Summary

Imagine a car garage with three car lifts that can be used to lift a car so it can be repaired. All the time, customers are coming in with their broken cars. Some are very broken and need a lot of time to repair while others may only need a short amount of time. The manager of the garage needs to decide which cars he is going to repair first. His goal is to get the most cars out of his garage as fast as possible.

Unfortunately, deciding what cars to repair first is a very hard problem. It is comparable to solving the famous Travelling Salesperson Problem, where a businessperson has to to visit a lot of cities and wants to find the shortest route between them.

A simple way of deciding the order of repairing the cars is starting with the cars that will take the shortest time to repair, and leave the longer repairs for later. This may seem like a very logical rule, but it may produce a repair schedule that is not ideal. But, how far from ideal could this simple schedule be? That is the main topic of my thesis.

In the thesis, I first explain what research has been done already on the subject. Then I use the program I wrote to make schedules using the simple rule and optimal schedules for a lot of examples. What I found out is that the examples where the difference between the two schedules is the largest all look similar. Finally, I analyze the general form of the examples with large differences. I show in the paper that when we restrict ourselves to the form of the examples with the largest differences, the simple rule schedule can be at most 1,105 times worse than the ideal schedule. In practice, this means that at the very worst, the cars in the garage will only have to stay 10,5% longer in the garage.

This last result is new and if someone can prove that the form of the examples I found is indeed the worst form of example, we know that the simple rule produces quite well preforming schedules that can be at most 1,105 times worse than the ideal schedule. Then, if the small factor of 1,105 is not a problem, garage managers and other people who encounter problems like this can apply the "short jobs first" rule with confidence.

3.1 Introduction

In this paper, I will study the shortest remaining running time algorithm for preemptive scheduling. SRPT, which stand for Shortest Remaining Processing Time, is a well-known and simple online procedure to generate a feasible schedules for instances of this type of problem.

Recently, it has been shown that the competitive ratio of SRPT is between $\frac{21}{19}$ and $\frac{5}{4}$ [Chung et al. (2010), Sitters (2010)]. In this paper, I will focus on the lower bound of $\frac{21}{19}$ and will try to analyze and improve on it. The ultimate goal is to find an instance with a greater ratio than $\frac{21}{19}$.

The goals for this paper are to find the characteristics of instances where SPRT gives a greater objective value than an optimal schedule would give and to show that SRPT is much faster than solving the Integer Linear Program to find an optimal schedule. In addition, this paper wants to give a range of instances for which the competitive ratio of SRPT is not greater than $\frac{21}{19}$ and to analyze a general form of instances for which the ratios are high.

In this paper, I will first introduce the problem and the known literature about the subject. Second, I will present the methods used for the analysis required by this paper; I will give the integer linear program I used to find optimal schedules in this paper and show how the process to generate the instances worked. Third, I will present the results from the analysis of some ranges of small instances and the results for large instances. Last, I will algebraically analyze the general form of some particular high ratio instances found in the analysis of small instances.

All analysis was done on my laptop, bought in fall 2011. It has an Intel Core i5 2520M processor, rated at 2,5 GHz and has a turbo function which increases the frequency to 3,0 GHz under load. The laptop has 4 GB of DDR3 RAM.

3.2 Literature

3.2.1 Scheduling problems

The field of scheduling problems was introduced in the early fifties. Since then, there has been a lot of literature on many different kinds of scheduling problems.

The only thing that all scheduling problems have in common is that there is a set of n jobs J that have to be processed on m machines. Further specifications vary greatly. To the end of classifying scheduling problems, a

notation was introduced by [Graham et al. (1979)]. The notation has three parts: $\alpha|\beta|\gamma$.

The α in the Graham notation specifies the machine environment used in the problem. Choices for α include: P signifying identical parallel machines, Q signifying parallel machines with specific speed factors and R signifying unrelated parallel machines.

The β in the Graham notation indicates the type of jobs used. This field can be a combination of multiple specifications. Preemption (job splitting) and precedence relationships are included in β.

The γ specifies the optimality criterion chosen. Often, the optimality criterion is based on the completion times of jobs or on their lateness regarding their due date.

