Universiteit Leiden
The Netherlands

Scheduling under weighted squared deviation from due dates for unit sized jobs

Koopmans, C.

Citation

Koopmans, C. *Scheduling under weighted squared deviation from due dates for unit sized jobs.*

Version:	Not Applicable (or Unknown)
License:	License to inclusion and publication of a Bachelor or Master thesis in the Leiden University Student Repository
Downloaded from:	https://hdl.handle.net/1887/4171487

Note: To cite this publication please use the final published version (if applicable).

C.M.P. Koopmans

Scheduling under weighted squared deviation from due dates for unit sized jobs.

Master thesis

August 31, 2021

Supervisors: Prof. Dr. F. M. Spieksma, Prof. Dr. T. Vredeveld, MSc. M.Y. Buchem

Leiden university
Mathematical Institute

Abstract

We study the Just-In-Time scheduling problem minimizing $\sum_j w_j(C_j - d_j)^2$ for jobs with unit processing times. The focus lays on the problem with a single machine, but when possible, we expand our findings to the identical parallel machine environment. It is unknown whether this problem, for a single machine, is in NP. An application of this problem is to minimize fuel use over all transport freighters passing a lock or parallel locks. We explain how the problem for fixed starting, and completion times, can be solved as an assignment problem. Thus, the problem with integral input and integral starting and completion times, can be solved in polynomial time. We give an effective so-called block merging algorithm to solve the problem for a given job sequence. We apply this procedure to problems with unit weight. Then we show how to distribute the jobs over the m machines, such that the problem can be solved as m single machine problems. For a single common due date, we can effectively solve the problem for a single machine. In the identical parallel machine environment, we show how to effectively solve the problem for a large enough due date. When the due date is smaller, we will give an exponential procedure to solve the problem. For the general problem we give a local search heuristic and a polynomial time greedy heuristic. Finally, for a fixed number of distinct due dates, we give a fully polynomial additive approximation scheme.

Contents

1 Introduction **3**
- 1.1 The problem of interest . . . 3
  - 1.1.1 Notation . . . 4
  - 1.1.2 The integral scheduling problem . . . 4
  - 1.1.3 Constrained vs unconstrained problems . . . 4
- 1.2 Related work . . . 4
- 1.3 The Research . . . 5

2 Complexity **7**
- 2.1 Reduction from the partition problem . . . 7
- 2.2 Reduction from the 3-partition problem . . . 8

3 Structural properties and solution methods for fixed order or fixed starting times **9**
- 3.1 Structural properties of optimal schedules . . . 9
- 3.2 The problem for a fixed job sequence . . . 10
  - 3.2.1 Quadratic programming for a given job sequence . . . 10
  - 3.2.2 An alternative algorithm to schedule a job sequence optimally . . . 12
- 3.3 Fixed starting and completion times . . . 15

4 Unit weight problems **17**
- 4.1 Structural Properties . . . 17
- 4.2 Finding an optimal integral schedule . . . 18
- 4.3 Finding an optimal schedule for general starting times . . . 18
- 4.4 Extension the identical parallel machine environment . . . 19

5 Single due date problems **24**
- 5.1 Optimal schedules in the single machine environment . . . 24
  - 5.1.1 The unconstrained problem . . . 25
  - 5.1.2 The constrained problem . . . 26
- 5.2 Optimal schedules in the identical parallel machine environment . . . 26
  - 5.2.1 The unconstrained problem . . . 27
  - 5.2.2 The constrained problem . . . 27

6 The general single machine problem **29**
- 6.1 A local search heuristic . . . 29
  - 6.1.1 Counterexample of optimality . . . 29
- 6.2 A greedy heuristic . . . 31
  - 6.2.1 Counterexample of optimality . . . 32
- 6.3 An additive approximation scheme . . . 32
  - 6.3.1 Idle periods and structural properties . . . 33
  - 6.3.2 The additive approximation scheme . . . 34

7 Concluding remarks **38**

A The assignment problem **43**
- A.1 The balanced assignment problem . . . 43
- A.2 The unbalanced assignment problem . . . 43

B The problem formulated as a quadratically constrained quadratic program **44**

C Formal proofs **46**
- C.1 Formal proof of Property 4.3 . . . 46
- C.2 Optimal starting time after merging for general processing times . . . 46
- C.3 Improving pairwise interchanges . . . 46

D Algorithms **47**
- D.1 The block merging algorithm for the integral scheduling problem with unit weights . . . 47
- D.2 The greedy heuristic for identical parallel machines . . . 47

Preface

This is the Master thesis "Scheduling under weighted squared deviation from due dates for unit sized jobs." by Camiel Koopmans. This report is written at Leiden University.

I want to thank Prof. Dr. T. Vredeveld and PhD candidate M.Y. Buchem for weekly meetings where we talked about this problem and developed the ideas that came to be in this thesis. During six full months this came to be one of the few certainties in an odd year where the corona epidemic was at hand. Furthermore I want to thank Prof. Dr. F.M. Spieksma for helping me write this thesis. After the good help for my bachelor thesis, I knew I would like her help on this thesis as well. Via several meetings we talked about the thesis and she helped me especially well with writing nice mathematics texts.

1 Introduction

For financial and environmental reasons, it is preferred to efficiently transport goods with a low fuel consumption. A paper by Buchem et al. [8], considers fuel consumption of vessels when passing locks for inland waterway transportation. The paper optimizes the fuel use of a given ship, in a stochastic setting. One can also consider to schedule the arrival of multiple vessels at a lock to minimize the total fuel use of vessels. Let d_j be the optimal time for vessel j to undock, and let the lock be scheduled such, that vessel j undocks at time C_j. It is mentioned that fuel use is cubic in the velocity of the vessel, and as a result, a deviation δ in the planned operation time of the lock results in an extra amount of fuel needed for the vessel, that is quadratic in δ. As fuel use differs between distinct vessels, the extra fuel use is scaled by a certain weight w_j for vessel j. As a result the total extra fuel necessary is $w_j(C_j - d_j)^2$. Minimizing the overall fuel use of all vessels is now equivalent to minimizing $\sum_j w_j(C_j - d_j)^2$. We assume that p_j, the time between docking and undocking at a lock, is a constant unit of time for any vessel j. Furthermore, we assume that the capacity of the lock is only a single vessel, implying that the values C_j need to differ by at least one.

We will formalise this problem as a scheduling problem, and present, exact solution approaches for special cases of this problem, and heuristic algorithms and an additive approximation scheme for the general problem.

Earlier research on the problem minimizing $\sum_j w_j(C_j - d_j)^2$ has not yet been focused on jobs with unit processing times. Many special cases of the problem, including the general problem itself, are easier to solve for jobs with unit processing times than for jobs with general processing times. As a result, we will find a lot of new results by restricting to jobs with unit processing times. For a lot of special cases we will find a polynomial solution method. Another distinction between this thesis and other research is that many papers only consider heuristics that give schedules without idle time between jobs (for example the following papers [21],[29],[35],[36],[37],[38]). By contrast we will research occurrences of idle time between jobs in optimal schedules in order to give an additive approximation scheme. Except when all the jobs share a common due date, idle time can occur between jobs in an optimal schedule. Therefore, we will also consider schedules with idle time between jobs.

Furthermore Tjark Vredeveld, Moritz Buchem and I are planning to write a paper on the scheduling problem researched in this thesis. This will include multiple special cases of this problem which we have managed to solve up to optimality and the found additive approximation scheme for the general problem. For one of the special cases we have found an exponential solution method. We will revisit this special case in the hope to find a polynomial solution method instead.

1.1 The problem of interest

We model this lock scheduling problem as a machine scheduling problem. We have n jobs, representing the vessels, that need to be scheduled on one machine, representing a lock. The time for the machine to process job j is the processing time p_j, which we assume to be $p_j = 1$, for each job j. Furthermore, each job is given a due date d_j, representing the preferred time of undocking, and a weight w_j, representing the proportional fuel use. We assume that all values are positive rationals, i.e., $w_j, d_j \in \mathbb{Q}_+$, for each job j.

A schedule σ can be specified by either the starting, or completion times of all jobs. A schedule σ is feasible when each job starts at or after time 0, and every machine processes at most one job at a time. The starting time S_j and completion time C_j of job j represent the time of docking and the time of undocking respectively, for all j. For a given feasible schedule σ, we refer to the objective value $\sum_j w_j(C_j - d_j)^2$ as the *cost of the schedule*. When a feasible schedule σ^* attains the minimal cost over all feasible schedules σ, the schedule σ^* is called an *optimal schedule*. We note that for any given instance, an optimal schedule exists, as stated by Theorem 3.5 in Section 3.2.1.

We observe, that the cost increases when a job is scheduled further from its due date. Likewise, it decreases when a job is scheduled closer to its due date. This is a fundamental basic property of our objective function and for Just-In-Time scheduling problems in general. These Just-In-Time scheduling problems give an incentive for jobs to be completed close to their due date. Reasons for this are to ensure fresh goods, prevent storage costs or to deliver to clients with a limited availability for example.

In general we consider the single machine problem. Whenever it is possible to extend our findings to m identical parallel machines, we will do this. These m machines then represent m identical locks in parallel.

1.1.1 Notation

We let D denote the set of different due dates, $D = \{d_1, \ldots, d_n\}$, and k to be the number of distinct due dates, $|D| = k$. The i-th distinct due date is also denoted by d^i, i.e., $D = \{d^1, \ldots, d^k\}$, where we assume that $d^1 < d^2 < \ldots < d^k$. When this is not the case, we can order the due dates in $\mathcal{O}(k \log k)$ time.

We consider multiple cases for our scheduling problem.

1.1.2 The integral scheduling problem

In this thesis, we will also study the restriction to so-called integral scheduling problems, in which the data and the starting/completion times are integers.

Definition 1.1. The *integral scheduling problem* is the above defined scheduling problem with the additional restriction to integral due dates and processing times, that is $d_j, w_j \in \mathbb{N}$ for jobs $j \in \{1, 2, \ldots, n\}$, including the requirement that starting times and completion times must be integral as well.

An *integral schedule* is a schedule σ that is feasible for the integral scheduling problem.

1.1.3 Constrained vs unconstrained problems

For a schedule to be feasible, one needs all starting times to be non-negative, that is, $S_j \geq 0$ for all jobs $j \in \{1, 2, \ldots, n\}$. It might occur that the constraints of non-negative starting times lead to a higher optimal cost than for the same problem without these constraints.

Definition 1.2. A scheduling problem is called *constrained* if the optimal cost of the problem relaxing the constraints of non-negative starting times is strictly lower than the optimal cost. Otherwise, the problem is called *unconstrained.*

Only for unconstrained problems, the optimal cost can be found as the cost of an optimal schedule for the problem without the constraint of non-negative starting times. As a result we find following corollary.

Corollary 1.3. A scheduling problem with optimal cost Z^* is *unconstrained* iff for for every $\epsilon > 0$, a translation of the due dates to $d_j + \epsilon$ for all jobs j, results in a problem with optimal cost equal to Z^* as well.

This definition follows as a problem is constrained, if and only if, there exists a schedule σ with a lower cost than Z^*, that becomes feasible by a large enough translation $\epsilon > 0$ on the starting times and due dates.

1.2 Related work

A wide variety of scheduling problems with cost for late and early jobs, has been researched. Baker and Scudder [5] review such problems.
The scheduling problem minimizing $\sum_j w_j|C_j - d_j|$ for general processing times is closely related to our problem of interest. For a common due date $d_j = d$, Hall and Posner [18] have shown that the recognition version of this problem is NP-complete, when the problem is unconstrained. Hall et al. [17] have shown that the problem is also NP-complete when constrained. Alidaee [1] proved, that if weights of jobs are proportional to their processing times, the problem minimizing $\sum_j w_j|C_j - d|$ is equivalent to the total weighted tardiness problem. This also shows that this problem is NP-hard.
Hall et al. [17] have further proved that the constrained problem minimizing $\sum_j |C_j - d|$ for general processing times is NP-complete as well, by giving a reduction from (the) odd-even partition. Independently, Hoogeveen and van de Velde [19] have given this reduction as well. Furthermore, they give a pseudopolynomial dynamic programming algorithm to solve the problem. Bagchi et al. [3] give an algorithm that minimizes $\sum_j |C_j - d|$ under restrictive assumptions. Likewise, Kim et al. [22] have developed an optimal

algorithm and multiple heuristic algorithms. Computational tests indicate that optimal solutions can be found for problems with up to 20 jobs, and that two of the heuristic procedures provide optimal or very near optimal solutions in many instances.

For earliness $E_j = \max\{0, d - C_j\}$ of job j, and tardiness $T_j = \max\{0, C_j - d\}$ of job j, an asymmetric objective function $\sum_j(\alpha_j E_j + \beta_j T_j)$ has been studied. Saravanan and Nooral Haq [32] give a heuristic based on scatter search to find good solutions of this problem. For unit sized jobs and a common due date, the unconstrained problem can be solved in $\mathcal{O}(n^3)$ time. This applies to identical parallel machines as well, as found by Mosheiov and Yovel [25]. Huynh Tuong et al. [20] showed that the unconstrained problem is also solvable in $\mathcal{O}(n^3)$ time for a single machine. The objective function $\sum_j \alpha_j E_j + \beta_j T_j$ can be minimized for unit sized jobs and a restricted set of due dates D with $|D| = k$, in $\mathcal{O}(k^{k+1} n^{k+2})$ time, for a single machine, see the same paper by Huynh Tuong et al. [20]. Panwalker et al. [27] have shown that the objective function $\sum_j \alpha E_j + \beta T_j$ can be minimized in $\mathcal{O}(n \log n)$ time, for a single due date.

By considering the objective function $\sum_j(\alpha_j E_j^2 + \beta_j T_j^2)$, we obtain an asymmetric version of our problem. For a common due date $d_j = d$, and weights $\alpha_j = \alpha,\ \beta_j = \beta$, Bagchi et al.[2] have given an exponential procedure to solve the problem.

Bagchi et al. [4], also studied the problem with unit weights and a single due date, that is the problem with objective function $\sum_j(C_j - d)^2$. They have suggested heuristic algorithms for this problem. Furthermore, they have proved that the problem is equivalent to the Completion Time Variance problem. This Completion Time Variance problem was later shown to be NP-hard by Kubiak [23].
In the single machine environment, this problem has been well studied in the literature, with many found heuristics to find good solutions, for example by Ventura and Weng [40].
Several more papers give heuristic algorithms for the completion time variance problem in the identical parallel machine environment. For example the papers by Brundavanam et al. [7], by Federgruen and Mosheiov [11] and by Brundavanam and Srirangacharyulu [6]. The weighted variant has been studied by Nessa and Chu [26], for a single machine. Sun et al. [34] study this problem in the identical parallel machine environment.

Many heuristics and exponential algorithms have been found for the weighted mean squared deviation problem minimizing $\sum_j w_j(C_j - d_j)^2$ on a single machine. In general these solution methods only consider schedules without idle time between jobs. Kianfar and Moslehi [21] give a branch and bound algorithm. Pereira and Vásquez [29] study upper and lower bounds of (sub)problems and follow by giving multiple heuristic procedures. The following papers give heuristics as well [35],[36],[37],[38].

1.3 The Research

In Section 2, we consider the complexity of the lock scheduling problem. In the further literature we have found problems that are alike and are NP-complete. For the problem with general processing times, we give reductions from the Partition problem in Section 2.1 and from 3-Partition in Section 2.2. This shows NP-completeness of the more general problem. For unit processing times it is unknown whether the problem is polynomially solvable.
In Section 3.1, we consider structural properties of optimal schedules and introduce the concept of a block structure. In Section 3.2 we find a solution method to find optimal schedules for a given sequence. Two different methods are given, a method using quadratic programming in Section 3.2.1, and the so-called block merging algorithm we give in Section 3.2.2. As a consequence we know that an optimal schedule must exist. Furthermore, the findings in Section 3.2.2, give that all optimal schedules must have rational starting, and completion times. Then, in Section 3.3, we explain how the assignment problem can be used to solve the scheduling problem, when restricting to a fixed set of starting and completion times.
In Section 4, we consider the problem for jobs with unit weight. We start by giving structural properties of optimal schedules in Section 4.1. As a result we can find an optimal integral schedule as explained in Section 4.2. Then, in Section 4.3, we explain how the block merging algorithm for sequences of jobs, can be used for unit weight problems. This gives a solution method with low running time. In Section 4.4, we consider the unit weight problem in the identical parallel machine environment. By showing an optimal schedule exists for a certain distribution of jobs to the m machines, the problem can be solved as m single machine problems. Furthermore, the found solution methods are applicable to the integral scheduling problem with unit weights.
In Section 5, we consider the problem, for a single common due date d, for all jobs. For a large enough

due date, the problem is known to be unconstrained. This gives more freedom in constructing optimal schedules. We give an effective method to solve both unconstrained single machine problems, as explained in Section 5.1.1, and constrained single machine problems, as explained in Section 5.1.2. In the identical parallel machine environment, unconstrained problems can be solved efficiently as explained in Section 5.2.1. In Section 5.2.2, we give a method that is exponential in m to find an optimal schedule for constrained problems.
In Section 6, the general problem is studied. In Section 6.1 we define, and study, locally optimal solutions. This gives a heuristic for the general problem with a potentially very high running time. Locally optimal solutions are, however, not necessarily optimal. In Section 6.2, a greedy polynomial heuristic algorithm is suggested. Thus a good solution is expected to be found. Again, the found schedules are not necessarily optimal. Then, we study an additive approximation scheme in Section 6.3. To achieve this we first study the positions and lengths of idle periods in optimal schedules. The additive approximation scheme considers different schedules by fixing the starting time of the schedule, the length of the blocks of consecutive jobs, and the length of idle periods between these blocks. For each schedule, we solve the assignment problem between jobs and positions in the schedule. Then we pick the schedule with the lowest cost. Finally, this chosen schedule can be improved by finding the optimal schedule for its job sequence.
In Section 7, we revisit all special cases we considered and the found results. For the general problem we consider the complexity of the heuristics and additive approximation scheme and whether we have a performance guarantee. For both heuristics we shortly remark how they can be extended to the identical parallel machine problem. Finally we state a number of possible questions for further research.

2 Complexity

The question whether the considered problem is solvable in polynomial time is open. From the literature, it is not known whether the problem minimizing $\sum_j w_j(C_j - d_j)^2$ is polynomial solvable when the jobs have unit processing times.

From Bagchi et al. [4] and Kubiak [23], we know that for general processing times, a single common due date, and unit weights, the problem minimizing $\sum_j (C_j - d)^2$, is NP-hard.

The problem is much simpler for unit processing times. In Section 4.3 we show that we can solve the problem with unit processing times and unit weights, in $\mathcal{O}(k)$ time for k different ordered due dates. When the weights are general, the problem becomes harder to solve. We still don't know whether the problem is in P or in NP.

For completeness, we do include two simple reductions to the problem minimizing $\sum_j w_j(C_j - d_j)$, for general processing times.

2.1 Reduction from the partition problem

Theorem 2.1. *The scheduling problem with general processing times is NP-complete.*

Proof. First we consider the Partition problem, which is NP-complete.
Given a set of integers $S = \{a_1, a_2, \ldots, a_n\}$, we seek a partition of S into S_1 and S_2 such that $\sum_{a \in S_1} a = \sum_{a \in S_2} a = A$.

We will reduce this problem to a problem deciding whether a single machine schedule exists with cost $\sum_j w_j(C_j - d_j)^2 \leq nA^2$.

We take n jobs $j_1, j_2, \ldots, j_n$ with unit weight, due date $A + 1$, and processing times $p_1 = a_1,\ p_2 = a_2, \ldots, p_n = a_n$. Then we introduce job j_{n+1} with unit processing time, due date $A + 1$ and weight $W = 8nA^2$. Finally we take job j_{n+2} with processing time $P = nA$, due date $2A + P + 1$ and weight W.

When a solution for the partition problem exists, we can take such a solution and schedule the jobs in $P_1 = \{j_i : a_i \in S_1\}$ before job j_{n+1} and the jobs in $P_2 = \{j_i : a_i \in S_2\}$ between job j_{n+1} and job j_{n+2}, resulting in the schedule portrayed in Figure 1.

