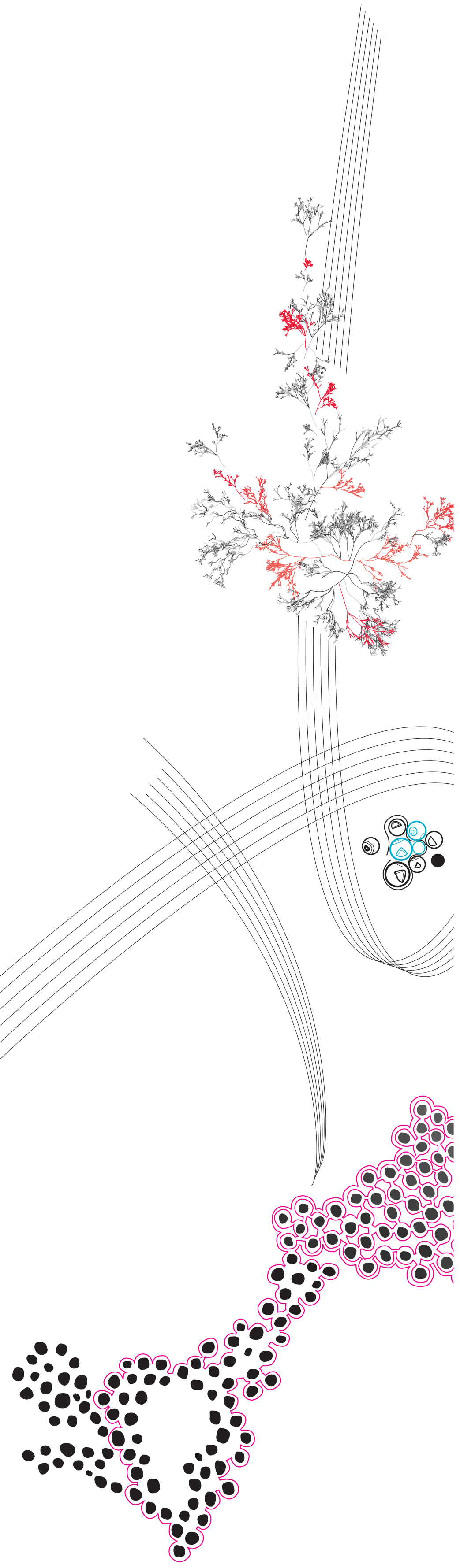
BSc Thesis Applied Mathematics



The Block Structure of Linear Programming Solutions to a Single Machine Scheduling Problem

M.H. Kruimer

Supervisor: Dr. M. Walter

July 6, 2024

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

Integer programming is an effective mathematical tool used for optimizing a linear objective function, subject to linear equality and inequality constraints. The discipline is distinguished from linear programming by the requirement that all decision variables should be integers. Integer programming has widespread applications across various fields including engineering, transportation and manufacturing. Despite its well-established theoretical foundation, efficient solving techniques and new applications of integer programming are still being developed.

This paper delves into a very peculiar solving technique of integer programs, namely using linear programming. Over the past few decades, the development of powerful algorithms have significantly improved the efficiency and scalability of solving integer programs using linear programming methods. This research aims to provide both theoretical and practical insights that demonstrate the structure of these linear programming solutions to the single machine scheduling problem.

I would like to extend my gratitude to Professor Matthias Walter for his continuous support and encouragement.

This paper is structured as follows: Section 1 introduces the research goal and formulates the main research question. Section 2 describes the formulation of the linear program that is used to obtain the results. In Section 3 we derive results based on a weighted starting time objective function, with the ultimate goal to prove an extended version of Smith's Rule. Section 4 dives deeper into the structure of linear programming solutions, as we introduce the concept of minimal blocks, prove a property for schedules to be a vertex solution and introduce a graph algorithm that will be used in the computational experiments in Section 5. The experiments focus on 2 aspects, firstly the relevance of the block structure and secondly the amount of positive decision variables in the obtained solutions. We conclude in Section 6 with a discussion of the results that were found and potential future research in this topic.

It is my hope that this work will contribute to the ongoing dialogue in the field of integer programming and inspire further research and application of linear programming techniques in this area.

The Block Structure of Linear Programming Solutions to a Single Machine Scheduling Problem

M.H. Kruimer

July 6, 2024

Abstract

This paper investigates the structure of linear programming solutions to a single machine scheduling problem. We prove an extension of Smith's rule, a property of schedules that are vertex solutions to the underlying feasible region and perform some computational experiments. The presented results will show the relevance of the so-called block structure in these linear programming solutions.

Keywords: linear programming, (minimal) block, schedule, Smith's rule, vertex, solution structure

Contents

1 Introduction

Integer programming offers powerful techniques for solving complex decision-making problems, distinguished by its requirement that some or all decision variables should be integers. This constraint, to which we refer as the 'integer constraint', is critical in accurately modeling real-world problems where solutions must be whole numbers, such as in scheduling, resource allocation, and network design. The complexity of integer programming makes it a challenging but essential field of study within Operations Research and applied mathematics.

An essential element of this research is the intersection of integer programming with linear programming. While linear programming permits continuous variables and is efficiently solvable with well-known algorithms like the Simplex method [2], integer programming problems are NP-hard [4]. It turns out, however, that linear programming serves as a part of the toolbox of techniques for tackling integer programs.

This paper exemplifies how linear programming can be leveraged to address an integer programming challenge, namely the Single Machine Scheduling Problem. We allow for a relaxation of the integer constraint when scheduling jobs on a machine. Using the model of time discretization we derive results that hold for general objective functions, while we also dive deeper into a special weighted starting time objective function.

Our goal is to provide a comprehensive understanding of how linear programming solutions to this single machine scheduling problem are structured. The main question that we answer in this paper is *What is the structure of a linear programming solution to the single machine scheduling problem?* We derive a generalization for Smith's rule, prove a claim about whether a solution to the problem corresponds to a vertex in the underlying feasible region and construct several computational experiments that improve the understanding of these linear program solutions.

In Section 2 first the single machine scheduling problem will be properly defined. Also some contraints are constructed, which together form a linear program that will be the base of the research in the successive sections. Section 3 discusses a special objective function for the linear program that was constructed in Section 2, namely the weighted starting time objective function. We derive an extension of Smith's rule, which states that an optimal schedule is equivalent to the jobs in that schedule being sorted in non-decreasing order of their ratios. Section 4, which is the last theoretical section, will includes discussion about the structure in the solution to the linear program as defined in Section 2. We define the notion of minimal blocks and show that minimal blocks preserve the property that a solution is a vertex solution. In Section 5 we discuss results of some computational experiments that were done based on the results derived in Section 3 and Section 4. The goal of these experiments is firstly to gain more insight into the relevance of the block structure that is discussed in Section 4, but also to investigate the amount of positive decision variables in the solutions. The latter aims to interpret the relevance of so-called column generating algorithms, which can be used to solve large linear programs. The results of the computational experiments form the base of our conclusion.

2 Formulation of the problem

We consider instances where we schedule a set J of jobs on a machine. Each job should be executed once over a time horizon $\mathcal{T} = \{0, 1, \ldots, T\}$, the machine can handle at most one complete job at a time and jobs cannot be interrupted. Each job $j \in J$ has a *processing time* p_j, which is a positive integer, i.e. $p_j \in \mathbb{Z}_+$. Lenstra, Rinnooy and Brucker (1977) showed that, as soon as arbitrary objective functions are allowed, this variant of the single machine scheduling problem is NP-hard [5].

When scheduling jobs, we compute a starting time for each job. These starting times form a *feasible schedule*, i.e. a schedule that satisfies all requirements stated. A schedule $x \in \mathbb{R}^{J \times \mathcal{T}}$ consists of variables, one for each (j, t) pair, for all $j \in J$ and for all $t \in \mathcal{T}$. The variables $x_{j,t}$, which are called *starting values*, are a way of encoding a schedule, and are defined as follows:

$$x_{j,t} = \begin{cases} 1, & \text{if job } j \text{ is started at time } t \\ 0, & \text{if job } j \text{ is not started at time } t \end{cases} \tag{1}$$

This definition, together with the capacity of the machine, which states that at most one job at a time can be handled, implies that if $x_{j,t} = 1$ for a job $j \in J$, then no other jobs are started in the interval $[t, t + p_j - 1]$. In other words: $x_{j^*,s} = 0$ for all $j^* \in J \setminus \{j\}$ and for $t \leq s < t + p_j$. We call some time t an *idle time* if none of the jobs is being processed at t. In other words, t is an idle time if $\sum_{j \in J} \sum_{t - p_j \leq s \leq t} x_{j,s} = 0$.

In the context of this problem, solutions can only have integer values for their $x_{j,t}$ variables i.e. $x_{j,t} \in \mathbb{Z}$. In this paper however, we will not limit ourselves to the context of this problem and its constraint that all $x_{j,t}$-values should be integers, towards our goal of discussing linear programming solutions. We allow for a relaxation of this binary defined $x_{j,t}$, more specifically, we allow $x_{j,t}$ to have a non-integer value. Because the definition as stated in (1) and the formulation of the problem do not allow $0 < x_{j,t} < 1$, the program that we would be solving is an integer program. The reason for the relaxation of (1) is that a linear program, a program that allows the decision variables to be non-integer, is easier to solve in practice, as several algorithms like the Simplex method are designed to compute solutions to these programs.