3.2.2 Specific scheduling problem

The scheduling problem addressed in this paper is notated in the Graham notation by $P|r_j, pmtn|\sum C_j$.

As indicated in 3.2.1, the P signifies parallel identical machines, where the number of machines is specified in the instance of the problem. The r_j indicates release times for the jobs: each job j becomes known at time r_j and can not be processed before this point in time. The $pmtn$ indicates that preemption is allowed. This means that jobs can be split up as often as needed without penalty. Finally, $\sum C_j$ means that the objective in this problem is minimizing the average completion time of jobs, which is equivalent to minimizing the overall sum of completion times.

In short, the class of problems studied is the class of scheduling problems where there are some number of identical parallel machines which are doing jobs that have a certain release time and that can be split up at any point. The objective is to minimize the sum of completion times of the jobs.

[Baptiste et al. (2007)] proved that the $P|r_j, pmtn|\sum C_j$ problem is unary NP-hard. This means that even if the inputs can be bounded by a polynomial in the input size, the problem is still NP-hard. Before that, [Du et al. (1990)] already proved the problem is NP-hard.

3.2.3 Approximation ratio

In Operations Research, often algorithms are studied that may not always give the optimal solution to a problem. When studying such an algorithm, it might be of interest how much worse the solution that algorithm gives can be than the optimal solution. In other words, to compare the worst

possible solution from the algorithm to the optimal solution. This is called the performance of an algorithm.

The instrument that is used to measure this performance is the approximation ratio. If OPT is the objective value of the optimal solution and ALG is the objective value of the worst solution the algorithm can provide, the approximation ratio is defined as $\frac{ALG}{OPT}$.

3.2.4 SRPT

The scheduling problem introduced in 3.2.2 can not be solved in polynomial time, as proven by [Baptiste et al. (2007)]. Thus, solving the problem to optimality could prove to take a long time. [Philipps et al. (1998)] introduced an algorithm to solve $P|r_j, pmtn|\sum C_j$ which they called SPT, for Shortest Processing Time. By that time, it was already known that this algorithm solved the problem for one machine to optimality in polynomial time. SPT is the same algorithm that we call SRPT now, where the R stands for Remaining.

SRPT schedules the jobs in such a way that at every time slot the jobs with the shortest remaining running time are processed. A schedule is called an SRPT schedule if for every machine at each time slot the jobs with the lowest possible remaining processing time are selected.

SRPT is a deterministic algorithm. This implies that SRPT gives a schedule that has for each point in time which jobs the machines have to process. This also requires an arbitrary method to choose from jobs that have the same remaining processing times. Any method will work and will not affect the performance of the algorithm.

SRPT can be seen as an online algorithm. The performance of the algorithm is not affected by having knowledge of future jobs.

Ideally, the processing time of the SRPT algorithm is $O(n \log n)$, where n is the number of jobs. [Philipps et al. (1998)]

3.2.5 Upper bound

When trying to find the approximation ratio of an algorithm, trying to find an upper bound is most desirable. If an upper bound is known, there is a guarantee that for any instance, the performance of SRPT can be at most the upper bound times the optimal objective value. In other words, if an upper bound is known, it is known how "bad" SRPT can perform at worst. When SRPT was first studied for this particular problem by [Philipps et al. (1998)], they proved that SRPT has an approximation ratio of at most 2. The idea used in that proof is that the SRPT schedule has finished at least the same

amount of jobs by time $2t$ that any other schedule could have finished by time t.

Later, in [Chung et al. (2010)], [Chung et al. (2010)] published a paper that both included a new lower bound and a new upper bound for the approximation ratio of SRPT. They proved that SRPT is 1.86-competitive for this problem. In the article, they call their method probabilistic but note that although the method is probabilistic, the result is deterministic. The proof uses an algorithm that transforms an optimal schedule to a SRPT schedule. They prove this algorithm increases the expectation of the objective value by at most 1.86.

The most recent result is from [Sitters (2010)], which improves the upper bound to $\frac{5}{4}$ with a proof based on the proof by [Philipps et al. (1998)].

