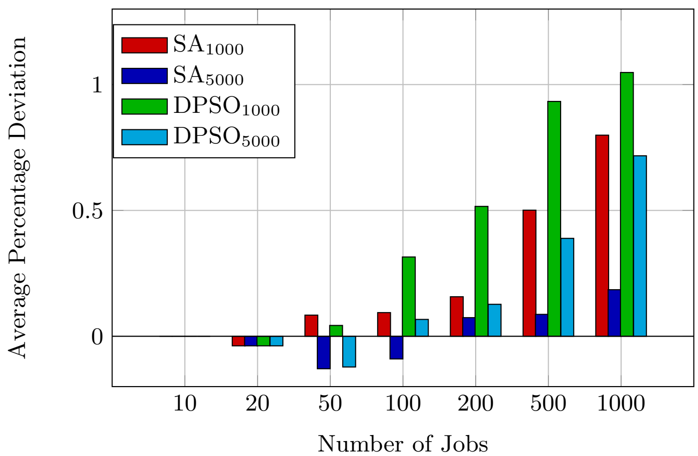


Fig. 7.11. Comparative average percentage deviation of our four parallel algorithms relative to the CPU implementation of UCDDCP mentioned in Chapter 6.

As in the case for CDD, we again observe that DPSO computes worse results from job size 100 and above, compared to SA, for 1000 iterations. Table 7.5 shows that both versions of DPSO obtains equal results to SA for input sizes of 10 and 20 jobs, while the results for 50 jobs are better for DPSO with 1000 iterations and slightly worse for 5000 iterations. For higher job sizes, SA consistently achieves better results than DPSO for both 1000 and 5000 iterations. For 1000 jobs the deviation for $DPSO_{1000}$ is 1.05 percent, while that of SA_{1000} is just 0.8 percent. The $DPSO_{5000}$ obviously performs better than $DPSO_{1000}$ as far as the solution quality is concerned, however the improve obtained by $DPSO_{5000}$ is still comparable to SA_{1000}. Simulated Annealing on the other hand, again achieves better results for all instances of 100 jobs and above. Although the solution quality of DPSO is only slightly worse than the SA, the comparison of the speed-up obtained by the two approaches is highly contrasting. Table 7.6 presents the speed-ups obtained by our four parallel ap-

Table 7.6. Obtained speed-ups of the parallel algorithms for the UCDDCP problem relative to CPU implementation mentioned in Chapter 6

Jobs	SA_{1000}	SA_{5000}	$DPSO_{1000}$	$DPSO_{5000}$
10	1.032	0.217	0.688	0.142
20	4.717	1.007	3.154	0.595
50	6.288	1.294	4.043	0.766
100	10.285	2.104	7.710	1.266
200	15.927	3.310	11.863	2.373
500	34.907	7.083	23.419	4.697
1000	65.909	13.892	44.051	8.881

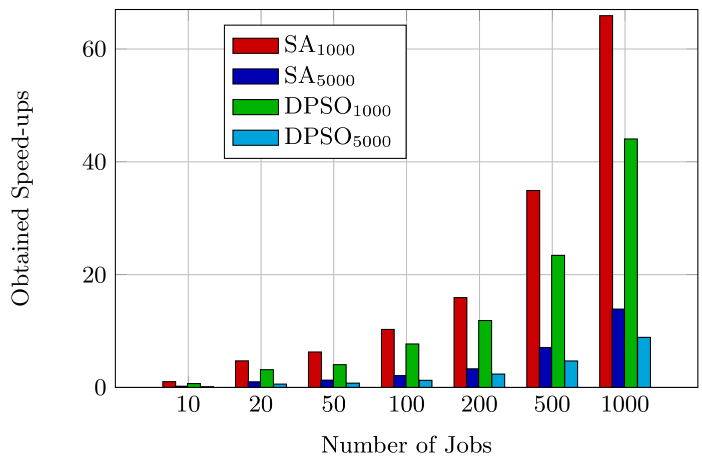


Fig. 7.12. Graphical representation of the obtained speed-ups of the parallel algorithms for the UCDDCP problem relative to CPU implementation mentioned in Chapter 6.

proaches with the CPU runtime of Simulated Annealing presented Chapter 6. As can be seen, the parallel asynchronous SA version with 1000 iteration offers the highest amount of speed-up among all the benchmark instances, with the highest speed-up reaching a value of 65 for jobs size of 1000. The corresponding DPSO implementation with 1000 iterations achieves a speed-up of 44 times, while the solution quality is not as good as SA_{1000}. Comparing the speed-up of SA and DPSO for 5000 iterations, SA again outperforms DPSO by being around 1.5 times faster. We also present these speed-ups in a graphical representation in Figure 7.12, where the superiority of SA over DPSO is evident. Additionally, as in the case of CDD we present the measures of central tendency for our parallel approaches in terms of minimum, mean, maximum, median mode and the standard deviation against the percentage deviation of the results, compared to CPU implementation. Clearly, the superiority of SA is again proved as it not only performs in the best case best also in the worst

Table 7.7. Measures of central tendency with respect to the percentage error of the best solution obtained by GPU implementation of Simulated Annealing and Discrete Particle Swarm Optimization for 1000 and 5000 iterations, for the UCDDCP problem.

Simulated Annealing 1000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.000	0.000	0.000	0.000	0.000	0.000
20	-0.038	0.035	0.011	0.034	0.031	0.035
50	0.084	0.133	0.094	0.017	0.090	0.089
100	0.094	0.165	0.118	0.019	0.113	0.112
200	0.157	0.191	0.177	0.009	0.178	0.174
500	0.501	0.548	0.529	0.012	0.530	0.521
1000	0.799	0.831	0.820	0.008	0.821	0.804
Simulated Annealing 5000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.000	0.000	0.000	0.000	0.000	0.000
20	-0.038	-0.038	-0.038	0.000	-0.038	-0.038
50	-0.129	-0.121	-0.128	0.002	-0.129	-0.129
100	-0.090	-0.047	-0.074	0.011	-0.077	-0.079
200	0.074	0.078	0.076	0.001	0.077	0.077
500	0.087	0.093	0.090	0.001	0.090	0.090
1000	0.185	0.203	0.195	0.005	0.196	0.188
Discrete Particle Swarm Optimization 1000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.000	0.000	0.000	0.000	0.000	0.000
20	-0.038	0.437	0.126	0.161	0.115	0.026
50	0.043	0.310	0.179	0.072	0.179	0.169
100	0.315	0.379	0.349	0.017	0.350	0.341
200	0.516	0.594	0.561	0.020	0.562	0.557
500	0.933	0.979	0.958	0.012	0.959	0.938
1000	1.048	1.076	1.065	0.006	1.066	1.055
Discrete Particle Swarm Optimization 5000 Generations						
Jobs	**Minimum**	**Maximum**	**Mean**	**Std.**	**Median**	**Mode**
10	0.000	0.000	0.000	0.000	0.000	0.000
20	-0.038	-0.038	-0.038	0.000	-0.038	-0.038
50	-0.122	-0.071	-0.112	0.014	-0.118	-0.118
100	0.067	0.179	0.133	0.028	0.136	0.124
200	0.127	0.143	0.136	0.004	0.136	0.135
500	0.389	0.434	0.415	0.011	0.416	0.394
1000	0.717	0.748	0.736	0.008	0.738	0.720

