UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

FREDERICO CORREA

Minimizing the weighted number of late jobs on a single machine with equal processing times and agreeable release and due dates

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Advisor: Prof. Dr. Marcus Ritt

Porto Alegre
October 2022

ACKNOWLEDGEMENTS

First and foremost, my sincere and greatest thanks to my advisor, Prof. Dr. Marcus Rolf Peter Ritt, for all knowledge shared and all guidance provived for my accomplishing of this work. It was also Prof. Ritt introduced me to many scheduling problems and made the suggestion as main topic of my final graduation work. Along these many years of continued developing, I was able to learn too much from him, about computer science, generic science-related topics, and a bit about life as well. For his continued support and understanding along all these years, I here express my deepest feelings of gratitude.

My sincere thanks to the many teachers I've met along the path to graduation, for their unmatched domain of knowledge, and their utmost competence in teaching about it. And for this top-notch level of education provided, I express my sincere thanks for Universidade Federal do Rio Grande do Sul (UFRGS) and the Instituto de Informática (INF) as well.

My very special thanks to Danúsia Braeske Pulla, "Dadá", my best friend in the last thirty years or so, and to "Tia Benta" as well, to whom I owe just too much to try and describe with words.

Many people were instrumental in making possible for me to be here, in more than one way: my numerous family, my friends from both the real life and the Internet, with special mentions for "estevam", Caio ("oiacz"), and Matheus ("Striker"). Also, my deepest thanks for Dr. Vivian Denise Cazerta Vaitses, as well as her daughter, Dr. Anna Martha Vaitses Fontanari. For all of them, thank you very much for your help.

To my grandfathers, grandmothers, and to my parents, mother and late father, for everything.

"If one cannot prove that a thing is, he may try to prove that it is not. And if he succeeds in doing neither (as often occurs), he may still ask whether it is in his interest to accept one or the other of the alternatives hypothetically, from the theoretical or the practical point of view. Hence the question no longer is as to whether perpetual peace is a real thing or not a real thing, or as to whether we may not be deceiving ourselves when we adopt the former alternative, but we must act *on the supposition of its being real."*

— Immanuel Kant

ABSTRACT

Consider the problem of a set of n jobs which needs to be processed by a single machine. The processing time for each job is identical to all the others, and predefined. Once on the machine, preemptions are not allowed. Every job has a release date before which it can not be scheduled. They also have due dates, before which they are supposed to have completed processing by the machine. The jobs are weighted, and the goal is to find a schedule which maximize the sum of weights of jobs complete in time. Baptiste [1] approached a generic instance of this problem with a dynamic programming solution which runs in $O(n^7)$ time. We use an additional hypothesis related to release and due dates: for any job j, its release date is denoted by r_j and its due date by d_j. We say that a set of jobs and release and due dates are agreeable if, and only if, for two jobs j_1 and j_2, $r_{j_1} < r_{j_2} \Leftrightarrow d_{j_1} < d_{j_2}$. We model this problem as an integer linear programming and run in general solvers like glpsol and CPLEX. Finally, we present an alternative solution inspired on Baptiste's original dynamic programming to solve only instances whose release and due dates are "agreeable" like we defined earlier. Our solution outperforms the original and the solvers when the set of dates is agreeable, running in $O(n^3)$ time.

Keywords: Combinatorial Optimization. Scheduling. Single Machine Scheduling Problem. SMSP. Dynamic Programming. Integer Linear Programming. ILP. Computational Complexity Theory. Theoretical Computer Science.

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Given a set of n jobs, we call a *schedule* the mapping of each job to a starting time, at which the job starts executing on a machine up to its completion. The general problem is about finding an optimal schedule of jobs in terms of predefined optimality criteria. For example, to minimize makespan. Variations can be obtained by adding other restrictions to this general concept, such as imposing for each job a release date before which that job is not considered available for scheduling. Many practical, real-life situations can be modeled as a scheduling problem with the proper set of constraints.

Among these variations there are many for which a reasonable solution has yet to be found. We will refer to each of these variations of the scheduling problem by using the popular notation introduced by [2]. The problem of minimizing the number of late jobs is written as $1 \mid \sum U_j$ using the aforementioned notation. This problem, without further constraints, consists in alloting penalties for jobs overdue, and can be solved in polynomial time using the algorithm introduced by [3] in time $O(n \log n)$ steps. In this particular case, to minimize unit time penalties altogether means the same as maximizing the number of jobs completed before due. The weighted version of this same problem mean some jobs are less desired to be on time than others. This, and adding of *release dates* so a job cannot be scheduled arbitrarily early, is noted as $1 \mid r_j \mid \sum w_j U_j$ and shown to be strongly NP-hard [4]. The concept of *tardiness*, which, rather than *lateness*, consider overdue jobs to be still of value, the difference between its due date and completion time, is none easier in overall terms of complexity [5], with its weighted version being also strongly NP-hard [6].

For the special case $1 \mid p_j = p, r_j \mid \sum w_j U_j$, with all jobs having identical processing time, a dynamic programming algorithm was proposed in [1]. This solution runs in polynomial time though having primarily $O(n^5)$ space complexity as well as a $O(n^7)$ time complexity. It considers the problem of measuring up smaller job sequences in different time intervals, later combining those in a bottom-up fashion.

This work will focus on a special case of $1 \mid p_j = p, r_j \mid \sum w_j U_j$, with sets of release and due date ordered in a similar fashion. Namely, $\forall i, j \; s.t. \; i \leq j, r_i \leq r_j \Leftrightarrow d_i \leq d_j$. Henceforth, we will refer to this property as having sets of

agreeable release and due dates. We study and implement the solution for the former case, without agreeable sets, for purpose of comparing performance. We also model this problem as a time-indexed ILP using AMPL, then solve them using CPLEX. The results are compared to those obtained through the said implementation of Baptiste's solution. Furthermore, we propose a better solution for the special case of agreeable release and due date sets, and generate another set of instances according to said restriction. We show our solution is more efficient when Baptiste's dynamic programming runs over the same subset of instances having agreeable release and due date sets. Finally, we present results for this subproblem that outperform CPLEX for inputs of considerable size.