3.2.6 Lower bound

In addition to proving the 1.86 upper bound on the approximation ratio, [Chung et al. (2010)] also obtained a lower bound. Obtaining a lower bound is easier, as an instance for which the algorithm performs badly is enough to prove that the approximation ratio is at least the ratio in the instance. [Chung et al. (2010)] found an instance for which the ratio is $\frac{21}{19}$. This improves on the previous best known lower bound, $\frac{12}{11}$. It is also known that no deterministic online algorithm can have a lower bound less than $\frac{22}{21}$ [Vestjens (1997)].

The instance by [Chung et al. (2010)] with a ratio of $\frac{21}{19}$ has $m = 2$ and $n = 7$.

j	1	2	3	4	5	6	7
p_j	1	1	2	1	1	1	1
r_j	1	1	1	3	3	3	3

Table 3.1: The instance with the greatest approximation ratio known

An SRPT schedule for the instance could look like this:

t	1	2	3	4	5
m_1	1	3	4	6	3
m_2	2		5	7	

The objective value for this instance is 21. In this case, doing the short jobs first creates a hole in the schedule at time slot 2. This becomes inefficient at time slot 3 because a block of new jobs is released. The completion of job 3 is postponed until all the new jobs are complete.

Solving to optimality gives this schedule with an objective value of 19.

t	1	2	3	4	5
m_1	1	2	4	6	
m_2	3	3	5	7	

3.3 Methods

To achieve the goals set by this paper, analyzing a lot of instances was required. To that extent, I built a program that can generate an SRPT schedule and an optimal schedule for an instance of the problem. In addition, I created a program that can generate all instances in a specified range for analysis and then uses the methods of the first program to find the two schedules for each instance.

3.3.1 Solving to optimality

First, a method to solve an instance of the scheduling problem to optimality was needed. In this paper, this is done by using CPLEX technology to solve an integer linear program which corresponds to the instance that needs to be solved.

Let T be an integer number at least as large as the maximum possible time needed by a feasible solution. In the implementation of the program, $T = \sum_{j=0}^{n} p_j + \max_j r_j$ is used. Then the integer linear program looks like:

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{j=1}^{n} C_j \\
\text{Subject to} \quad & \forall j: \sum_{t=1}^{T} B_{j,t} \geq p_j \\
& \forall j: \sum_{t=1}^{r_j - 1} B_{j,t} = 0 \\
& \forall j: C_j \geq \sum_{t=0}^{T} B_{j,t} \\
& \forall t: \sum_{j=1}^{n} B_{j,t} \leq m \\
\text{Bounds} \quad & \forall j,t: B_{j,t} \in \{0,1\} \\
& \forall j: 0 \leq C_j \leq T
\end{aligned}
$$

Where $B_{j,t}$ represents a boolean schedule for all jobs and C_j represents the completion time of job j. As before, n is the number of jobs and m is the number of machines. t is used to indicate time and j to indicate job number.

The boolean schedule is visualized below. For each time slot, there is a column of variables of length n which can take the values 0 and 1. So, if $T = 20$ and $j = 7$, the amount of variables in the schedule is 140.

		Time				
		1	2	3	4	...
Job	1					
	2					
	3					
	4					
	...					

In this visualization, the solution the linear program will find has a 0 or 1 in every empty cell. The first constraint means that at least p_j cells in row j should be 1. The second constraint means that the first $r_j - 1$ time slots on a row should have a 0 to accommodate for the release time. The third constraint defines the completion time of each job as the time of the last cell that has a 1 for that row. The last constraint means that the sum over a column should be at most the number of machines.

3.3.2 Finding an SRPT schedule

To find an SRPT schedule, the following technique was used:

Start at time slot 1. Add all jobs that have release time 1 to a list of available jobs sorted by their remaining processing time. Then select the first m jobs from this sorted list. For each selected job, decrease the remaining processing time by 1. If for any of the selected jobs the remaining processing time is now 0, remove those jobs from the list of available jobs.

Now go to the next time slot and repeat the same operations. Continue until all jobs are completed.