The relaxation of (1) as discussed is quite common in practice, as Schulz used a similar technique in 1996 [1] and this was also done by Hall, Schulz, Shmoys and Wein in 1997 [8]. An immediate consequence of this relaxation is that if $0 < x_{j,t} < 1$ for some job j at some time t, that $x_{j,t}$-th part of j is started at time t. This implies that there exists at least one more different $t^* \in \mathcal{T}$, such that $t^* \neq t$, that satisfy $x_{j,t^*} > 0$. Moreover, as job j should be executed fully, we conclude that $\sum_{t \in \mathcal{T}} x_{j,t} = 1$.

Another consequence of the decision to relax the assumption of integer values for $x_{j,t}$ is the appearance of fractional schedules. In a *fractional schedule*, there is at least one (j, t) pair such that $0 < x_{j,t} < 1$. Conversely, in a *non-fractional schedule*, we have that all $x_{j,t} \in \{0, 1\}$ for all $j \in J$ and for all $t \in \mathcal{T}$. It is worth pointing out that a fractional schedule is not a feasible solution to the single machine scheduling problem that we stated at the beginning of this section. Nevertheless, the aim of this paper is to improve our understanding of the linear programming solutions to the single machine scheduling problem, which demands us to discuss the intricacies of these fractional schedules.

Although we allow that a schedule may have multiple positive starting values at some time t, we still cannot exceed the machine's capacity that is set to 1 job at a time. In other words, (2c) should hold for all $t \in \mathcal{T}$. We combine these constraints into a *linear* program, notice that changing (2d) into $x_{j,t} \in \{0,1\}$ would result in an *integer* program that would be a feasible solution to the single machine scheduling problem that we defined.

$$\begin{aligned}
&\min \text{ [objective function]} && && \text{(2a)}\\
&\text{s.t.} && \sum_{t \in \mathcal{T}} x_{j,t} = 1 \quad \forall j \in J && \text{(2b)}\\
& && \sum_{j \in J} \sum_{t-p_j \leq s \leq t} x_{j,s} \leq 1 \quad \forall t \in \mathcal{T} && \text{(2c)}\\
& && x_{j,t} \in [0,1] && \text{(2d)}
\end{aligned}$$

In the subsequent sections we will derive several results using this LP-formulation, which has been introduced in 1992 by Sousa and Wolsey [7]. In these sections we will, amongst others, reason about the feasibility of fractional schedules. As was stated before, a fractional schedule is not a feasible solution to the single machine scheduling problem. However, when a fractional schedule x is called feasible, this means that both (2b) and (2c) are satisfied for this schedule. This allows us to reason about fractional schedules and their characteristics. We start by deriving an extension of Smith's rule in the context of the weighted starting time objective function substituted in (2a).

3 Weighted Starting Time

In this section we will derive a useful result for a specific variant of the single machine scheduling problem, namely the weighted starting time formulation. To define the objective function of the corresponding linear program, we define the *weight* of a job j, denoted as w_j, to be a non-negative number for all $j \in J$. The weighted starting time problem is the problem of finding a schedule in which each w_j is multiplied with the t for which $x_{j,t} > 0$ holds. The *cost of a schedule* $C(x)$ is a function that has as argument the schedule x, as defined in Section 2. The cost is determined as follows:

$$C(x) = \sum_{j \in J} \sum_{t \in \mathcal{T}} x_{j,t} \cdot t \cdot w_j \tag{3}$$

The goal of this scheduling problem is to find an optimal schedule, i.e. a schedule with minimal cost. This means that (3) will be substituted in (2a). An *optimal schedule* x^* that schedules J jobs has cost $C(x^*)$ such that $C(x^*) \leq C(x)$ for all schedules x that schedule the J jobs, in other words, all x such that $\sum_{t \in \mathcal{T}} x_{j,t} = 1$ for all $j \in J$, also see (2b). As can be concluded from (1) and (3), starting (part of) a job contributes to the total cost. Moreover, because the cost depends on t, we would like to schedule the jobs as soon as possible, while of course not exceeding the machine capacity as stated in (2c). The following observation follows immediately.

Observation 3.1. *An optimal non-fractional schedule only has idle times after all jobs have been scheduled.*

We define the *ratio of a job* as $r_j = \frac{p_j}{w_j}$ for $j \in J$. We say that jobs are *sorted in non-decreasing order of their ratios* when the jobs with a lower ratio are scheduled before jobs with a higher ratio. This implies that for all jobs $j, j^* \in J$ in a schedule x such that $r_j > r_{j^*}$ we have that, if $x_{j,t} > 0$ and $x_{j^*,t^*} > 0$ both hold, then $t > t^*$ for $t, t^* \in \mathcal{T}$. Similarly, when jobs are *sorted in non-increasing order of their ratios*, we schedule the jobs with a high ratio before the jobs with a low ratio. In the mathematical formulation above we would have that $t < t^*$ holds. Note that jobs in a fractional schedule can also be sorted in non-decreasing or non-increasing order of their ratios.

3.1 Job Swapping

We are interested in the cost of an optimal schedule, but first we provide a condition that will later prove to be sufficient for a schedule being optimal. This condition relies on the notion of *swapping* two jobs in a non-fractional schedule. Suppose a non-fractional schedule that only has idle times after all jobs have been scheduled, schedules jobs j_i for all $i \in \{1, \ldots, n\}$ such that job j_1 is scheduled first, job j_2 is started whenever job j_1 ended, and so forth. Now we can represent the schedule by the sequence:

$$j_1, j_2, \ldots, j_{n-1}, j_n.$$

Now swapping jobs j_p and j_q, where $1 \leq p < q \leq n$, results in the sequence:

$$j_1, \ldots, j_{p-1}, j_q, j_{p+1}, \ldots, j_{q-1}, j_p, j_{q+1}, \ldots, j_{n-1}, j_n.$$

In other words, after job j_{p-1} has ended, we start job j_q and job j_{p+1} is started after job j_q has been executed. We now derive the difference of the cost of a schedule when swapping two jobs that are adjacent in the schedule. We call two jobs *adjacent* when one of these jobs is started immediately when some part of the other job has been finished. In other words, if j_p and j_q are adjacent in a schedule x, then there is a $t \in \mathcal{T}$ such that $x_{j_p,t} > 0$ and $x_{j_q,t+p_{j_p}} > 0$.

Lemma 3.2. *In a non-fractional schedule, swapping two adjacent jobs with the same ratio will not change the cost, swapping adjacent jobs that are sorted in increasing order of their ratios, increases the cost and swapping adjacent jobs that are sorted in decreasing order of their ratios, decreases the cost.*

Proof. Consider a non-fractional schedule x that only has idle times after all jobs $j_1, \dots, j_n$ have been scheduled. Denote with t_i the starting time of job j_i, for $i \in \{1, \dots, n\}$. Without loss of generality assume that $t_1 < t_2 < \dots < t_n$. We now show the difference in cost after any swap of adjacent jobs.

Suppose that any two adjacent jobs j_p and j_{p+1} in x are swapped to obtain schedule x^*, where $1 \leq p \leq n-1$. For all jobs except j_p and j_{p+1} we know that x and x^* have the same values. For the jobs j_p and j_{p+1} we know that $x \neq x^*$ only for $t = t_p$, $t = t_p + p_{j_{p+1}}$ or $t = t_{p+1}$. In other words:

$$\begin{aligned} x_{j,t} &= x^*_{j,t} && \forall j \in J \setminus \{j_p, j_{p+1}\}, \forall t \in \mathcal{T} \\ x_{j,t} &= x^*_{j,t} && \text{for } j \in \{j_p, j_{p+1}\}, \forall t \in \mathcal{T} \setminus \{t_p, t_{p+1}, t_p + p_{j_{p+1}}\} \end{aligned}$$

Because of the swap of j_p and j_{p+1} we know that x and x^* have these values for these jobs at the times $t \in \{t_p, t_{p+1}, t_p + p_{j_{p+1}}\}$:

$$\begin{aligned} x_{j_p,t_p} &= 1 && \text{and} && x^*_{j_p,t_p} = 0 \\ x_{j_p,t_{p+1}} &= 0 && \text{and} && x^*_{j_p,t_{p+1}} = 0 \\ x_{j_p,t_p+p_{j_{p+1}}} &= 0 && \text{and} && x^*_{j_p,t_p+p_{j_{p+1}}} = 1 \\ x_{j_{p+1},t_p} &= 0 && \text{and} && x^*_{j_{p+1},t_p} = 1 \\ x_{j_{p+1},t_{p+1}} &= 1 && \text{and} && x^*_{j_{p+1},t_{p+1}} = 0 \\ x_{j_{p+1},t_p+p_{j_{p+1}}} &= 0 && \text{and} && x^*_{j_{p+1},t_p+p_{j_{p+1}}} = 0 \end{aligned}$$