2 PROBLEM

2.1 Overview

A *Single Machine Scheduling Problem* refers to the more general problem of optimization of a given objective function applied to given a set of *tasks* or *jobs*, and which has the only predetermined restriction of being a single machine alone, not allowing assignments such that more than one job is processed at the same time.

A *Multiple Machine Scheduling Problem* refers to the somewhat relaxed version of the aforementioned problem, since having more than one machine allow for assignment of jobs to be processed at the same time on different machines ·

Note that both definitions above are far from complete, and several constraints shall arise for a more specific version of both general problems as is. We will use a more precise and formal definition for a **Scheduling Problem**:

A *Scheduling Problem* is defined as follows: suppose that m machines $M_J(j = 1, \ldots, m)$ have to process n **jobs** $j_i(i = 1, \ldots, n)$. A **schedule** is for each job an allocation of one or more intervals to one or more machines. The corresponding scheduling problem is to find a schedule satisfying certain restrictions. Classes of scheduling problems may be specified in terms of a three-field notation $\alpha \mid \beta \mid \gamma$.

2.1.1 The α, β, γ Notation

Here α specifies the machine environment (i.e. single versus multiple machines, along with the possibilities for dedicated machines and parallel machines of different sorts, such as identical, uniform, or unrelated). Field β relates to the job specific characteristics, such as possibility of preemption, precedence relation between them, presence of release and/or due dates, and deliberations about their processing times. Field γ denotes the optimality criterion, that is, the objective function. Examples are minimizing **makespan**. C_i=completion time of job j_i. Minimizing the *makespan* is to $max(C_i|i = 1, ..., n)$. Other objective functions of interest are minimizing the $\sum_{i=1}^{j} C_i$, that is, the **total flow time**, and minimizing $\sum_{i=1}^{j} w_i C_i$, that is **weighted total flow time**. Other common objective functions

are measuring of tardiness, lateness, among several others.[7].

A *Single Machine Scheduling Problem* is our topic of interest of in this paper. It can be defined by having only one machine, which can process only one job at time. Thus, a SMSP consists of a set J of n jobs $j_1, j_2, \ldots, j_n$ that have to be processed on a single machine M. In our case of interest, we will be interested in minimizing the number of late jobs, rather than minimizin the makespan as in the referenced paper.

Field β is related to the many possible *constraints* of the problem. These range from allowing or not e.g. preemption, or if there are release and due dates for each job.

The last field, γ is related to one of many possible optimizing function. We already mentioned minimizing $\max C$ where C is the completion time of jobs. This is know as minimizing the *makespan*. Other important bottleneck function is minimizing $Lmax$ where L stand for *lateness*. Formally, $L_J = \max 0, C_J - d_J$ where d_j stands for the due date of job J.

Summmary of optimizing functions of interest:

- Tardiness: $c_J - d_J$
- Lateness: $\max\{0, c_J - d_J\}$
- Unit Penalty: $0\ if\ c_J \leq d_J$, $otherwise\ 1$

2.2 Baptiste's approach for the general case without agreeable release and due date sets

Baptiste [1] approached the general problem $1 \mid p_J = p, r_J \mid \sum w_j U_j$ using a dynamic programming iterating over a set $\Theta = \{t = r_i + lp \mid \exists\ r_i \exists\ l \in \{0, \ldots, n\}\}$ of possible start and completion times for every job. Baptiste proved that in any left-shifted schedule the starting times of the jobs belong to Θ. Furthermore, he defined sets $U_k(s, e)$ and maximal $W_k(s, e)$ such that $\forall k, s, e,\ W_k(s, e)$ can be computed

through:

$$\max\left\{W_{k-1}(s,e), \max_{\substack{s'\in\Theta \\ \max\{r_k,s+p\}\leq s'\leq \min\{d_k,e\}-p}} w_k + W_{k-1}(s,s') + W_{k-1}(s',e)\right\},$$

A bottom-up dynamic programming which calculate all possibilities and combinations for s, s' and s', e so to determine the max. The main loop iterates from 1 *to* k, which is $O(k)$. Each s' pairs with an s making an (s, s') then iterates through every possible values of e for (s', e). $s, s', e \in \Theta$ and, therefore have $O(n^2)$ possibilities each. $O(n^2)^3 = O(n^6)$. Since this is an inner loop iterating from 1 *to* k the overall time complexity of the algorithm is $O(n^7)$.

From this point onwards, what will we be referring to as "our problem", are instances of $1 \mid p_J = p, r_J \mid \sum w_j U_j$ which have sets agreeable release and due dates. Out of the scope of our work, the sets are supposed not to be agreeable unless otherwise explicitly noted.

2.3 Approaching our Problem through ILP

Following the brief review about the subject at hand, we will now at out problem of intetest, that is: $1 \mid, P_J = p, r_J \mid \sum w_J U_J$, minimizing the weighted number of late jobs, represented as a problem of maximization using a **penalty unit set** U_j. Moreover, we add to our problem a unique processing time $\forall j \in J, p_j = p$, release and due dates. Specifically to the intererests of this work, we propose a relation between r_j and d_j such that:

$$\forall j_i, j_j \in J, r_i \leq r_j \Leftrightarrow d_i \leq d_j$$

Henceforth, we may refer to this by simply stating that the problem has "agreeable r_k, d_k for any given job k", that is, *agreeable release and due dates*. Furthermore, given the context, for the sake of clarity through avoidance of unnecessary repetitions, any problem instance conformant to the prior definition may be refered just as being **agreeable**.

2.4 Our ILP Model

Discrete sets of agreeable and normal instances were used but this approach was not particularly designed so to benefit from either. They were required, however, as our main solution works only for the agreeable subset.