The STD library of C++ includes the multimap class which is very well suited to store the available jobs. It stores pairs of information while keeping the data sorted by the key variable. For this application, it can keep the pair of remaining processing time and job number together while sorting the remaining processing times.

3.3.3 Generating the instances

A big challenge when analyzing the competitiveness of SRPT was generating all instances for given parameters. Skipping certain instances may lead to missing some particularly high approximation ratio instances while analyzing duplicate instances will cost too much time.

An instance of the problem class $P|r_j, pmtn|\sum C_j$ includes a number of machines and a vector of processing times and a vector of release times of equal length. The length of the vectors is equal to the number of jobs. Ideally, the program will only generate unique instances. In this sense, unique instances are instances that have a different set of jobs regardless of their order.

The process used to generate all instances uses a different generation process for the processing times and the release times. The generation process for the processing times used in this paper is a process that generates increasing numbers that have the characteristic that each digit is at least as large as the digit before it. The generation process used for the release times is just increasing numbers until all jobs have the maximum release time.

To be certain no instance will be checked twice, a repository is used that holds all previously tried instances as a long integer number with for each job the processing time and release time, sorted by the release time. Before any instance is analyzed by the program, the program first checks if the long integer of the current instance is already in the repository. If it is, the instance is skipped because it is a duplicate of an earlier instance. If not, the program analyzes the instance.

3.3.4 The number of machines

Each instance has a list of jobs with for each job a processing time and release time. These processing and release times will be generated by the generating process described in 3.3.3. In addition to the jobs, each instance has a certain number of machines m. In this paper, I chose to set the number of machines for each instance to 2. The choice for $m = 2$ stems from the fact that the worst known instance also has 2 machines.

Preliminary analysis shows that if we set $m = 3$, many more jobs are needed to create the same difference between the objective values given by SRPT and OPT. For example, the most intuitive translation of the instance given by [Chung et al. (2010)] to 3 machines would create one more job at release time 1 with a processing time of 1 and two new jobs at release time 3 with a processing time of 1. All three new jobs finish at the same time in the SRPT schedule and optimal schedule, increasing both objective values by the same amount. Thus, decreasing the ratio between them.

My belief is that the greatest ratio that can be found for $m = 3$ is less than the greatest ratio for $m = 2$. To get to ratios greater than 1, for $m = 3$, at least 7 jobs are needed following from the characteristics presented in 3.4.1, with even more jobs required when m is increased further. As processing power is limited, analysis on instances with $m = 2$ will give better results.

3.3.5 Reducing the amount of instances

Generating all instances with maximum processing time and maximum release time of 5 using the process described in 3.3.3 produces too many instances to analyze in reasonable time for $n > 4$. Therefore, it is desirable to reduce the amount of instances to check for high competitive ratios.

In search for high competitive ratios, problem instances that have no jobs with release time 1 can be disregarded. Any problem instance that has no jobs with release time 1 has only jobs with release times of 2 or more. By decreasing all release times in the instance by 1, a new, valid problem instance is created that will have a greater competitive ratio than the original instance. In this section I will use a function $V(\cdot)$ which gives the objective value of the schedule of its argument. In this section, $OPT(A)$ and $SRPT(A)$ denote the schedules solving to optimality and SRPT would give respectively.

Assume a problem instance A with release times such that $r_j > 1$. Use the program described in 3.3.1 and 3.3.2 to find an optimal schedule $OPT(A)$ and an SRPT schedule $SRPT(A)$. The objective values corresponding to these schedules are $V(OPT(A))$ and $V(SRPT(A))$. The

competitive ratio for instance A is $\frac{V(SRPT(A))}{V(OPT(A))} \geq 1$.