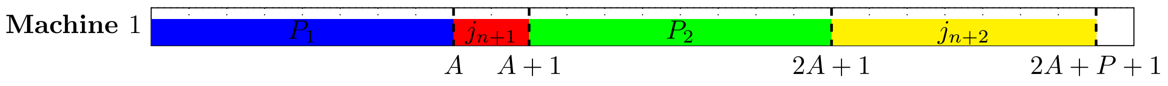

Figure 1: A feasible schedule for the given partition

Here job j_{n+1} and job j_{n+2} are scheduled at their due date. Also $(C_j - d_j)^2 \leq A^2$ for any job. Thus this schedule has a cost of $\sum_j w_j(C_j - d_j)^2 \leq nA^2$.

We now consider that no solution for the partition problem exists.
Assume that a schedule σ exists with cost of at most nA^2.

Jobs j_{n+1} and j_{n+2} can't differ from the due date by $\frac{1}{2}$ or more, as $W \cdot (\frac{1}{2})^2 = 2nA^2 > nA^2$. The total processing time of the jobs scheduled before job j_{n+1} can't be A as no solution of the partition problem exists, it also can't be more than A as job j_{n+1} is then scheduled at least 1 later than its due date. No job j can be scheduled after job j_{n+2} as job j would then have cost

$$w_j(C_j - d_j)^2 \geq w_j(A + P)^2 \geq P^2 = n^2A^2 > nA^2.$$

Thus the total processing time between job j_{n+1} and job j_{n+2} must be at least $A+1$. As a result either job j_{n+1} or job j_{n+2} differs by at least $\frac{1}{2}$ from its due date. This contradicts that schedule σ has cost of at most nA^2.

We may conclude that a schedule with cost of at most nA^2 exists, if and only if there exists a solution to the partition problem.
This completes the reduction from the NP-complete partition problem to the problem whether a single machine schedule exists with cost $\sum_j w_j(C_j - d_j)^2 \leq nA^2$. □

There does, however, exist a pseudopolynomial algorithm to solve the partition problem. This is not possible for problems that are strongly NP-complete, unless P=NP [14].

2.2 Reduction from the 3-partition problem

Theorem 2.2. *The scheduling problem with general processing times is strongly NP-complete.*

Proof. We consider the 3-partition problem which is strongly NP-complete.
Given a set of integers $S = \{a_1, a_2, \dots, a_{3m}\}$ with $\sum_{i=1}^{3m} a_i = mA$, the goal is to find a partition of S into $S_1, S_2, \dots S_m$ such that $\sum_{a \in S_i} a = A$ for $i \in \{1, 2, \dots, m\}$.

We now will reduce this problem to a problem whether a single machine schedule exists with cost $\sum_j w_j(C_j - d_j)^2 \leq 3m^2(A+1)^2$.

We consider jobs $j_1, j_2, \dots j_{3m}$ to have unit weight, due date 0 and processing times
$p_1 = a_1,\ p_2 = a_2, \dots,\ p_{3m} = a_{3m}$. Then we introduce jobs j_{3m+i} with $i \in \{1, 2, \dots, m-1\}$, to have unit processing time, due date $k(A+1)$ and weight $W = 20m^3(A+1)^2$. Finally job j_{4m} has processing time $P = 2m^2(A+1)$, due date $m(A+1) + P - 1$ and weight W.

Assume a solution to the 3-partition problem to exist. Considering such a solution, we can schedule the jobs in $P_i = \{j_\ell : a_\ell \in S_i\}$ directly before job j_{3m+i} for $i \in \{1, 2, \dots, m\}$. The resulting schedule is portrayed in Figure 2.

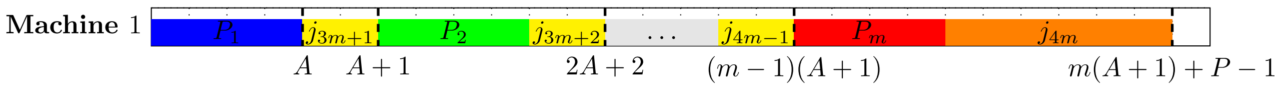

Figure 2: A feasible schedule for the given 3-partition

The jobs j_{3m+i} are completed at their due dates for $i \in \{1, 2, \dots, m\}$. An upperbound for the cost of this schedule can be given by $\sum_j w_j(C_j - d_j)^2 = \sum_{i=1}^{3m}(C_j - d_j)^2 \leq 3m \cdot (m(A+1) - 1)^2 \leq 3m^3(A+1)^2$.

We now consider that no solution to the 3-partition problem exists.
Assume that a schedule σ exists, with cost of at most $3m^3(A+1)^2$.
As $W \cdot (\frac{1}{2})^2 = 5m^3(A+1)^2 > 3m^3(A+1)^2$, all of the jobs $j_{3m}, j_{3m+1}, \dots, j_{4m}$ must be completed with less than $\frac{1}{2}$ distance from their due date. When no job with unit weight finishes after job j_{4m}, the total processing time between job j_{3m+i} and j_{3m+i+1} must be at most equal to the integer A, for $i \in \{1, 2, \dots, m-1\}$. There does not exist a partition of jobs $j_1, j_2, \dots j_{3m}$ in m sets, such that for each set, the total processing time of the jobs in that set is A. Thus, a job j with unit weight must be scheduled after job j_{4m}. The cost of this job j is

$$w_j(C_j - 0)^2 \geq P^2 = 4m^4(A+1)^2 > 3m^3(A+1)^2.$$

This contradicts the existence of σ.

We can conclude that a schedule σ exists with cost of at most $3m^3(A+1)^2$, if and only if there exists a solution to the 3-partition problem.

This completes the reduction from the strongly NP-complete 3-partition problem to the problem whether a single machine schedule exists with cost $\sum_j w_j(C_j - d_j)^2 \leq 3m^2(A+1)^2$. □

3 Structural properties and solution methods for fixed order or fixed starting times

In this section we will first give some structural properties of optimal schedules for the general problem and introduce the concept of a block structure.
Next, we will study optimal schedules, on a single machine, for a given job sequence. For a given job sequence, quadratic programming can be used to find an optimal schedule. Another result using the quadratic programming formulation, is that an optimal schedule must exist.
Then we will investigate so called block structures in our problem. We derive optimal starting times of blocks, and, as a consequence, we show that all starting times in an optimal schedule must be rational. These findings lead to an alternative algorithm, that gives an optimal schedule for a fixed job sequence as well. The algorithm finds the block structures, and their starting times, thus determining the optimal schedule. With a running time of $\mathcal{O}(n)$, this algorithm is more efficient than quadratic programming, moreover, using the structure of the scheduling problem it gives more insight in optimal schedules.
Finally, in the identical parallel machine environment, we will show how to find an optimal schedule for fixed starting and completion times. For this we explain that the problem can be solved as an assignment problem. Background knowledge on assignment problems is provided in Appendix A.

3.1 Structural properties of optimal schedules

We first recollect that scheduling a job closer to its due date results in lower cost. This has the following simple implication for the single machine environment and the identical parallel machine environment.

Property 3.1. Let an optimal schedule σ^*, be given. No job j with $C_j < d_j$ is followed by idle time. Furthermore, no job j with $C_j > d_j$ is preceded by idle time.

For the single machine environment, we consider two jobs j, j', with equal due dates $d_j = d_{j'}$. Consider job j to be scheduled directly before job j'. Then job j is completed before d_j, or job j' must be scheduled after the common due date d_j. Then Property 3.1 implies the following.

Property 3.2. In any optimal schedule σ^*, for every pair of consecutively scheduled jobs j, j' with equal due date $d_j = d_{j'}$, there will be no idle time between the processing of these two jobs.

Likewise, we will find and prove Property 6.4 in Section 6.3.1, stating that in any optimal schedule σ^*, between any consecutive due dates $d^\ell, d^{\ell+1}$, there can only be one single interval of idle time.

For jobs j, j' with a common due date, we now show that in an optimal schedule σ^*, the job with highest weight must be scheduled with completion time closest to the common due date. If this is not the case, interchanging jobs j and j' improves the schedule. As every job has a unit processing time, interchanges don't give conflicts with other scheduled jobs.

Property 3.3. Let a pair of jobs j, j' be given with equal due date $d_j = d_{j'}$. In any optimal schedule σ^*

$$w_j > w_{j'} \implies |C_j - d_j| \leq |C_{j'} - d_{j'}|.$$

For objective functions that are increasing in the completion times of the jobs (regular objective functions), an optimal schedule for a single machine may be determined by giving a job sequence. The machine will start processing the first job in the sequence and keep processing until the last job has been processed, as idle time can't reduce the cost. To demonstrate, job sequence $[j_1, j_2, \ldots, j_n]$ specifies that the machine processes job j_1 first, then job j_2 and so on, until job j_n, which is the last job to be processed.
For the objective function we study, idle time may in fact be beneficial. As it will be useful to study sequences of jobs without any idle time in between consecutive jobs, we introduce the following structure.

Definition 3.4. A *block* of jobs, is a (maximal) sequence of jobs that are processed without intermediate idle time.

We will specify, whenever a job sequence $[j_1, j_2, \dots, j_n]$ is in fact a block of jobs.

3.2 The problem for a fixed job sequence

We now consider how to schedule jobs optimally, given a job sequence. That is to give the schedule with lowest cost for which the jobs are processed in the order specified by a certain job sequence $JS = [j_1, j_2, \dots, j_n]$. We show that an optimal schedule can be found using quadratic programming. We show that this implies the existence of an optimal schedule. Then we also show that we can use the structural properties of optimal schedules to find an alternative algorithm that will give an optimal schedule as well. Furthermore the exploited structural properties of optimal schedules also give that every optimal schedule must have rational starting and completion times.

3.2.1 Quadratic programming for a given job sequence

First, we note that the general scheduling problem can be formulated as a quadratically constraint quadratic program as is shown in Appendix B. In Appendix B, we also show that this does not provide us with efficient techniques to solve the scheduling problem in general. In this section we show that in the specific case of a given job sequence, quadratic programming can be used to find an optimal schedule.

Consider schedules that follow the order of jobs being processed as specified by job sequence JS. For such a schedule to be feasible, we need non-negative starting times and at most one job to be processed at the same time.
The constraint $S_{j_1} \geq 0 \iff C_{j_1} \geq 1$ specifies that the first job to be processed has a non-negative starting time and thus all jobs will have non-negative starting times.
Furthermore we have the constraints that no two jobs can be processed simultaneously. Thus all completion times must differ by at least 1. This is the case when the completion times of every pair of consecutive jobs differ by at least 1. For the given order of processing, this leads to the following constraints: $C_{j_{a-1}} \leq C_{j_a} - 1$ or $C_{j_{a-1}} - C_{j_a} + 1 \leq 0$ for $a \in \{2, 3, \dots, n\}$.

This leads to a Quadratic Program with n linear constraints. Notice that in the literature Quadratic Programming (QP) problems are always assumed to have only linear constraints [39]. This leads to a problem of the following form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & x^T H x + c^T x \\ \text{s.t.} \quad & Ax \geq b, \quad \text{for } i = 1, 2, \dots, m, \end{aligned} \tag{3.1}$$

where H is a symmetric $n \times n$ matrix, A is an $m \times n$ matrix, b is an m-vector and c is an n-vector.

We let

$$x = \begin{pmatrix} C_{j_1} - d_{j_1} \\ C_{j_2} - d_{j_2} \\ \vdots \\ C_{j_n} - d_{j_n} \end{pmatrix},$$

and the objective is then to minimize

$$\sum_j w_j (C_j - d_j)^2 = \begin{pmatrix} C_{j_1} - d_{j_1}, & C_{j_2} - d_{j_2}, & \dots & C_{j_n} - d_{j_n} \end{pmatrix} \begin{pmatrix} w_{j_1} & 0 & \dots & 0 \\ 0 & w_{j_2} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & w_{j_n} \end{pmatrix} \begin{pmatrix} C_{j_1} - d_{j_1} \\ C_{j_2} - d_{j_2} \\ \vdots \\ C_{j_n} - d_{j_n} \end{pmatrix}.$$

This is equivalent to minimizing $x^T H x + c^T x$ with $c = 0$ and

$$H = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & w_n \end{pmatrix}.$$

The non-negative starting time constraint takes the form

$$C_{j_1} \geq 1 \iff C_{j_1} - d_{j_1} \geq 1 + d_{j_1} \iff x_1 \geq 1 + d_{j_1}.$$

The other $n-1$ constraints take the form

$$\begin{aligned} C_{j_{a-1}} \leq C_{j_a} - 1 &\iff C_{j_a} - C_{j_{a-1}} \geq 1 \iff C_{j_a} - d_{j_a} - C_{j_{a-1}} + d_{j_{a-1}} \geq 1 - d_{j_a} + d_{j_{a-1}} \\ &\iff x_a - x_{a-1} \geq 1 - d_{j_a} + d_{j_{a-1}}. \end{aligned}$$

These constraints can be expressed in the form (3.1) by constraint matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ -1 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & 0 & \dots & -1 & 1 \end{pmatrix},$$

and constraint vector

$$b = \begin{pmatrix} 1 + d_{j_1} \\ 1 - d_{j_2} + d_{j_1} \\ 1 - d_{j_3} + d_{j_2} \\ \vdots \\ 1 - d_{j_n} + d_{j_{n-1}} \end{pmatrix}.$$

Therefore, the scheduling problem, for a given job sequence, is indeed a Quadratic Programming problem. As matrix H is positive definite, the objective function of our problem is convex. Furthermore the set $\{x \mid Ax \geq b\}$ of feasible solutions is a convex polyhedral set. This leads to our QP problem being convex.

For QP problems, Frank and Wolfe [12] have shown that an optimal solution exists, when the set of feasible solutions and this set is non-empty and the objective function is bounded from below on this set. In our case, for any job sequence $JS = [j_1, j_2, \dots, j_n]$, we have a feasible schedule given by $C_{j_a} = a$ for $a \in \{1, 2, \dots, n\}$, and thus the set of feasible solutions is non-empty. Secondly, the objective function $\sum_j w_j (C_j - d_j)^2$ is bounded from below by 0 on the set of feasible solutions, because all summands are non-negative. Thus, we may conclude that for every job sequence, there exists an optimal schedule.
This leads to a fundamental property of our problem.

Theorem 3.5. *For every instance of our general scheduling problem there exists an optimal schedule.*

Proof. For every given job sequence there exists an optimal feasible schedule minimizing the objective function. There is a finite number of job sequences, $n!$ to be exact. Then a job sequence, for which the optimal schedule has minimal cost, must exist. This optimal solution then specifies an optimal schedule of the general problem. □

We note that this theorem still holds, when considering the identical parallel machine environment and/or when we relax the job characteristic of unit processing times. This follows from the fact that the constraints are linear, also without unit processing times. In the identical parallel machine environment, there is a finite number of ways to assign the n jobs to the m machines. Given such an assignment, the problem is equivalent to solving the now m specified single machine problems. An optimal schedule, for this assignment of jobs to machines, is a combination of m optimal schedules of the m single machine problems. Again we can conclude that the best of these finite number of optimal schedules, given an assignment of jobs to machines, is an optimal schedule for the identical parallel machine problem.

To solve convex QP problems, the algorithm with the best known running time (as of 2001 [39]), has running time $\mathcal{O}(m^{\frac{7}{2}}L)$. This is an interior point algorithm by Renegar [31]. In the running time, the factor m denotes the number of constraints, and the factor L in the running time of this algorithm denotes the number of digits in the input (H, A, b, c).
We note that, for the QP problem we consider, the number of constraints m is equal to the number of

jobs n. Thus the running time of the algorithm by Renegar is $\mathcal{O}(n^{\frac{7}{2}}L)$.

When the number of arithmetic operations of an algorithm is polynomial in m and n, this algorithm is said to be strongly polynomial [16]. When the number of arithmetic operations of an algorithm is not polynomial in m and n but is polynomial in m, n and L, this algorithm is said to be weakly polynomial [16]. Therefore, the algorithm of Renegar is weakly polynomial. As a consequence we have a weakly polynomial algorithm to find an optimal schedule, given a job sequence.
It is still an open problem to decide whether a strongly polynomial algorithm exists to solve Quadratic Programs, and even Linear Programs [33].

The importance of the distinction between strongly, and weakly polynomial algorithms becomes clear in the following quote [41]: "The existence of a strongly polynomial-time algorithm for linear programming is a cross century international mathematical problem, whose breakthrough will solve a major theoretical crisis for the development of artificial intelligence."
Thus, we prefer a strongly polynomial algorithm over the weakly polynomial algorithm by Renegar. To achieve this we must further investigate block structures in optimal schedules.

3.2.2 An alternative algorithm to schedule a job sequence optimally

We will exploit the block structures within job sequence JS for optimal schedules. This will give an alternative algorithm to find an optimal schedule given job sequence JS. The advantages of this algorithm will be its low running time and the fact that it is strongly polynomial.

Optimal starting time of a block of jobs

We will give the optimal starting time of any block of jobs. These optimal starting times will be rational, resulting in the fact that all jobs in an optimal schedule have rational starting and completion times.

We consider one fixed block $[j_1, j_2, \ldots, j_\ell]$ of $\ell > 0$ jobs. We will first give the optimal starting time $\hat{t}$ of this block, without the non-negativity constraint $\hat{t} \geq 0$. This is not a feasible starting time of the block but this value will be needed in later calculations like Equation (3.4).

Let t be the starting time of j_1, and thus the starting time of this block. This block has cost:

$$\sum_{a=1}^{\ell} w_{j_a}(t + a - d_{j_a})^2.$$

This is a quadratic function in t and the minimum is obtained by setting the derivative to zero, which yields

$$\begin{aligned} \frac{d}{dt}\sum_{a=1}^{\ell} w_{j_a}(t + a - d_{j_a})^2 &= \sum_{a=1}^{\ell} 2w_{j_a}(t + a - d_{j_a}) = 0 \implies \\ t\sum_{a=1}^{\ell} w_{j_a} &= \sum_{a=1}^{\ell} w_{j_a}(d_{j_a} - a). \end{aligned}$$

And thus the optimal starting time $\hat{t}$ is attained for

$$\hat{t} = \frac{\sum_{a=1}^{\ell} w_{j_a}(d_{j_a} - a)}{\sum_{a=1}^{\ell} w_{j_a}}. \tag{3.2}$$

For general processing times, we note that the value a needs to be replaced with $\Pi_a = \sum_{i=1}^{a} p_{j_i}$. This yields the following equation.

$$\hat{t}_{general} = \frac{\sum_{a=1}^{\ell} w_{j_a}(d_{j_a} - \Pi_a)}{\sum_{a=1}^{\ell} w_{j_a}}. \tag{3.3}$$

Note that $\hat{t}$ must be rational, as the weights, due dates and values of a are rational as well.

The quadratic function is increasing after the optimum $\hat{t}$. Thus, when including the constraint of a non-negative starting time of the block block, this gives the optimal feasible starting time $t^* = \max\{0, \hat{t}\}$.

Furthermore, as an optimal schedule consists of multiple blocks at their optimal rational starting time, this has the consequence that an optimal schedule on a single machine must be rational.
In the identical parallel machine environment, all m single machine schedules must be optimal. As a result all jobs have rational starting and completion times.

Proposition 3.6. Any optimal schedule σ^* of the scheduling problem, in the identical parallel machine environment, has only rational starting times and completion times.

We will try to find the subsequences of JS that form a block. Thus an optimal schedule is found by scheduling these blocks optimally. At first we don't know which subsequences of JS must form blocks to be optimally scheduled, therefore we start by assuming that every job is a block by itself. For all of the blocks we have an optimal starting time, being $\max\{0, d_j - 1\}$ for the block only consisting of job j. When possible we schedule every block at its optimal starting time. If some of the block-starting times are *conflicting*, i.e., block $\ell + 1$ would start before block ℓ is completed, this choice of starting times does not give a feasible schedule.

We will show that in this case, in an optimal schedule, these conflicts can be resolved by *merging* two blocks such that they form one larger block for which the optimal starting time can be found. Repeating this procedure results in a schedule in which there are no conflicting blocks. Then a feasible optimal schedule is found. We show how this can be done efficiently in $\mathcal{O}(n)$ time.