2.4.1 Explaining our modeling

Let $N = \{0 \dots n\}$. The processing time p remains constant for all n jobs. We define $\Theta = \{t \mid \forall j \in J, \exists l \in \{0 \dots n\}, t = r_j + lp\}$ as the set of all possible times for a job to start or to finish processing, provided it starts as soon as possible. Jobs are scheduled either at their release time, or are waiting for completion of another job. If this other job which is currently being processes did not start at its release date either, the same may apply to a subschedule like an "waiting queue", so inductively backwards up to a very first one. Now we assume, as in [1], that the jobs are sorted by their due dates, increasingly. Therefore, a unique first job is determined, provided, if more than one job is made available simultaneously, the one with the lowest index will be chosen. So, this job is guaranteed to run at its release date, and to be completed at $r_1 + lp = (1)p = r_j + p$. Notice both starting and ending times are members of Θ, with values of 0 and 1 for l, respectively.

For any job made available there are only two possibilities:

- The machine is free and the job is immediately selected and treated in a way analogous to the first one
- The machine is busy processing another job. Let us say this waiting job is the last in a "waiting queue" of m jobs. The one before it also had to wait, and so on up to the first, which we will suppose to have been the kth to be scheduled.

For the latter case, we assume this kth job to have been immediately scheduled at its release date r_{kth} and finished processing at $r_{kth} + p$, both being members of Θ with $l = 0$ and $l = 1$, respectively again. Regarding the "waiting queue" aforementioned, the next on line will start at the former's completion time and be completed itself at $r_{kth} + (l = 2)p$, which is also in Θ. In such situation, the last job in the m-lengthed

queue would start at $r_{kth} + (m-1)p$ and be finished at $r_{kth} + mp$. Once again, we have only two possibilities:

- $m + k = n$
- $m + k < n$

Removing the $k-1$ jobs scheduled before the kth, the latter inequality is shown to be sound, for a kth plus the m-sized subschedule would need them to equal n.

The first possibility is trivial, as the kth job is the very first one, and the last job in the queue of size m was already shown to start and end at instant times alike those in Θ. $k = 1$ thus $m - 1 < m < n$, which are both valid values for $l \in \{0 \dots n\}$.

For the second possibility, either there were $k-1$ jobs before the kth, or there are more jobs waiting to be released after the last in the waiting queue. In the first case, it follows that $m + k < m < n$ for any k, and both $m - 1$ and m are valid as l again. In the second case, the total number of jobs is just shown yet bigger, making the former inequalities and their implications of validity sound all the same.

With that, we had shown Θ has every and only possibilities for the starting and ending times of every job.

Late jobs are scheduled arbitrarily late in timem, as they're already late.

We, now, define another set, parameterized by Θ and also a subset of it: Let $H_t = \{t' \mid \forall t \in \Theta, t \leq t' \leq t + p\}$. As discussed so far, any $t \in \Theta$ is a possible time for a job to start being processed. It will take $t + p$ time units until its completion. Therefore, the members of a given H_t are the subset of times conflicting with a job scheduled on t.

2.4.2 The ILP

The original problem is thus presented as the following ILP:

$$\text{maximize} \quad \sum_{j \in J} \sum_{t \in \Theta} w_j X_{j,t} \tag{2.4.1}$$

$$\text{subject to} \quad \sum_{t \in \Theta} X_{j,t} \leq 1, \qquad \forall j \in J, \tag{2.4.2}$$

$$\sum_{j \in J} \sum_{t' \in H_t} X_{j,t'} \leq 1, \qquad \forall t \in \Theta \tag{2.4.3}$$

$$X_{j,t} = 0, \qquad \forall j \in J, \forall t \in \Theta, t \leq r_j \tag{2.4.4}$$

$$X_{j,t} = 0, \qquad \forall j \in J, \forall t \in \Theta, d_j \leq t + p \tag{2.4.5}$$

$$0 \leq r_j, d_j, w_j, \qquad \forall j \in J \tag{2.4.6}$$

$$X_{j,t} \in \{0, 1\}. \tag{2.4.7}$$

- The objective function maximizes the weighted number of jobs completed in due time, iterating over all jobs and times to find the optimal solution.
- The first constraint iterates over times in Θ for all jobs and, obviously, for a given job, only one starting time shall be assigned, hence the sum is 1 at most.
- The second constraint iterates over every job and their parameterized set of conflicting times, for all possible starting time $t \in \Theta$. Therefore, no jobs shall overlap in time.
- The third constraint states, for every job and for all $t \in \Theta$, a job can not be scheduled if it was not yet released at that point in time.
- The fourth constraint states, for every job and for all $t \in \Theta$, a job with a due date smaller than its starting time plus the time it will take up to completion, is already a late job.
- The remaining constraints defines X to be either 0 or 1 for every j and t.

2.5 An Example

To illustrate the purpose of each constraint, we made a small example of a set J with seven jobs, described in Table 2.1:

Table 2.1 – An illustrative example

job	release date	due date	weight
j_1	1	7	61
j_2	1	7	50
j_3	2	8	59
j_4	2	9	45
j_5	2	10	54
j_6	3	10	83
j_7	4	10	27

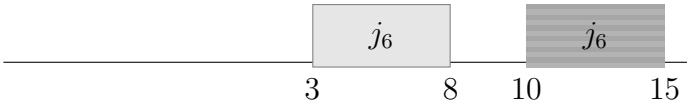

Figure 2.1 – $X_{6,3} = 1$ means that j_6 starts at $t = 3$ then again at $t = 10$, since $X_{6,10} = 1$, thus violating constraint (2.3.2) and making no sense, for each job only needs to be processed once.

Now, suppose constraint (2.3.2) is violated. That is, there exists a $j \in J$ for which their sum over members of Θ is greater than 1. Let us try and see an example with j_6 in Figure 2.1.