and average cases. Moreover, the standard deviation of Simulated Annealing is also better than DPSO. Our experimental results certainly conclude that GPU parallelization is very powerful and efficient. Another observation which cannot be overlooked is that GPU technology has proven to be worthwhile only for large instances as shown in [102] and [33]. However, we show that with an efficient strategy for data transfer and algorithm parameters, high level of speed-ups are also possible for small instances. Concluding, our results and comparisons show that the best solutions can be achieved with SA and 5000 generations. To improve the values for 20 and 50 jobs, DPSO can be used but for larger problem instances DPSO does not work well with the given parameters. Reason for this could be that SA is an *intensification* oriented metaheuristic which searches intensively on a promising part of the domain, where as the DPSO is a *diversification* oriented metaheuristic which works more scattered [24].

7.8 Summary

This work presents an efficient parallelization of the Simulated Annealing algorithm for the Common Due Date (CDD) problem and the Un-restricted CDD with Controllable Processing Times. We utilize the 2-layered approach to break up the NP-hard problem in two components to parallelize the metaheuristic algorithms. Henceforth, two strategies for parallelizing the SA algorithm are explained, based on Ferreiro *et al.* [58]. Later on in the chapter, we focus on exhaustively explaining our parallel SA algorithm and its exact implementation. The NP-hard problems which are covered in this work are the CDD and the UCDDCP. We effectively use the polynomial algorithms provided in recent works of Lässig *et al.* [86] and Awasthi *et al.* [6], to optimize the given sequences for both these problems and to develop the parallel metaheuristic algorithms. Section 7.5 describes how the SA metaheuristic algorithm is mapped on to the CUDA programming model. Finally, we present our extensive evaluations of our parallel strategies for the two NP-hard scheduling problems. The algorithms are implemented over the benchmark instances provided in the OR-library [14] and by Awasthi *et al.* [6]. The efficiency of our parallel Simulated Annealing algorithm is proven by the comparison of our results with the previous CPU implementations, as well as the parallel DPSO algorithm on the same GPU architecture.

Not only do we obtain high speed-ups, our parallel algorithms also provide improvements to the best known solution values for several benchmark instances, in comparison to its corresponding CPU implementation. It is evident from our results that parallel DPSO does not perform as well as the parallel SA, at least for the studied problems. The speed-ups obtained with SA are massive compared to the very recent work of [86] and [6]. The speed-up values obtained are of the order of 100 and 50, even for a relatively small problem instance of 1000 jobs. However, DPSO is not just slow compared to the SA but it is also not able to find solutions of high quality, compared to the parallel SA. With this work, we show that the two-layered approach is not only easily parallelizable but it is also highly effective in utilizing the

GPGPU parallelization. Parts of this chapter have been adapted, modified and improved from our previous publication [8].

8

Discussions and Conclusion

Scheduling is an important aspect of industrial production, transportation, transshipment, allocation of resources and many more commercial/non-commercial businesses. Developments in scheduling processes have improved efficient utilization and management of available resources and aided in smoother planning of several industries as well as provided economic benefits. Efficient import/export freight transport of endless commodities, transportation management from traffic to flight, mass productions of goods, *etc.*, involve utilization of scheduling concepts. There is no doubt that scheduling plays an important role in our everyday life, directly or indirectly. Many of this scheduling problems are NP-hard, and hence to solve these problems deterministically, is impossible, if $\boldsymbol{P} \neq \boldsymbol{NP}$.

8.1 Contribution and Synopsis

In this research work, we deal with several real world scheduling problems in production and transportation. We implement a two-layered approach which is the product of splitting the integer programming formulation of an NP-hard scheduling problem in two layers. We develop novel polynomial algorithms to solve the resulting linear program and utilize metaheuristic heuristic algorithms to obtain an optimal or near-optimal solution. We first work on the Aircraft Landing Problem, where we utilize the two-layered approach and break the 0-1 mixed IP formulation in two parts. For the ALP, fixing the binary decision variables provides us with a landing sequence, and the resulting LP is solved polynomially. We carry out extensive theoretical analysis of the problem and provide an efficient $O(N^3)$ algorithm, where N is the number of aircraft in the landing sequence. Our algorithm not only returns optimal landing times for more practical case of safety constraint between consecutively landing planes, but also offers high quality solution for the general case of the safety constraint, which is highlighted by our results. The algorithm basically works by initializing the worst possible landing times to all the air-planes in the sequence, and then reducing the time between blocks of aircraft. Each such block of planes is identified with the help of our theoretical analysis of the problem. After developing the algorithm, we carry experimental analysis of our approach and compare our results with the state-of-the-art results on this problem. The significance of the two-layered approach is justified over several other recent and famous works. Developing a polynomial algorithm for the ALP also provides a solution to other scheduling problems against due-dates to minimizes weighted earliness/tardiness (E/T). We discuss this similarity of the ALP with the general E/T problem comprising of release-dates, distinct due-dates, distinct integer/non-integer penalties with sequence dependent set up times, for all the jobs.