Merging blocks of jobs together

We consider block $A = [j_1, j_2, \ldots, j_{n_a}]$ and block $B = [j_{n_a+1}, j_{n_a+2}, \ldots, j_{n_a+n_b}]$, consecutively scheduled in an optimal schedule, such that block A is scheduled first. When block A has optimal starting time t_a^* and block B has optimal starting time $t_b^* < t_a^* + n_a$, we know by convexity that no idle time exists between these blocks in an optimal schedule. Thus we can merge block A and block B resulting in the new block $C = [j_1, j_2, \ldots, j_{n_a+n_b}]$ of jobs that are scheduled together in an optimal schedule. This leads to the following theorem.

Theorem 3.7. *Let block A and block B be scheduled consecutively, in that order, in an optimal schedule consisting of n_a and n_b jobs respectively. When these blocks have optimal starting times t_a^* and t_b^* respectively and $t_b^* \leq t_b^* + n_a$, block A and block B must be scheduled without idle time in between.*

In other words, this theorem states that, the merged block C must be processed as such in an optimal schedule.

We will now show, how to find the optimal starting time of block C after merging blocks A and B.
When $t_a^* = 0$, the new optimal starting time of block C is $t_c^* = 0$ as well. Otherwise the new optimal starting time t_c^* can be constructed from the (possibly negative) optimal starting times $\hat{t}_a, \hat{t}_b$ and number of jobs n_a, n_b for block A and block B respectively. The total weight of a blocks A and B will be denoted by W_a and W_b respectively. The total number of jobs in block C is $n_a + n_b$.
As in Equation (3.2) we can derive the optimal starting time $\hat{t}_c$ as follows

$$\begin{aligned}
\hat{t}_c &= \frac{\sum_{i=1}^{n_a+n_b} w_{j_i}(d_{j_i} - i)}{W_a + W_b} \\
&= \frac{W_a}{W_a + W_b} \cdot \frac{\sum_{i=1}^{n_a} w_{j_i}(d_{j_i} - i)}{W_a} + \frac{W_b}{W_a + W_b} \cdot \frac{\sum_{i=n_a+1}^{n_a+n_b} w_{j_i}(d_{j_i} - i)}{W_b} \\
&= \frac{W_a}{W_a + W_b} \cdot \hat{t}_a + \frac{W_b}{W_a + W_b} \cdot \frac{\sum_{i=1}^{n_b} w_{j_{i+n_a}}(d_{j_{i+n_a}} - (i + n_a))}{W_b} \\
&= \frac{W_a}{W_a + W_b} \cdot \hat{t}_a + \frac{W_b}{W_a + W_b} \cdot (\hat{t}_b - n_a).
\end{aligned} \tag{3.4}$$

This is a weighted average of $\hat{t}_a$ and the translated $\hat{t}_b$, where this translation follows as, after merging, block B starts n_a time units later than block A.

For general processing times, the optimal starting time $(\hat{t_c})_{general}$ after merging is similar. It is given by

$$(\hat{t_c})_{general} = \frac{W_a}{W_a + W_b} \cdot (\hat{t_a})_{general} + \frac{W_b}{W_a + W_b} \cdot ((\hat{t_b})_{general} - \Pi_{n_a}), \tag{3.5}$$

here Π_{n_a} is the total processing time of block A. For completeness, the derivation is included in Appendix C.2.

The block merging algorithm

We now show that these findings lead to an algorithm for an optimal schedule given job sequence $JS = [j_1, j_2, \ldots, j_n]$.

As explained, we start with the n blocks $B_1, B_2, \ldots, B_n$ of jobs where block B_ℓ only consists of job j_ℓ. We find the n optimal (possibly negative) starting times $\hat{t}_\ell = d_{j_\ell} - 1$ for each block B_ℓ. Furthermore we keep track of the total weight W_ℓ and the number of jobs n_ℓ of the blocks.
We will consider the optimal schedule σ^*_ℓ, for subsequence $JS_\ell = [j_1, j_2, \ldots, j_\ell]$ only, for $\ell = 1, \ldots n$. We start with schedule σ^*_1, only scheduling job j_1 with starting time $\max\{0, d_{j_1} - 1\}$. Iteratively we expand the subsequence to schedule until having found optimal schedule $\sigma^*_n = \sigma^*$, given job sequence JS. Having found schedule σ^*_ℓ, we include the block consisting of job $j_{\ell+1}$ and try to schedule it as well. The block we are pursuing to schedule is called the *active* block. This active block is possibly merged multiple times until it does not conflict anymore with blocks of lower index, and is then scheduled at its optimal starting time becoming inactive. We note that when a block merges with the active block, the new block may conflict with an earlier block. After scheduling the active block the optimal schedule $\sigma^*_{\ell+1}$ is found.

This leads to the following algorithm.

Algorithm 1: The block merging algorithm for sequences of jobs

```
Initialize the n blocks $B_1, B_2, \ldots, B_n$ with as data the optimal starting times
  $\hat{t}_\ell = d_{j_\ell - 1}, t^*_\ell = \max\{0, d_\ell - 1\}$, and weights $W_\ell = w_{j_\ell}$. Every block consists of 1 job.
Schedule block $B_1$ at its optimal starting time;
for $\ell = 2, 3, \ldots, n$ do
    $ActiveBlock = B_\ell$;
    while Optimal starting time of ActiveBlock conflicts with the block directly before it in job
      sequence JS do
        Merge ActiveBlock with the conflicting block creating NewBlock;
        Update data of NewBlock and find its optimal starting time using Equation (3.4);
        $ActiveBlock = NewBlock$;
    end
    Schedule block containing block $B_\ell$ at its optimal non-negative starting time;
end
```

The initialization of the original n blocks takes $\mathcal{O}(n)$ time. We enter the for loop $n-1$ times. The while-loop can be entered multiple times in the same iteration of the for-loop. In a given iteration the while-loop can be entered at most $\ell - 1$ times. Over all iterations of the for-loop, however, the while-loop can be entered only a total of at most $n-1$ times, as after $n-1$ iterations we merged blocks a total of $n-1$ times. When this has happened, the schedule would consist of a single block and is not able to merge with other blocks. Including a new block or merging two blocks takes $\mathcal{O}(1)$ time of updating data, this leads to the algorithm having running time $\mathcal{O}(n)$.

This algorithm can be applied to instances with general processing times as well. As changes, we need to keep track of the total processing times of the blocks. The optimal starting times after merging then result from Equation (3.5) instead. These changes have no effect on the time complexity of the algorithm.

Theorem 3.8. *An optimal schedule σ^*, given a fixed job sequence, can be found in $\mathcal{O}(n)$ time.*

Thus we have found a strongly polynomial algorithm to schedule a given job sequence optimally. Also the complexity of Algorithm 1 is lower than the complexity for the quadratic programming formulation. These are two reasons that we will use Algorithm 1 instead of quadratic programming to find an optimal schedule given a job sequence.

We can adapt Algorithm 1 to solve the integral scheduling problem.

The block merging algorithm for the integral scheduling problem

Again we start with n ordered blocks to be scheduled and their (integral) optimal completion times. Consider an active block. Consider the block directly before it in job sequence JS, to have completion time $z \in \mathbb{N}$, and the active block to have optimal starting time $t^* \in \mathbb{Q}_+$. By integrality and convexity the optimal feasible starting time of the active block is either $\lfloor t^* \rfloor$ or $\lceil t^* \rceil$. When $t^* \leq z$ we know that the optimal feasible starting time of the active block is at most z and we merge the active block with the block directly before it in job sequence JS. When $t^* > z$ we know that the optimal feasible starting time of the active block is at least z and we will not (yet) merge the two blocks.
Then we choose the starting time $\lfloor t^* \rfloor$ or $\lceil t^* \rceil$, that results in the least cost. As the quadratic cost of a block is symmetric around its optimum, the value of $\lfloor t^* \rfloor$ or $\lceil t^* \rceil$ closest to t^* is the optimal integral starting time. The resulting block merging algorithm for the integral scheduling problem is given below.

Algorithm 2: The block merging algorithm for the integral scheduling problem

```
Initialize the $n$ blocks $B_1, B_2, \ldots, B_n$ with as data the optimal starting times
  $\hat{t}_\ell = d_{j_\ell} - 1 \in \mathbb{Z}, t^*_\ell = \max\{0, d_\ell - 1\} \in \mathbb{Z}$, and weight $w_\ell = w_{j_\ell}$. Every block consists of 1 job.
Schedule block $B_1$ at its optimal starting time;
for $\ell = 2, 3, \ldots, n$ do
    $ActiveBlock = B_\ell$;
    while Optimal (rational) starting time of ActiveBlock conflicts with the block directly before
      it in job sequence $JS$ do
        Merge ActiveBlock with the conflicting block creating NewBlock;
        Update data of NewBlock and find its optimal (rational) starting time using
          Equation (3.4);
        $ActiveBlock = NewBlock$;
    end
    Schedule ActiveBlock at its optimal non-negative integral starting time;
end
```

Likewise this algorithm has time complexity $\mathcal{O}(n)$ as well.

3.3 Fixed starting and completion times

Consider the general scheduling problem on identical parallel machines.
Assume that we restrict the problem to the case that for each machine i a set of completion times T_i is known at which the jobs are allowed to complete. We will show that when there are in total at least n available completion times, the scheduling problem reduces to an assignment problem. We will then give the complexity in which we can solve these reduced problems. Such an application of the assignment problem for job scheduling has been used earlier [20].

We consider the sets of allowed completion times T_i for machine i. These sets can arise, for instance, by integrality constraints. W.l.o.g. we may assume these sets to be finite as, of all completion times smaller than the earliest due date, the largest n completion times need to be taken into account. Likewise only the n smallest completion times, larger than the largest due date need to be considered.
We take the union $T = \bigcup_i \{(t, i) : \ t \in T_i\}$ as set of completion times to assign to the n jobs. Then $a = (t, i) \in T$ corresponds to specified completion time t on machine i. To be able to process each job we need $|T| \geq n$.

We consider a schedule σ. Then we define job j_{ti} as the job that finishes at time t on machine i, and $j_{ti} = \times$ if no job is completed at time t on machine i. Then the cost of the schedule σ is given by $\sum_{(t,i)\in T} w_{j_{ti}}(t - d_{j_{ti}})^2$, where $w_{j_{ti}} = 0$ whenever $j_{ti} = \times$.

When job j is assigned to slot (t,i) this adds $w_j(t-d_j)^2$ to the cost. We seek the minimum cost over all possibilities of assigning all n jobs to a unique time slot. This minimization problem is a matching problem between jobs and time slots.

When the number of available time slots is equal to the number of jobs, $|T|=n$, this matching problem is known as the balanced assignment problem. The Hungarian method solves this problem to optimality in $\mathcal{O}(n^3+n^2\log n)=\mathcal{O}(n^3)$ time, as given in Section A.1. The assignment of completion times to jobs then specifies an optimal schedule under the restricted completion times.

When the number of available time slots is larger than the number of jobs, $|T|>n$, the matching problem is known as the unbalanced assignment problem. As found in Section A.2, this problem can be solved to optimality with computation time of complexity $\mathcal{O}((n\cdot|T|)n+n^2\log n)=\mathcal{O}(n^2|T|)$, where this last equality uses $|T|>n$. Again, this assignment of completion times to jobs, then specifies an optimal schedule under the restricted completion times.

For very large sets T, we will show that we can restrict ourselves to a relatively small subset of T. Consider consecutive due dates d^i, d^{i+1}. When there are more than $2n$ time slots in A, between d^i and d^{i+1}, we note that we only need to consider the n earliest, and n latest of these time slots. The other time slots between d^i and d^{i+1} can be excluded from T. Any of the excluded time slots is further away from any due date, than n of these included time slots. For the same reason, we only need to consider the n latest time slots before d^1. Likewise, we only need to consider the n earliest time slots after d^k. Thus w.l.o.g. we may assume $|T|=\mathcal{O}(kn)$ for any instance with k distinct due dates.

In general this gives the following theorem.

Theorem 3.9. *The scheduling problem over a restricted set T of completion times, can be solved in $\mathcal{O}(\min\{|T|n^2, kn^3\})$ time as an assignment problem for $|T|\geq n$.*

The integral scheduling problem is a scheduling problem over the restricted set $\mathbb{Z}\times m$. As a consequence of Theorem 3.9, any integral scheduling problem can be solved in $\mathcal{O}(kn^3)$ time, both for a single machine and in the identical parallel machine environment.

While the use of the assignment problem on scheduling problems with unit sized jobs is not new, this solution method we found is the solution method with lowest complexity as far as we know.

4 Unit weight problems

In this section we will consider the scheduling problem with unit weights. Thus the objective function is $\sum_j (C_j - d_j)^2$. Taking the jobs to have unit weights models that for every job it is equally important to be completed close to its due date.

We will start by considering the single machine environment for this problem. For this machine environment we will give some structural properties of optimal schedules for the problem. Then we will use the known properties to find an optimal integral schedule. After this, we will show that Algorithm 1 can be used to find an optimal schedule for general starting times and completion times. By using the structural properties of optimal schedules for the unit weight problem we can improve the algorithm to have lower running time. This algorithm can be adapted to determine an optimal integral schedule. Finally, we will extend the previous results to the identical parallel machine environment. Here we find that there must exist an optimal schedule where the jobs are distributed in a balanced matter between the m machines. Finally we find an exact distribution of jobs to machines for which an optimal schedule must exist. This optimal schedule can be found by solving m single machine problems.

We assume an instance I to be given by the k due dates $d^1, d^2, \ldots, d^k$ and for each due date d^i the number of jobs with this due date to be given by n_i.
Again, we take $d^1, d^2, \ldots, d^k$ to be ordered increasingly. Ordering the k due dates can be done in $\mathcal{O}(k \log k)$ time.

4.1 Structural Properties

The first structural property for unit weights is a general property that even holds in the identical parallel machine environment.

Property 4.1. For any optimal schedule σ^*, it holds that, for every pair of jobs $j \neq j'$:

$$d_j < d_{j'} \implies C_j \leq C_{j'}$$

Proof. Let σ^* be an optimal schedule. Suppose that there exists a pair of jobs j and j' with $d_j < d_{j'}$, such that $C_{j'} < C_j$.
Now, we construct a new schedule σ' in which we swap jobs j and j'. The difference in objective values is given by:

$$\begin{aligned}&(C_j - d_{j'})^2 + (C_{j'} - d_j)^2 - (C_{j'} - d_{j'})^2 - (C_j - d_j)^2\\ &= -2C_j d_{j'} - 2C_{j'} d_j + 2C_{j'} d_{j'} + 2C_j d_j\\ &= 2(d_j - d_{j'})(C_j - C_{j'}) < 0,\end{aligned}$$

as $d_j - d_{j'} < 0$ and $C_j - C_{j'} > 0$.
Thus, this interchange of jobs would lead to an improvement contradicting the optimality of σ^*. □

This means that in an optimal schedule, jobs have to be scheduled in order of non-decreasing due dates. For completeness we have derived when exactly an improving pairwise interchange of jobs is possible in Appendix C.3.

We will now further study the problem, restricting to the single machine environment.

Property 4.1, in combination with Property 3.2 give the following structural property of an optimal schedule.

Property 4.2. In any optimal schedule, jobs with the same due date are consecutively scheduled in a single block of jobs.

Note, that the order between jobs of the same due date, does not affect the cost of a schedule, as all jobs are of unit weight. Thus the specific order between these jobs does not matter and can be taken in increasing order of index.
With this known optimal sequence of jobs, the problem can either be solved by Algorithm 1. Using

Property 4.2 and the fact that the jobs have unit weights makes it possible to simplify Algorithm 1, resulting in a block merging algorithm for unit weights.

Using Property 4.2, we can portray an optimal schedule differently by giving the starting time of each block of jobs with a common deadline. For this matter we let t_i denote the starting time of the block of jobs with deadline d^i for $i = 1, 2, \dots, k$. This notion results in a method to find an optimal integral schedule.

4.2 Finding an optimal integral schedule

We will show a way to find an optimal integral schedule. First we consider instances where all jobs j share a common due date $d_j = d$. Then we consider instances with $k > 1$ distinct due dates.

Consider a block of jobs having common due date d. These jobs need to be scheduled in a single block of jobs by Property 3.2. For constrained problems, the number of jobs finishing before the due date differs by at most one from the number of jobs that complete after the due date. This can be easily seen by moving a job from one side of the due date to the other side when this is not the case. This means that either we start the first job at the end of the block instead or start the last job 1 time unit before the start of the block instead (such that it is completed at the start of the block). Hence, we have the following property.

Property 4.3. Consider the integral scheduling problem, where all jobs j have common due date, i.e., $d_j = d$. Then an optimal schedule consists of a single block that starts at time $t = \max\{0, d - \frac{n+1}{2}\}$ when n is odd, and starts at time $t \in \{\max\{0, d - 1 - \frac{n}{2}\}, \max\{0, d - \frac{n}{2}\}\}$ in case n is even.

A short formal proof is included in Appendix C.1.

Let an instance I of the integral scheduling problem be given. Then we let I_a be an instance derived from I by only considering jobs j with due date $d_j \leq d^a$. We will now consider schedule σ_a^* as optimal integral schedule of instance I_a. An optimal integral schedule for jobs with the smallest due date, σ_1^*, is given by Property 4.3. Then in an iterative manner we can find an optimal schedule σ_k^*, which is an optimal schedule of instance I.

Theorem 4.4. *Given an optimal schedule σ_ℓ^*, of instance I_ℓ, an optimal schedule $\sigma_{\ell+1}^*$ can be found in $\mathcal{O}(n^2)$ time.*

Proof. Let t_ℓ be the starting time of the first job with due date d^ℓ, to be processed in schedule σ_ℓ^*. Let t be the largest optimal starting time, of the block of jobs corresponding to due date $d^{\ell+1}$, as given by Property 4.3. When $t \geq t_\ell + n_\ell$, an optimal schedule $\sigma_{\ell+1}^*$ is found by including this block with starting time t in schedule σ_ℓ^*. Now we consider when this is not the case.
As in σ_ℓ^* no idle period can exist after d^l, we know that $t_\ell + n_\ell \leq d^l + n < d^{\ell+1} + n$. Furthermore, earlier scheduled jobs will never decrease the optimal starting time of later jobs. Thus one of the $\mathcal{O}(n)$ starting times $t, t+1, \dots, t_\ell + n_\ell$ must be optimal for instance $I_{\ell+1}$. For these starting times, the starting times of earlier jobs may need to shift to the left. Then for the $\mathcal{O}(n)$ schedules resulting from the different choices of starting times $t, t+1, \dots, t_\ell + n_\ell$, we can find the cost in $\mathcal{O}(n)$ time, resulting in $\mathcal{O}(n^2)$ time to find the optimal starting time $t_{\ell+1}$, and its corresponding schedule. □

As a result, we can iterate until finding optimal schedule σ_k^* in $\mathcal{O}(kn^2)$ time.

In the following section we show how Algorithm 1 can be implemented for unit weight instances to have running time $\mathcal{O}(k)$. We will show that this algorithm can be extended to find an optimal integral schedule in $\mathcal{O}(k)$ time. This algorithm is given in Appendix D.1.