Violating this constraint means that a single j start twice. It is not even necessary to consider the possibility of overlapping, since our next example will talk about this. Now, we will schedule j_1 to start at $t = 1$, that is, as soon as possible. It makes sense, since its weight is the second greatest overall. This simple scenario is shown in Figure 2.2:

In order to violate constraint (2.3.3), another job would have to start in a time $t' \in H_t$, for example j_3 starting at 3. That would be represented with $X_{3,3}$. Figure 2.3 shows the case where $X_{3,3} = 1$, that is, j_3 is starting at $t' = 3 \in H_1$:

Clearly, constraint (2.3.3) is needed to avoid job overlapping. Constraints (2.3.4) and (2.3.5) apply for all $j \in J$ and $t \in \Theta$. The first part needs no illustration: it is impossible for any job to be scheduled ahead of its own release. As for the latter, Figure 2.4 illustrates the problem of a job having a due date smaller than its starting time plus the time it would take up to completion. Let j_7 start at $t > d_7 - p$:

It is impossible for j_7 to be due. The two vertical lines representing $d_7 - p$

Figure 2.2 – j_6 starting at $t = 1$

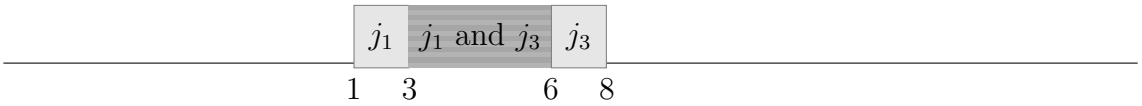

Figure 2.3 – $X_{1,1} = 1$ means that j_1 starts at $t = 1$, but $X_{3,3} = 1$ means j_3 starts at $t = 3 \in H_1$, thus violating constraint (2.3.3) and jobs overlap

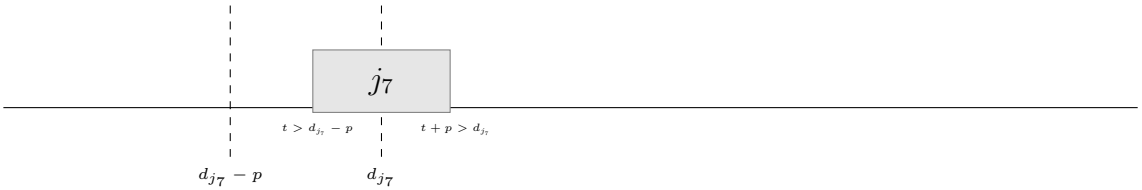

Figure 2.4 – Constraint (2.3.6) is violated by having $X_{7,t>d_7-p} = 1$ meaningless, since it's impossible for j_7 to be due

and j_7 help visualizing it. In short, constraint 2.3.4 is needed so the ILP modelled work as expected. There are no issues with quantifying universally both j and t, for the other aforementioned constraints guarantee no unfit combination of values can happen. h

3 ALGORITHMIC SOLUTION OF THE CASE WITH AGREEABLE RELEASE AND DUE DATES

3.1 Problem

Mimimizing weighted late jobs with the same processing times or $1 \mid p_j = p, r_j \mid \sum w_j U_j$ in $\alpha \mid \beta \mid \gamma$ notation, for the special case of "agreeable" sets of release and due dates. Two sets of release and due dates are said to be *agreeable* with each other if, and only if, $\forall j_i, j_j \in J, r_i \leq r_j \Leftrightarrow d_i \leq d_j$

3.2 Input

A set $J = \{j_1, j_2, \ldots, j_n\}$ of n jobs, the processing time p that is the same for every job, and their respective:

- Release dates $r_1, r_2, \ldots, r_n$
- Due dates $d_1, d_2, \ldots, d_n$
- Weights $w_1, w_2, \ldots, w_n$

3.3 Definitions

We assume that jobs are ordered by their due dates:

$$i \leq j \Leftrightarrow d_i \leq d_j$$

Hence, by adding the hypothesis of agreeable sets of release and due dates, we have

$$r_i \leq r_j \Leftrightarrow d_i \leq d_j \Leftrightarrow i \leq j \tag{3.3.1}$$

We define any schedule of a subset $J' \subseteq J$ to be *feasible* if and only if:

- No job starts before its release date

- No jobs are late
- The execution intervals of the jobs do not overlap

We are looking for the subset X for which there exists a feasible schedule $\sigma(X)$ and whose weight is maximal. Every $j_k \in J \setminus X$ that is late can be scheduled arbitrarily late for it's already late.

Proposition 1. *If $i < j$ then j_i precedes j_j in $\sigma(X)$*

Proof. Either $\sigma(X)$ is already ordered like this or it can be modified to be as such. Let us call any feasible, optimal yet unordered schedule to be $\sigma'(X)$. We show that for every j_j happening before j_i in $\sigma'(X)$ such that $i < j$, their positions can be switched like follows:

- j_j is surely available at the time j_i starts, since it was scheduled before.
- If we swap the jobs, then j_j completion time is the same as j_i was before. $\sigma'(X)$ is feasible by definition, therefore j_i was not late. From (3.3.1) it follows that $d_j \geq d_i$, therefore j_j is not late as well after the swap, and the resulting schedule is still feasible
- j_i can not become late since it is moving backwards in time
- All that is left is to show j_i was already available at the starting time of j_j in $\sigma'(X)$. Since $\sigma'(X)$ is feasible, we know that j_j started after its release date. From (3.3.1) it follows that $r_i \leq r_j$. Hence, j_i is released either earlier or at the same time as j_j, and the schedule resulting from the switch is still feasible.