We can calculate the cost difference between x and x^* using these values:

$$\begin{aligned} C(x^*) - C(x) &= \sum_{j \in J} \sum_{t \in \mathcal{T}} x^*_{j,t} \cdot t \cdot w_j - x_{j,t} \cdot t \cdot w_j \\ &= x^*_{j_{p+1},t_p} \cdot t_p \cdot w_{j_{p+1}} + x^*_{j_p,t_p+p_{j_{p+1}}} \cdot (t_p + p_{j_{p+1}}) \cdot w_{j_p} \\ &\qquad - x_{j_p,t_p} \cdot t_p \cdot w_{j_p} - x_{j_{p+1},t_{p+1}} \cdot t_{p+1} \cdot w_{j_{p+1}} \\ &= t_p \cdot w_{j_{p+1}} + (t_p + p_{j_{p+1}}) \cdot w_{j_p} - t_p \cdot w_{j_p} - t_{p+1} \cdot w_{j_{p+1}} \\ &= w_{j_p} \cdot (t_p + p_{j_{p+1}} - t_p) + w_{j_{p+1}}(t_p - t_{p+1}) \\ &= p_{j_{p+1}} \cdot w_{j_p} - p_{j_p} \cdot w_{j_{p+1}}. \end{aligned}$$

If we now assume that $r_{j_p} > r_{j_{p+1}} \implies \frac{p_{j_p}}{w_{j_p}} > \frac{p_{j_{p+1}}}{w_{j_{p+1}}} \implies p_{j_{p+1}} \cdot w_{j_p} - p_{j_p} \cdot w_{j_{p+1}} < 0$, i.e. $C(x^*) < C(x)$. We conclude that if j_p and j_{p+1} were sorted in decreasing order of their ratios that swapping them will decrease the cost. It also follows that $r_{j_p} < r_{j_{p+1}} \implies$

$p_{j_{p+1}} \cdot w_{j_p} - p_{j_p} \cdot w_{j_{p+1}} > 0$, i.e. $C(x^*) > C(x)$. We conclude that jobs if j_p and j_{p+1} were sorted in increasing order of their ratios, swapping them will increase the cost. Lastly, if we suppose that $r_{j_p} = r_{j_{p+1}}$, we conclude that $p_{j_p} \cdot w_{j_{p+1}} - w_{j_p} \cdot p_{j_{p+1}} = 0$, i.e. $C(x^*) = C(x)$, hence swapping adjacent jobs with the same ratio will not change the cost. □

3.2 Smith's Rule

We can now make an interesting claim about optimality in non-fractional schedules. We state the following Lemma, which is also known as *Smith's rule* [9].

Lemma 3.3. *A non-fractional schedule is optimal among all non-fractional schedules if and only if the jobs in a non-fractional schedule are sorted in non-decreasing order of their ratios.*

Proof. We first prove the "only if"-direction.

Suppose for contradiction that schedule x is optimal but that the jobs in x are not sorted in non-decreasing order of their ratios. In x, there must be at least one pair of adjacent jobs that are sorted according to decreasing order of their ratios. By Lemma 3.2 we conclude that the cost of x can be decreased by swapping these adjacent jobs, this contradicts that x is optimal.

We now prove the "if"-direction.

Consider a schedule x where jobs are sorted in non-decreasing order of their ratios. Now suppose that x^* is a non-fractional schedule that is optimal among all non-fractional schedules. By the "only if"-direction we know that the jobs in x^* must be sorted in non-decreasing order of their ratios. This implies that schedule x^* can be obtained by a series of swaps between adjacent jobs with the same ratio executed in x. By Lemma 3.2, we know that these swaps do not change the cost of x, which implies that $C(x) = C(x^*)$, we conclude that x is optimal. □

At the end of this section we aim to derive a more general result, more specifically, the goal is to show that Lemma 3.3 can also be applied to fractional schedules.

3.3 Job Splitting

In order to ultimately arrive at this conclusion, we now introduce the notion of *splitting a job*, which means that we replace a job j in an instance with p_j-many jobs j_i such that $p_{j_i} = 1$ and $w_{j_i} = \frac{w_j}{p_j}$, for $i \in \{1, \ldots, p_j\}$. If some schedule contained j with $p_j > 1$ that was fully or partly executed at time $t \in \mathcal{T}$, i.e. $x_{j,t} = q$ for $q \in (0, 1]$, splitting j results in a new schedule x^*. The new schedule does not contain job j, i.e. the variable $x^*_{j,t}$ does not exist, but instead contains jobs j_i, such that $x^*_{j_i,t_i} = q$, where $t_i = t + i - 1$, for $i \in \{1, \ldots, p_j\}$ and for all $t \in \mathcal{T}$ such that $x_{j,t} > 0$. In turns out that the property that jobs are sorted in non-decreasing order of their ratios is preserved after splitting a job. This is stated in the following proposition:

Proposition 3.4. *The jobs in a fractional or non-fractional schedule are sorted in non-decreasing order of their ratios after splitting a job in a schedule in which the jobs are sorted in non-decreasing order of their ratios.*

Proof. Consider that schedule x^* is obtained by splitting job j in schedule x, where x is a schedule in which J jobs are sorted in non-decreasing order of their ratios. This means that in x^* jobs j_i are scheduled with $p_{j_i} = 1$ and $w_{j_i} = \frac{w_j}{p_j}$ for $i \in \{1, \ldots, p_j\}$. It immediately follows that $r_{j_i} = \frac{p_{j_i}}{w_{j_i}} = \frac{p_j}{w_j} = r_j$.

It is known that $x^*_{j_i,t_i} = x_{j,t}$ for time $t_i = t - 1 + i$, for $i \in \{1, \dots, p_j\}$ for all $t \in \mathcal{T}$ such that $x_{j,t} > 0$. This implies, using the fact that $r_{j_i} = r_j$ and that $x_{j',t} = x^*_{j',t}$ for all $j \in J \setminus \{j\}$ for all $t \in \mathcal{T}$, that if the jobs in x are sorted in non-decreasing order of their ratios, that the jobs in x^* are sorted in non-decreasing order of their ratios. □

Using Lemma 3.3, the next observation immediately follows:

Observation 3.5. *Splitting a job in an optimal non-fractional schedule results in an optimal non-fractional schedule.*

We now know that optimality is preserved after a job has been split in a non-fractional schedule. This does not necessarily imply that the cost of a schedule after splitting a job has in- or decreased with regards to the original instance. It turns out that this increase in cost can be derived for both fractional and non-fractional schedules.

Lemma 3.6. *Splitting a job j in a fractional or non-fractional schedule increases the cost by $\sum_{i=0}^{p_j-1} i \cdot \frac{w_j}{p_j}$.*

Proof. Consider a fractional or non-fractional schedule x and denote with x^* the schedule after job j has been split. This means that j is replaced by jobs j_i in x^*, such that $x^*_{j_i,t_i} = q$, where $t_i = t + i - 1$, for $i \in \{1, \dots, p_j\}$, for all $t \in \mathcal{T}$ such that $x_{j,t} = q$.

We now calculate the difference in cost between these schedules, $C(x^*) - C(x)$. Because all other jobs $j' \in J \setminus \{j\}$ have not been changed or rescheduled when splitting job j, we conclude that

$$\begin{aligned}
C(x^*) - C(x) &= \sum_{j \in J} \sum_{t \in \mathcal{T}} x^*_{j,t} \cdot t \cdot w_j - \sum_{j \in J} \sum_{t \in \mathcal{T}} x_{j,t} \cdot t \cdot w_j \\
&= \sum_{i=1}^{p_j} \sum_{t \in \mathcal{T}} x^*_{j_i,t_i} \cdot t_i \cdot w_{j_i} - \sum_{t \in \mathcal{T}} x_{j,t} \cdot t \cdot w_j = \sum_{t \in \mathcal{T}} (\sum_{i=1}^{p_j} x^*_{j_i,t_i} \cdot t_i \cdot w_{j_i} - x_{j,t} \cdot t \cdot w_j) \\
&= \sum_{t \in \mathcal{T}} (\sum_{i=1}^{p_j} t_i \cdot w_{j_i} - t \cdot w_j) = \sum_{t \in \mathcal{T}} (\sum_{i=1}^{p_j} ((t + i - 1) \cdot \frac{w_j}{p_j}) - t \cdot w_j) \\
&= \sum_{t \in \mathcal{T}} (t \cdot w_j + \sum_{i=0}^{p_j-1} (i \cdot \frac{w_j}{p_j}) - t \cdot w_j) = \sum_{i=0}^{p_j-1} i \cdot \frac{w_j}{p_j}
\end{aligned}$$

□

3.4 Exchanging Starting Values

We now consider a special case of the set of jobs J, namely that $p_j = 1, \forall j \in J$. In other words, we consider schedules where all job have the same unit processing time. Before we can start proving some necessary condition for a fractional schedule to be optimal, we first state a sufficient condition, similar to Lemma 3.2, but generalized to fractional schedules as well. To ultimately end up at this goal we introduce the notion of *exchanging starting values* between two jobs over some time period by some value. When exchanging starting values of jobs j_p and j_q over the time period t_p, t_q with the value M in a schedule x, this creates a new schedule x^* where the starting value of job j_p at time t_p is decreased by M, but at time t_q is increased by M, in other words we have $x^*_{j_p,t_p} = x_{j_p,t_p} - M$ and $x^*_{j_p,t_q} = x_{j_p,t_q} + M$. The opposite happens for job j_q, its starting value decreases at t_q and increases at t_p by M, this implies that $x^*_{j_q,t_p} = x_{j_q,t_p} + M$ and $x^*_{j_q,t_q} = x_{j_q,t_q} - M$. A necessary condition for this exchange of starting values is of course that $x_{j_p,t_p} > M$ and $x_{j_q,t_q} > M$ both hold.