4.3 Finding an optimal schedule for general starting times

In this section, we consider the general unit weight problem, that is, the starting and completion times do not need to be integral. We use Property 4.2 and the optimal starting times of blocks of jobs given

in Equation (3.2) in Section 3.2.2 to improve the running time of Algorithm 1.
When, in an optimal schedule, $t_i > t_{i-1} + n_{i-1}$, the block of jobs with due date d^i can be scheduled at its optimal starting time. For a block of n_ℓ jobs with due date d^ℓ, the optimal starting time t^*_ℓ, can be simplified to

$$t^*_\ell = \max\left\{0, \frac{\sum_{a=1}^{n_\ell}(d^\ell - a)}{n_\ell}\right\} = \max\left\{0, d^\ell - \frac{n_\ell + 1}{2}\right\}. \tag{4.1}$$

If some of these block-starting times are *conflicting*, i.e., $t_{\ell+1} < t_\ell + n_\ell$ for one or more values of ℓ, the choice of these starting times for the k blocks does not give a feasible schedule. Then, we can start the block merging algorithm with the k blocks $B_1, B_2, \dots, B_k$ of jobs having a common due date, ordered on increasing due date. This gives a computational advantage over starting with n blocks. We find the k optimal (possibly negative) starting times $\hat{t}_1, \hat{t}_2, \dots, \hat{t}_k$ for each block. Again we keep track of the number of jobs and the optimal starting time of each block. As the total weight of block B_i is equal to the number of jobs it contains, this gives another computational advantage. When merging block A of n_a jobs and with optimal starting time $\hat{t}_a$ with block B of n_b jobs and with optimal starting time $\hat{t}_b$, the new optimal starting time of block C follows from equation (3.4) and is equal to

$$\hat{t}_c = \frac{n_a}{n_a + n_b}\hat{t}_a + \frac{n_b}{n_a + n_b}(\hat{t}_b - n_a) \tag{4.2}$$

This leads to the following algorithm

Algorithm 3: The block merging algorithm for unit weights

```
Initialize the $k$ blocks $B_1, B_2, \dots, B_k$ with as data the optimal starting times $\hat{t}, t^*$ and the
  number of jobs. Schedule block $B_1$ at its optimal starting time;
for $\ell = 2, 3, \dots, k$ do
    $ActiveBlock = B_\ell$;
    while Optimal starting time of ActiveBlock conflicts with the block directly before it in job
      sequence $JS$ do
        Merge $ActiveBlock$ with the conflicting block creating $NewBlock$;
        Update data of $NewBlock$ and find its optimal starting time using Equation (4.2);
        $ActiveBlock = NewBlock$;
    end
    Schedule block containing block $B_\ell$ at its optimal non-negative starting time;
end
```

The initialization of the original k blocks takes $\mathcal{O}(k)$ time. We enter the for loop $k-1$ times. Over all iterations of the for-loop, the while-loop can be entered a total of $k-1$ times, as this would mean we merge blocks a total of $k-1$ times. When this happens, the schedule would consist of a single block and is not able to merge with other blocks. Including a new block or merging two blocks takes $\mathcal{O}(1)$ time of updating data, this leads to the algorithm having running time $\mathcal{O}(k)$ for ordered due dates.

Theorem 4.5. *The scheduling problem with unit weights can be solved in $\mathcal{O}(k)$ time for ordered due dates.*

In general the idea of starting with $b \neq n$ blocks in sequence $[j_1, j_2, \dots, j_n]$ gives a running time of $\mathcal{O}(b)$ to find an optimal schedule.

Again, Algorithm 3 can be adapted to the integral scheduling problem, much like explained in Section 3.2.2, adapting Algorithm 1. This adapted algorithm is displayed in Appendix D.1. As this scheduling problem has a running time of $\mathcal{O}(k)$ as well for ordered due dates, the integral scheduling problem with unit weights can be solved in $\mathcal{O}(k)$ time as well.

4.4 Extension the identical parallel machine environment

We consider the same problem for unit weights, but in the identical parallel machine environment.
We have shown that on a given machine we can solve the unit weight problem in $\mathcal{O}(k)$ time for ordered due dates. Therefore, for any distribution of jobs over the m machines, we can solve the m single machine problems to get the optimal schedule for the identical parallel machine problem given that distribution.

In this section we give some properties of an optimal solution, from which we can easily derive an optimal assignment of the jobs to the machines.

First, we define a structure for schedules, wherein jobs with a common due date are spread evenly over the machines.

Definition 4.6. A *balanced* schedule, is a schedule such that, for any due date d^ℓ, the number of jobs with due date d^ℓ to be processed on any machine is either $\lfloor \frac{n_\ell}{m} \rfloor$ or $\lceil \frac{n_\ell}{m} \rceil$.

We now show that there exists an optimal schedule with this structure.

Lemma 4.7. There exists an optimal schedule that is balanced.

Proof. Given any optimal schedule σ^*, we can construct an optimal schedule σ^*_b that is balanced.
If there exists a due date d^ℓ and a machine i that processes less than $\lfloor \frac{n_\ell}{m} \rfloor$ jobs, then there must exist a machine that processes more than $\lfloor \frac{n_\ell}{m} \rfloor$ jobs and vice versa.
Therefore, the schedule is not balanced, if and only if there exists a due date d^ℓ, and machines i, i', such that machine i processes at least two jobs more with due date d^ℓ than machine i' does.

Thus, when optimal schedule σ^* is not balanced, we consider a due date d^ℓ, and machines i, i', such that machine i processes at least two jobs more with due date d^ℓ than machine i' does.
Let job j_f, j_{last} be the first and last jobs with due date d^ℓ to be processed on machine i respectively. Note, that $S_{j_{last}}$ is the completion time of an earlier scheduled job with due date d^ℓ. Proposition 4.1 gives that no job j with due date $d_j \neq d^\ell$ can be processed at time C_{j_f}, and no job j with due date $d_j \neq d^\ell$ can be processed at time $S_{j_{last}}$. Machine i processes at least 2 jobs with due date d^ℓ more than machine i' does, then by Property 3.2, machine i' has not started processing jobs with due date d^ℓ before C_{j_f}, or has ended processing jobs with due date d^ℓ before $S_{j_{last}}$. The former is illustrated in Figure 3, and the latter in Figure 4. Here we use the convention that jobs with earlier due date are colored (dimmed) blue, and jobs with later due date are colored (dimmed) red.

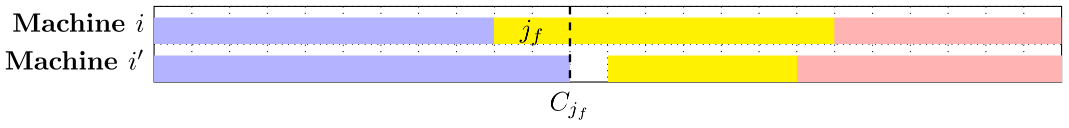

Figure 3: Late start on machine i'

In the situation in Figure 3 the schedules from C_{j_f} onwards can be exchanged between the machines, without affecting the cost or the number of jobs with due date other than d^ℓ. This reduces the difference between the number of jobs with due date d^ℓ that machine i and machine i' process.

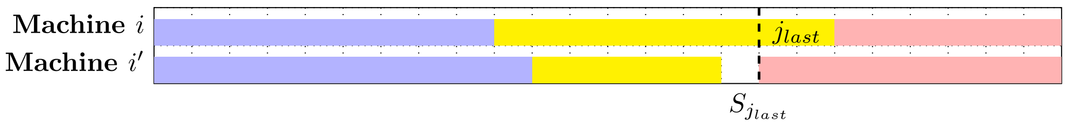

Figure 4: Early completion on machine i'

In the situation in Figure 4 the difference between the number of jobs with due date d^ℓ that machine i and machine i' process, can be reduced by exchanging the schedules, from $S_{j_{last}}$ onwards between the machines. This does not affect the cost or the number of jobs with due date other than d^ℓ.

We can repeat this procedure until we obtain a schedule σ^*_b, for which there does not exist a due date d^ℓ and two machines i, i', such that machine i' processes at least two jobs with due date d^ℓ more than machine i' does. Schedule σ^*_b must be an optimal schedule that is balanced. $\square$

Thus, by Lemma 4.7 there exists a balanced schedule that is optimal.

We note that any distribution of jobs for a balanced schedule, can be found as follows.
We first distribute $\lfloor \frac{n_\ell}{m} \rfloor$ jobs with due date d^ℓ to every machine for $\ell = 1, 2, \dots, k$. Note, that we have $n_\ell - m \cdot \lfloor \frac{n_\ell}{m} \rfloor$ remaining jobs with due date d^ℓ for $\ell = 1, 2, \dots, k$. In total this gives $r = \sum_{\ell=1}^{k} (n_\ell - m \cdot \lfloor \frac{n_\ell}{m} \rfloor)$ remaining jobs to distribute over the m machines. As we can find the number of remaining jobs for each due date in $\mathcal{O}(1)$ time, we can give the remaining jobs $j_1, j_2, \dots, j_r$ ordered on non-decreasing due date in $\mathcal{O}(k)$ time.

We introduce an assignment of remaining jobs $j_1, j_2, \dots, j_r$ to the machines. Consider a partition of the set of remaining jobs $\{j_1, j_2, \dots, j_r\}$ defined by $J_i = \{j_a \ : \ 1 \leq a \leq r, \ i \equiv a \mod m\}$ for $i = 1, 2, \dots, m$. Thus $J_i = \{j_i, j_{i+m}, j_{i+2m}, \dots\} \subseteq \{j_1, j_2, \dots, j_r\}$. Then we assign the job set J_i to machine i. Thus the remaining jobs are distributed evenly between the m machines. We now claim that an optimal balanced schedule σ^* exists for this assignment of the jobs to the machines.

Lemma 4.8. There exists an optimal balanced schedule σ^*, where machine i processes the remaining jobs in $J_i = \{j_a \ : \ 1 \leq a \leq r, \ i \equiv a \mod m\}$, for $i = 1, 2, \dots, m$.

Proof. We consider $\ell < r$ to be the maximal value for which there exists an optimal balanced schedule, such that machine i processes the jobs in $\{j_a \in J_i \ : \ a \leq \ell\}$ for $i = 1, 2, \dots, m$.
W.l.o.g. machine 1 processes job 1, and thus $\ell \geq 1$.
Let an optimal balanced schedule σ^* be given, such that machine i processes the jobs in $\{j_a \in J_i \ : \ a \leq \ell\}$ for $i = 1, 2, \dots, m$. Let i be the machine that processes job $j_{\ell+1}$, and let $i' \in \{1, 2, \dots, m\}$ such that $i' \equiv \ell + 1 \mod m$. We consider machine i and machine i'.
Note that $j_{\ell+1}$ is not the remaining job with lowest index scheduled on machine i or machine i', as otherwise we can exchange the schedules between the machine, creating an optimal schedule that contradicts the maximality of ℓ. We note that the remaining jobs that are assigned to machines i, i', are alternatingly assigned in index to machine i and machine i' until machine i processes job $j_{\ell+1}$ as well as the remaining job before it j_{prev}.
We proceed by showing that an optimal schedule $\hat{\sigma}$ must exist where the remaining jobs are alternatingly assigned to machine i and machine i', at least until machine i processes job $j_{\ell+1}$.

When the remaining job j_{next} following job $j_{\ell+1}$ has the same due date as $j_{\ell+1}$, job j_{next} must be scheduled on machine i' as the schedule σ^* is balanced. We can switch jobs j_{next} and $j_{\ell+1}$ in schedule σ^*, resulting in optimal schedule $\hat{\sigma}$.

Otherwise, let $J_{\ell+1}, J'_{\ell+1}$ be the blocks of jobs with due date $d_{j_{\ell+1}}$ scheduled on machine i and machine i' respectively. As σ^* is balanced and jobs j_{prev} and $j_{\ell+1}$ are both scheduled on machine i, $d_{j_{prev}} < d_{j_{\ell+1}}$. Thus $J_{\ell+1}$ consists of one job more than $J'_{\ell+1}$ does. Consider that $J'_{\ell+1}$ does not start processing later than $J_{\ell+1}$. W.l.o.g. job $j_{\ell+1}$ is the latest scheduled job in $J_{\ell+1}$. Then no job in $J'_{\ell+1}$ is scheduled at time $S_{j_{\ell+1}}$, and by Property 4.1, no job is scheduled on machine i' at $S_{j_{\ell+1}}$. This is illustrated in Figure 5.

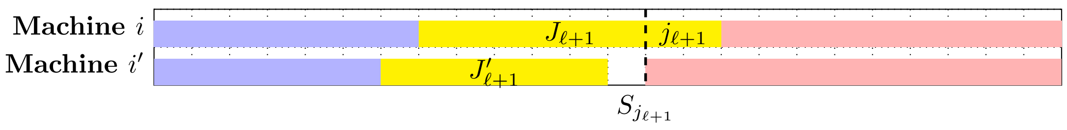

Figure 5: Early start of $J'_{\ell+1}$

In this case we swap the schedules from $S_{j_{\ell+1}}$ onwards, between the machines , giving optimal schedule $\hat{\sigma}$.

Then we consider that $J'_{\ell+1}$ does start processing later than $J_{\ell+1}$. W.l.o.g. $j_{\ell+1}$ is the job to be processed first, of the jobs in $J_{\ell+1}$. Here we distinguish between two cases.

Case 1

Suppose that no job j' is being processed at $S_{j_{\ell+1}}$ on machine i'. This is illustrated in Figure 6.

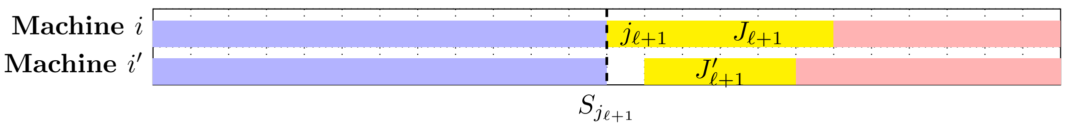

Figure 6: Case 1

We swap the schedules from $S_{j_{\ell+1}}$ onwards between the machines, giving optimal schedule $\hat{\sigma}$.

Case 2

Suppose that a job j' is being processed at time $S_{j_{\ell+1}}$ on machine i'. This is illustrated in Figure 7

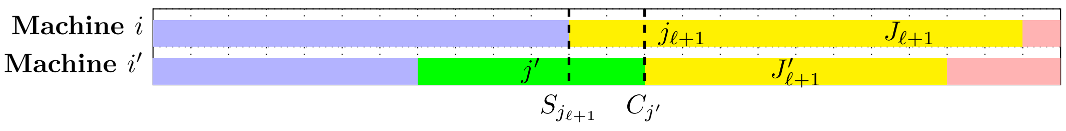

Figure 7: Case 2

Note that there is at most one remaining job with due date $d_{j'}$ assigned to machine i, and at most one such remaining job to machine i'. Let $j_{secprev}$ be the remaining job second in index to remaining job j_{prev} (that was the job second in index to remaining job $j_{\ell+1}$) As $d_{j_{secprev}} \leq d_{j_{prev}}$, Property 4.1 gives $d_{j'} = d_{j'_{secprev}} \implies d_{j'} = d_{j_{prev}}$.
We now show that $d_{j'} = d_{j_{prev}} \implies d_{j'} = d_{j_{secprev}}$.
Consider to contradiction that $d_{j'} = d_{j_{prev}} \neq d_{j_{secprev}}$, then machine i processes one job more with due date $d_{j'}$ and completes processing the jobs with due date $d_{j'}$ before machine i' does. We consider job j_{prev} to be the first processed job of its due date on machine i. This is illustrated in Figure 8

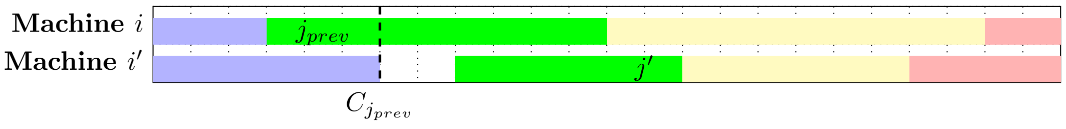

Figure 8: Case 2: $d_{j'} = d_{j_{prev}} \neq d_{j_{secprev}}$

By Property 4.1, no job is being processed on machine i' at time $C_{j_{prev}}$. Thus we can exchange the schedules from time $C_{j_{prev}}$ onwards, between the machines, resulting in the schedule in Figure 9.

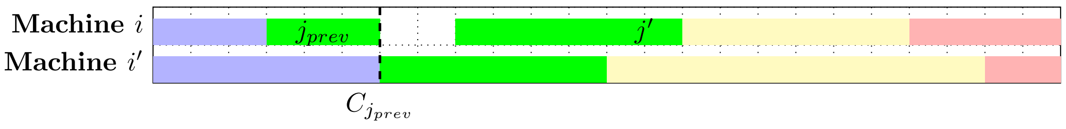

Figure 9: Case 2: $d_{j'} = d_{j_{prev}} \neq d_{j_{secprev}}$

This schedule has the same cost as σ^* and is not optimal by Property 3.1
Thus also $d_{j'} = d_{j_{prev}} \implies d_{j'} = d_{j_{secprev}}$ and therefore $d_{j'} = d_{j_{prev}} \iff d_{j'} = d_{j_{secprev}}$.
We conclude that the two machines process equally many jobs with due date $d_{j'}$. Either both machines process 1 remaining job with due date $d_{j'}$ or no such remaining job. We illustrate case 2 with this expanded knowledge in Figure 10.

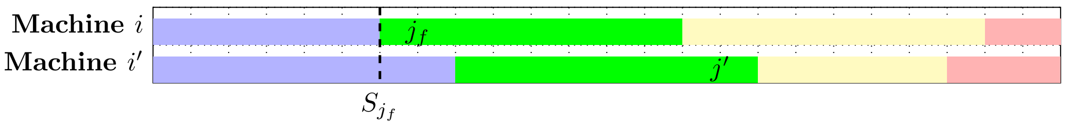

Figure 10: Case 2: expanded

Here j_f is the first job to be processed on machine i with due date $d_{j'}$. When reaching case 2 we try to swap the schedules from S_{j_f} onwards, between the two machines. Again we are either in case 1 and it can be done, or we reach case 2 again. Whenever we reach case 2 we repeat this procedure until reaching case 1. This process will terminate as there are only finitely many jobs scheduled of unit size, when there are no jobs before j_f we are in a case 1 situation.
After execution of this method, we will swap the pairs of remaining jobs with equal due date in case 2 situations back and have found the optimal schedule $\hat{\sigma}$.

The existence of optimal schedule $\hat{\sigma}$ contradicts the maximality of ℓ, and thus $\ell = n$.

□

Finding a distribution of jobs between the m machines for which an optimal schedule exists is an important step in determining an optimal schedule. By Lemma 4.4 an optimal balanced schedule can be found by first distributing $\lfloor \frac{n_\ell}{m} \rfloor$ to each machine for $\ell = 1, 2, \dots, k$, then distributing the remaining job set J_i to machine i, after which, we solve the m single machine problems optimally. With Algorithm 3, the block merging algorithm for unit weights, we can find the optimal schedule for each machine in $\mathcal{O}(k)$ time. Therefore the identical parallel machine problem can be solved to optimality in $\mathcal{O}(mk)$ time. Another approach is to use the job sequences for every machine. Every machine processes $\mathcal{O}(\frac{n}{m})$ jobs, and thus for every machine an optimal schedule can be found in $\mathcal{O}(\frac{n}{m})$ time using Algorithm 1, the block merging algorithm for sequences. Thus the identical parallel machine problem can also be solved in $\mathcal{O}(m \cdot \frac{n}{m}) = \mathcal{O}(n)$ time.

Theorem 4.9. *An optimal balanced schedule σ^* for the identical parallel machine problem with unit weights, can be found in $\mathcal{O}(\min\{mk, n\})$ time, for ordered due dates.*

Likewise, for the integral scheduling problem both Algorithm 6 and Algorithm 2 can be used to find optimal integral schedules. These algorithms do not differ in complexity from the algorithms with general starting and completion times. Thus, in the parallel machine environment, the integral scheduling problem with unit weights can be solved in $\mathcal{O}(\min\{mk, n\})$ time as well.

5 Single due date problems

In this section we consider all jobs j to share a single due date $d_j = d$. The jobs are assumed to have general weights. First, we show that an optimal schedule of the integral scheduling problem can be found in $\mathcal{O}(n \log n)$ time. Then we study the problem with general starting times. First we consider unconstrained problems. Then we consider constrained problems. In both cases, an optimal schedule can be found in $\mathcal{O}(n \log n)$. We will extend these findings to the identical parallel machine environment. For unconstrained problems we can find an optimal schedule in $\mathcal{O}(n \log n)$ time. For constrained problems we can find an optimal schedule in $\mathcal{O}(m^2 \cdot 2^m n + n \log n)$ time.