□

For $k, e \in \mathbb{N}, k \leq n$ we define

$$U_k(e) = \{j_i \mid i \leq k \wedge r_i \leq e - p\} \tag{3.3.2}$$

That is, the set of jobs:

- whose index is k or less; and
- whose release date is, at most, p time units less than e

Also, we will define a function $weight()$ returning either the sum of all job weights in either a subset $X \subseteq J$, or in a schedule $\sigma(X)$ of any subset $X \subseteq J$

Now, let $W_k(e)$ be the maximal weight of a subset $X \subseteq U_k(e)$ such that there is a feasible schedule $\sigma(X)$ of these jobs:

- $\sigma(X)$ starts within $[rmin, rmin + p]$ that is the time interval of length p beginning at the smallest release date $rmin$
- $\sigma(X)$ ends at most at time e

Proposition 2. *For $k, e \in \mathbb{N}$ and $k \leq n$, $W_k(e)$ is equal to $W_{k-1}(e)$ if $r_k > e - p$ and to the following expression otherwise:*

$$W' = \max\{W_{k-1}(e), w_k + W_{k-1}(\min\{d_k, e\} - p)\}$$

Proof. We first prove $W' \leq W_k(e)$, then $W_k(e) \leq W'$.

1. $W' \leq W_k(e)$
   If $W' = W_{k-1}(e)$ then $W' \leq W_k(e)$, since $U_{k-1}(e) \subseteq U_k(e)$.
   Otherwise, let X be the subset of jobs that has a feasible schedule $\sigma'(X)$ that realizes $W_{k-1}(\min\{d_k, e\} - p)$. Then, $\sigma'(X)$ either ends at $\min\{d_k, e\} - p$ or earlier. Hence, there exists a feasible schedule of $X \cup \{j_k\}$ that ends exactly at time d_k, and that weighs exactly $w_k + W_{k-1}(\min\{d_k, e\} - p)$, but is not necessarily maximal. Therefore, $W' \leq W_k(e)$.

2. $W_k(e) \leq W'$

   Let Z be a subset that realizes $W_k(e)$. If $j_k \notin Z$ then

   $$W_k(e) = W_{k-1}(e) \leq \max\{W_{k-1}(e), w_k + W_{k-1}(\min\{d_k, e\} - p)\}$$

   Otherwise, $j_k \in Z$. Let $X = Z \setminus \{J_k\}$. The subschedule $\sigma(X)$ immediately before j_k is formed by jobs with indexes up to $k - 1$, though not necessarily

all of them.

$\sigma(X)$ is feasible, and must end at $\min\{d_k, e\} - p$ or before. Hence, $X \subseteq U_{k-1}(\min\{d_k, e\} - p)$, and $weight(\sigma(X))$ won't surpass $W_{k-1}(\min\{d_k, e\} - p)$, for the latter is maximal by definition

$$\begin{aligned} W_k(e) &= weight\left(\sigma(Z)\right) \text{ for } Z \text{ realizes } W_k(e) \\ &= weight\left(\sigma(X \cup \{j_K\})\right) \\ &= weight\left(\sigma(X)\right) + w_k \\ &\leq W_{k-1}(\min\{d_k, e\} - p) + w_k \\ &\leq \max\{W_{k-1}(e), w_k + W_{k-1}(\min\{d_k, e\} - p)\} \end{aligned} \tag{3.3.3}$$

Q.E.D.

□

Notice there are at most n^2 elements in Θ. Starting with the last, w_k iterates backwards, and $1 \leq k \leq n$. Since for each k we have e iterating over values in Θ, the overall complexity of our solution is $O(n^3)$.

```
1  Function W(k,e):
       Input  : A set of n jobs j_1, j_2, ..., j_n with processing time p for
                every job, due dates d_1, d_2, ..., d_n, release times
                r_1, r_2, ..., r_n, and weights w_1, w_2, ..., w_n.
2      if table.find(key) then                          // key is in table
3          return value[key]
4
5      if k = 0 then                                     // recursive base
6          return 0
7
8      if r_k > e - p then                          // job released too late
9          result ⟵ W(k-1, e)                          // from proposition 2
            table.insert(W(k-1, e))
           // register in table for memoization
10         return result
11
12     W(k,e) = max{W(k-1, e), w_k + W(k-1, min{d_k, e} - p)}
       // from proposition 2
13     table.insert(W(k,e))      // register in table for memoization
       return W(k,e)
```

Algorithm 1: Outline of the function W(k,e), which implements our top-down solution recursively

4 COMPUTATIONAL RESULTS

4.1 General Description

The experiments were performed based on three different approaches

- Original dynamic programming solution of Baptiste [1];
- An AMPL model for a time-indexed ILP run on CPLEX;
- Our own solution, also based on dynamic programming, but restricted to subproblems having the premise of agreeable release and due dates.

All experiments were conducted in a Intel Core i7 CPU 930 running at 2.80GHz 8-core 12 GiB RAM machine. The instances were randomly generated in a fashion resembling that used for example in [8], with n in the set $\{70, 80, 90, 100\}$ and processing times $p \in \{5, 10, 15, 20, 25, 30\}$. For each pair of (n, p) 50 instances were randomly generated. Hence a total of 1200 instances were tested for the general case. Likewise, a different set of 1200 instances with agreeable r_k, d_k was generated. We refer to the latter as the agreeable set and to the former as the "normal" set.

4.2 Representing Problem Instances

Both the implementation of Baptiste's original dynamic programming and our adapted solution for agreeable r_k, d_k have the same formatting for the input files. This format consists of a plain text file with an ".in" filename extension and such that:

- Zero or more *comment lines* are placed before anything else. A comment line must start by the character 'c' separated from the comment by a single blank space.
- Exactly one *parameter line* describing the number n of jobs and the common processing time p of that specific problem instance. This parameter line must start with the characters 'n' and 'p' immediately followed by the values intended for n and p, each separated by a single blank space.

- Exactly *n job lines* describing each of the jobs itself. A job line must start by the character 'j' and be followed by the values of release date, due date, and weight associated to that job, in this same order, each separated by a single blank space

  Example of a valid input file:

```
c This is a comment line
c Another comment line
c Next line has values for n and p (number of jobs and duration time)
c Jobs are "j r d w" r = release date, d = due date, w = weight
n p 3 9
j 3 42 2
j 1 46 7
j 4 43 1
j 1 43 3
j 5 45 4
```

Input files formatted other than the above description, such as with a number of job lines mismatching the parameter line or with multiple parameter lines, are considered malformed. The implementation is correct (that is, returns the max weight of an optimal schedule) for any input that is not malformed. Trying to pass a malformed input file as argument to an implementation yields an undefined behaviour.