We now introduce the vector operator $\Psi : \mathbb{Q}^{J\times T} \to \mathbb{R}$ such that $\Psi x = M$ holds for the biggest M such that for all $x_{j,t} \in X = \{x_{j,t} | x_{j,t} > 0\}$, we have that $x_{j,t} = k \cdot M$ for some $k \in \mathbb{Z}$. The existence of such an M is proven in the next proposition and would make sure that in finitely many steps we can exchange starting values to obtain any schedule x^* that satisfies for all $x^*_{j,t} \in X^* = \{x^*_{j,t} | x^*_{j,t} > 0\}$, we have that $x^*_{j,t} = k^* \cdot M$ for some $k^* \in \mathbb{Z}$. In other words with such an M, we would have that $x_{j,t} \in \frac{1}{M}\mathbb{Z}$ and $x^*_{j,t} \in \frac{1}{M}\mathbb{Z}$ if $x_{j,t} \in X$ and $x^*_{j,t} \in X^*$.

It turns out that if we assume that $x \in \mathbb{Q}^{J\times T}$, that the following proposition guarantees the existence of such an M.

Proposition 3.7. *For each finite set of rationals $x \in \mathbb{Q}$ there exists an M such that each element in the set is in $\frac{1}{M}\mathbb{Z}$.*

Proof. Consider a set $X \in \mathbb{Q}$ such that $X = \{x_1, x_2, \dots, x_n\}$. We can write each x_i as $\frac{a_i}{b_i}$ for $i \in \{1, \dots, n\}$ and for $a_i, b_i \in \mathbb{Z}$. We now set $M = \operatorname{lcm}\{b_i | 1 \leq i \leq n\}$, which implies that $x_i = \frac{a_i}{b_i} = \frac{a_i \cdot k_i}{b_i \cdot k_i} = \frac{a_i \cdot k_i}{M}$, where k_i is an integer for $i \in \{1, \dots, n\}$. $a_i \cdot k_i \in \mathbb{Z}$ which implies that $x_i \in \frac{1}{M}\mathbb{Z}$. □

An immediate consequence of Proposition 3.7 is that we now know that for all sets of schedules $X = \{x_{j,t} | j \in J, t \in T\}$ such an M exists. This will help us proving the following result:

Lemma 3.8. *Exchanging starting values between two adjacent jobs in a fractional or non-fractional schedule where all jobs have unit processing times will not change the cost if the jobs have the same ratios, increases the cost if the jobs are sorted in increasing order of their ratios and decreases the cost if the jobs are sorted in decreasing order of their ratios.*

Proof. Consider a schedule x where n jobs $j_1, \dots, j_n$ with $p_{j_i} = 1$ for all $i \in \{1, \dots, n\}$ are scheduled. Suppose schedule x^* is obtained by exchanging starting values of any two adjacent jobs j_p and j_q, over some time period $t_p, t_{p+1} = t_p, t_p + 1$ by M, where $M = \Psi x$, for some $1 \leq p \leq n$ and $1 \leq q \leq n$. Note that for the choice of p and q we require that $x_{j_p,t_p} > 0$ and $x_{j_q,t_p+1} > 0$, such that these jobs are really adjacent. By the choice of M we know that both $x_{j_p,t_p} \geq M$ and $x_{j_q,t_p+1} \geq M$.

It is known that x and x^* are similar for jobs $j \in J \setminus \{j_p, j_q\}$ and for $t \in \mathcal{T} \setminus \{t_p, t_p + 1\}$, in other words:

$$\begin{aligned} x_{j,t} &= x^*_{j,t} && \text{for } j \in J \setminus \{j_p, j_q\}, \text{ for } t \in \mathcal{T} \\ x_{j,t} &= x^*_{j,t} && \text{for } j \in \{j_p, j_q\}, \text{ for } t \in \mathcal{T} \setminus \{t_p, t_p + 1\} \end{aligned}$$

For the jobs j_p and j_q we have the following starting values at $t \in \{t_p, t_p + 1\}$:

$$\begin{aligned} x_{j_p,t_p} &\geq M && \text{and} && x^*_{j_p,t_p} = x_{j_p,t_p} - M \geq 0 \\ x_{j_p,t_p+1} &\geq 0 && \text{and} && x^*_{j_p,t_p+1} = x_{j_p,t_p+1} + M > 0 \\ x_{j_q,t_p} &\geq 0 && \text{and} && x^*_{j_q,t_p} = x_{j_q,t_p} + M > 0 \\ x_{j_q,t_p+1} &\geq M && \text{and} && x^*_{j_q,t_p+1} = x_{j_q,t_p+1} - M \geq 0 \end{aligned}$$

It is worth to note that if x is feasible, which would imply that $x_{j_p,t_p} \leq 1$ and $x_{j_q,t_p+1} \leq 1$, we know that x^* is feasible as well, as this follows from $x_{j_p,t_p+1} \leq 1 - M$ and $x_{j_q,t_p} \leq 1 - M$.

Having established that x^* still is a feasible fractional or non-fractional schedule, we can now calculate the difference in cost between x and x^*:

$$\begin{aligned}
C(x^*) - C(x) &= \sum_{j\in J}\sum_{t\in\mathcal{T}} \left(x^*_{j,t}\cdot t\cdot w_j - x_{j,t}\cdot t\cdot w_j\right)\\
&= x^*_{j_p,t_p}\cdot t_p\cdot w_{j_p} + x^*_{j_p,t_p+1}\cdot(t_p+1)\cdot w_{j_p} + x^*_{j_q,t_p}\cdot t_p\cdot w_{j_q}\\
&\quad + x^*_{j_q,t_p+1}\cdot(t_p+1)\cdot w_{j_q} - x_{j_p,t_p}\cdot t_p\cdot w_{j_p} - x_{j_p,t_p+1}\cdot(t_p+1)\cdot w_{j_p}\\
&\qquad - x_{j_q,t_p}\cdot t_p\cdot w_{j_q} - x_{j_q,t_p+1}\cdot(t_p+1)\cdot w_{j_q}\\
&= (x^*_{j_p,t_p} - x_{j_p,t_p})\cdot t_p\cdot w_{j_p} + (x^*_{j_p,t_p+1} - x_{j_p,t_p+1})\cdot(t_p+1)\cdot w_{j_p}\\
&\quad + (x^*_{j_q,t_p} - x_{j_q,t_p})\cdot t_p\cdot w_{j_q} + (x^*_{j_q,t_p+1} - x_{j_q,t_p+1})\cdot(t_p+1)\cdot w_{j_q}\\
&= -M\cdot t_p\cdot w_{j_p} + M\cdot(t_p+1)\cdot w_{j_p} + M\cdot t_p\cdot w_{j_q} - M\cdot(t_p+1)\cdot w_{j_q}\\
&= M(w_{j_p} - w_{j_q}).
\end{aligned}$$

Since $p_j = 1$, we have that $r_j = \frac{1}{w_j}$ for all $j \in J$. Now suppose that the jobs j_p and j_q had equal ratio, we conclude that $C(x^*) - C(x) = 0$, hence the cost has not changed. If jobs j_p and j_q were sorted in increasing order of their ratios however, we would have that $\frac{1}{w_{j_q}} > \frac{1}{w_{j_p}} \implies w_{j_p} - w_{j_q} > 0$, which implies that $C(x^*) - C(x) > 0$, the cost of x^* is bigger, hence the cost of x has increased. Similarly we conclude that if j_p and j_q were sorted in decreasing order of their ratios, that the cost of the schedule x^* has decreased. □

It turns out that optimality of a fractional or non-fractional schedule where all jobs have unit processing time follows in a similar manner as Lemma 3.3. We state and prove this in the following Lemma:

Lemma 3.9. *A fractional or non-fractional schedule where all jobs have unit processing times is optimal if and only if the jobs are sorted in non-decreasing order of their ratios.*

Proof. We first prove the "only if"-direction.

Suppose for contradiction that schedule x is optimal but that the jobs in x are not sorted in non-decreasing order of their ratios. In x, there must be at least one pair of adjacent jobs that are sorted according to decreasing order of their ratios. By Lemma 3.8 we conclude that the cost of x can be decreased by exchanging the starting values of these adjacent jobs with $M = \Psi x$, this contradicts that x is optimal.

We now prove the "if"-direction.