For the integral scheduling problem with identical parallel machines, there is no distinction between the machines: every machine has the same set of completion times. According to Property 3.3, in an optimal schedule, the jobs with highest weight must be completed closest to d. This means that the m jobs with highest weight must be scheduled to be completed at time d, the jobs with next highest weight must be scheduled directly afterwards or (if possible) directly before these m jobs. This process can be continued until all jobs are scheduled.

Theorem 5.1. *An optimal schedule σ^* for the integral scheduling problem with a single common due date d, can be found in $\mathcal{O}(n \log n)$ time.*

Proof. We order the n jobs on non-increasing weight, this can be done in $\mathcal{O}(n \log n)$ time (for example using block sort). Then we schedule the jobs, in this order, over the available completion times closest to due date d, over all of the m machines, which can be done in $\mathcal{O}(n)$ time. $\square$

This gives a solution method for the problem with general starting and completion times and $d \leq 1$ as well. Whenever $d \leq 1$, Property 3.1 gives that in any optimal schedule, all machines start processing at time 0 without idle time between consecutive jobs. Thus, every optimal schedule is an integral schedule. Therefore we assume $d > 1$ for the rest of Section 5.

We recall that we only demand integral starting and completion times when explicitly stating that we consider an integral scheduling problem. We now will proceed with single due date problems in the single machine environment (with general starting and completion times).

5.1 Optimal schedules in the single machine environment

We will start by introducing a certain symmetry around the due date.

Definition 5.2. Given a schedule σ we define the mirrored schedule σ^M by $C_j^M = 2d - C_j$.

We will explain the reasoning for the new completion time $C_j^M = 2d - C_j$.
Note that $2d - C_j = d - (C_j - d) = d + (E_j - T_j)$. That is, when job j is late, we subtract the lateness of job j from d, when job j is early the earliness of job j is added to d. For early jobs j, the earliness in σ will be the lateness in σ^M, for late jobs j, the lateness in σ will be the earliness in σ^M.
The cost of σ is equal to the cost of σ^M as for every job j the cost $w_j(C_j - d)^2 = w_j(C_j^M - d)^2$ is not affected by mirroring.

We will show how this works by giving an example:
Let a single due date problem be given with four jobs and common due date $d = \frac{18}{5}$.
Let the schedule σ be defined by $C_1 = \frac{8}{5}$, $C_2 = \frac{17}{5}$, $C_3 = \frac{22}{5}$ and $C_4 = \frac{31}{5}$.
This schedule is illustrated in Figure 11.

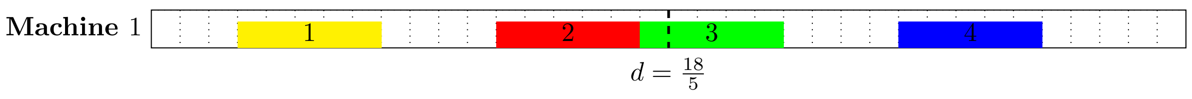

Figure 11: Schedule σ

The completion times of the mirrored schedule are $C_1^M = \frac{28}{5}$, $C_2^M = \frac{19}{5}$, $C_3^M = \frac{14}{5}$ and $C_4^M = \frac{5}{15}$. The mirrored schedule σ^M is illustrated in Figure 12.

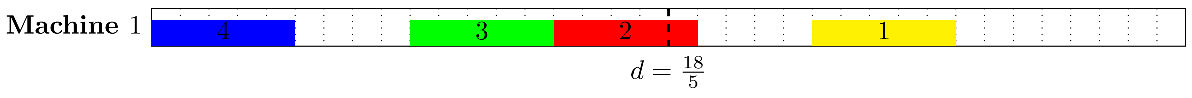

Figure 12: The mirrored schedule σ^M

By Property 3.2 any optimal schedule σ consists of one single block of jobs to be scheduled.
Consider an optimal schedule σ with job j_f to be processed first and job j_ℓ to be processed last.
We note that $C_{j_\ell} = S_{j_f} + n$. Then the mirrored schedule σ^M starts processing at time
$S_{j_\ell}^M = C_{j_\ell}^M - 1 = 2d - (S_{j_f} + n) - 1$. When $S_{j_f} \leq d - \frac{n+1}{2}$ we have $S_{j_\ell}^M \geq 2d - (d - \frac{n+1}{2} + n) - 1 = d - \frac{n+1}{2}$.
We note that j_f is the first job to start processing in σ and j_ℓ is the first job to start processing in the mirrored schedule σ^M. Therefore for any optimal schedule σ the first job does not start processing before time $d - \frac{n+1}{2}$ or for its mirrored schedule σ^M, the first job does not start processing before time $d - \frac{n+1}{2}$. As the mirrored schedule σ^M is also optimal and when $d \geq \frac{n+1}{2}$, $S_{j_f} \geq 0$ or $S_{j_\ell}^M \geq 0$, this gives the following theorem.

Theorem 5.3. *The single due date problem with $d \geq \frac{n+1}{2}$ is unconstrained.*

Furthermore, by definition of unconstrained problems, increasing the due date of a problem will not reduce the optimal cost of the problem. Thus, in order to find the cost of an optimal schedule, we may assume $d \geq \frac{n+1}{2}$ for unconstrained problems.

We will first show how unconstrained problems with a single common due date can be solved up to optimality. Then we will show how to solve constrained problems with a single common due date.

5.1.1 The unconstrained problem

We consider an unconstrained problem, for which we may assume $d \geq \frac{n+1}{2}$. We will show how to find an optimal schedule σ^*. As seen before we can disregard the non-negativity constraints for the starting times as this will result in an optimal schedule σ^* for which the constraints hold or otherwise these constraints hold for the mirrored schedule $(\sigma^*)^M$.

Let an optimal schedule σ^* be given. Let job j be a job that is scheduled with completion time closest to d from all completion times in σ^*. This job can not be completed too far away from d.

Property 5.4. For an optimal schedule σ^*, where job j is a job scheduled with completion time closest to the due date, it holds that $|C_j - d| \leq \frac{1}{2}$.

W.l.o.g. such a job j is scheduled in σ^*, to be late or completed at its due date. When this is not the case job j is scheduled late in the mirrored schedule $(\sigma^*)^M$, which is optimal as well.
We consider job j to have completion time $d + \epsilon$ with $0 \leq \epsilon \leq \frac{1}{2}$.
By Property 3.3, job j must be the job of highest weight. Furthermore this property gives that the job completed second closest to d must have second highest weight and so on.
Let the n jobs $j_1, j_2, \ldots, j_n$ ordered on non-increasing weight. Then the single block of jobs that is scheduled in σ^* is $[j_{n-1}, \ldots, j_2, j_1, j_3, \ldots, j_n]$ for odd n and $[j_n, \ldots, j_2, j_1, j_3, \ldots, j_{n-1}]$ for even n. Thus the order of jobs in this block can be find in $\mathcal{O}(n \log n)$ time, as this is the time it takes to order the jobs.
Given the composition of the block of jobs scheduled in σ^*, we can find the optimal starting time as in Equation (3.2) given in Section 3.2.2 in $\mathcal{O}(n)$ time. This optimal starting time is given by

$$\hat{t} = \frac{\sum_{a=1}^{n} w_{j_a}(d-a)}{\sum_{a=1}^{n} w_{j_a}} = d - \frac{\sum_{a=1}^{n} w_{j_a} a}{\sum_{a=1}^{n} w_{j_a}}.$$

And thus we have shown how to find an optimal schedule σ^* effectively.

Theorem 5.5. *An optimal schedule σ^* for the scheduling problem with a single common due date $d \geq \frac{n+1}{2}$, can be found in $\mathcal{O}(n \log n)$ time.*

5.1.2 The constrained problem

Now we consider constrained problems for which necessarily $d < \frac{n+1}{2}$, by Theorem 5.3.

Let $1 < d < \frac{n+1}{2}$. We consider an optimal schedule σ^* that consists of a single block of jobs to be processed. The first job j_f must have starting time $0 \leq S_{j_f} \leq 1$. Otherwise the last job j_ℓ has completion time $C_{j_\ell} \geq n+1$ and $|C_{j_\ell} - d| \geq n+1-d > \frac{n+1}{2}$. Thus, job j_ℓ could instead be processed at the start with completion time $C'_{j_\ell} = 1$, so that $|C'_{j_\ell} - d| = d - 1 < \frac{n+1}{2} - 2 < |C_{j_\ell} - d|$ contradicting optimality of schedule σ^*.
As $0 \leq S_{j_f} \leq 1$, either $\lfloor d \rfloor$ or $\lfloor d \rfloor - 1$ jobs must be completed at time d. Furthermore job j_1 can be one of the jobs completed at time d, or it can be completed late such that it isn't.

This leads to four possible cases for the block in an optimal solution. For each case, a different block is optimal. We will construct these four blocks. One of these blocks must be the block scheduled in σ^*, thus we can find an optimal schedule.

Theorem 5.6. *An optimal schedule σ^* for the scheduling problem with a single common due date $d < \frac{n+1}{2}$, can be found in $\mathcal{O}(n \log n)$ time.*

Proof. Consider $\lfloor d \rfloor$ jobs are completed at time d, including job j_1. Property 3.3 then gives that the optimal block must be $[j_{2\lfloor d \rfloor - 1}, \ldots, j_3, j_1, j_2, j_4, \ldots, j_{2\lfloor d \rfloor}, j_{2\lfloor d \rfloor + 1}, j_{2\lfloor d \rfloor + 2}, \ldots, j_n]$.

Secondly we consider that $\lfloor d \rfloor$ jobs are completed at time d, and this doesn't include job j_1. Property 3.3 shows that the optimal block must be $[j_{2\lfloor d \rfloor}, \ldots, j_4, j_2, j_1, j_3, \ldots, j_{2\lfloor d \rfloor + 1}, j_{2\lfloor d \rfloor + 2}, j_{2\lfloor d \rfloor + 3}, \ldots, j_n]$.

Then we consider that $\lfloor d \rfloor - 1$ jobs are completed at time d, including job j_1. Property 3.3 then gives that the optimal block must be $[j_{2\lfloor d \rfloor - 3}, \ldots, j_3, j_1, j_2, j_4, \ldots, j_{2\lfloor d \rfloor - 2}, j_{2\lfloor d \rfloor - 1}, j_{2\lfloor d \rfloor}, \ldots, j_n]$.

Finally, we consider that $\lfloor d \rfloor - 1$ jobs are completed at time d, and this does not include job j_1. Property 3.3 then gives that the optimal block must be $[j_{2\lfloor d \rfloor - 2}, \ldots, j_4, j_2, j_1, j_3, \ldots, j_{2\lfloor d \rfloor - 1}, j_{2\lfloor d \rfloor}, j_{2\lfloor d \rfloor + 1}, \ldots, j_n]$.

Let $\sigma_1, \sigma_2, \sigma_3$ and σ_4 be the schedules resulting from scheduling these blocks at their optimal starting time given by Equation (3.2). As these four schedules are the optimal schedules for each of the four possible cases of an optimal schedule, one of the schedules $\sigma_1, \sigma_2, \sigma_3$ and σ_4 must be optimal. Thus, we may take any of these schedules with minimal cost compared to the other 3 schedules to be the optimal schedule σ^*.
After sorting the jobs in non-increasing weight in $\mathcal{O}(n \log n)$ time, these four blocks can be found and then scheduled at their optimal starting time in $\mathcal{O}(n)$ time. $\square$

In the following section we will use these methods to find an optimal schedule for the single machine case to find an optimal schedule for m identical parallel machines.

5.2 Optimal schedules in the identical parallel machine environment

Much like the unit weight problem in the identical parallel machine environment, there exists an optimal schedule that is balanced. As defined in Definition 4.4, this means that every machine processes either $\lfloor \frac{n}{m} \rfloor$ or $\lceil \frac{n}{m} \rceil$ jobs.

Theorem 5.7. *For the problem with a single due date, in the identical parallel machine environment, any optimal schedule σ^* is balanced.*

Proof. Let an optimal schedule σ be given that is not balanced. Assume machine i processes at least two jobs more than machine i', we note that either machine i starts processing at least 1 time unit earlier,

or it ends at least 1 time unit later.
In this first case, let job j' be the first job to be processed on machine i. In the latter case, let j' be the last job to be processed on machine i. Job j' can be processed at the same time on machine i' instead. Let schedule σ' be the optimal schedule that results from this change. Note that $C_{j'} \neq d$ as otherwise a job $j \neq j'$ scheduled on machine i can be scheduled at its due date instead lowering cost. This contradicts optimality. Let B be the block of jobs on machine i in schedule σ Then $C_{j'} \neq d$ implies that when removing job j' from block B, the optimal position of the block of all other jobs in B must be different than before, as can be seen in Equation (3.2). This constradicts optimality of σ' and therefore also of schedule σ. $\square$

This leads that in an optimal schedule with $m_1 = n \mod m$ machines that process $\lceil \frac{n}{m} \rceil$ jobs, and $m_2 = m - m_1$ machines that process $\lfloor \frac{n}{m} \rfloor$ jobs.

As $d \geq \frac{\lceil \frac{n}{m} \rceil + 1}{2} \implies d \geq \frac{\lfloor \frac{n}{m} \rfloor + 1}{2}$, Theorem 5.3 gives that all single machine problems are unconstrained when $d \geq \frac{\lceil \frac{n}{m} \rceil + 1}{2}$. As a result the identical parallel machine problem with $d \geq \frac{\lceil \frac{n}{m} \rceil + 1}{2}$ is unconstrained. Again, to find the optimal cost of unconstrained problems we may assume $d \geq \frac{\lceil \frac{n}{m} \rceil + 1}{2}$ (if this is not the case we may increase the due date to any arbitrary value). We first consider unconstrained problems. Then we consider constrained problems.

5.2.1 The unconstrained problem

Let the problem be unconstrained. We may assume $d \geq \frac{\lceil \frac{n}{m} \rceil + 1}{2}$.

Consider an optimal schedule σ^*.
Property 5.4 gives that on every machine i we may assume that the job scheduled to complete closest to its due date completes at time $d + \epsilon_i$ with $-\frac{1}{2} \leq \epsilon_i \leq \frac{1}{2}$. For negative values of ϵ_i we consider the mirrored schedule, and thus $0 \leq \epsilon_i \leq \frac{1}{2}$. W.l.o.g. we assume $0 \leq \epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_m$. From Property 3.3 it follows that job j_i is scheduled on machine i to be completed at time $d + \epsilon_i$. Job j_{m+1} must be scheduled to complete in the next closest position. This is directly before job j_m on machine m. Likewise, job $j_{m+\ell}$ is scheduled on machine $m + 1 - \ell$ before job $j_{m+1-\ell}$ for $\ell \in \{1, 2, \ldots, m\}$. Then job j_{2m+1} must be scheduled on machine 1 directly after job j_1. Job $j_{2m+\ell}$ is schedule on machine ℓ directly after job j_ℓ for $\ell \in \{1, 2, \ldots, m\}$. Continue this procedure until the job sequence is known for all m machines.
For machine i the job sequence is $[\ldots, j_{4m+1-i}, j_{2m-i+1}, j_i, j_{2m+i}, j_{4m+i}, \ldots]$. The m sequences can be found in a total of $\mathcal{O}(n)$ time. Then the blocks that consist of these job sequences can be scheduled at their optimal starting time given in Equation (3.2). This takes in total $\mathcal{O}(n)$ time. We do need the jobs to be ordered however, which can be done in $\mathcal{O}(n \log n)$ time.

Theorem 5.8. *An optimal schedule σ^* for the unconstrained problem, in the identical parallel machine environment, can be found in $\mathcal{O}(n \log n)$ time.*

5.2.2 The constrained problem

Let the problem be constrained, then we may consider $1 < d < \frac{\lceil \frac{n}{m} \rceil}{2}$.

In an optimal schedule σ^*, every machine processes either $\lceil \frac{n}{m} \rceil$, or $\lfloor \frac{n}{m} \rfloor$ jobs. By adding $m \cdot \lceil \frac{n}{m} \rceil - n$ jobs with weight 0, we may consider every machine to process $\lceil \frac{n}{m} \rceil$ jobs in an optimal schedule.

Consider optimal schedule σ^*.
Again, Property 5.4 gives that on every machine i we may assume that the job scheduled to be completed closest to its due date completes at time $d + \epsilon_i$ with $-\frac{1}{2} \leq \epsilon_i \leq \frac{1}{2}$. This time, mirrored schedules might be infeasible however, and thus we consider the order of $|\epsilon_i|$ for $i = 1, 2, \ldots, m$.
W.l.o.g. we assume $0 \leq |\epsilon_1| \leq |\epsilon_2| \leq \ldots \leq |\epsilon_m|$.
From Property 3.3 it follows again, that job j_i is scheduled on machine i to be completed at time $d + \epsilon_i$.

The job sequences on any of the machines, must take one of the 4 forms discussed in Section 5.1.2. For machine i, job j_i can be scheduled late or early. Furthermore, either $\lfloor d \rfloor$ or $\lfloor d \rfloor - 1$ jobs are scheduled early.
Thus, there are 4^m possible combinations of these cases, for the m single machine schedules. By fixing

a combination, the order of the machines and Property 3.3, the m job sequences can be found in $\mathcal{O}(n)$ time for ordered jobs. Then any job sequence can be scheduled optimally as a block in $\mathcal{O}(\frac{n}{m})$ time, and the whole schedule can be found in $\mathcal{O}(m \cdot \frac{n}{m}) = \mathcal{O}(n)$ time. Thus, any combination of cases for the m machines, gives a candidate for an optimal schedule in $\mathcal{O}(n)$ time.
By considering the 4^m distinct possibilities, an optimal schedule can be found in $\mathcal{O}(4^m n)$ time.

We will show, however, that it suffices to consider a subset of $m^2 \cdot 2^m$ combinations of cases.

Let M_{early} be the set of machines where the job scheduled to complete closest to the due date (job j_i for machine i) is scheduled early or on time. Likewise, let M_{late} be the set of machines where the job scheduled to complete closest to the due date is scheduled late.

Lemma 5.9. An optimal schedule σ^* and integers $0 \leq \ell_1, \ell_2 \leq m$ exist, such that the first ℓ_1 machines in M_{early}, and the last ℓ_2 machines in M_{late} process $\lfloor d \rfloor$ jobs before d, and all other machines process $\lfloor d \rfloor - 1$ job before d.

Proof. Let an optimal schedule σ^* be given.
Consider machines $i, i' \in M_{early}$ with $i < i'$ such that machine i processes $\lfloor d \rfloor - 1$ jobs before d and machine i' processes $\lfloor d \rfloor$ jobs before d.
Note that jobs j_i and $j_{i'}$ are the jobs scheduled to complete closest to due date d on machine i and machine i' respectively. As both $i, i' \in M_{early}$ jobs $j_i, j_{i'}$ are scheduled early. Then, by the ordering between the machines, the fact that $i < i'$ implies that job j_i is completed at least as close to the due date d as $j_{i'}$ does, i.e., $C_{j_{i'}} \leq C_{j_i}$. Machine i must process $\lfloor d \rfloor - 2$ jobs before job j_i and machine i' must process $\lfloor d \rfloor - 1$ jobs before job $j_{i'}$. As every machine processes a single block of jobs the first job on machine i' is processed during idle time on machine i. Thus we will process this job on
machine i instead. As a result machine i processes $\lfloor d \rfloor$ jobs before d and machine i' processes $\lfloor d \rfloor - 1$ jobs before d.

Likewise, we consider machines $i, i' \in M_{late}$ with $i < i'$ such that machine i processes $\lfloor d \rfloor$ jobs before d and machine i' processes $\lfloor d \rfloor - 1$ jobs before d. As $i < i'$ we have completion times $C_{j_i} \leq C_{j_{i'}}$ for the late jobs j_i and $j_{i'}$. This time we can process the first job on machine i on machine i' instead. In the resulting optimal schedule, machine i processes $\lfloor d \rfloor - 1$ jobs before d and machine i' processes $\lfloor d \rfloor$ jobs before d.