A small set of utilities was implemented to aid the handling of input files and formats:

- The *gen* tool takes two mandatory arguments n and p for generating a random problem instance with n jobs and common processing time of p time units. When invoked with the -G (or "–agreeable") option, the generated file satisfies the property of agreeable r_k, d_k.

- The $inst-chkr$ tool tests if a given input file satisfies the property of agreeble r_k, d_k. The agreeable instance set is one that was thereby tested for each of its 1200 files and none failed.

- The *conv* tool takes an input file.in as argument then outputs the corresponding instance written in AMPL (more specifically, GNU MathProg) language. For convenience, AMPL/GNU MathProg-formatted inputs have the .ampl filename extension. An AMPL input file can either be solved by *glpsol* directly or converted into an .lp file first then solved by CPLEX.

Several tests ran simultaneously using *GNU parallel*. The tests were automatized through bash scripting and their overall performance measured by *time*. The *gen* tool used the mt19937 implementation for the Mersenne Twister PRNG, seeded with a generic random device. Job weights, release dates, and due dates were randomly chosen from an uniform integer distribution, with slight different ranges. Weights were in the range of the interval $[1, 120]$. The range for release dates was different: $[1, (n \times 6) - p]$. The interval from which the due date of job j_k depended on its release date: $[r_k + p, (n - 5) - p]$. After generating n triples of release date, due date, and weight, one for each job, they were sorted accordingly if the option for generating an agreeable set was passed.

4.3 Table of Results

In this section we will provide and comment on the results of our experiments. We will show tables with the (average) times for each combination of problem size n and the processing time p, for purpose of comparison between proposed solutions.

Table 4.1 – These are random instances lacking the agreeable premise, hence our own solution do not apply. CPLEX results are shown in two columns corresponding to running multi-threaded (default) and single-threaded, respectively. All times are in seconds.

n	p	bap	cpx1	cpx2
70	5	3.46	0.30	0.26
70	10	19.66	0.98	0.84
70	15	51.14	2.31	2.06
70	20	92.53	2.41	2.14
70	25	137.93	3.29	3.01
70	30	188.79	3.78	3.49
80	5	5.82	0.39	0.34
80	10	35.63	1.44	1.24
80	15	93.16	2.98	2.62
80	20	178.78	3.99	3.54
80	25	290.40	5.32	4.70
80	30	404.72	7.04	6.04
90	5	9.86	0.55	0.49
90	10	58.50	2.66	2.25
90	15	168.71	3.53	3.14
90	20	325.57	5.58	4.88
90	25	545.80	7.98	7.11
90	30	780.66	9.71	9.66
100	5	16.05	0.74	0.65
100	10	95.80	3.59	2.97
100	15	280.29	4.73	4.11
100	20	592.01	8.06	6.54
100	25	951.72	11.59	10.24
100	30	1367.37	15.47	15.22

Table 4.2 – These instances have agreeable release and due dates, therefore our solution was added for comparison. The columns for CPLEX mean the same as in 4.1. All times are in seconds.

n	p	agbap	bap	cpx1	cpx2
70	5	0.00	3.69	0.32	0.27
70	10	0.01	20.41	1.01	0.87
70	15	0.01	51.24	1.80	1.60
70	20	0.01	91.83	2.33	2.01
70	25	0.01	144.92	3.16	2.72
70	30	0.01	200.74	3.88	3.33
80	5	0.01	6.31	0.44	0.37
80	10	0.01	37.94	1.75	1.48
80	15	0.02	95.39	2.35	1.98
80	20	0.02	191.25	3.71	3.14
80	25	0.02	315.28	5.65	4.66
80	30	0.02	447.09	8.13	7.49
90	5	0.01	10.61	0.57	0.48
90	10	0.02	62.25	2.61	2.21
90	15	0.03	172.41	3.58	2.91
90	20	0.03	366.69	6.01	5.23
90	25	0.03	609.65	7.90	7.01
90	30	0.04	869.02	10.71	9.27
100	5	0.01	15.83	0.78	0.64
100	10	0.03	100.42	3.64	2.96
100	15	0.04	290.57	5.23	4.90
100	20	0.05	625.77	9.09	7.83
100	25	0.05	1063.28	12.87	10.80
100	30	0.06	1522.97	16.63	14.50

Baptiste's proposed solution is outperformed by our own for agreeable instances, and, overall, by CPLEX. Its $O(n^7)$ time is costly, and for $n = 100, p = 30$ had run for over 20 minutes, whilst CPLEX took about 15 seconds.

Our own solution could not be satisfactorily measured by the aforementioned tests. In order to fix this, we increased the number of jobs and applied further testing.

4.4 More Testing

The tests above failed to substantially measure our own solution, which lasted no more than 0.06 sec for the greatest input size with $n = 100$ and $p = 30$. Nevertheless, Baptiste's original algorithm already hit about 30 min. Therefore, a new set of tests was prepared, with n ranging from 120 to 200 (with increments of 20), and p restricted to 20, 25, and 30. It compares our solution against CPLEX running with default settings. The results were as follows:

Table 4.3 – Testing our solution and CPLEX for up to 200 jobs. CPLEX run default (multi-threaded) mode for all these tests. All times are in seconds.

n	p	agbap	CPLEX
120	20	0.13	16.01
120	25	0.14	20.72
120	30	0.16	41.12
140	20	0.24	34.98
140	25	0.27	46.44
140	30	0.30	61.37
160	20	0.39	52.17
160	25	0.45	77.05
160	30	0.49	125.61
180	20	0.59	68.60
180	25	0.68	148.51
180	30	0.75	218.35
200	20	0.89	101.84
200	25	1.04	200.55
200	30	1.15	258.19