Consider a scheduling problem where n jobs have to be processed on a single machine and each job possesses a processing time, distinct due-date and asymmetric earliness/tardiness penalties. Any job which is completed before or after its corresponding due-date incurs an earliness or tardiness penalty, respectively. Additionally, a release-date is associated with each job, before which a job can not be processed by the machine. Apart from these constraints, each job also requires a non-zero set-up time for the machine and a deadline before which the machine has to complete the processing of the job. The objective of the problem is similar to the ALP, *i.e.*, minimizing the weighted earliness/tardiness penalties, while maintaining all the constraints. Using algorithmic reduction, any given feasible job sequence of this scheduling problem can be easily optimized with our algorithm for the Aircraft Landing Problem. First of all, considering the similarities between the two problems, the obvious parameters are the due-dates, release-dates, deadlines, processing times, set up times and the earliness/tardiness penalties. It easy to observe that the due-date of a job in the E/T problem corresponds to the target landing of an aircraft in the ALP. The release-date of the E/T problem is the earliest time at which the machine can start the processing of a job, likewise the earliest landing time of an aircraft is the earliest time at which it can land at the runway. The deadline of a job is the latest time at which the machine must complete its processing, which is equivalent to the latest landing time of an aircraft in the ALP. The processing time and the set-up time of a job correspond to the safety distance constraint between any two aircraft. As a machine has to wait for a certain time to start a job processing, on the same lines an aircraft has to maintain some safety time with its preceding aircraft. However, in the E/T problem the time required for the setup and processing effects only its immediate following job, while an aircraft in the ALP has to maintain a safety distance with all its preceding planes. Hence, it is apparent to observe the clear similarity between the two problems and our polynomial algorithm is just as well suited for this general Earliness/Tardiness scheduling problem.

Henceforth in this work, we show the importance and benefit of the two-layered approach for the Common Due-Date problem. We utilize our ALP algorithm and develop an $O(n \log n)$ algorithm for the CDD scheduling problem. This is also done by reducing the CDD to ALP. Parameters corresponding to the earliest landing time, latest landing time do not exist in the CDD. Moreover, CDD contains a common due-date for all the jobs and time between any two jobs is only defined by the processing time of the preceding job, unlike the ALP where the landing time of an air-plane is constrained by all the planes ahead of it. Hence, the complexity of our algorithm for the common due-date problem, with the help of exponential search reduces to $O(n \log n)$, where n is the problem size, *i.e.*, the number of jobs in the processing sequence. In addition to this specialized polynomial algorithm for the resulting linear program of the CDD, we also carry out theoretical analysis of the CDD and prove an important property which states that the position of the due-date in the optimal schedule of any job sequence of the CDD is independent to the processing times of the jobs. With the help of this property we develop a linear algorithm for the CDD job sequence. Furthermore, we utilize the V-shaped property and provide an improvement heuristic to locally improve any job sequence to obtain a better solution value for the objective function. Our computational analyses show that the two-layered approach in conjunction with a modified simulated annealing algorithm is effective in solving all the benchmark instances to the best known solution values with a competitive runtime to other approaches mentioned in the literature. On the same lines of the CDD problem, we study the Common Due-Window scheduling problem along with its theoretical analysis. Once again we work on developing a specialized polynomial algorithm for the resulting linear program by fixing the binary decision variables of the 0-1 mixed integer programming formulation of the CDW, to feasible set of values. With the help of the property which is also valid for the CDD, we develop an $O(n)$ algorithm to optimize an job sequence. Our experimental analysis over the benchmark instances shows that we certainly perform better by obtaining better results than the best known solutions for several benchmark instances.

In the next chapter, we discuss and develop linear algorithms for the Unrestricted Common Due-Date problem with Controllable Processing Times. This scheduling problem is a variant of the CDD and we present and prove two important properties which help us to develop an $O(n)$ algorithm for any given processing sequence. One of these properties states that if reducing the processing time of a job can fetch a better solution value to the objective function of the UCDDCP, then this reduction in the processing time has to be made to its maximum possible value. In addition to this property, we also prove that if the due-date falls at the completion time of some job r in the optimal schedule for the CDD, then the position of the due-date relative to the jobs remains unchanged for the UCDDCP job sequence. We derive this property by utilizing the CDD property of Chapter 4 and the property of maximum compression. These two properties are then exploited to develop linear algorithm for any given job sequence of the UCDDCP. In this work, we also provide a complete set of benchmark instances for the problem which are derived from the CDD instances of Biskup and Feldmann [20], and provide our best results by combining Simulated Annealing and Threshold Accepting with our polynomial algorithms. All these NP-hard scheduling problems have been dealt with the two-layered approach which requires the development of efficient specialized polynomial algorithms for the resulting linear program of the specific scheduling problem. Moreover, we also highlight the added advantage of this approach for parallel processing to solve the NP-hard problems.

8.2 Utilization of Parallel Computing

One of the best tools available today to solve the NP-hard optimization problems are the metaheuristic algorithms. However, implementing these heuristic algorithms solely, both to optimize the job sequences and search for the optimal/near-optimal job sequence leads to a fair amount of unnecessary computation. Hence, the possible search space gets quite large. However, with this approach we make sure that any processing sequence is solved to optimal solution value in polynomial time and thus requirement of the metaheuristic essentially boils down to searching for the job sequences only. With this in mind, it paves a clear way for any population based metaheuristic algorithm to solve a problem with the two-layered approach, to be parallelized with ease.

Hence, we exploit this idea and incorporate our linear algorithms for the common due-date problem and the un-restricted common due-date problem with controllable processing times, with parallel metaheuristic algorithms, namely the simulated annealing and discrete particle swarm optimization on the graphical processing units. GPUs have recently evolved from being graphics-only processing units to become a general purpose parallel computing architecture. Although, a single core of a CPU is faster than a single GPU core, utilizing the high number of computing threads in a GPU clearly outruns a CPU, provided the GPU must carry out a large amount of simple and repetitive computations. Our results for the two exemplary cases of the NP-hard problems reflect the utility of the specialized LP algorithms and yield speed-ups of several folds in comparison to the exact same implementation on a CPU.