By repeating these procedures, we can alter the optimal schedule σ^*, until we find an optimal schedule as specified in Lemma 5.9. □

As a result, we can consider the 2^m partitions of the machines in M_{early} and M_{late}. Then we may consider the $\mathcal{O}(m^2)$ possible values of ℓ_1 and ℓ_2. This specifies a subset of the 4^m combinations that we considered at first. By virtue of Lemma 5.9, these $\mathcal{O}(m^2 \cdot 2^m)$ combinations are sufficient to consider. Thus, after ordering the jobs in $\mathcal{O}(n \log n)$ time, a total of $\mathcal{O}(m^2 \cdot 2^m)$ combinations are considered giving $\mathcal{O}(m^2 \cdot 2^m)$ candidates for an optimal schedule. Every candidate can be constructed in $\mathcal{O}(n)$ time using Property 3.3.

Theorem 5.10. *An optimal schedule σ^* for the constrained problem, in the identical parallel machine environment, can be found in $\mathcal{O}(m^2 \cdot 2^m n + n \log n)$ time.*

For many machines and a very small due date $1 < d << \frac{\lceil \frac{n}{m} \rceil}{2}$, we would advice a simple heuristic to find a schedule with reasonable cost in polynomial time. The assumption of many machines implies that the running time to find an optimal schedule is high. The assumption $d << \frac{\lceil \frac{n}{m} \rceil}{2}$ will lead us to believe that the heuristic finds a good result. In fact we will use the same method as for instances with $d \leq 1$.
Just like before we schedule a single block of jobs on every machine. Many jobs must be late, every late job reduces the optimal starting of the block of jobs that contains it. As a result we expect every block to start at, or close to, 0. Therefore we choose to consider every machine to start processing at time 0. As every job sequence forms a single block, this means we seek an optimal integral schedule which can be found in $\mathcal{O}(n \log n)$ time. This optimal integral schedule is then the schedule we expect to have a reasonable cost for the problem with general starting and completion times.

6 The general single machine problem

In this section we consider the general problem on a single machine and introduce two heuristics and an additive approximation scheme.

6.1 A local search heuristic

Minimization problems can sometimes be solved by local search techniques. In local search, a neighbourhood for a given solution is considered. The simplest form of local search; iterative improvement, moves from one solution to an improving neighbour until no improving neighbours exist. A well-known example of iterative improvement is the simplex algorithm. By convexity, the simplex algorithm will terminate with an optimal solution. For the scheduling problem we will define the neighbourhood of a schedule, thus giving a local search algorithm. Then, we point out certain disadvantages of this approach.

First, we note that we can represent candidates of optimal schedules by a single job sequence JS. This follows as we can find an optimal schedule given job sequence JS in only $\mathcal{O}(n)$ time using Algorithm 1. When saying that a schedule σ is represented by a job sequence, we will in turn mean that schedule σ is an optimal schedule given that job sequence.

Definition 6.1. Let a schedule σ be represented by job sequence $JS = [j_1, j_2, \dots, j_n]$. The neighbouring schedules σ' are the schedules represented by JS', where JS' results from a single interchange of jobs in JS.

Thus, one could start with an arbitrary schedule that is optimal given its sequence and keep moving from schedule to neighbouring schedule with lower cost (as soon as we find one) until no neighbour with lower cost exists. A choice of schedule to start with could be in order of non-decreasing due dates to possibly reduce the number of iterations.
The resulting Algorithm is given below.

Algorithm 4: The local search algorithm

```
Let $JS$ be the initial job sequence and let $\sigma$ be the optimal schedule given job sequence $JS$,
  found with Algorithm 1.
while There exists an improving neighbouring schedule $\sigma'$ of schedule $\sigma$ do
    Replace schedule $\sigma$ with schedule $\sigma'$;
end
```

Here we note, that all neighbouring schedules σ can be found by performing a job interchange in JS, and running Algorithm 1 on the newly found job sequence JS'.

The number of iterations of this algorithm is bounded by the number of possible job sequences given by $n!$. This would lead to the algorithm having worst case complexity $\mathcal{O}(n \cdot n!)$ as Algorithm 1 has running-time $\mathcal{O}(n)$. In $\mathcal{O}(n \cdot n!)$ time we could also calculate the cost of all $n!$ schedules that are optimal given their sequence, thus finding an optimal schedule in $\mathcal{O}(n \cdot n!)$ time.

With an example we will show that for our definition of neighbouring schedules, a locally optimal solution does not have to be optimal.

6.1.1 Counterexample of optimality

By a counterexample, we show that locally optimal solutions are not necessarily optimal.

Consider the problem with the 6 jobs $j_1, j_2, \dots, j_6$. Let $d_1 = -4,\ d_2 = 1\frac{5}{11},\ d_3 = 2\frac{4}{11},\ d_4 = 4,$
$d_5 = 5,\ d_6 = 6$. For jobs j_4, j_5 and j_6 we let the weight $j_4 = j_5 = j_6 = W$ approach infinity such that these jobs need to be completed at their due date (which is equal to their index) in an optimal schedule. Furthermore we take $w_1 = 1,\ w_2 = w_3 = 110$.
Note that this problem is equivalent to the problem translated 5 to the right with 5 added jobs with due date 1 and weight W. Thus a counterexample with only positive due dates exist. For simplicity we prefer the counterexample with 5 jobs and $d_1 = -4$, as only 6 jobs need to be scheduled instead of 11.

Now we consider the schedule σ_1 given by $C_{j_i} = i$ for $i \in \{1, 2, \dots, 5\}$. This schedule is shown in

Figure 13.

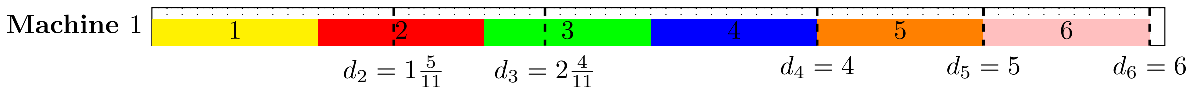

Figure 13: Schedule σ_1

This schedule has cost

$$\begin{aligned}\sum_j w_j(C_j - d_j)^2 &= w_1(1-(-4))^2 + w_2(2-1\frac{5}{11})^2 + w_3(3-2\frac{4}{11}) \\ &= 5^2 + 110\cdot(\frac{6}{11})^2 + 110\cdot(\frac{7}{11})^2 = 25 + \frac{360}{11} + \frac{490}{11} = 102\frac{3}{11}.\end{aligned}$$

If job j_1 would be scheduled after job j_6 in an optimal schedule it would have cost of at least $w_1(7-(-4))^2 = 121 > 102\frac{3}{11}$. For job j_2 and j_3 we also see that scheduling one of these jobs after job j_6 gives cost of at least $110(7-2\frac{4}{11}) > 77\frac{3}{11}$.
Thus it is clear that in an optimal schedule jobs j_1, j_2 and j_3 need to be scheduled between time 0 and job j_4 starting at time 3. This is the case in σ_1.

A pairwise interchange with at least one of the jobs j_4, j_5 and j_6 will not give a better schedule (the cost will approach infinity). Thus, we consider the pairwise interchanges of jobs j_1, j_2 and j_3.

First, we look at the interchange of jobs j_1 and j_2. This gives schedule σ_2 illustrated in Figure 14
.

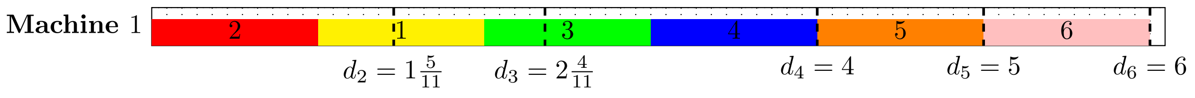

Figure 14: Schedule σ_2

Schedule σ_2 has cost

$$\begin{aligned}\sum_j w_j(C_j - d_j)^2 &= w_1(2-(-4))^2 + w_2(1-1\frac{5}{11}) + w_3(3-2\frac{4}{11})^2 \\ &= 6^2 + 110\cdot(-\frac{5}{11})^2 + 110\cdot(\frac{7}{11})^2 = 36 + \frac{250}{11} + \frac{490}{11} = 103\frac{3}{11} > 102\frac{3}{11}.\end{aligned}$$

Thus this interchange does not reduce the cost.

Then we consider the interchange of jobs j_1 and j_3. As both jobs will be completed farther from their respective due dates, this pairwise interchange will not decrease the cost.

Finally, we consider the interchange of jobs j_2 and j_3 resulting in schedule σ_3 illustrated in Figure 15.

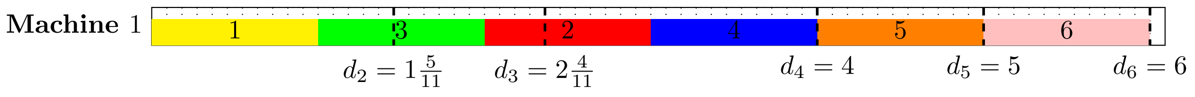

Figure 15: Schedule σ_3

Schedule σ_3 has cost

$$\begin{aligned}\sum_j w_j(C_j - d_j)^2 &= w_1(1-(-4))^2 + w_2(3-1\frac{5}{11}) + w_3(2-2\frac{4}{11})^2 \\ &= 5^2 + 110\cdot(\frac{17}{11})^2 + 110\cdot(-\frac{4}{11})^2 = 25 + \frac{2890}{11} + \frac{160}{11} = 302\frac{3}{11} > 102\frac{3}{11}.\end{aligned}$$

Thus this last interchange does not reduce the cost either.

This shows that schedule σ_1 is locally optimal.

We can however consider schedule σ^* illustrated in Figure 16.

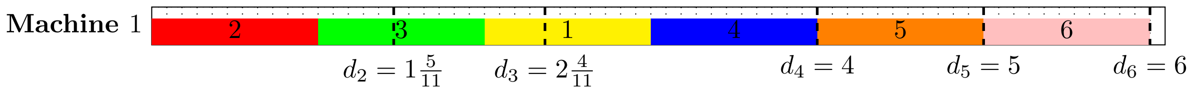

Figure 16: Schedule σ^*

This schedule σ^* is not the result of a pairwise interchange of schedule σ.

Schedule σ^* has cost

$$\begin{aligned}\sum_j w_j(C_j - d_j)^2 &= w_1(3-(-4))^2 + w_2(1-1\frac{5}{11}) + w_3(2-2\frac{4}{11})^2 \\ &= 7^2 + 110\cdot(-\frac{5}{11})^2 + 110\cdot(-\frac{4}{11})^2 = 49 + \frac{250}{11} + \frac{160}{11} = 86\frac{3}{11} < 102\frac{3}{11}.\end{aligned}$$

Therefore the locally optimal schedule σ is not an optimal schedule.
It can in fact easily be verified that schedule σ^* is the (unique) optimal schedule of this problem.

6.2 A greedy heuristic

We present a polynomial time, greedy heuristic to find a schedule $\hat{\sigma}$ with low cost. First, we sort the jobs $j_1, j_2, \ldots, j_n$ in order of non-increasing weight. Then the heuristic iteratively includes the unscheduled job with highest weight and inserts it at a place in the job sequence of scheduled jobs such that, when scheduled by Algorithm 1, the cost is minimal.

To initialize, the job sequence $JS = [j_1]$ is taken. Then job j_2 is either inserted in the first or second position giving job sequences $[j_2, j_1]$ and $[j_1, j_2]$ respectively. The job sequence JS is then updated with the new job sequence of lowest cost. Then this insertion is done for job j_3 and so on, until we have found a job sequence that includes all n jobs and is scheduled optimally by Algorithm 1.
This gives the following algorithm.

Algorithm 5: The greedy heuristic algorithm

```
Start with job sequence $JS = [j_1]$.
for $\ell = 2, 3, \ldots, n$ do
    $ActiveJob = j_\ell$;
    for $a = 1, 2, \ldots, \ell$ do
        Insert job $j_\ell$ at position $a$ in the job sequence, find the optimal schedule and its cost with
          Algorithm 1.
    end
    Insert job $j_\ell$ in job sequence $JS$ at the position that results in the lowest cost.
end
```

We note that the jobs can be sorted in $\mathcal{O}(n \log n)$ time. Then we have $\mathcal{O}(n)$ jobs to add to the job sequence. For each of the job we test $\mathcal{O}(n)$ positions, and find the optimal schedules and cost of these sequences which takes $\mathcal{O}(n)$ time each, thus every insertion is done in $\mathcal{O}(n^2)$ time. This leads to running time $\mathcal{O}(n^3)$ for this greedy algorithm.

As the jobs with highest weight are scheduled at the best possible place in the sequence, we hope to achieve that the jobs with highest weight are correctly ordered relatively to each other. When jobs with lower weight are not scheduled at the same position as in an optimal schedule, this does not affect the cost that much. Thus we hope to find a schedule with relatively low cost.

We do note however that this heuristic does not always give an optimal solution, as the following example shows.

6.2.1 Counterexample of optimality

We take 6 jobs. Let $w_1 = w_2 = w_3 = w_4 = W$ with W approaching infinity, and let the last 2 jobs have weights $w_5 = 10$ and $w_6 = 9$ respectively. Then as due dates we take $d_1 = 2,\ d_2 = 3,\ d_3 = 4,$ $d_4 = 6,\ d_5 = 5.8$ and $d_6 = 4$. After 3 iterations the job sequence $[j_1, j_2, j_3, j_4]$ is found, the corresponding optimal schedule is illustrated in Figure 17.

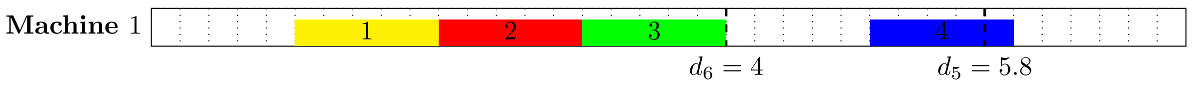

Figure 17: Job sequence $[j_1, j_2, j_3, j_4]$

These four jobs are scheduled with zero cost, as they also must be in the optimal schedule.

Algorithm 5 proceeds by inserting job j_5 at the fourth position giving sequence $[j_1, j_2, j_3, j_5, j_4]$, with optimal schedule illustrated in Figure 18.

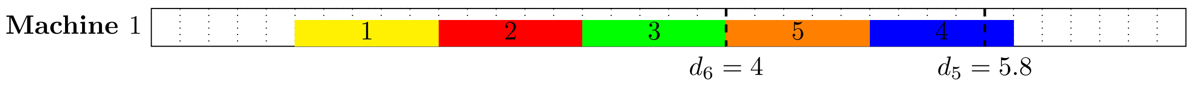

Figure 18: Job sequence $[j_1, j_2, j_3, j_5, j_4]$

We note that scheduling job j_5 at any other position gives $|C_5 - 5.8| \geq 1.2 > 0.8$, and thus gives higher cost. Algorithm 5 proceeds by inserting job j_6 at the first position giving sequence $[j_6, j_1, j_2, j_3, j_5, j_4]$, with optimal schedule illustrated in Figure 19.

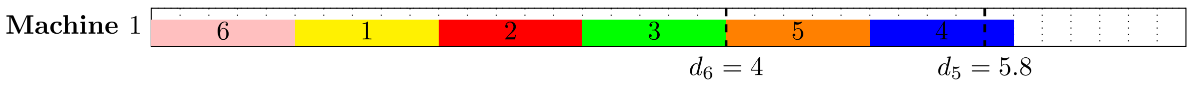

Figure 19: Job sequence $[j_6, j_1, j_2, j_3, j_5, j_4]$

This schedule σ has cost $w_5 \cdot 0.8^2 + w_6 \cdot 3^2 = 6.4 + 81 = 87.4$.

Another feasible schedule is given in Figure 20.

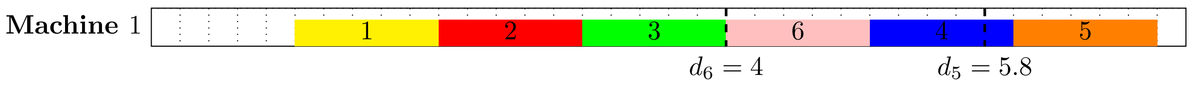

Figure 20: Alternative schedule with $[j_1, j_2, j_3, j_6, j_4, j_5]$

This schedule has cost $w_5 \cdot 1.2^2 + w_6 \cdot 1^2 = 14.4 + 9 = 23.4$, which is lower than schedule σ found by Algorithm 5. Thus Algorithm 5 does not always give an optimal schedule.

6.3 An additive approximation scheme

We give an additive approximation scheme to find a solution with cost close to the optimal cost. Unlike the previously introduced heuristics, this will yield a performance guarantee. The additive approximation scheme will construct many schedules by picking a starting time of the schedule and fixing several idle periods. The starting time of the schedule and the lengths of the constructed idle periods will be multiples of a suitably chosen value α. To know which possible starting times and idle periods suffice, we first need to give further structural properties of idle periods in optimal schedules.

The definition we use of an additive approximation scheme is taken from a paper by Buchem et al. [9].

Definition 6.2. An *additive approximation scheme* is a family of algorithms that finds on any instance I and for every $\epsilon > 0$ a solution with value $A(I)$ satisfying

$$|A(I) - \text{OPT}(I)| \leq \epsilon h,$$

where h is a suitable chosen parameter of instance I.

Often this choice of h arises naturally for a problem. In our case we let $h = W$ with W the maximal weight $W = \max_j\{w_j\}$ of the jobs in the given instance. When all the weights in instance I are multiplied by a certain value λ, both the value $|A(I) - OPT(I)|$ and the value h will be multiplied by λ. Thus, this does not have an effect on the performance guarantee.

In order to derive the additive approximation scheme, we first need some structural remarks on idle periods.

6.3.1 Idle periods and structural properties

The concept of idle periods helps to find the structure of (near) optimal schedules.

Definition 6.3. An *idle period* is a maximal time interval between the processing of two jobs on a machine.

We will give an example of an optimal schedule with an occurrence of an idle period.
We consider 7 jobs to be scheduled with (integral) due dates $d_1 = d_2 = d' = 7$ and
$d_3 = d_4 = d_5 = d_6 = d_7 = d = 5$. Let $w_1 = w_2 = w_3 = 1$ and $w_4 = w_5 = w_6 = w_7 = \epsilon$, where ϵ approaches 0.
To find the optimal schedule, we start by scheduling the first 3 jobs optimally. The block with due date d and the block with due date d' can be scheduled at their optimal starting times given by Equation (3.2). This gives the partial schedule σ' illustrated in Figure 21.

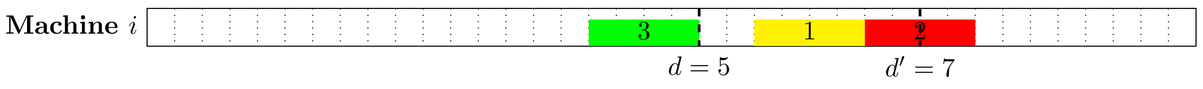

Figure 21: Partial schedule σ'

We note that there is an idle period between job 3 and job 1.

As ϵ approaches 0, the latter 4 jobs will not affect the scheduled starting times of the first 3 jobs. Now the latter 4 jobs will be scheduled before job 3 and after job 2 without any more idle periods (by Property 3.1). Doing this such that these 4 jobs are scheduled as close to d as possible gives the optimal schedule σ^* illustrated in Figure 22.

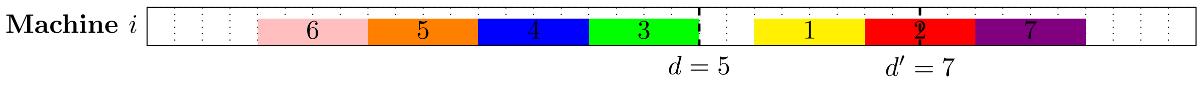

Figure 22: Optimal schedule σ^*

Now we know that idle periods can occur in optimal schedules, we will study where idle periods can occur and with what length.

There will not be an idle period between jobs before the smallest due date (otherwise shifting the jobs before the idle period to the right reduces the value of the objective function). Likewise there will not be an idle period between jobs after the largest due date. Furthermore by Property 3.1 we know that there is no idle time between two consecutively scheduled jobs with the same due date. Related to this property, we also find that there is at most one idle period between two consecutive due dates.

Property 6.4. For any optimal schedule σ^*, there will be at most one idle period intersecting with the interval $(d^\ell, d^{\ell+1})$.