Consider a schedule x where jobs are sorted in non-decreasing order of their ratios. Now suppose that x^* is a schedule that is optimal among all fractional and non-fractional schedules. By the "only if"-direction we know that the jobs in x^* must be sorted in non-decreasing order of their ratios. This implies that schedule x^* can be obtained from x by a series of exchanges of starting values of adjacent jobs with the same ratio by $M^* = \Gamma\begin{bmatrix}\Psi x\\ \Psi x^*\end{bmatrix}$, where $\Gamma : \mathbb{R}^2 \to \mathbb{R}$ returns the lowest M^* such that $M^* = \Psi x\cdot k$ and $M^* = \Psi x^*\cdot k^*$ for some integers $k, k^* \in \mathbb{Z}$ in x. By Lemma 3.2, we know that these exchanges do not change the cost of x, which implies that $C(x) = C(x^*)$, we conclude that x is optimal. □

3.5 Optimality in Fractional and Non-Fractional Schedules

We now may extend the statements as in Lemma 3.3 and in Lemma 3.9 to a more general, combined version. It is noteworthy that Theorem 3.10 has been proven before by Van den

Akker in 1994 [6], the existence of this proof came to my attention only after the proof presented here was established. Nevertheless, the proof in this paper provides an intuitive approach to the problem, using the notions of splitting jobs and exchanging starting values. The proof provided by Van den Akker considers dual programs of the LP provided in (2a)-(2d).

Theorem 3.10. *A fractional or non-fractional schedule is optimal if and only if the jobs in that schedule are sorted in non-decreasing order of their ratios.*

Proof. We first prove the "if"-direction.

Consider a schedule x that schedules J jobs. Let n be the number of jobs in x that have a processing time of at least 2. In other words: $n = |\{j \in J | p_j > 1\}|$. Suppose that the jobs in x are sorted in non-decreasing order of their ratios. We now prove that x is optimal, by induction on n. For the base case we suppose $n = 0$. By Lemma 3.9 we can immediately conclude that x is optimal. We now assume that for $n = k$ the statement does hold, i.e. x is optimal. As an induction step we now set $n = k + 1$. We denote with x' an optimal schedule that schedules the same jobs as x, note that both x and x' schedule $k + 1$ jobs that have a processing time of at least 2. x' being optimal implies that $C(x') \leq C(x)$. We now split one of the jobs j' in both schedules, for which it holds that $p_{j'} > 1$. Let x_{new} be the schedule after splitting j' in x and x'_{new} be the schedule after splitting j' in x'. Note that both x_{new} and x'_{new} now contain k jobs with a processing time of at least 2. By Proposition 3.4 the jobs in x_{new} are sorted in non-decreasing order of their ratios if the jobs in x are sorted in non-decreasing order of their ratios. By the induction hypothesis we conclude that x_{new} must be optimal. By Lemma 3.6 we know that $C(x'_{\text{new}}) = C(x') + \sum_{i=0}^{p_{j'}-1} i \cdot \frac{w_{j'}}{p_{j'}} \leq C(x) + \sum_{i=0}^{p_{j'}-1} i \cdot \frac{w_{j'}}{p_{j'}} = C(x_{\text{new}})$. Because $C(x_{\text{new}})$ is optimal we conclude that $C(x_{\text{new}}) = C(x'_{\text{new}}) \implies C(x') = C(x)$. We conclude by optimality of x' that x is optimal. This completes the induction step and the proof, as we have shown that a fractional or non-fractional schedule where the jobs are ordered in non-decreasing order of their ratios is optimal.

We now prove the "only if"-direction.

Again, let n be the number of jobs in a schedule x that have a processing time of at least 2. In other words: $n = |\{j \in J : p_j > 1\}|$. Suppose that x is optimal. We now prove that the jobs in x are sorted in non-decreasing order of their ratios, by induction on n. For the base case we suppose $n = 0$. By Lemma 3.9 we can immediately conclude that the jobs in x are sorted in non-decreasing order of their ratios. We now assume that for $n = k$ the statement does hold, i.e. the jobs in x are sorted in non-decreasing order of their ratios. As an induction step we now set $n = k + 1$. We denote with x' a schedule that schedules the same jobs as x such that the jobs in x' are sorted in non-decreasing order of their ratios. Note that both x and x' schedule $k + 1$ jobs that have a processing time of at least 2. By the "if"-direction, we know that x' is optimal, which implies that $C(x') = C(x)$. We now split one of the jobs j' in both schedules, for which it holds that $p_{j'} > 1$. Let x_{new} be the schedule after splitting j' in x and x'_{new} be the schedule after splitting j' in x'. Note that both x_{new} and x'_{new} now contain k jobs with a processing time of at least 2. By Proposition 3.4 the jobs in x'_{new} are sorted in non-decreasing order of their ratios if the jobs in x' are sorted in non-decreasing order of their ratios. By the "if"-direction we conclude that x'_{new} is optimal. By Lemma 3.6 we know that $C(x'_{\text{new}}) = C(x') + \sum_{i=0}^{p_{j'}-1} i \cdot \frac{w_{j'}}{p_{j'}} = C(x) + \sum_{i=0}^{p_{j'}-1} i \cdot \frac{w_{j'}}{p_{j'}} = C(x_{\text{new}})$. We conclude that $C(x_{\text{new}})$ must be optimal as well. By the induction hypothesis we conclude that the jobs in x_{new} are sorted in non-decreasing order of their ratios, which implies that the jobs in x are sorted in non-decreasing order of their ratios as well. $\square$

4 Block Structures

In this section we start by defining what we mean with the block structure of a solution to the single machine scheduling problem. After proving a few useful results we are ready to establish the relevance of this block structure by doing some computational experiments.

4.1 Minimal Blocks

Because we are interested in the solution structure of the problem as stated in Section 2, we start by defining a block. A *block* in a schedule x is a subset $B \subseteq \mathcal{T}$ where B is an interval over the integers, such that each job is either fully started and completed in B or not started in B. This means that if $x_{j,t} > 0$ for some job j and $t \in B$, then $x_{j,t^*} = 0$ if either $t^* \notin B$ or $t^* + p_j - 1 \notin B$. As B is an interval we have that if $a \in B$ and $c \in B$ and we have $a < b < c$, then $b \in \mathcal{T} \implies b \in B$. Consequently, we have that $\sum_{t \in B} x_{j,t} = 0$ or $\sum_{t \in B} x_{j,t} = 1$ for all jobs j.

Observation 4.1. *Every feasible schedule contains at least one block.*

Consider a schedule x scheduling J jobs that contains a block B, then in this block a subset $J^* \subseteq J$ jobs are scheduled. Scheduling jobs J^* over the interval $B \subseteq \mathcal{T}$ is a single machine scheduling problem of itself. Such a schedule, which schedules the jobs in a block on the time horizon of that block, is called a *subschedule of a block.* By the definition of a block, we know that for each subschedule constraints (2b) and (2c) are satisfied. As Observation 4.1 implies, a schedule itself is a block. Before we can make some general statements about blocks, we would like to extend the definition of a block, to separate blocks that do strictly contain other blocks and blocks that do not, in other words blocks that are as “small" as possible. A block is a *minimal block* if it does not strictly contain any block, i.e. none of the strict subsets of the block are blocks. This brings us to the following observation.

Observation 4.2. *The time horizon of every block can be partitioned into minimal blocks and a set of times at which the machine is idle.*

In Proposition 4.6 we discuss an algorithm that computes this partitioning efficiently. We call such a partition, consisting of minimal blocks and idle times, a *minimal block decomposition.* Also related to Observation 4.1, we can conclude that because each feasible schedule is a block, that Observation 4.2 also applies to schedules, i.e. each schedule has a minimal block composition, something that will be useful in a later proof.

4.2 Vertex Schedules

A schedule x is called a *vertex schedule* if it is a unique minimizer over some linear function of the underlying polyhedron of the linear program. This name comes from the fact that such a schedule corresponds to a vertex of the feasible region of the underlying single machine scheduling problem. The next observation immediately follows from this definition, which can be applied to fractional and non-fractional schedules.

Observation 4.3. *A schedule is a vertex schedule if and only if it cannot be written as a convex combination of two different schedules.*

Something that follows from this definition and Observation 4.3 is that each non-fractional schedule is a vertex schedule. As we are interested in the solution structure of the solution to the single machine scheduling problem, we aim to provide a condition for a schedule to be a vertex schedule. We can show that vertex schedules are built out of blocks that are vertex schedules.

Theorem 4.4. *A schedule is a vertex schedule if and only if the subschedule of every minimal block that is contained in the schedule is a vertex schedule within its minimal block.*

Proof. We first prove the "only if" direction. By Observation 4.2 vertex schedule x over $\mathcal{T}$ can be divided into k minimal blocks B_i with corresponding subschedules $x^{(i)}$, for $i \in \{1, \dots, k\}$ such that $\mathcal{T} = (\bigcup_{i=1}^{k} B_i) \cup \mathcal{T}^*$, where $\mathcal{T}^*$ is the set of times at which the machine is idle. Suppose now for contradiction that there is a block B_j that is not a vertex schedule. This implies by Observation 4.3 that for $t \in B_j$, subschedule $x^{(j)}$ is a convex combination of two different schedules $x^{(j,1)}$ and $x^{(j,2)}$ that schedule the jobs in B_j over the time interval B_j. In other words, there exists a λ such that $0 < \lambda < 1$ and $x^{(j)} = \lambda x^{(j,1)} + (1-\lambda)x^{(j,2)}$. We conclude that

$$x = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(j)} \\ \vdots \\ x^{(k)} \end{bmatrix} = \begin{bmatrix} x^{(1)} \\ \vdots \\ \lambda x^{(j,1)} + (1-\lambda)x^{(j,2)} \\ \vdots \\ x^{(k)} \end{bmatrix} = \lambda \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(j,1)} \\ \vdots \\ x^{(k)} \end{bmatrix} + (1-\lambda) \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(j,2)} \\ \vdots \\ x^{(k)} \end{bmatrix}$$

for some λ such that $0 < \lambda < 1$. Hence x is a convex combination of two schedules and not a vertex schedule, we have reached a contradiction.