8.3 Adaptability to Other Optimization Problems

This work highlights and proves the importance of the mentioned approach for several NP-hard scheduling problems, both for single and multi-core processing units. Moreover, we now show that this approach is not only effective for the scheduling problems but also for other optimization problems. We discuss one such problem known as the Warehouse Location Problem (WLP). The WLP consists of allocating resources of distributed customers to capacitated or uncapacitated warehouses, such that the overall transportation cost as well as the storage cost is minimized. Each customer j requires its goods in the amount of d_j to be stored at the storage facilities such that its storage demand is met completely. Each of these customers is situated at different locations and at the same time there are several possible locations for the warehouses, with each warehouse i may or may not be restricted to a maximum possible storage capacity. The distance of each customer to every warehouse is represented in terms of the transportation cost c_{ij}, such that the cost of transporting a unit amount of good from customer j to warehouse i is c_{ij}. We discuss the capacitated WLP where each warehouse is limited with a maximum storage capacity of b_i. In addition to this capacity constraint there is an additional fixed cost F_i associated with each warehouse, which is incurred if the warehouse is serviced for storage. Hence, the optimization problem is to figure out the location of these warehouses which fulfills the demands of all the customers completely and each warehouse does not exceed its capacity limit, such that the total transportation cost and the fixed cost of opening a warehouse i is minimized. We now present the 0-1 mixed integer programming formulation of this problem below and demonstrate the utilization of the two-layered approach for the WLP.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\cdot x_{ij} + \sum_{i=1}^{m} F_i \cdot y_i \\
\text{subject to} \quad & \sum_{i=1}^{m} x_{ij} \geq d_j, && \forall j \in \{1,2,\ldots,n\} \\
& \sum_{j=1}^{n} x_{ij} \leq y_i \cdot b_i, && \forall i \in \{1,2,\ldots,m\} \\
& x_{ij} \geq 0, && \forall i \in \{1,2,\ldots,m\}, \forall j \in \{1,2,\ldots,n\} \\
& \boldsymbol{y_i \in \{0,1\}}, && \forall i \in \{1,2,\ldots,m\},
\end{aligned}
$$

In the above formulation, the binary decision variable y_i is equal to 1 if the warehouse i is opened and 0 otherwise. Clearly, the above formulation is similar to the formulation to other IP formulations of scheduling problems discussed in this work, in the sense that if we have a feasible set of warehouses which need to be opened, then the above MIP converts to a linear programming formulation for that set of warehouses. In other words, fixing the decision variable y_i to a set of feasible set of values, we are then left to solve the resulting LP, which is polynomial. We do not go into the details of development of the specialized polynomial algorithm for the above problem, but only prove that the two-layered approach is not restricted to scheduling problems but also to other NP-hard optimization problems.

References

[1] Ahmadizar, F., Farahani, M.H.: A novel hybrid genetic algorithm for the open shop scheduling problem. The International Journal of Advanced Manufacturing Technology **62**(5-8), 775–787 (2012)

[2] Appelgren, L.H.: A column generation algorithm for a ship scheduling problem. Transportation Science **3**(1), 53–68 (1969)

[3] Awasthi, A., Kramer, O., Lässig, J.: Aircraft landing problem: An efficient algorithm for a given landing sequence. In: 16th IEEE International Conferences on Computational Science and Engineering (CSE 2013), pp. 20–27 (2013). DOI 10.1109/CSE.2013.14

[4] Awasthi, A., Lässig, J., Kramer, O.: Common due-date problem: Exact polynomial algorithms for a given job sequence. In: 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013, pp. 258–264 (2013). DOI 10.1109/SYNASC.2013.41

[5] Awasthi, A., Lässig, J., Kramer, O.: Solving Computationally Expensive Engineering Problems: Methods and Applications, chap. A Novel Approach to the Common Due-Date Problem on Single and Parallel Machines, pp. 293–314. Springer International Publishing (2014). DOI 10.1007/978-3-319-08985-0_13

[6] Awasthi, A., Lässig, J., Kramer, O.: Un-restricted common due-date problem with controllable processing times: Linear algorithm for a given job sequence. In: 17th International Conference on Enterprise Information Systems (ICEIS), pp. 526–534 (2015)

[7] Awasthi, A., Lässig, J., Kramer, O., Weise, O.: Common due-window problem: Polynomial algorithms for a given processing sequence. In: IEEE Symposium on Computational Intelligence in Production and Logistics Systems (IEEE SSCI-CIPLS), pp. 32–39 (2014). DOI 10.1109/CIPLS.2014.7007158

[8] Awasthi, A., Lässig, J., Leuschner, J., Weise, T.: GPGPU-based parallel algorithms for scheduling against due date. In: 2016 IEEE International

Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 766–775 (2016). DOI 10.1109/IPDPSW.2016.66

[9] Banisadr, A.H., Zandieh, M., Mahdavi, I.: A hybrid imperialist competitive algorithm for single-machine scheduling problem with linear earliness and quadratic tardiness penalties. The International Journal of Advanced Manufacturing Technology **65**(5–8), 981–989 (2013)

[10] Bauman, J., Jzefowska, J.: Minimizing the earlinesstardiness costs on a single machine. Computers & Operations Research **33**(11), 3219 – 3230 (2006)

[11] Beasley, J., Krishnamoorthy, M., Sharaiha, Y., Abramson, D.: Scheduling aircraft landings - the static case. The Management School, Imperial College, London SW7 2AZ, England (1995)

[12] Beasley, J., Krishnamoorthy, M., Sharaiha, Y., Abramson, D.: Displacement problem and dynamically scheduling aircraft landings. Journal of the operational research society **55**(1), 54–64 (2004)

[13] Beasley, J., Sonander, J., Havelock, P.: Scheduling aircraft landings at london heathrow using a population heuristic. Journal of the Operational Research Society **52**(5), 483–493 (2001)

[14] Beasley, J.E.: OR-library: Distributing test problems by electronic mail. Journal of the Operational Research Society **41**(11), 1069–1072 (1990)

[15] Bector, C., Gupta, Y., Gupta, M.: V-shape property of optimal sequence of jobs about a common due date on a single machine. Computers & Operations Research **16**(6), 583 – 588 (1989)

[16] Bellman, R.: Mathematical aspects of scheduling theory. Journal of the Society for Industrial & Applied Mathematics **4**(3), 168–205 (1956)

[17] Bencheikh, G., Boukachour, J., Alaoui, A.: Improved ant colony algorithm to solve the aircraft landing problem. International Journal of Computer Theory and Engineering **3**(2), 224–233 (2011)

[18] Bencheikh, G., Boukachour, J., Alaoui, A., Khoukhi, F.: Hybrid method for aircraft landing scheduling based on a job shop formulation. International Journal of Computer Science and Network Security **9**(8), 78–88 (2009)