Proof. Let σ^* and ℓ be given such that the theorem does not hold.
As there are multiple idle periods between d^ℓ and $d^{\ell+1}$, there must be a block of jobs B between two idle periods. The last job in B, denoted by j_{last} must have a due date from at most d^ℓ, as otherwise j_{last} can be scheduled later and closer to its due date. Likewise the first job in B denoted by j_{first} must have a due date from at least $d^{\ell+1}$. Thus a pairwise interchange between jobs j_{first} and j_{last} reduces the cost. This contradicts optimality of σ^*. □

Recall that there are no idle periods before d^1 or after d^k in an optimal schedule. Together with Property 6.4, this gives the following corollary.

Corollary 6.5. For any optimal schedule σ^*, there will be at most $k-1$ idle periods.

Then, we consider the length of idle periods. When an idle period $[a, b]$ is of length at least 1 and a job j scheduled before this idle period has due date of at least b, it can be processed in the idle period instead, reducing the cost. Likewise, when a job j scheduled after this idle period has due date of at most a, it can be processed in the idle period instead, again reducing the cost. This gives the following property.

Property 6.6. Let idle period $[a, b]$ be of length 1 or more in optimal schedule σ^*. Every job scheduled before time a must have due date of at most a and every job scheduled after time b must have due date of at least b.

6.3.2 The additive approximation scheme

The additive approximation scheme we introduce, fixes the starting time of the schedule, the length of the blocks, and the positions of the (at most $k-1$) idle periods. This in turn fixes all possible starting and completion times, such that we can use the assignment problem as explained in Section 3.3. This gives an optimal assignment of jobs to time slots, which then specifies a schedule. For the job sequence that is processed in this schedule we can find the optimal schedule with Algorithm 1.

Let $\alpha = \min\{1, \frac{\epsilon}{2n^2}\}$, with $\frac{1}{\epsilon}$ being integral. When $\frac{1}{\epsilon}$ is not integral, we choose ϵ' instead, such that $\frac{1}{\epsilon'} = \lceil \frac{1}{\epsilon} \rceil$. Then we note that $\epsilon' < \epsilon$ and $|A(I) - OPT(I)| \leq \epsilon' W \implies |A(I) - OPT(I)| \leq \epsilon W$.

Given an optimal schedule σ^*, we note that we can round all starting times up to the nearest multiple of α, and this will result in a schedule σ' with cost of at most ϵW more than the optimal cost.

Theorem 6.7. *Let an optimal schedule σ^* be given with cost $OPT(I)$ and the schedule σ' be the result of rounding the starting times of the jobs up to the nearest multiple of α. Schedule σ' has a cost of at most $OPT(I) + \epsilon W$.*

Proof. First, we show that σ' is a feasible schedule. As $1 = 2n^2 \cdot \frac{1}{\epsilon} \cdot \alpha$, the value 1 must be an integer multiple of α. Consider two jobs j and j' such that job j is processed before job j' in schedule σ^*. In schedule σ we have $S_j + 1 \leq S_{j'}$ and as 1 is an integer multiple of α we know $S'_j + 1 \leq S'_{j'}$ in σ'. Thus the machine will process at most one job at a time. Of course the starting times are still non-negative and thus σ' is feasible.

We recall the assumption of positive (rational) due dates. An important observation for the approximation scheme is, that in an optimal schedule for any job j, for $0 < \alpha' \leq \alpha \leq 1$ we have

$$\begin{aligned} ((C_j + \alpha') - d_j)^2 = (C_j - d_j)^2 + 2(C_j - d_j)\alpha' + (\alpha')^2 &\overset{(1)}{\leq} (C_j - d_j)^2 + (n + k - 1)\alpha' + \alpha' \\ &\leq (C_j - d_j)^2 + 2n\alpha' \leq (C_j - d_j)^2 + 2n\alpha, \end{aligned} \tag{6.1}$$

where inequality (1) uses the fact that there can be at most $k-1$ idle time intervals of (the $k-1$ idle periods of Corollary 6.5) after d_j before processing job j. Each of these have length less than 1 as otherwise job j can be scheduled closer to its due date. Finally, the time the machine is not idle the machine must be processing jobs which takes a total of n time.
By rounding up the starting times to the nearest multiple of α, the jobs are shifted at most α to the right. Thus Equation (6.1) gives that the cost for any job increases by at most $w_j \cdot 2n\alpha \leq 2Wn\alpha \leq \frac{W\epsilon}{n}$, thus the total cost increases by at most $n \cdot \frac{W\epsilon}{n} = W\epsilon$. □

When in a schedule, the starting and completion times of jobs are multiples of α, the starting time of the schedule and the length of the idle periods are multiples of α, and vice versa (as the processing time of jobs is 1 which is a multiple of α).

Consider the schedule σ', as specified in Theorem 6.7. In the additive approximation scheme we consider several schedules, including a schedule with the same starting and completion times as σ'. For these starting, and completion times we assign the jobs optimally to these completion times. As a result the found schedule does not have a higher cost than the schedule σ'. We will explain how the additive approximation scheme constructs these schedules.

We let s denote the starting time of the first block, and let $z_i \in \mathbb{N}$ denote the number of jobs in the $i-$th block of jobs for $1 \leq i \leq k$. Then we let ℓ_i denote the length of the i-th period for $1 \leq i \leq k-1$. As explained we let s and $\ell_1, \ell_2, \ldots, \ell_{k-1}$ be multiples of α. We will now further specify which possible values we need to consider for these variables.

First, we note that in an optimal schedule σ^* the first block does not start processing before $d^1 - n$. Recall that $d^1 \in \mathbb{Q}_+$. The first block can't start after d^1+1, as then a job with due date d^1 can be processed earlier, from d^1 to d^1+1 instead, reducing the cost. When rounding the starting time up to a multiple of α, resulting in starting time s, we know $\alpha\lceil \frac{d^1-n}{\alpha} \rceil \leq s \leq \alpha \lceil \frac{d^1+1}{\alpha} \rceil$.

Next, we consider the number of jobs in the first block, given by z_1. As the first idle period can not take place before d^1, we let z_1^{min} be the smallest non-zero value such that $s + z_1^{min} \geq d^1$. This leads to $z_1^{min} \leq z_1 \leq n$.

For the $i-$th block with $i \in \{2, 3, \ldots, k-1\}$, we need to choose its length z_i after the $(i-1)-$th idle period has ended. We consider values up to $n - \sum_{p=1}^{i-1} z_p$. We know the $i-$th block does not fully lay between two consecutive due dates $d^\ell, d^{\ell+1}$ for $\ell = 1, 2, \ldots, k-1$. Otherwise two idle periods would intersect with $(d^\ell, d^{\ell+1})$ contradicting Property 6.4. This gives a lower bound (except when the block starts at a due date), that may be stronger than the lower bound 1. Thus, we consider the resulting lower bound restricting z_i to $1 \leq z_i^{min} \leq z_i$.
For the choice $z_i = n - \sum_{p=1}^{i-1} z_p$, the size of all the blocks up to the $i-$th block is n in total and we stop constructing blocks and idle periods. As a result we have now specified the n starting and completion times for the n jobs. When choosing the length of the $k-$th block, we will always choose length $z_k = n - \sum_{p=1}^{k-1} z_p$, such that n jobs will be processed in total.

After constructing the $i-$th block with $i \in \{1, 2, \ldots, k-1\}$ and with completion time t, we need to choose the length $\alpha \leq \ell_i$ of the $i-$th idle period. When the $i-$th idle period is longer than 1, we know, by Property 6.6, that the already scheduled blocks consist of only jobs with due dates of at most t. We note that this is only possible when the total size of the scheduled blocks is equal to the number of jobs with due date of at most t. The problem to find a schedule for the jobs with due date higher that t, can be seen as a scheduling problem where these jobs may only start at time $t+1$ or later. After a translation of $t+1$ to the left on the due dates, this is equivalent to the non-negativity of the starting times. Thus, when an idle period, starting at time t, is longer than 1, we see the problem after time $t+1$ as a separate instance. We leave the earlier jobs to be assigned at the end, when we assign all jobs to the fixed starting, and completion times.

Summarizing the procedure; we will construct schedules, by choosing the starting time s, and then, alternatingly choosing the length z_i of the $i-$th block and the length ℓ_i of the $i-th$ idle period (which follows the $i-$th block), until the total length of the blocks is n. When the total length of the blocks before the $k-th$ block is less than n, the length of the $k-$th block is chosen such that the total length of these k blocks is n. The possible values of the choices for the variables is specified above. After constructing the blocks and idle periods, all starting times and completion times are fixed. Thus, an optimal solution over these starting, and completion times can be found in $\mathcal{O}(n^3)$ time, by solving a balanced assignment problem, as explained in Section 3.3. Of all these found schedules we choose the schedule with lowest cost. Finally, we will schedule the job sequence in this schedule optimally using Algorithm 1. The resulting schedule σ_ϵ is our final schedule with a cost that is an approximation of the optimal cost.

Theorem 6.8. *The described procedure is an additive approximation scheme, in the sense that, for any $\epsilon > 0$ with $\frac{1}{\epsilon}$ integral, a schedule σ_ϵ is found with cost at most ϵW higher than the cost of an optimal schedule σ^*. Furthermore, the running time of this procedure is $\mathcal{O}((\frac{n^4}{\epsilon})^{k-1} \cdot \frac{n^6}{\epsilon})$.*

Proof. First, we show that this algorithm is indeed an additive approximation scheme.
Let σ' be a schedule as in Theorem 6.7.
As explained, the procedure will find a schedule with the same starting, and completion times, for all jobs, that σ' has. The assignment problem, for these starting, and completion times, is solved up to optimality. Thus the constructed schedule with lowest cost must have cost less than or equal to the cost of σ'. Scheduling the job sequence in this schedule optimally using Algorithm 1, can only reduce the cost and results in schedule σ_ϵ. Theorem 6.7 yields that the procedure finding σ_ϵ is an additive approximation scheme.

By induction we will show this additive approximation scheme has a running time of $\mathcal{O}((\frac{n^4}{\epsilon})^{k-1} \cdot \frac{n^6}{\epsilon})$.

For $k = 1$, we note that we check $\mathcal{O}(\frac{n}{\alpha}) = \mathcal{O}(\frac{n^3}{\epsilon})$ different starting times of the single block. For every starting time we solve the balanced assignment problem in $\mathcal{O}(n^3)$ time. Thus the procedure has a processing time of $\mathcal{O}(\frac{n^6}{\epsilon})$.
As induction hypothesis, we suppose the procedure to have running time of $\mathcal{O}((\frac{n^4}{\epsilon})^{k-1}\frac{n^6}{\epsilon})$, for $k \leq \ell < n$. Then we consider the case $k = \ell + 1$.
Again we check $\mathcal{O}(\frac{n^3}{\epsilon})$ starting times for the first block. Then we consider $\mathcal{O}(n)$ possible lengths of this first block. Combined, this gives $\mathcal{O}(\frac{n^4}{\epsilon})$ possible combinations for the first block.
If the first block is followed by an idle period with length of at least 1, we use Property 6.6 and construct the later blocks and idle periods as a separate problem after which the assignment is done once for all jobs. We only need to consider solving this separate problem when the size of the already constructed blocks is equal to the number of jobs with due date before this idle period. Then, when starting to construct the blocks of this separate problem, the iteration is completed in $\mathcal{O}((\frac{n^4}{\epsilon})^{\ell-1} \cdot \frac{n^6}{\epsilon})$ time, as given by the induction hypothesis.
When the first block is followed by an idle period that is shorter than 1, we consider $\mathcal{O}(\frac{1}{\alpha}) = \mathcal{O}(\frac{n^2}{\epsilon})$ distinct lengths of that idle period. For any of the $\mathcal{O}(\frac{n^2}{\epsilon})$ optional lengths of this first idle period, the complexity to complete the iteration is $\mathcal{O}(((\frac{n^4}{\epsilon})^{\ell-1} \cdot \frac{n^6}{\epsilon})/(\frac{n^3}{\epsilon})) = \mathcal{O}((\frac{n^4}{\epsilon})^{\ell-1} \cdot n^3)$, as the starting time of the second block is already fixed at $s + z_1 + \ell_1$.
Thus after fixing the first block the remaining running time of that iteration is
$\mathcal{O}((\frac{n^4}{\epsilon})^{\ell-1} \cdot \frac{n^6}{\epsilon} + \frac{n^2}{\epsilon} \cdot (\frac{n^4}{\epsilon})^{\ell-1} \cdot n^3) = \mathcal{O}((\frac{n^4}{\epsilon})^{\ell-1} \cdot \frac{n^6}{\epsilon})$. As we have $\mathcal{O}(\frac{n^4}{\epsilon})$ iterations fixing the first block, this gives a total running time of $\mathcal{O}(\frac{n^4}{\epsilon} \cdot (\frac{n^4}{\epsilon})^{\ell-1} \cdot \frac{n^6}{\epsilon}) = \mathcal{O}((\frac{n^4}{\epsilon})^{\ell} \cdot \frac{n^6}{\epsilon})$.
Finally, we run Algorithm 1 once to optimally schedule the job sequence in the constructed schedule with lowest cost, resulting in σ_ϵ. This is done in $\mathcal{O}(n)$ time.
By induction, we may conclude that the running time of the additive approximation scheme, is
$\mathcal{O}((\frac{n^4}{\epsilon})^{k-1} \cdot \frac{n^6}{\epsilon})$, for $1 \leq k \leq n$.

□

We note that the running time of this additive approximation scheme is polynomial in $n, \frac{1}{\epsilon}$, but is exponential in k. This means we have a fully polynomial time additive approximation scheme when k is a constant.

Furthermore, we note that there are a finite number of possible job sequences. Thus, we may consider JS^2 to be the job sequence that, when scheduled optimally, gives cost that is closest to, but not equal to, the optimal cost. Let g be the gap between this cost and the cost of an optimal schedule.

When $\epsilon W < g$, we can conclude that the additive approximation scheme yields an optimal schedule. The value of g is however unknown. When a lowerbound of g is found in further research, this additive approximation scheme can be used as an algorithm solving the general problem up to optimality instead.

7 Concluding remarks

In this thesis we studied the scheduling problem minimizing $\sum_j w_j(C_j - d_j)^2$ for unit sized jobs. This problem can be used to model the fuel use of all vessels passing a lock. We have found many special cases to be polynomially solvable.

Special cases

We have been able to solve all special cases up to optimality. The found results for the special cases are summarized in Table 1.

Table 1: Results for special cases of the scheduling problem

	Complexity in the single machine environment	Complexity in the identical parallel machine environment
The problem for a fixed job sequence	$\mathcal{O}(n)$[1]	$\mathcal{O}(n)$ for m job sequences[1]
The problem for a set T of fixed completion times	$\mathcal{O}(\min\{\lvert T\rvert \cdot n^2, kn^3\})$	$\mathcal{O}(\min\{\lvert T\rvert \cdot n^2, kn^3\})$
The unit weight problem	$\mathcal{O}(k)$, for ordered due dates	$\mathcal{O}(\min\{mk, n\})$, for ordered due dates
The unconstrained single due date problem	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
The constrained single due date problem	$\mathcal{O}(n \log n)$	$\mathcal{O}(m^2 \cdot 2^m n + n \log n)$

For a set of fixed starting, and completion times, the problem can be solved as an assignment problem. For a fixed sequence, we gave a $\mathcal{O}(n)$ block merging algorithm to solve the problem. For m job sequences this algorithm can be used on all job sequences in $\mathcal{O}(n)$ time as well. We showed how to apply the block merging algorithm to unit weight problems. For a single due date we scheduled a single block of jobs on every machine. The order of jobs in these blocks can be found efficiently, except when constrained in the identical parallel machine environment.

The integral scheduling problem is generally easier to solve. In this thesis we have found the results as summarized in Table 2.

Table 2: Results for special cases of the integral scheduling problem

	Complexity in the single machine environment	Complexity in the identical parallel machine environment
The problem for a fixed job sequence	$\mathcal{O}(n)$[1]	$\mathcal{O}(n)$ for m job sequences[1]
The problem for a set T of fixed completion times	$\mathcal{O}(\min\{\lvert T\rvert \cdot n^2, kn^3\})$	$\mathcal{O}(\min\{\lvert T\rvert \cdot n^2, kn^3\})$
The unit weight problem	$\mathcal{O}(k)$, for ordered due dates	$\mathcal{O}(\min\{mk, n\})$, for ordered due dates
The single due date problem	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
The general problem	$\mathcal{O}(kn^3)$	$\mathcal{O}(kn^3)$

These findings resulted either from the found results for general starting, and completion times, or by solving an assignment problem.

[1] Also for general processing times

The general problem

For the general single machine problem, we have given a local search heuristic, a greedy heuristic and an additive approximation scheme, the results of which are summarized in Table 3.

Table 3: Results for the heuristics and approximation scheme

	Complexity	Performance guarantee
The local search heuristic	$\mathcal{O}(n \cdot n!)$	No
The greedy heuristic	$\mathcal{O}(n^3)$	No
The approximation scheme	$\mathcal{O}((\frac{n^4}{\epsilon})^{k-1}\frac{n^6}{\epsilon})$	Within ϵW of optimum

By representing schedules by a job sequence and defining a neighbourhood we have given a local search heuristic. This heuristic has a bad worst case running time. We have also given a greedy heuristic, iteratively inserting jobs into sequences. Finally, we have given an additive approximation scheme, using the position of blocks and idle periods. This additive approximation scheme is exponential in the number of distinct due dates, but polynomial in all other input. Furthermore, the two heuristics and the additive approximation scheme will always give a schedule that is optimal given the sequence of jobs.

Extension of the heuristics to the identical parallel machine environment

We note that the additive approximation scheme can not easily be extended to the identical parallel machine environment. The starting times of the first block on any machine, are not as restricted for a single machine. Furthermore, the consequences of an idle period with length of 1 or more, are different.

The local search heuristic and the greedy heuristic can easily be extended to give schedules with a reasonable cost in the identical parallel machine environment.

In order to do this we let a schedule be represented by m job sequences $JS_1, JS_2, \ldots, JS_m$ for the m machines. These can be optimally scheduled in $\mathcal{O}(n)$ time by running Algorithm 1 on all m job sequences.

The local search heuristic

To extend the definition of a neighbouring schedule, a neighbouring schedule is either a schedule that is the result of a single interchange of jobs within a job sequence, or the result of an insertion of a job in another job sequence. For this extended notion, Algorithm 4 gives a locally optimal schedule for the identical parallel machine environment. We note that a neighbouring schedule can be found after running Algorithm 1 at most twice.

We note that there are $n!$ different ways to order the n jobs in a single job sequence. Then, there are $\mathcal{O}(n^m)$ ways to split the job sequence in m consecutive subsequences for the m machines. As there are $m!$ distinct orders between the m machines, this gives $\mathcal{O}(\frac{n^m}{m!} \cdot n!)$ possible schedules to consider with distinct weight. Each needing $\mathcal{O}(n)$ computation time. Therefore this heuristic can have the long running time of $\mathcal{O}(\frac{n^{m+1}}{m!} \cdot n!)$.

The greedy heuristic

For the parallel machines, we construct the m job sequences by allowing insertions in any of the m job sequences.

At any time there are $\mathcal{O}(m + n) = \mathcal{O}(n)$ possible places to insert a new job, this is the same as in the single machine environment. We keep track of the cost on all m machines. The cost on a machine after an insertion can be found by running Algorithm 1 only on the affected job sequence.
Thus, none of the computational properties is different from the greedy heuristic in the single machine environment. Therefore, the running time of this extension is $\mathcal{O}(n^3)$ as well.

For completeness, the algorithm resulting from this extension is displayed in Appendix D.2.

The result for the two extended heuristics are summarized in Table 4.

Table 4: Results for the extended heuristics

	Complexity	Performance guarantee
The local search heuristic	$\mathcal{O}(\frac{n^{m+1}}{m!} \cdot n!)$	No
The greedy heuristic	$\mathcal{O}(n^3)$	No

Open problems

The results in this thesis leave open a number of possible questions for further research.