4.5 Correlation between processing time p and general results

The data so far presented seems to imply a small correlation between the execution time for several instantes of n jobs, however variating p. The processing time p is a *parameter* of the problem and, at first glance, there seems to be no relation whatsoever with the input. We claim our algorithm to be polynomial as a foundation hypothesis. Hence, if expressed solely in terms of the input size n, $T(n) = n^{\beta}$. Finally, we claim that, if expressed in terms of both n and p, $T(n,p) = n^{\beta_1} \cdot p^{\beta_2} \rightarrow log(T(n,p)) = \beta_1\ log(n) + \beta_2\ log(p)$. Data was input in R and processed so to estimate the interference of p as another variable in a multiple linear regression. The results are shown in Figures 4.1 and 4.2. Figures 4.3 and 4.4 show the regression along with points in Table 4.3.

```
Call:
lm(formula = log(cplex$t) ~ log(cplex$n) + log(cplex$p))

Residuals:
     Min       1Q   Median       3Q      Max
-0.20971 -0.11450  0.02406  0.07150  0.19673

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -22.7617     1.1625  -19.58 1.79e-10 ***
log(cplex$n)   3.9499     0.1891   20.89 8.37e-11 ***
log(cplex$p)   2.2033     0.2058   10.70 1.71e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1322 on 12 degrees of freedom
Multiple R-squared:  0.9787,    Adjusted R-squared:  0.9751
F-statistic: 275.5 on 2 and 12 DF,  p-value: 9.369e-11
```

Figure 4.1 – Summary of multiple linear regression over data obtained from CPLEX.

```
Call:
lm(formula = log(tcc$t) ~ log(tcc$n) + log(tcc$p))

Residuals:
      Min        1Q    Median        3Q       Max
-0.047017 -0.017460  0.006012  0.016603  0.031498

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -22.00287    0.21163  -104.0  < 2e-16 ***
log(tcc$n)    3.81197    0.03442   110.8  < 2e-16 ***
log(tcc$p)    0.56976    0.03748    15.2 3.33e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02407 on 12 degrees of freedom
Multiple R-squared:  0.999,     Adjusted R-squared:  0.9989
F-statistic:  6249 on 2 and 12 DF,  p-value: < 2.2e-16
```

Figure 4.2 – Summary of multiple linear regression over data obtained from our solution.

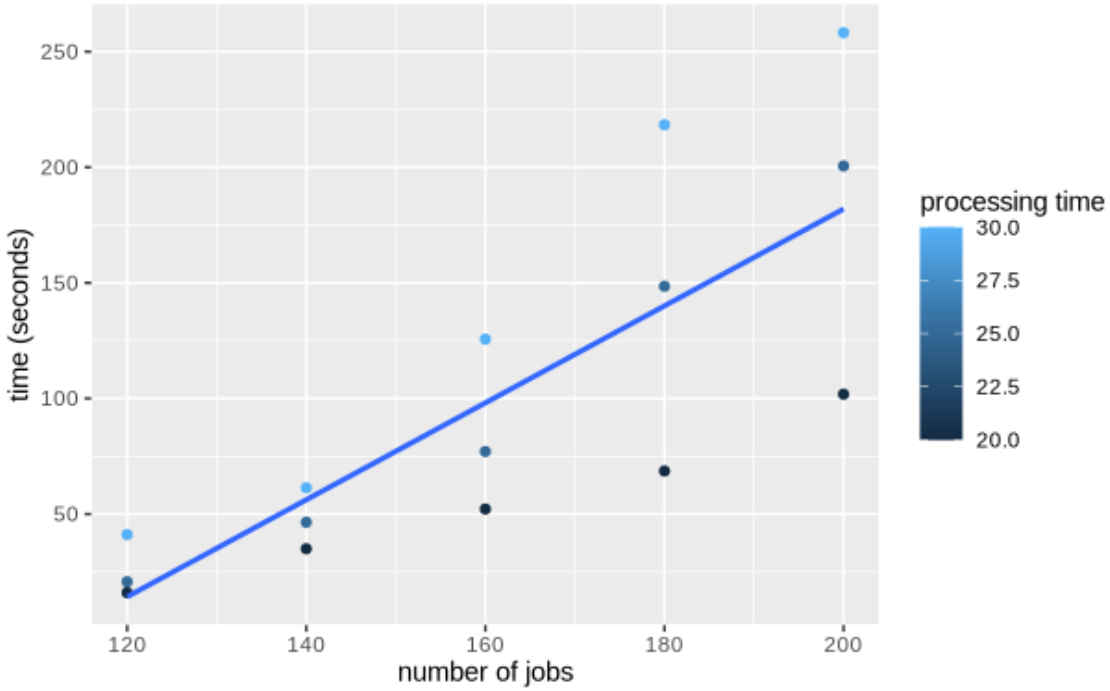

Figure 4.3 – CPLEX linear regression and data plot.