We now prove the "if" direction. Again, schedule x over $\mathcal{T}$ can be divided into k minimal blocks B_i and a set of times at which the machine is idle. Denote the corresponding subschedules of the blocks as $x^{(i)}$ for $i \in \{1, \dots, k\}$. Because $B_1, ..., B_k$ are vertex schedules, we know that for each block B_i there exists a c_i such that $x^{(i)}$ is the unique c_i-minimizer for $i \in \{1, \dots, k\}$. We now prove that x is the unique c^*-minimizer for some c^* that we construct. For this, we use the $b(j)$-function, which takes a job j as parameter and returns i, when j is scheduled in block B_i, for $i \in \{1, \dots, k\}$. We now introduce

$$M = \begin{cases} 1, & \text{if } \{x_{j,t}^{(i)} | x_{j,t}^{(i)} > 0\} = \varnothing \\ \min_{j \in J, t \in \mathcal{T}, i \in \{1,\dots,k\}} \{x_{j,t}^{(i)} | x_{j,t}^{(i)} > 0\}, & \text{else} \end{cases}$$

Because there is a finite number of positive $x_{j,t}^{(i)}$, we know that the minimum value of all these variables must be finite, which implies that $M > 0$. Let

$$c^*(x) = \sum_{j \in J, t \in \mathcal{T}} \bar{c}_{j,t}^{\intercal} x_{j,t} + \varepsilon \sum_{i=1}^{k} c_i^{\intercal} x^{(i)},$$

where

$$\bar{c}_{j,t} = \begin{cases} 0, & \text{if } t \in B_{b(j)} \\ 2T^2 \cdot |J| \cdot \frac{1}{M}, & \text{else} \end{cases}$$

$$\varepsilon = [2k \cdot \max_{1 \le i \le k} c_i^{\intercal} x^{(i)}]^{-1} \tag{4}$$

We now consider some other schedule $x' \neq x$ that schedules the same jobs J as x. We consider two cases, firstly that x' does not have the same block structure as x and secondly that the block structures of x' and x correspond.

Suppose that x' that does not have the same block structure as x, in other words there is a $t' \in \mathcal{T}$ and a job j such that $x'_{j,t'} \geq M > 0$ and $[t', t' + p_j - 1] \notin B_{b(j)}$. Then

$$\begin{aligned}\sum_{j \in J, t \in \mathcal{T}} \bar{c}_{j,t} \cdot x'_{j,t} = \bar{c}_{j,t'} \cdot x'_{j,t'} &= 2T^2 \cdot |J| \cdot \frac{1}{M} \cdot x'_{j,t'} \\ &\geq 2T^2 \cdot |J| \cdot \frac{1}{M} \cdot M = 2T^2 \cdot |J|.\end{aligned}$$

Recall that using (4) we can conclude that:

$$c^*(x') = \sum_{j \in J, t \in \mathcal{T}} \bar{c}_{j,t} \cdot x'_{j,t} + \varepsilon \sum_{i=1}^{k} c_i x'^{(i)} \geq 2T^2 \cdot |J| \geq 1.$$

Because $\bar{c}_{j,t} = 0$ for all $t \in B_{b(j)}$, we know that by construction of ε in (4), we have that:

$$\begin{aligned}c^*(x) &= \sum_{j \in J, t \in \mathcal{T}} \bar{c}_{j,t} x_{j,t} + \varepsilon \sum_{i=1}^{k} c_i x^{(i)} = \varepsilon \sum_{i=1}^{k} c_i x^{(i)} \\ &= [2k \cdot \max_{1 \leq i \leq k} c_i^\intercal x^{(i)}]^{-1} \sum_{i=1}^{k} c_i x^{(i)} \\ &\leq [2k \cdot \max_{1 \leq i \leq k} c_i^\intercal x^{(i)}]^{-1} \sum_{i=1}^{k} \max_{1 \leq i \leq k} c_i^\intercal x^{(i)} \\ &= [2k \cdot \max_{1 \leq i \leq k} c_i^\intercal x^{(i)}]^{-1} \cdot k \cdot \max_{1 \leq i \leq k} c_i^\intercal x^{(i)} = \frac{1}{2}\end{aligned}$$

This implies that $c^*(x') > c^*(x)$. We conclude that each c^*-minimizer must schedule job j in block $B(j)$, i.e. it should have the same block structure as x.

Now suppose that x' with subschedules $x'^{(i)}$ for $i \in \{1, \dots, k\}$ such that $\sum_{j \in J, t \in \mathcal{T}} \bar{c}^\intercal_{j,t} x'_{j,t} = \sum_{j \in J, t \in \mathcal{T}} \bar{c}^\intercal_{j,t} x_{j,t} = 0$. In other words, in x' each block is scheduled in $b(j)$.

Because $x' \neq x$, we know that there exists an $p \in \{1, \dots, k\}$ such that $x'^{(p)} \neq x^{(p)}$. This implies that $c_p^\intercal x^{(p)} < c_p^\intercal x'^{(p)}$, as $x^{(p)}$ is the unique c_p-minimizer. As $x^{(i)}$ is the unique c_i-minizimer for all $i \in \{1, \dots, k\}$, we can conclude that $c_i^\intercal x^{(i)} \leq c_i^\intercal x'^{(i)}$ for all $i \in \{1, \dots, k\}$, which implies that $\sum_{i=1}^{k} c_i^\intercal x^{(i)} < \sum_{i=1}^{k} c_i^\intercal x'^{(i)}$, which means that x is the unique c^*-minimizer and hence must be a vertex schedule. □

4.3 Graph of a Minimal Block Decomposition

To be able to measure the relevance of the so-far discussed features of block structures, we are interested in constructing some algorithm that computes the minimal block decomposition of a schedule. It turns out that graph theory allows us to define an efficient algorithm that solves this problem. The pseudocode for this algorithm is written in Algorithm 1.

Algorithm 1 *MinimalBlockDecomposition*(x)

```
Variables: $x_{j,t}$ for $j \in J$ for $t \in \mathcal{T}$
Create graph $G = (J, \varnothing)$
for $j \in J$ do
    $t_{j,\min} := \min_{t\in\mathcal{T}}\{x_{j,t} | x_{j,t} > 0\}$
    $t_{j,\max} := \max_{t\in\mathcal{T}}\{x_{j,t} | x_{j,t} > 0\} + p_j - 1$
    Interval $I_j := [t_{j,\min}, t_{j,\max}]$
    for $k \in J \setminus j$ do
        if $I_j \cup I_k$ is non-empty then
            Add the edge $\{j, k\}$
    end
end
Compute the connected components $V_1$, $V_2$, ... $V_m$ of $G$ using BFS
return $V_1$, $V_2$, ... $V_m$
```

The complexity of the graph algorithm as written in Algorithm 1 is stated in the following observation:

Observation 4.5. *The graph algorithm as defined in Algorithm 1 terminates in $O(|J|^2)$-time.*

Before we can start using *MinimalBlockDecomposition* in a computational setting, we first prove that the algorithm as written in Algorithm 1 does work correctly. This leads us to the following proposition:

Proposition 4.6. *Algorithm 1 correctly returns minimal block decomposition of a schedule.*

Proof. If suffices to show that jobs j_i and j_k belong in the same minimal block if $|I_i \cup I_k| \geq 1$, where I_i and I_k are the intervals as defined in Algorithm 1.

Consider a schedule x that schedules n jobs j_i, for $i \in 1, \ldots, n$. Assume that for two jobs j_i and j_k, such that $1 \leq k < i \leq n$ we have that $|I_i \cup I_k| \geq 1$. Now suppose for contradiction that x has a minimal block decomposition consisting of a set of idle times $\mathcal{T}^*$ and H minimal blocks B_h for $h \in \{1, \ldots, H\}$ such that $x_{j_i,t_i} > 0$ for some $t_i \in B_{h_i}$ and $x_{j_k,t_k} > 0$ for some $t_k \in B_{h_k} \neq B_{h_i}$. Because B_{h_i} and B_{h_k} are part of the minimal block decomposition of x, we know that $B_{h_i} \cup B_{h_k} = \varnothing$, they can not contain other blocks. By the definition of a minimal block, we have that $\sum_{t\in B_{h_i}} x_{j_i,t} = 1$ and $\sum_{t\in B_{h_k}} x_{j_k,t} = 1$, hence we can assume without loss of generality that $t_k \leq t_i$ for all t_k, t_i such that $x_{j_k,t_k} > 0$ and $x_{j_i,t_i} > 0$. Moreover, we can state that if $t_i \in B_{h_i}$ and $t_k \in B_{h_k}$ then $t_i + p_{j_i} \in B_{h_i}$ and $t_k + p_{j_k} \in B_{h_k}$. The fact that $|I_i \cup I_k| > 1$ implies that $t_{k,\max} = \max_{t\in\mathcal{T}}\{x_{j_k,t} | x_{j_k,t} > 0\} + p_{j_k} - 1 \geq \min_{t\in\mathcal{T}}\{x_{j_i,t} | x_{j_i,t} > 0\} = t_{i,\min}$. We conclude that $t_{k,\max} \geq t_{i,\min}$ and that $t_{k,\max} \in B_{h_i} \cup B_{h_k}$, this contradicts that $B_{h_i} \cup B_{h_k} = \varnothing$. We conclude that j_i and j_k must be scheduled inside the same minimal block. This implies that the number of components of the graph as created in Algorithm 1 is the number of minimal blocks and that the size of such a component equals the number of jobs scheduled inside that minimal block. □

We are now ready to perform some computational experiments, the results of which are denoted in the next section.