1. Find a polynomial algorithm to solve the general scheduling problem for a single machine, or prove this problem is in NP.
2. Find a polynomial algorithm to solve the general scheduling problem in the identical parallel machine environment, or prove this problem is in NP.
3. Find a polynomial algorithm to solve the unconstrained single due date problem in the identical machine environment, or prove this problem is in NP.

Furthermore, the found additive approximation scheme, for the general problem with a single machine, is not polynomial in k. An additive approximation scheme that is fully polynomial when k is part of the input would be preferred.

In the identical parallel machine environment, an open problem is to give an additive approximation scheme.

References

[1] B. Alidaee. Minimizing absolute and squared deviation of completion times from due dates. *Production and Operations Management*, 3(2):133–147, 1994.

[2] U. Bagchi, Y. Chang, and R. Sullivan. Minimizing absolute and squared deviations of completion times with different earliness and tardiness penalties and a common due date. *Naval Research Logistics*, 34(5):739–751, 1987.

[3] U. Bagchi, R. Sullivan, and Y. Chang. Minimizing mean absolute deviation of completion times about a common due date. *Naval Research Logistics Quarterly*, 33(2):227–240, 1986.

[4] U. Bagchi, R. Sullivan, and Y. Chang. Minimizing mean squared deviation of completion times about a common due date. *Management Science*, 33(7):894–906, 1987.

[5] K. Baker and G. Scudder. Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38(1):22–36, 1990.

[6] S. Brundavanam. Completion time variance minimisation on two identical parallel processors. *Computers and Operations Research*, 86, 05 2017.

[7] S. Brundavanam and G. Srinivasan. Minimising mean squared deviation of job completion times about a common due date in multimachine systems. *European Journal of Industrial Engineering*, 5:424–447, 01 2011.

[8] M. Buchem, J. Golak, and A. Grigoriev. Vessel velocity decisions in inland waterway transportation under uncertainty. *European Journal of Operational Research*, 2021.

[9] M. Buchem, L. Rohwedder, T. Vredeveld, and A. Wiese. Additive approximation schemes for load balancing problems. *arXiv*, abs/2007.09333, 2020.

[10] A. Orden D. Votaw. *Symposium on Linear Inequalities and Programming*, chapter The personnel assignment problem, pages 155–163. US Air Force, 1951.

[11] A. Federgruen and G. Mosheiov. Heuristics for multi-machine scheduling problems with earliness and tardiness costs. *Management Science*, 42:1544–1555, 11 1996.

[12] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[13] M. Fredman and R. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[14] M. Garey and D. Johnson. “ strong ” np-completeness results: Motivation, examples, and implications. *Journal of the ACM*, 25(3):499–508, July 1978.

[15] M. Garey and D. Johnson. Computers and intractability. *A Guide to the Theory of NP-Completeness*, 1979.

[16] M. Grötschel, L. Lovász, and A. Schrijver. *Complexity, Oracles, and Numerical Computation*, pages 21–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

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

[18] N. Hall and M. Posner. Earliness-tardiness scheduling problems, I: Weighted deviation of completion times about a common due date. *Operations Research*, 39(5):836–846, 1991.

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

[20] N. Huynh Tuong and A. Soukhal. Some new polynomial cases in just-in-time scheduling problems with multiple due dates. In *2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*, pages 36 – 41, 08 2008.

[21] K. Kianfar and G. Moslehi. A branch-and-bound algorithm for single machine scheduling with quadratic earliness and tardiness penalties. *Computers and Operations Research*, 39(12):2978–2990, 2012.

[22] Y. Kim and C. Yano. Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. *Naval Research Logistics*, 41(7):913–933, 1994.

[23] W. Kubiak. Completion time variance minimization on a single machine is difficult. *Operations Research Letters*, 14(1):49–59, 1993.

[24] H. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[25] G. Mosheiov and U. Yovel. Minimizing weighted earliness–tardiness and due-date cost with unit processing-time jobs. *European Journal of Operational Research*, 172(2):528–544, 2006.

[26] R. Nessah and C. Chu. A lower bound for weighted completion time variance. *European Journal of Operational Research*, 207:1221–1226, 2010.

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

[28] J. Park and S. Boyd. General heuristics for nonconvex quadratically constrained quadratic programming. *arXiv*, 2017.

[29] J. Pereira and Ó. Vásquez. The single machine weighted mean squared deviation problem. *European Journal of Operational Research*, 261(2):515–529, 2017.

[30] L. Ramshaw and R. Tarjan. On minimum-cost assignments in unbalanced bipartite graphs. *HP Research Labs*, 2012.

[31] J. Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical Programming*, 40:59–93, 1988.

[32] M. Saravanan and A. Haq. Single machine scheduling for minimising earliness and tardiness penalties by scatter search approach. *International Journal of Electronic Transport*, 1:5 − 25, 01 2011.

[33] S. Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20, 02 2000.

[34] Y. Sun, X. Li, and J. Fowler. Minimizing weighted completion time variance on homogeneous servers. *61st Annual IIE Conference and Expo Proceedings*, 01 2011.

[35] J. Valente. Beam search heuristics for quadratic earliness and tardiness scheduling. *Journal of the Operational Research Society*, 61(4):620–631, 2010.

[36] J. Valente and R. Alves. Heuristics for the single machine scheduling problem with quadratic earliness and tardiness penalties. *Computers and Operations Research*, 35(11):3696–3713, 2008. Part Special Issue: Topics in Real-time Supply Chain Management.

[37] J. Valente and M Moreira. Greedy randomised dispatching heuristics for the single machine scheduling problem with quadratic earliness and tardiness penalties. *The International Journal of Advanced Manufacturing Technology*, 44:995–1009, 10 2009.

[38] J. Valente, M. Moreira, A. Singh, and R. Alves. Genetic algorithms for single machine scheduling with quadratic earliness and tardiness costs. *International Journal of Advanced Manufacturing Technology*, 54:251–265, 01 2011.

[39] S. Vavasis. *Complexity theory: quadratic programming*, pages 304–307. Springer US, Boston, MA, 2001.

[40] J. Ventura and M. Weng. Minimizing single-machine completion time variance. *Management Science*, 41(9):1448–1455, 1995.

[41] P. Wang, J. He, H. Lui, Y. Shi, Q. Kong, and S. Guo. Tri-skill variant simplex and strongly polynomial-time algorithm for linear programming. *arXiv*, 01 2021.

Appendices

A The assignment problem

In this section the balanced and unbalanced assignment problems are given. These are of use for finding optimal schedules for fixed starting and completion times.

A.1 The balanced assignment problem

In 1951 D.F. Votaw and A. Orden introduced the balanced assignment problem [10]. Later on, a more general assignment problem has been studied. Which we will now give.

Let a set of agents A be given, and a set of tasks T. We want to assign agents to tasks. An edge set $E \subseteq A \times T$, with cardinality $|E| = m$, models which tasks can be done by which agent. When $|A| = |T| = n$, the assignment problem is *balanced*. For each edge $(i, j) \in E$ we have cost c_{ij} and the problem is given by

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} x_{ij} c_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j)\in E} x_{ij} = 1 \text{ for } i \in \{1, 2, \ldots, n\}, \\ & \sum_{i:(i,j)\in E} x_{ij} = 1 \text{ for } j \in \{1, 2, \ldots, n\}, \\ & x_{ij} \geq 0 \text{ for } (i, j) \in E, \\ & x_{ij} \in \mathbb{Z} \text{ for } (i, j) \in E. \end{aligned} \tag{A.1}$$

This assigment problem models, that we seek a maximal matching (of cardinality n), with minimal cost.

The first polynomial algorithm to solve this balanced assignment problem is known as the Hungarian method [24]. Using Fibonacci heaps it is also, the algorithm with the best known running time, given by $\mathcal{O}(mn + n^2 \log n)$ [13].

A.2 The unbalanced assignment problem

When the amount of agents $|A|$ is not equal to the amount of tasks $|T|$, the assignment problem is called *unbalanced*. For unbalanced assignment problems we take $n = \max\{|A|, |T|\}$ and $r = \min\{|A|, |T|\}$. Again we have a set $E \subseteq A \times T$ of edges with cardinality $|E| = m$. Each edge (i, j) has cost c_{ij}.

In this case the goal of the assignment problem is to give a matching of cardinality r, with minimum total cost of its edges. When $n = r$, this is equivalent to formulation (A.1).

For such unbalanced assignment problems, the Hungarian method can be extended and computes an optimal solution in time $\mathcal{O}(mr + r^2 \log r)$ [30].

B The problem formulated as a quadratically constrained quadratic program

We can formulate the general problem as a Quadratically Constrained Quadratic Program (QCQP). To do this we use the following formulation of Quadratically Constrained Quadratic Programs from [28]:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & x^T P_0 x + q_0^T x + r_0 \\ \text{s.t.} \quad & x^T P_i x + q_i^T x + r_i \leq 0 \quad \text{for } i = 1, 2, \dots, m, \end{aligned} \tag{B.1}$$

with $P_i \in \mathbb{R}^{n \times n}$, $q_i \in \mathbb{R}^n$ and $r_i \in \mathbb{R}$ for $i = 1, 2, \dots, m$.

In our problem we want to minimize

$$\sum_j w_j (C_j - d_j)^2 = \begin{pmatrix} C_1 - d_1, & C_2 - d_2, & \dots & C_n - d_n \end{pmatrix} \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & w_n \end{pmatrix} \begin{pmatrix} C_1 - d_1 \\ C_2 - d_2 \\ \vdots \\ C_n - d_n \end{pmatrix}.$$

Thus minimizing $\sum_j w_j (C_j - d_j)^2$ is equivalent to minimizing $x^T P_0 x$ where

$$x = \begin{pmatrix} C_1 - d_1 \\ C_2 - d_2 \\ \vdots \\ C_n - d_n \end{pmatrix}$$

and P_0 is the matrix given by

$$P_0 = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & w_n \end{pmatrix}.$$

We can complete the objective function of the form as in formulation (B.1) by taking $q_0 = 0$ and $r_0 = 0$.

We note that in this case P_0 is a diagonal positive definite matrix. The Hessian matrix of $x^T P_0 x$ is equal to P_0 and therefore also positive definite. Functions with a positive semidefinite Hessian matrix are convex and therefore the objective function of our general problem is convex.

In our general scheduling problem we have non-negativity constraints of the starting times. These constraints are

$$S_i \geq 0 \iff C_i \geq 1 \iff x_i + d_i \geq 1 \iff -x_i + (1 - d_i) \leq 0, \quad \text{for} \quad 1 \leq i \leq n.$$

In formulation (B.1) this corresponds with $P_i = 0$, $q_i = -e_i$ and $r_i = 1 - d_i$ for $i = 1, 2, \dots, n$.
Here e_i denotes the $i-$th standard unit vector. Furthermore, we have the constraints that no two jobs can be processed at the same time, meaning that the completion times must differ by at least 1.
This is equivalent to

$$\begin{aligned} |C_i - C_j| \geq 1 \quad & \iff \quad |(C_i - d_i) + d_i - ((C_j - d_j) + d_j)| \geq 1 \iff (x_i + d_i - (x_j + d_j))^2 \geq 1 \\ & \iff \quad 1 - (x_i - x_j + (d_i - d_j))^2 \leq 0 \quad \text{for } 1 \leq i < j \leq n. \end{aligned}$$

To write our problem in the form of formulation (B.1), we note that

$$1 - (x_i - x_j + (d_i - d_j))^2 = 1 - (x_i^2 + x_j^2 - 2x_i x_j + 2(x_i - x_j)(d_i - d_j) + (d_i - d_j)^2).$$

Let $a_{i,j}$ be the index of this constraint for pair $1 \leq i < j \leq n$.
Thus in formulation (B.1) constraint $a_{i,j}$ corresponds with $P_{a_{ij}} = -(e_i e_i^T - e_i e_j^T - e_j e_i^T + e_j e_j^T)$, using the outer product of standard unit vectors, $q_{a_{ij}} = -2(e_i - e_j)(d_i - d_j)$ and $r_i = 1 - (d_i - d_j)^2$.
The number of constraints of this type is equal to $\binom{n}{2} = \frac{(n-1)n}{2}$.

Thus the total number of constraints is $c = n + \frac{(n-1)n}{2} = \frac{(n+1)n}{2}$, which completes the Quadratically Constrained Quadratic Programming formulation of our general scheduling problem.

Quadratically Constrained Quadratic Programs have been shown to be NP-hard in general [15].
There are, however, multiple special cases of Quadratically Constrained Quadratic Programs that are polynomially solvable. Special cases given in [28] are;

1. QCQP with one variable,
2. QCQP with one constraint,
3. QCQP with one interval constraint,
4. Convex QCQP,
5. QCQP with homogeneous constraints with one negative eigenvalue.

The QCQP formulation of our problem is clearly not of the form of special cases 1, 2 or 3. We can also show, that our QCQP is neither of the form of special case 4. For an optimization problem to be convex, the objective function has to be a convex function and the set of feasible solutions has to be convex.
We can show that our problem is not a convex QCQP in general, by showing that the set of feasible solutions does not have to be convex. We do this by giving a simple counterexample.

Let $n = 2,\ d_1 = d_2 = 0$. In this case we have solutions of the form $x = \begin{pmatrix} C_1 - d_1 \\ C_2 - d_2 \end{pmatrix} = \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$.
Then $x^1 = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$ and $x^2 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$ are feasible solutions for our problem. Consider the convex combination x^3 of x^1 and x^2 given by $x^3 = \frac{1}{2}x^1 + \frac{1}{2}x^2 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$. This solution x^3 is not feasible, as x^3 corresponds to $C_1 = C_2 = 3$. Thus the constraint $|C_1 - C_2| \geq 1$ is not met as $|C_1 - C_2| = 0$.

Finally, special case 5 is only applicable when there are only (non-trivial) constraints of the form $x^T P_i x \leq 0$. The constraints of our QCQP are not of this form and therefore we can conclude that our QCQP is none of these 5 special cases.

Thus, using this approach, we can not determine whether there exists a polynomial algorithm to solve our scheduling problem.

C Formal proofs

C.1 Formal proof of Property 4.3

Proof. It is already given that the optimal schedule consists of a single scheduled block.
For n odd and $d \in \mathbb{Z}$ we have as optimal starting time $t^* = \max\{0, d - \frac{n+1}{2}\}$ for the block, as given in Section 3.2.2 by Equation (3.2), this value of t^* is integral. For even n we have that the optimum $t^* = \max\{0, d - \frac{n+1}{2}\}$ is not integral when $d \geq \frac{n+1}{2}$. As the objective function of the block is symmetric and increasing after its optimum, we get that $t_1 = d - \frac{n+1}{2} - \frac{1}{2} = d - 1 - \frac{n}{2}$ and $t_2 = d - \frac{n+1}{2} + \frac{1}{2} = d - \frac{n}{2}$ are the integral optimal starting times. □

C.2 Optimal starting time after merging for general processing times

We consider blocks $A = [j_1, j_2, \ldots, j_{n_a}]$ and $B = [j_{n_a}, j_{n_a+1}, \ldots, j_{n_a+n_b}]$ to be merged, resulting in block C. Let P_a be the total processing time of block A, and let Π_ℓ be $\sum_{i=1}^{\ell} p_{j_i}$. Then the optimal starting time of block C is given by

$$\begin{aligned}
(\hat{t}_c)_{general} &= \frac{\sum_{i=1}^{n_a+n_b} w_{j_i}(d_{j_i} - \Pi_i)}{W_a + W_b} \\
&= \frac{W_a}{W_a+W_b} \cdot \frac{\sum_{i=1}^{n_a} w_{j_i}(d_{j_i} - \Pi_i)}{W_a} + \frac{W_b}{W_a+W_b} \cdot \frac{\sum_{i=n_a+1}^{n_a+n_b} w_{j_i}(d_{j_i} - \Pi_i)}{W_b} \\
&= \frac{W_a}{W_a+W_b} \cdot (\hat{t}_a)_{general} + \frac{W_b}{W_a+W_b} \cdot \frac{\sum_{i=1}^{n_b} w_{j_{i+n_a}}(d_{j_{i+n_a}} - (\Pi_{i+n_a} - \Pi_{n_a}))}{W_b} \\
&= \frac{W_a}{W_a+W_b} \cdot (\hat{t}_a)_{general} + \frac{W_b}{W_a+W_b} \cdot ((\hat{t}_b)_{general} - \Pi_{n_a}).
\end{aligned}$$

C.3 Improving pairwise interchanges

Theorem C.1. *For a schedule σ and jobs j, j' with $C_{j'} = C_j + \delta$, a pairwise interchange reduces the cost iff*

$$w_j(-2\delta C_j - \delta^2 + 2\delta d_j) \ > \ w_{j'}(-2\delta C_j - \delta^2 + 2\delta d_{j'}).$$

Proof. Jobs j and j' have cost $w_j(C_j - d_j)^2 + w_{j'}(C_j + \delta - d_{j'})^2$ in σ. After a pairwise interchange, jobs j and j' have cost $w_j(C_j + \delta - d_j)^2 + w_{j'}(C_j - d_{j'})^2$. Thus the pairwise interchange reduces the cost iff

$$\begin{aligned}
& w_j(C_j - d_j)^2 + w_{j'}(C_j + \delta - d_{j'})^2 > w_j(C_j + \delta - d_j)^2 + w_{j'}(C_j - d_{j'})^2 \\
\iff \ & w_j(C_j^2 - 2C_jd_j + d_j^2) + w_{j'}((C_j+\delta)^2 - 2(C_j+\delta)d_{j'} + d_{j'}^2) \\
& > \ w_j((C_j+\delta)^2 - 2(C_j+\delta)d_j + d_j^2) + w_{j'}(C_j^2 - 2C_jd_{j'} + d_{j'}^2) \\
\iff \ & w_j(C_j^2 - 2C_jd_j + d_j^2) - w_j((C_j+\delta)^2 - 2(C_j+\delta)d_j + d_j^2) \\
& > \ w_{j'}(C_j^2 - 2C_jd_{j'} + d_{j'}^2) - w_{j'}((C_j+\delta)^2 - 2(C_j+\delta)d_{j'} + d_{j'}^2) \\
\iff \ & w_j(C_j^2 - (C_j+\delta)^2 + 2\delta d_j) \ > \ w_{j'}(C_j^2 - (C_j+\delta)^2 + 2\delta d_{j'}) \\
\iff \ & w_j(-2\delta C_j - \delta^2 + 2d_j) \ > \ w_{j'}(-2\delta C_j - \delta^2 + 2\delta d_{j'}).
\end{aligned}$$

□

D Algorithms

D.1 The block merging algorithm for the integral scheduling problem with unit weights

Again we say that the optimal (rational) starting time t^* conflicts with the (integral) completion time z of the previously scheduled block when $t^* \leq z$. Also the optimal integral starting time is the value of $\lfloor t^* \rfloor$ and $\lceil t^* \rceil$ closest to t^*.

Algorithm 6: The block merging algorithm for the integral scheduling problem with unit weights

```
Initialize the $k$ blocks $B_1, B_2, \ldots, B_k$ with as data the optimal starting times $\hat{t}, t^*$ and the
  number of jobs;
Schedule block $B_1$ at its optimal starting time;
for $\ell = 2, 3, \ldots, k$ do
    ActiveBlock = $B_\ell$;
    while Optimal (rational) starting time of ActiveBlock conflicts with the block directly before
      it in job sequence JS do
        Merge ActiveBlock with the conflicting block creating NewBlock;
        Update data of NewBlock and find its optimal starting time using Equation (3.4);
        ActiveBlock = NewBlock;
    end
    Schedule ActiveBlock at its optimal non-negative integral starting time;
end
```

D.2 The greedy heuristic for identical parallel machines

Algorithm 7: The greedy heuristic algorithm for identical parallel machines

```
Start with job sequences $JS_i = [j_i]$, for $i = 1, 2, \ldots m$. Keep track of the total cost and of the
  cost for each of the machines.
for $\ell = m + 1, m + 2, \ldots, n$ do
    for All possible insertions of job $j_\ell$ do
        Find the cost after insertion by running Algorithm 1 on the affected job sequence.
    end
    Insert job $j_\ell$, in the job sequence, at the position in that job sequence, that result in the
      lowest cost.
end
```