[19] Biskup, D., Cheng, T.: Single-machine scheduling with controllable processing times and earliness, tardiness and completion time penalties. Engineering Optimization **31**(3), 329–336 (1999)

[20] Biskup, D., Feldmann, M.: Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. Computers & Operations Research **28**(8), 787 – 801 (2001)

[21] Biskup, D., Feldmann, M.: On scheduling around large restrictive common due windows. European Journal of Operational Research **162**(3), 740 – 761 (2005)

[22] Biskup, D., Jahnke, H.: Common due date assignment for scheduling on a single machine with jointly reducible processing times. International Journal of Production Economics **69**(3), 317 – 322 (2001)

[23] Blum, C., Chiong, R., Clerc, M., Jong, K., Michalewicz, Z., Neri, F., Weise, T.: Variants of Evolutionary Algorithms for Real-World Applications, 1 edn., chap. Evolutionary Optimization, pp. 1–29. Springer-Verlag Berlin Heidelberg (2012)

[24] Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003). DOI 10.1145/937503.937505

[25] Bożejko, W., Hejducki, Z., Uchroński, M., Wodecki, M.: Solving the flexible job shop problem on multi-GPU. Procedia Computer Science **9**, 2020–2023 (2012)

[26] Bożejko, W., Uchroński, M., Wodecki, M.: Fast parallel cost function calculation for the flow shop scheduling problem. In: Proceedings of Artificial Intelligence and Soft Computing, pp. 378–386. Springer (2012)

[27] Bożejko, W., Uchroński, M., Wodecki, M.: Parallel cost function determination on GPU for the job shop scheduling problem. In: Parallel Processing and Applied Mathematics, pp. 1–10. Springer (2012)

[28] Bożejko, W., Uchroński, M., Wodecki, M.: Artificial Intelligence and Soft Computing: 12th International Conference, ICAISC 2013, Zakopane, Poland, June 9-13, 2013, Proceedings, Part II, chap. Parallel Neuro-Tabu Search Algorithm for the Job Shop Scheduling Problem, pp. 489–499. Springer Berlin Heidelberg (2013)

[29] Brucker, P.: Scheduling Algorithms. Springer (2007)

[30] Bukata, L., Sucha, P.: A GPU algorithm design for resource constrained project scheduling problem. In: Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, pp. 367–374 (2013)

[31] Bukata, L., Sucha, P., Hanzalek, Z.: Solving the resource constrained project scheduling problem using the parallel tabu search designed for the CUDA platform. Journal of Parallel and Distributed Computing **77**, 58 – 68 (2015)

[32] Chakroun, I., Melab, N.: An adaptative multi-GPU based branch-and-bound. a case study: The flow-shop scheduling problem. In: Proceedings of IEEE International Conference on High Performance Computing and Communication & Embedded Software and Systems (HPCC-ICESS), pp. 389–395 (2012)

[33] Chakroun, I., Melab, N., Mezmaz, M., Tuyttens, D.: Combining multi-core and GPU computing for solving combinatorial optimization problems. Journal of Parallel and Distributed Computing **73**(12), 1563–1577 (2013)

[34] Chen, Z., Lee, C.: Parallel machine scheduling with a common due window. European Journal of Operational Research **136**(3), 512 – 527 (2002)

[35] Cheng, T.: Optimal due-date assignment and sequencing in a single machine shop. Applied Mathematics Letters **2**(1), 21–24 (1989)

[36] Cheng, T., Ouz, C., Qi, X.: Due-date assignment and single machine scheduling with compressible processing times. International Journal of Production Economics **43**(2-3), 107 – 113 (1996)

[37] Cheng, T., Yang, S., Yang, D.: Common due-window assignment and scheduling of linear time-dependent deteriorating jobs and a deteriorating maintenance activity. International Journal of Production Economics **135**(1), 154 – 161 (2012)

[38] Cheng, T.C.E., Kahlbacher, H.G.: A proof for the longest-job-first policy in one-machine scheduling. Naval Research Logistics (NRL) **38**(5), 715–720 (1991)

[39] Choong, A., Beidas, R., Zhu, J.: Parallelizing simulated annealing-based placement using GPGPU. In: Field Programmable Logic and Applications (FPL), 2010 International Conference on, pp. 31–34. IEEE (2010)

[40] Chrétienne, P.: Minimizing the earliness and tardiness cost of a sequence of tasks on a single machine. RAIRO-Operations Research-Recherche Opérationnelle **35**(2), 165–187 (2001)

[41] Chrétienne, P., Sourd, F.: PERT scheduling with convex cost functions. Theoretical Computer Science **292**(1), 145 – 164 (2003)

[42] Ciesielski, V., Scerri, P.: An anytime algorithm for scheduling of aircraft landing times using genetic algorithms. Australian Journal of Intelligent Information Processing Systems **4**, 206–213 (1997)

[43] Coelho, I., Haddad, M., Ochi, L., Souza, M., Farias, R.: A hybrid cpu-gpu local search heuristic for the unrelated parallel machine scheduling problem. In: Third Workshop on Applications for Multi-Core Architectures (WAMCA), pp. 19–23 (2012)

[44] Cook, S.: CUDA Programming: A Developer's Guide to Parallel Computing with GPUs, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2013)

[45] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Third Edition, 3rd edn. The MIT Press (2009)

[46] Czapiński, M., Barnes, S.: Tabu search with two approaches to parallel flowshop evaluation on CUDA platform. Journal of Parallel and Distributed Computing **71**(6), 802–811 (2011)

[47] Dali, N., Bouamama, S.: Gpu-pso: Parallel particle swarm optimization approaches on graphical processing unit for constraint reasoning: Case of max-csps. Procedia Computer Science **60**, 1070 – 1080 (2015)

[48] Dantzig, G., Thapa, M.: Linear Programming 1: Introduction. Operations Research and Financial Engineering. Springer-Verlag, New York (1997)

[49] Davis, J.S., Kanet, J.J.: Single-machine scheduling with early and tardy completion costs. Naval Research Logistics (NRL) **40**(1), 85–101 (1993)

[50] Della Croce, F., Tadei, R., Volta, G.: A genetic algorithm for the job shop problem. Computers & Operations Research **22**(1), 15–24 (1995)

[51] Dueck, G., Scheuer, T.: Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Jour-

nal of Computational Physics **90**(1), 161 – 175 (1990). DOI http://dx.doi.org/10.1016/0021-9991(90)90201-B