Now, transform A by decreasing all release times by 1. The original schedule $OPT(A)$ can be transformed to a feasible schedule for A' by shifting all $B_{j,t}$ to $B_{i,t-1}$. As the schedule $OPT(A)$ was optimal for instance A and A' only changes the release times all by the same amount and the production times are shifted by the same amount, this schedule is an optimal schedule for A'. In $OPT(A')$, every job is completed 1 time slot ealier than in $OPT(A)$, thus $V(OPT(A')) = V(OPT(A)) - n$. For SRPT, a similar argument holds. Schedule $SRPT(A)$ can be transformed by shifting all production times to the front by 1 to obtain an SPRT schedule for A'. Again, the objective value has a similar relation: $V(SRPT(A')) = V(SRPT(A)) - n$. The competitive ratio for A' is now $\frac{V(SRPT(A'))}{V(OPT(A'))} = \frac{V(SRPT(A))-n}{V(OPT(A))-n}$. As the numerator and denominator are decreased by the same amount, the value of the ratio for A' will be greater. As $\frac{V(SRPT(A))}{V(OPT(A))} \geq 1$, now $\frac{V(SRPT(A'))}{V(OPT(A'))} \geq \frac{V(SRPT(A))}{V(OPT(A))}$, where equality only holds if $\frac{V(SRPT(A))}{V(OPT(A))} = 1$. Therefore, if the objective is to find a high ratio, instance A can be disregarded.

Now, the amount of instances to analyze to find the high competitive ratio instances is reduced. Table 3.2 shows how many instances are left for n jobs with a maximum processing time and maximum release time of 5. The restriction reduces the amount of instances for every amount for every group of instances with the same amount of jobs. However, the reduction becomes relatively smaller when n increases. The last column shows how many instances are left after the restriction is applied to instances with maximum processing time of 3 and maximum release time of 5. This restriction is needed to check a range of instances with more than 6 jobs, as it now becomes reasonable to check all instances in that range for 7 jobs.

Jobs	Instances	With a release time $= 1$	Max processing time 3
1	25	5	3
2	325	115	42
3	2925	1383	316
4	20475	11620	1695
5	118755	76251	7260
6	593775	416675	26384
7	2629575	1971775	84456
8	10518300	8298225	649230

Table 3.2: Numbers of instances to check for given parameters

Especially to analyze ranges of even larger instances, a further reduction

of the amount of instances to check would be helpful. In this paper, no further reduction of the amount of instances to check is performed. One additional way to reduce the amount of instances to check is to ignore all instances with jobs with all the same release time as it is known that those instances will be solved optimally by SRPT.

3.4 Findings

3.4.1 Small instances

For the ranges of instances specified in 3.3.5, this paper will now present the results that were obtained.

Table 3.3 sums up the results for small instances. The first column shows the amount of instances analyzed for that amount of jobs.

Jobs	Instances	"Bad" instances	Worst ratio	Average Ratio
1	5	0	1	
2	115	0	1	
3	2925	0	1	
4	11620	0	1	
5	76251	1316	$\frac{12}{11}$	1,0007
6	26384	886	$\frac{16}{15}$	1,0014
7	84456	4254	$\frac{21}{19}$	1,0020

Table 3.3: Results for small intances and $m = 2$

For 1 till 5 jobs, all instances with release time and processing time per job at most 5, applying the restrictions form 3.3.5 were considered. For 6 and 7 jobs, the instances all had release times of at most 5 and processing times per job of at most 3.

3 jobs

For 3 jobs and two machines, when limited to a maximum processing time per job of 5 and a maximum release time of 5, all instances are solved to optimality by SRPT.

Solving 2925 instances to optimality took 1315.41 seconds, while finding the SRPT solutions took 1.006 seconds. The average time per instance to solve for optimality is 0,45 seconds while the time for SRPT per instance is under a microsecond.

4 jobs

For 4 jobs and two machines, when limited to a maximum processing time per job of 5 and a maximum release time of 5, all instances are solved to optimality by SRPT.

Solving 11620 instances with 4 jobs to optimality took 1782,17 seconds. This means that the average time per job to solve to optimality was 0,15 seconds. The time it took for SRPT to find an SRPT schedule for all instances was 1,778 seconds, again less than a millisecond per job.

5 jobs

For 5 jobs and two machines, when limited to a maximum processing time per job of 5 and a maximum release time of 5, there are 1316 instances with a ratio > 1. The average ratio above 1 was about 1,04 while the average ratio over all instances was about 1,00073; less than $\frac{1}{1000}$. The overall performance of SRPT on average is close to optimal for these instances with 5 jobs with the given parameters.