5 Computational Experiments

We design a computational experiment with the main goal to obtain more insight into the relevance of the block structure that was discussed in the previous sections. The linear program that we construct is:

$$\begin{aligned} \min \sum_{j\in J}\sum_{t\in\mathcal{T}} & x_{j,t}(\lambda \cdot t \cdot w_j + (1-\lambda)\cdot u_{j,t}) \\ \text{s.t.} \qquad & (2b), (2c), (2d) \end{aligned}$$

We set up this linear program in the Gurobi module integrated in Python, also using the Gurobi module to solve the LP [3]. For this objective function we have that $0 \leq \lambda \leq 1$, the variables $U_{j,t}$ and w_j are drawn independently from a uniform distribution over $[-1, 1]$ and $[0, 1]$ respectively and $p_j \sim \text{Uniform}(1, 5)$. Furthermore we calculate T after generating the processing times p_j, that is $T = \sum_{j\in J} p_j$. This is not very relevant for a pure weighted starting time formulation, as for bigger T the last few slots will always be empty. Using the randomly distributed $u_{j,t}$-variables however, implies that schedules might contain idle times.

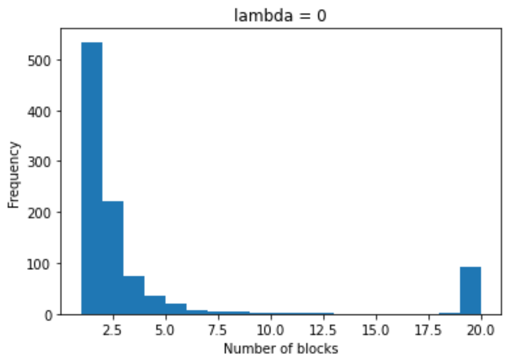
One of the experiments that we execute is varying the value of λ between 0 and 1. Running the linear program for multiple instances and recording the results of the graph algorithm as defined in Algorithm 1 will result in data on the number of minimal blocks that schedules contain and the number of jobs per block. Because the objective function is a convex combination of the weighted starting time objective function and some random $u_{j,t}$ dependent function, by varying the λ we expect to gain more insight into the relative importance of the block structure in weighted starting time formulated programs.

Another experiment that we perform is for some fixed values of λ that the number of positive $x_{j,t}$-variables in a schedule will be stored. For these specific λ values, we will be varying the length of the time-horizon T. As stated before, in combination with the randomly generated part of the objective function, we expect to witness some different scheduling behaviour whenever the time-horizon is bigger than the sum of all the processing times. This is interesting, because recent developments have shown that so-called column-generating algorithms are a powerful tool to help solve large linear programs. As the speed of the column-generating algorithm depends on the number of positive decision variables, it is convenient to have an idea of the size of the number of positive decision variables.

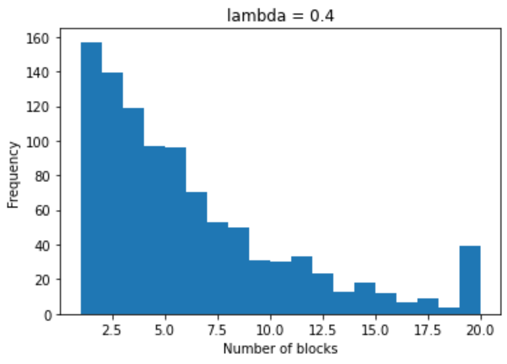
5.1 Relevance of Block Structures

We choose $\lambda \in \{0, 0.4, 0.6, 0.8, 0.9, 1\}$. This way, we hope to envision the influence of the objective function on the number of blocks. We run $N = 1000$ instances with $n = 20$ jobs for each λ. For each instance we record the number of blocks in the minimal block decomposition of the schedule, as proven in Lemma 4.6. The result is shown in Figure 1, which has Subfigures 1a until 1f.

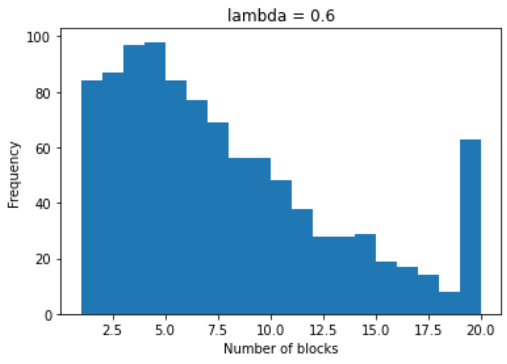
In the histograms one can easily see that the higher the λ, the higher the average of the number of blocks. Recall that in case of $\lambda = 0$, the objective function represents a completely randomized schedule. Concluding from Figure 1a, we can say that such an objective function results in a low number of minimal blocks in the minimal block decomposition, as the linear program will find a way to efficiently schedule jobs over multiple starting values. Once we start increasing λ, we can see that the number of blocks is a lot more spread over the possible $[1, 20]$-interval. In Figure 1b the most frequent encountered



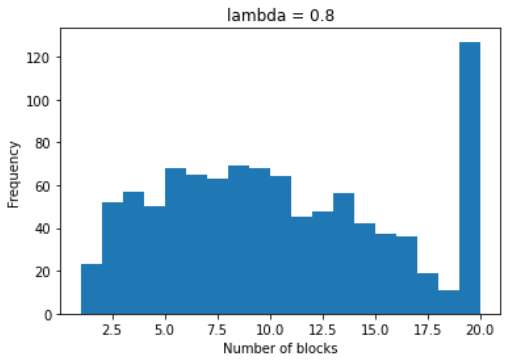
(A) The number of blocks in a schedule where the λ-value $= 0$



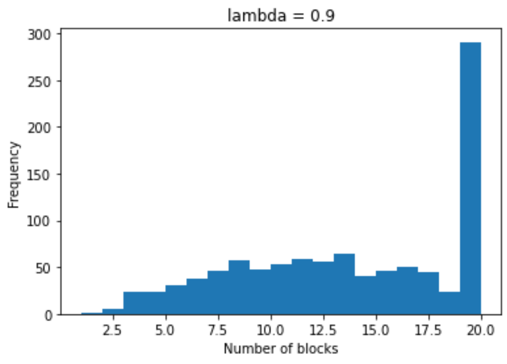
(B) The number of blocks in a schedule where the λ-value $= 0.4$



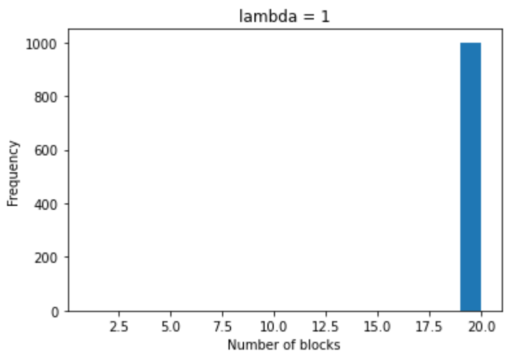
(C) The number of blocks in a schedule where the λ-value $= 0.6$



(D) The number of blocks in a schedule where the λ-value $= 0.8$



(E) The number of blocks in a schedule where the λ-value $= 0.9$



(F) The number of blocks in a schedule where the λ-value $= 1$

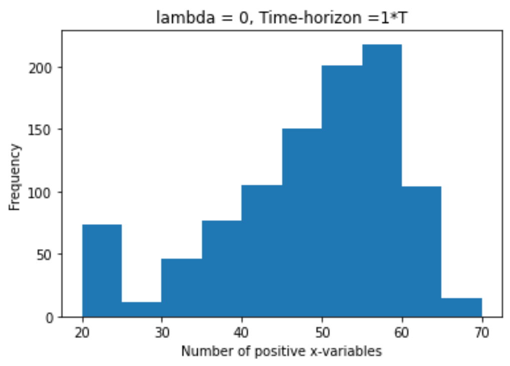
FIGURE 1: The number of blocks as recorded after running N instances of n jobs

number of blocks per schedule is still 1, but this changes when increasing λ to 0.6, this is visible in Figure 1c. The peak has shifted a bit towards 4 or 5 minimal blocks per schedule on average. It is interesting to see that the more influence that the weighted starting time objective function has, i.e. the higher λ is, that the peak at 20 starts growing. At some point, the influence of the weighted starting time objective is so big that most of the jobs are scheduled fully at one specific time, i.e. $x_{j,t} = 1$ for some $t \in \mathcal{T}$. In Subfigure 1d we can see that the peak at 20 is a lot higher than at any other number of blocks, while the other bins show some resemblance of a normal "bell-curve". When the λ-value approaches 1, in Subfigures 1e and 1f, we can observe that almost to all of the jobs are scheduled inside their 'own' block, significantly increasing the number of blocks in the minimal block decomposition of a schedule.