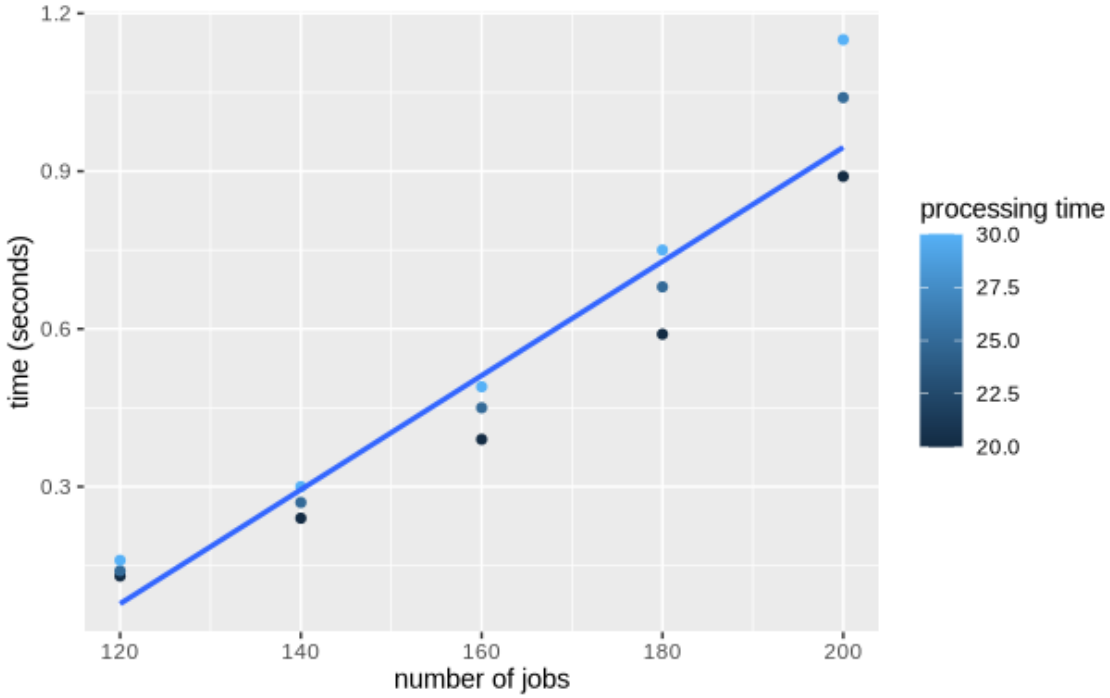

Figure 4.4 – Our solution linear regression and data plot.

The data obtained from R summarized in 4.1 and 4.2 is consistent with our polynomial hypothesis. We already shown our solution to run in time $O(n^3)$ before. The value of ≈ 3.8 is consistent with our polynomial hypothesis about our solution's time complexity. The results also indicate that p has a minor influence over the running time, having an exponent of ≈ 0.57. Both Multiple and Adjusted

R-Squared were ≈ 9.99, and the very small p-value attest for the whole procedure as both statistically reliable and significant.

5 CONCLUSIONS

The tests showed that our ILP model models the problem correctly and was used to solve instances in both glpsol and CPLEX. Results were compared for correctness, attesting for the model.

We claim that our solution outperforms the original Baptiste dynamic programming when release and due dates are agreeable. It also performs better than CPLEX for greater input size. Both these claims are objectively supported by observable evidence provided by experiments results.

The additional testing showed through multiple linear regression that, for both CPLEX and our solution, p has a small correlation but is negligible overall.

REFERENCES

[1] P. Baptiste, "Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times," *Journal of Scheduling*, vol. 2, no. 6, pp. 245–252, 1999.

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

[3] J. M. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs," *Management science*, vol. 15, no. 1, pp. 102–109, 1968.

[4] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of np-completeness. 1979," *San Francisco, LA: Freeman*, vol. 58, 1979.

[5] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," in *Annals of discrete mathematics*, vol. 1, pp. 343–362, Elsevier, 1977.

[6] E. Lawler, "A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness," *Ann. of Discrete Math.*, vol. 1, pp. 331–342, 1977.

[7] P. Brucker and P. Brucker, *Scheduling algorithms*, vol. 3. Springer, 2007.

[8] G. Diepen, J. van den Akker, and J. Hoogeveen, "Minimizing total weighted tardiness on a single machine with release dates and equal-length jobs," *Technical report CS-UU*, no. 2005-054, 2005.

[9] S. Dauzère-Pérès, "Minimizing late jobs in the general one machine scheduling problem," *European Journal of Operational Research*, vol. 81, no. 1, pp. 134–142, 1995.

[10] S. Dauzère-Pérès and M. Sevaux, "Various mathematical programming formulations for a general one machine sequencing problem," *JNPC'98*, pp. 63–68, 1998.

[11] S. Dauzère-Pérès and M. Sevaux, "A branch and bound method to minimize the number of late jobs on a single machine," 1998.

[12] E. L. Lawler, "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs," *Annals of Operations Research*, vol. 26, no. 1, pp. 125–133, 1990.

[13] R. Nessah and I. Kacem, "Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates," *Computers & Operations Research*, vol. 39, no. 3, pp. 471–478, 2012.

[14] P. Baptiste, L. Peridy, and E. Pinson, "A branch and bound to minimize the number of late jobs on a single machine with release time constraints," *European Journal of Operational Research*, vol. 144, no. 1, pp. 1–11, 2003.

[15] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval Research Logistics (NRL)*, vol. 1, no. 1, pp. 61–68, 1954.

[16] S. Verma and M. Dessouky, "Single-machine scheduling of unit-time jobs with earliness and tardiness penalties," *Mathematics of Operations Research*, vol. 23, no. 4, pp. 930–943, 1998.

[17] J. Van den Akker, C. A. Hurkens, and M. W. Savelsbergh, "Time-indexed formulations for machine scheduling problems: Column generation," *INFORMS Journal on Computing*, vol. 12, no. 2, pp. 111–124, 2000.

[18] E. L. Lawler, "Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the 'tower of sets' property," *Mathematical and Computer Modelling*, vol. 20, no. 2, pp. 91–106, 1994.

[19] M. Chrobak, C. Dürr, W. Jawor, Ł. Kowalik, and M. Kurowski, "A note on scheduling equal-length jobs to maximize throughput," *Journal of Scheduling*, vol. 9, no. 1, pp. 71–73, 2006.

[20] J. P. Sousa and L. A. Wolsey, "A time indexed formulation of non-preemptive single machine scheduling problems," *Mathematical programming*, vol. 54, no. 1, pp. 353–367, 1992.

[21] K. R. Baker and D. Trietsch, "Principles of sequencing and scheduling," 2009.

[22] R. Fourer, D. M. Gay, and B. W. Kernighan, "AMPL: A modeling language for mathematical programming," 2002.

[23] J. Y. Leung, *Handbook of scheduling: algorithms, models, and performance analysis*. CRC press, 2004.