[52] Ernst, A., Krishnamoorthy, M., Storer, R.: Heuristic and exact algorithms for scheduling aircraft landings. Networks **34**(3), 229–241 (1999)

[53] Etiler, O., Toklu, B., Atak, M., Wilson, J.: A genetic algorithm for flow shop scheduling problems. Journal of the Operational Research Society **55**(8), 830–835 (2004)

[54] Falkenauer, E., Bouffouix, S.: A genetic algorithm for job shop. In: Proceedings of IEEE International Conference on Robotics and Automation, pp. 824–829. IEEE (1991)

[55] Farber, R.: CUDA Application Design and Development. Applications of GPU computing. Morgan Kaufmann (2011)

[56] Faye, A.: Solving the aircraft landing problem with time discretization approach. European Journal of Operational Research **242**(3), 1028 – 1038 (2015)

[57] Feldmann, M., Biskup, D.: Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. Computers & Industrial Engineering **44**(2), 307–323 (2003)

[58] Ferreiro, A., García, J., López-Salas, J., Vázquez, C.: An efficient implementation of parallel simulated annealing algorithm in GPUs. Journal of Global Optimization **57**(3), 863–890 (2013)

[59] Gantt, H.L.: A graphical daily balance in manufacture. Transactions of the American Society of Mechanical Engineers **24**, 1322–1336 (1903)

[60] Gen, M., Tsujimura, Y., Kubota, E.: Solving job-shop scheduling problems by genetic algorithm. In: IEEE International Conference on Systems, Man, and Cybernetics, 1994. Humans, Information and Technology., vol. 2, pp. 1577–1582 (1994)

[61] Gerstl, E., Mosheiov, G.: Due-window assignment problems with unit-time jobs. Applied Mathematics and Computation **220**(0), 487 – 495 (2013)

[62] Glover, F., McMillan, C.: The general employee scheduling problem. an integration of MS and AI. Computers & operations research **13**(5), 563–573 (1986)

[63] Goldberg, D.E., Holland, J.H.: Genetic algorithms in search, optimization and machine learning. Machine learning **3**(2), 95–99 (1988)

[64] Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha and Vancouver, vol. 5, pp. 287 – 326. Elsevier (1979)

[65] Grey, M., Tarjan, R., Wilfong, G.: One-processor scheduling with symmetric earliness and tardiness penalties. Mathematics of Operations Research **13**(2), 330–348 (1988)

[66] Hall, N., Kubiak, W., Sethi, S.: Earliness–tardiness scheduling problems, II: deviation of completion times about a restrictive common due date. Operations Research **39**(5), 847–856 (1991)

[67] Hancerliogullari, G., Rabadi, G., Al-Salem, A., Kharbeche, M.: Greedy algorithms and metaheuristics for a multiple runway combined arrival-departure aircraft sequencing problem. Journal of Air Transport Management **32**, 39 – 48 (2013)

[68] Hendel, Y., Sourd, F.: An improved earlinesstardiness timing algorithm. Computers & Operations Research **34**(10), 2931 – 2938 (2007)

[69] Hillier, F., Lieberman, G.: Introduction to operations research. McGraw-Hill, New York (1982)

[70] Hoogeveen, J.A., Van de Velde, S.L.: Scheduling around a small common due date. European Journal of Operational Research **55**(2), 237–242 (1991)

[71] Jackson, J.R.: An extension of johnson's results on job IDT scheduling. Naval Research Logistics Quarterly **3**(3), 201–203 (1956)

[72] James, R.J.W.: Using tabu search to solve the common due date early/-tardy machine scheduling problem. Computers & Operations Research **24**(3), 199–208 (1997)

[73] Janiak, A., Janiak, W., Kovalyov, M., Kozan, E., Pesch, E.: Parallel machine scheduling and common due window assignment with job independent earliness and tardiness costs. Information Sciences **224**, 109 – 117 (2013)

[74] Janiak, A., Kwiatkowski, T., Lichtenstein, M.: Scheduling problems with a common due window assignment: A survey. International Journal of Applied Mathematics and Computer Science **23**(1), 231–241 (2013)

[75] Johnson, S.M.: Optimal two-and three-stage production schedules with setup times included. Naval research logistics quarterly **1**(1), 61–68 (1954)

[76] Kacem, I.: Fully polynomial time approximation scheme for the total weighted tardiness minimization with a common due date. Discrete Applied Mathematics **158**(9), 1035 – 1040 (2010)

[77] Kanet, J.: Minimizing the average deviation of job completion times about a common due date. Naval Research Logistics Quarterly **28**(4), 643–651 (1981)

[78] Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4), 373–395 (1984)

[79] Karp, R.M.: Reducibility Among Combinatorial Problems. Springer (1972)

[80] Kayvanfar, V., Komaki, G., Aalaei, A., Zandieh, M.: Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times. Computers and Operations Research **41**(1), 31–43 (2014)

[81] Kelley Jr, J.E., Walker, M.R.: Critical-path planning and scheduling. In: Proceedings of Eastern Joint IRE-AIEE-ACM Computer Conference, pp. 160–173. ACM (1959)
[82] Kim, J.: Genetic algorithm stopping criteria for optimization of construction resource scheduling problems. Construction Management and Economics **31**(1), 3–19 (2013)
[83] Kirk, D., Hwu, W.: Programming Massively Parallel Processors: A Hands-on Approach. Applications of GPU Computing Series. Elsevier Science (2010)
[84] Klee, V., Minty, G.: How good is the simplex algorithm. Tech. rep., DTIC Document (1970)
[85] Krämer, F., Lee, C.: Due window scheduling for parallel machines. Mathematical and Computer Modelling **20**(2), 69 – 89 (1994)
[86] Lässig, J., Awasthi, A., Kramer, O.: Common due-date problem: Linear algorithm for a given job sequence. In: 17th IEEE International Conferences on Computational Science and Engineering (CSE), pp. 97–104 (2014)
[87] Lässig, J., Sudholt, D.: Analysis of speedups in parallel evolutionary algorithms for combinatorial optimization. In: Proceedings of the 22nd International Conference on Algorithms and Computation, ISAAC'11, pp. 405–414. Springer-Verlag (2011)
[88] Lässig, J., Sudholt, D.: General upper bounds on the runtime of parallel evolutionary algorithms pp. 1–33 (2013). DOI 10.1162/EVCO_a_00114
[89] Lawler, E.L., Lenstra, J.K., Kan, A.R., Shmoys, D.B.: Sequencing and scheduling: Algorithms and complexity. Handbooks in Operations Research and Management Science **4**, 445–522 (1993)
[90] Yeung, W., Oguz, C., Cheng, T.: Single-machine scheduling with a common due window. Computers & Operations Research **28**(2), 157 – 175 (2001)
[91] Lenstra, J.K., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. Annals of Discrete Mathematics **1**, 343–362 (1977)
[92] Liao, C.J., Cheng, C.C.: A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date. Computers & Industrial Engineering **52**(4), 404–413 (2007)
[93] Lieder, A., Briskorn, D., Stolletz, R.: A dynamic programming approach for the aircraft landing problem with aircraft classes. European Journal of Operational Research **243**(1), 61 – 69 (2015)
[94] Lieder, A., Stolletz, R.: Scheduling aircraft take-offs and landings on interdependent and heterogeneous runways. Transportation Research Part E: Logistics and Transportation Review **88**, 167 – 188 (2016)
[95] Liman, S., Panwalkar, S., Thongmee, S.: Determination of common due window location in a single machine scheduling problem. European Journal of Operational Research **93**(1), 68 – 74 (1996)