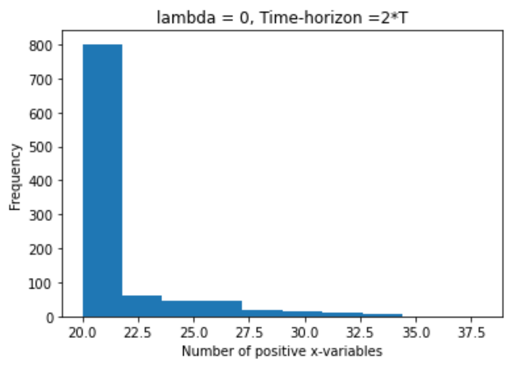
5.2 Influence of the Size of the Time Horizon

We choose for λ the following values: 0, 0.4 and 0.8, because the case with $\lambda = 1$ is not interesting, as stated before. For each λ, we run an instance with $T = \sum_{j\in J} p_j$ and an instance where the time horizon is equal to $2T$. We hope to envision the influence of the size of the time-horizon on the number of positive $x_{j,t}$-variables. We run $N = 1000$ instances with $n = 20$ jobs for each combination of λ and T. For each instance we record the number of positive x-variables. The result is shown in Figure 2, which has Subfigures 2a until 2f.

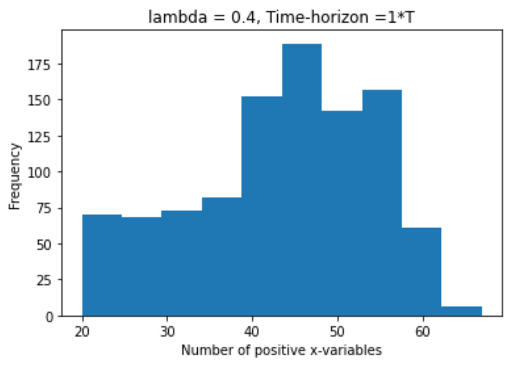
Something that was quite expected in these histograms is that the change in time-horizon has less effect for larger λ, in other words for a weighted starting time objective function, the size of the time-horizon does not matter as much. The difference in the Figures 2a and 2b is striking. Doubling the time-horizon has as effect that almost all of the jobs are scheduled at one time point, whereas the peak was around 55 in case of the normal T. We can safely say that the number of positive x-variables for a random objective function decreases significantly with the time-horizon increasing. This is not the case when the weighted starting time objective comes into play. In Figures 2c and 2d we see that the number of positive x-variables is more centered around one point, around 50, once the time-horizon is doubled. We can observe that in the last two Subfigures, 2e and 2f, the number of posititve x-variables is increasing after doubling the time-horizon. We conclude that for a low λ, that increasing the T yields less positive x-variables, while for a higher λ, increasing the T results in more positive x-variables.



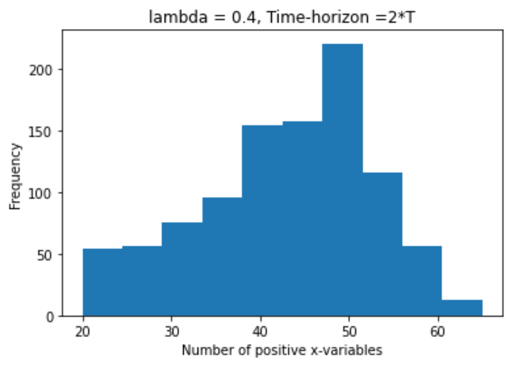
(A) The number of positive x-variables in a schedule where the λ-value $= 0$ and the time-horizon is T



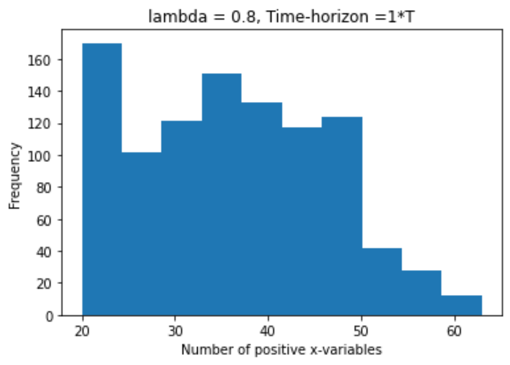
(B) The number of positive x-variables in a schedule where the λ-value $= 0$ and the time-horizon is $2T$



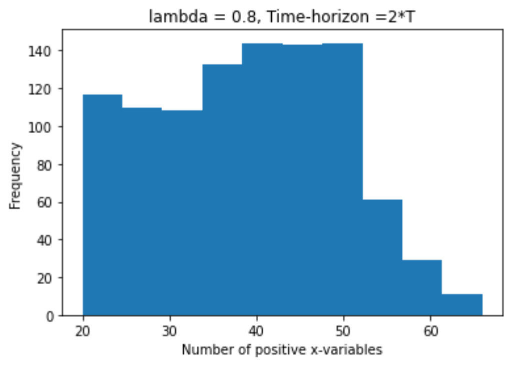
(C) The number of positive x-variables in a schedule where the λ-value $= 0.4$ and the time-horizon is T



(D) The number of positive x-variables in a schedule where the λ-value $= 0.4$ and the time-horizon is $2T$



(E) The number of positive x-variables in a schedule where the λ-value $= 0.8$ and the time-horizon is T



(F) The number of positive x-variables in a schedule where the λ-value $= 0.8$ and the time-horizon is $2T$

FIGURE 2: The number of positive $x_{j,t}$-variables as recorded after running N instances of n jobs

6 Conclusion

The results obtained show a variety of structures present in the linear programming solutions to the single machine scheduling problem. Using the weighted starting time objective function and some interesting techniques we could prove some intriguing properties of so-called fractional schedules. The main result in this section is that it was shown that Smith's rule can be extended to linear programming solutions. We also introduced the concept of a block, a way to grasp the structure of solutions to the single machine scheduling problem. Using this concept we were able to prove a nice result, namely that minimal blocks in a schedule preserve the property of being a vertex solution. Lastly, with some computational experiments we were able to deduce the relationship between the weighted starting time objective function and the number of minimal blocks in a schedule, as well as the link between the size of the time horizon and the number of positive variables in a solution to the single machine scheduling problem. These relations are interesting, as they not only provide insight into the relevance of the block structure that was discussed in Section 4, but also shed more light onto the performance of so-called column-generating algorithms that are a promising tool to solve large linear programs.

In further research, one might expect several other results regarding the block structure in linear programming solutions to be proven. It remains to show what properties are preserved by the blocks in a schedule and whether this block structure is relevant at all in different scheduling problems. More specifically, the concepts as introduced in this paper could be applied to different types of objective functions, for example by introducing deadlines or release dates. The relevance of the introduced block structure in these cases might be able to better estimate the relevance of the block structure in linear programming solutions as a whole. It is also not unlikely that in future research more specific bounds for the number of positive decision variables in a linear programming solution will be derived, as the technique of solving large linear programs using column-generating algorithms still has a long way ahead of it.

References

[1] A. S. Schulz. "Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds". en. In: *Integer Programming and Combinatorial Optimization*. Ed. by W. H. Cunningham, S. T. McCormick, and M. Queyranne. Berlin, Heidelberg: Springer, 1996, pp. 301–315. ISBN: 978-3-540-68453-4. DOI: 10.1007/3-540-61310-2_23.

[2] G. B. Dantzig. "Origins of the simplex method". In: *A history of scientific computing*. New York, NY, USA: Association for Computing Machinery, June 1990, pp. 141–151. ISBN: 978-0-201-50814-7. URL: https://doi.org/10.1145/87252.88081.

[3] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2024. URL: https://www.gurobi.com.

[4] H. W. Lenstra. "Integer Programming with a Fixed Number of Variables". In: *Math. Oper. Res.* 8.4 (Nov. 1983), pp. 538–548. ISSN: 0364-765X. DOI: 10.1287/moor.8.4.538.

[5] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. "Complexity of Machine Scheduling Problems". In: *Annals of Discrete Mathematics*. Ed. by P. L. Hammer et al. Vol. 1. Studies in Integer Programming. Elsevier, Jan. 1977, pp. 343–362. DOI: 10.1016/S0167-5060(08)70743-X.

[6] J. M. van den Akker. "LP-based solution methods for single-machine scheduling problems". Phd Thesis 1 (Research TU/e / Graduation TU/e). Eindhoven: Technische Universiteit Eindhoven, 1994. DOI: 10.6100/IR428838.

[7] J. P. Sousa and L. A. Wolsey. "A time indexed formulation of non-preemptive single machine scheduling problems". en. In: *Mathematical Programming* 54.1 (Feb. 1992), pp. 353–367. ISSN: 1436-4646. DOI: 10.1007/BF01586059.

[8] L. A. Hall et al. "Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms". In: *Mathematics of Operations Research* 22.3 (1997). Publisher: INFORMS, pp. 513–544. ISSN: 0364-765X. URL: https://www.jstor.org/stable/3690391.

[9] W. E. Smith. "Various optimizers for single-stage production". en. In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 59–66. ISSN: 1931-9193. DOI: 10.1002/nav.3800030106.