[96] Liman, S., Panwalkar, S., Thongmee, S.: A single machine scheduling problem with common due window and controllable processing times. Annals of Operations Research **70**(0), 145–154 (1997)

[97] Liu, L., Zhou, H.: Hybridization of harmony search with variable neighborhood search for restrictive single-machine earliness/tardiness problem. Information Sciences **226**, 68 – 92 (2013). DOI http://dx.doi.org/10.1016/j.ins.2012.11.007

[98] Lomnicki, Z.: A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem. Operational Research Society **16**(1), 89–100 (1965)

[99] Louis, S.J., Xu, Z.: Genetic algorithms for open shop scheduling and re-scheduling. In: Proceedings of 11th ISCA International Conference on Computers and Their Applicatons, pp. 99–102. Citeseer (1996)

[100] Lu, Y.Y., Li, G., Wu, Y.B., Ji, P.: Optimal due-date assignment problem with learning effect and resource-dependent processing times. Optimization Letters **8**(1), 113–127 (2012)

[101] Luong, T., Melab, N., Talbi, E.: GPU-based island model for evolutionary algorithms. In: Proceedings of 12th Annual Conference on Genetic and Evolutionary Computation, pp. 1089–1096. ACM (2010)

[102] Luong, T., Melab, N., Talbi, E.: Large neighborhood local search optimization on graphics processing units. In: Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, pp. 1–8. IEEE (2010)

[103] Luong, T., Melab, N., Talbi, E.: GPU computing for parallel local search metaheuristic algorithms. IEEE Transactions on Computers **62**(1), 173–185 (2013)

[104] Lutton, E., Levy Vehel, J.: Holder functions and deception of genetic algorithms. IEEE Transactions on Evolutionary Computation **2**(2), 56–71 (1998)

[105] Ma, W., Xu, B., Liu, M., Huang, H.: An efficient approximation algorithm for aircraft arrival sequencing and scheduling problem. Mathematical Problems in Engineering **2014** (2014)

[106] Melab, N., Chakroun, I., Mezmaz, M., Tuyttens, D.: A GPU-accelerated branch-and-bound algorithm for the flow-shop scheduling problem. In: Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), pp. 10–17. IEEE (2012)

[107] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of sstate calculations by fast computing machines. Chemical Physics **21**(6), 1087–1092 (1953)

[108] Moghaddam, R., Panah, M., F., R.: Scheduling the sequence of aircraft landings for a single runway using a fuzzy programming approach. Journal of Air Transport Management **25**, 15 – 18 (2012)

[109] Moser, I., Hendtlass, T.: Solving dynamic single-runway aircraft landing problems with extremal optimisation. In: IEEE Symposium on Computational Intelligence in Scheduling, 2007. SCIS'07., pp. 206–211 (2007)

[110] Murata, T., Ishibuchi, H., Tanaka, H.: Genetic algorithms for flow-shop scheduling problems. Computers & Industrial Engineering **30**(4), 1061–1071 (1996)

[111] Muth, J.F., Thompson, G.L.: Industrial Scheduling. Prentice-Hall (1963)

[112] Nearchou, A.: A differential evolution approach for the common due date early/tardy job scheduling problem. Computers & Operations Research **35**(4), 1329 – 1343 (2008). DOI http://dx.doi.org/10.1016/j.cor.2006.08.013

[113] Nearchou, A.: Scheduling with controllable processing times and compression costs using population-based heuristics. International Journal of Production Research **48**(23), 7043–7062 (2010)

[114] NVIDIA, C.: CUDA C programming guide version 7.5 (2016)

[115] Pan, Q., Tasgetiren, M., Liang, Y.: Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science 4150, chap. Minimizing Total Earliness and Tardiness Penalties with a Common Due Date on a Single-Machine Using a Discrete Particle Swarm Optimization Algorithm, pp. 460–467. Springer-Verlag Berlin Heidelberg (2006)

[116] Pan, Q., Tasgetiren, M., Liang, Y.: A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. Computers & Operations Research **35**(9), 2807 – 2839 (2008). Part Special Issue: Bio-inspired Methods in Combinatorial Optimization

[117] Panwalkar, S., Rajagopalan, R.: Single-machine sequencing with controllable processing times. European Journal of Operational Research **59**(2), 298 – 302 (1992)

[118] Panwalkar, S.S., Smith, M.L., Seidmann, A.: Common due date assignment to minimize total penalty for the one machine scheduling problem. Operations Research **30**(2), 391–399 (1982)

[119] Pinedo, M.: Scheduling: theory, algorithms, and systems. Springer Science+ Business Media (2012)

[120] Pinel, F., Dorronsoro, B., Bouvry, P.: Solving very large instances of the scheduling of independent tasks problem on the GPU. Journal of Parallel and Distributed Computing **73**(1), 101 – 110 (2013). Metaheuristics on GPUs

[121] Pinol, H., Beasley, J.: Scatter search and bionomic algorithms for the aircraft landing problem. European Journal of Operational Research **171**(2), 439–462 (2006)

[122] Rebai, M., Kacem, I., Adjallah, K.: Earliness-tardiness minimization on a single machine to schedule preventive maintenance tasks: metaheuristic and exact methods. Journal of Intelligent Manufacturing **23**(4), 1207–1224 (2012)

[123] Ronconi, D.P., Kawamura, M.S.: The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm. Computational & Applied Mathematics **29**, 107 – 124 (2010)

[124] Sabar, N., Kendall, G.: An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem. Omega **56**, 88 – 98 (2015)

[125] Salamon, P., Sibani, P., Frost, R.: Facts, Conjectures, and Improvements for Simulated Annealing. Society for Industrial and Applied Mathematics (2002). DOI 10.1137/1.9780898718300

[126] Salehipour, A., Modarres, M., Naeni, L.: An efficient hybrid metaheuristic for aircraft landing problem. Computers & Operations Research **40**(1), 207–213 (2012)

[127] Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming. Pearson Education (2010)

[128] Seidmann, A., Panwalkar, S., Smith, M.: Optimal assignment of due-dates for a single processor scheduling problem. The International Journal Of Production Research **19**(4), 393–399 (1981)

[129] Shabtay, D., Steiner, G.: A survey of scheduling with controllable processing times. Discrete Applied Mathematics **155**(13), 1643 – 1666 (2007)

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

[131] Somani, A., Singh, D.: Parallel genetic algorithm for solving job-shop scheduling problem using topological sort. In: IEEE International Conference on Advances in Engineering and Technology Research (ICAETR), pp. 1–8 (2014)

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

[133] Szwarc, W., Mukhopadhyay, S.K.: Optimal timing schedules in earliness-tardiness single machine sequencing. Naval Research Logistics (NRL) **42**(7), 1109–1114 (1995)

[134] Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)

[135] Tang, K., Wang, Z., Cao, X., Zhang, J.: A multi-objective evolutionary approach to aircraft landing scheduling problems. In: IEEE Conference on Evolutionary Computation, CEC, 2008., pp. 3650–3656 (2008)

[136] Tasgetiren, M.F., Pan, Q.K., Liang, Y.C., Suganthan, P.N.: A discrete differential evolution algorithm for the total earliness and tardiness penalties with a common due date on a single-machine. In: 2007 IEEE Symposium on Computational Intelligence in Scheduling, pp. 271–278 (2007). DOI 10.1109/SCIS.2007.367701

[137] Thompson, G.L.: Recent developments in the job-shop scheduling problem. Naval Research Logistics Quarterly **7**(4), 585–589 (1960)

[138] Toksari, M., Guner, E.: The common due-date early/tardy scheduling problem on a parallel machine under the effects of time-dependent learning and linear and nonlinear deterioration. Expert Systems with Applications **37**(1), 92–112 (2010)

[139] Tseng, C., Liao, C., Huang, K.: Minimizing total tardiness on a single machine with controllable processing times. Computers & Operations Research **36**(6), 1852 – 1858 (2009)

[140] Tsutsui, S., Fujimoto, N.: Solving quadratic assignment problems by genetic algorithms with GPU computation: a case study. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, pp. 2523–2530. ACM (2009)

[141] Van Luong, T., Melab, N., Talbi, E.G.: Parallel hybrid evolutionary algorithms on GPU. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2010)

[142] Wagner, H.M.: An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly **6**(2), 131–140 (1959)

[143] Wan, G.: Single machine common due window scheduling with controllable job processing times. In: First International Conference on Combinatorial Optimization and Applications (COCOA), pp. 279–290. Springer Berlin Heidelberg (2007). DOI 10.1007/978-3-540-73556-4_30

[144] Xie, J., Zhou, Y., Zheng, H.: A hybrid metaheuristic for multiple runways aircraft landing problem based on bat algorithm. Journal of Applied Mathematics **2013** (2013)

[145] Xu, Z., Zou, Y., Kong, X.: Meta-heuristic algorithms for parallel identical machines scheduling problem with weighted late work criterion and common due date. SpringerPlus **4**(1), 1–13 (2015)

[146] Yamada, T., Nakano, R.: Genetic algorithms for job-shop scheduling problems. In: Proceedings of Modern Heuristics for Decision Support, pp. 67–81 (1997)

[147] Yang, D., Lai, C., Yang, S.: Scheduling problems with multiple due windows assignment and controllable processing times on a single machine. International Journal of Production Economics **150**, 96 – 103 (2014)

[148] Yeung, W., Choi, T., Cheng, T.: Optimal scheduling of a single-supplier single-manufacturer supply chain with common due windows. IEEE Transactions on Automatic Control **55**(12), 2767–2777 (2010)

[149] Yin, Y., Cheng, T., Cheng, S., Wu, C.: Single-machine batch delivery scheduling with an assignable common due date and controllable processing times. Computers & Industrial Engineering **65**(4), 652 – 662 (2013)

[150] Yin, Y., Cheng, T., Wu, C., Cheng, S.: Single-machine batch delivery scheduling and common due-date assignment with a rate-modifying activity. International Journal of Production Research **52**(19), 5583–5596 (2014)

[151] Yunqiang, Y., Cheng, T., Hsu, C., Wu, C.: Single-machine batch delivery scheduling with an assignable common due window. Omega **41**(2), 216 – 225 (2013)

[152] Zelazny, D., Pempera, J.: Solving multi-objective permutation flowshop scheduling problem using cuda. In: 20th IEEE International Conference

on Methods and Models in Automation and Robotics (MMAR), pp. 347–352 (2015)

[153] Zhao, H., Ma, H., Han, G., Zhao, L.: A PTAS for common due window scheduling with window penalty on identical machines. In: Computer Application and System Modeling (ICCASM), 2010 International Conference on, vol. 10, pp. 648–652 (2010)

[154] Zhou, Y., Tan, Y.: GPU-based parallel particle swarm optimization. In: Evolutionary Computation, 2009. CEC'09. IEEE Congress on, pp. 1493–1500. IEEE (2009)

[155] Awasthi, A., Lässig, J., Weise, T., Kramer, O.: Tackling common due window problem with a two-layered approach. In: 10th International Conference on Combinatorial Optimization and Applications (COCOA 2016), Hong Kong, China, pp. 772–781 (2016). DOI 10.1007/978-3-319-48749-6

List of Tables

List of Figures

List of Algorithms