The processing time to solve for optimality for all instances together in seconds was 32752,7 seconds, which means an average of 0,43 seconds per instance. The total processing time for all instances together for SRPT was about 12,526 seconds, less than a millisecond on average.

To visualize the information produced by the program, Figure 3.1 plots the value of the competitive ratios on the y-axis in the order the program found the instances.

Interesting to see is that the instance with the greatest competitive ratio was found as the first instance with a competitive ratio greater than 1. Overall, this instance was the 45th instance to be checked. In addition, there was exactly one more instance, number 884, that had the same competitive ratio, namely $\frac{12}{11}$. The instances are shown in Table 3.4.

j	1	2	3	4	5
p_j	1	1	2	1	1
r_j	1	1	1	3	3

(a) Instance 1

j	1	2	3	4	5
p_j	2	2	4	2	2
r_j	1	1	1	5	5

(b) Instance 885

Table 3.4: Instances found for $n = 5$ with greatest approximation ratio

Instance 1 was actually the instance with the worst known approximation ratio before [Chung et al. (2010)] and was included in a paper by [Lu et al. (2003)]. Note that Instance 1 is very similar to the instance from

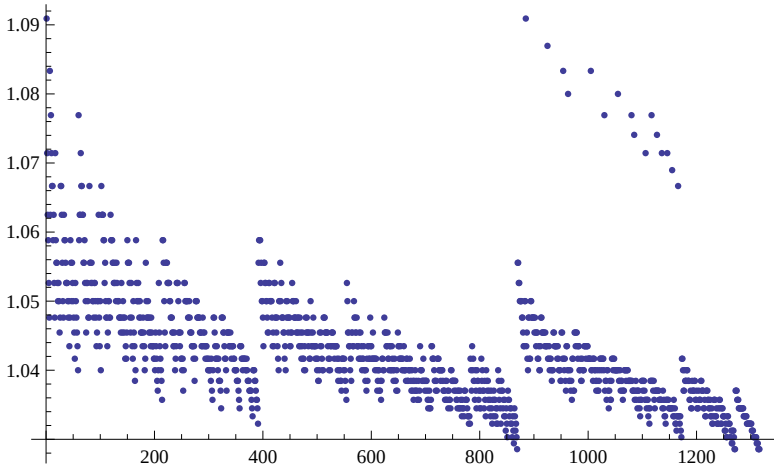


Figure 3.1: Approximation ratios > 1 for $n = 5$

[Chung et al. (2010)], only with a reduced block at release time 3. The SRPT and optimal schedules are shown in Table 3.5.

t	1	2	3	4
m_1	1	3	4	3
m_2	2		5	

(a) SRPT(12)

t	1	2	3	4
m_1	1	2	4	
m_2	3	3	5	

(b) Optimal(11)

Table 3.5: Schedules for instance 1

Essentially, instance 885 is the same instance as instance 1 with everything doubled. The processing times are all doubled and the release times are doubled as well. Release time 1 means the job becomes available in time slot 1, which starts at time 0. So, job 4 and 5 become available at time 2 in instance 1 and at time 4 in instance 885.

It seems useful to have a definition of a multiple of an instance.

Definition 3. *An instance A' is the c-multiple of instance A if:*

1. $m_{A'} = m_A$
2. $n_{A'} = n_A$
3. $\forall j \in A' : p_{j,A'} = cp_{j,A}$
4. $\forall j \in A' : r_{j,A'} = c(r_{j,A} - 1) + 1$

Where $c \in \mathbb{N}$.

All multiples of instance 1, including instance 885, have an approximation ratio of $\frac{12}{11}$.

Characteristics of instances with ratio greater than 1

For an SRPT schedule to be worse than an optimal schedule, it is required that for some time slot, say $\tilde{t}$, there are jobs of different lengths available and that OPT does not select the m shortest jobs to be processed at that time. For it not to be optimal to select the m shortest jobs first at time $\tilde{t}$, jobs have to be released at time $\tilde{t} + x$, such that the longer job selected by OPT is completed before $\tilde{t} + x$, but before that job is completed in an SPRT schedule. For further analysis is if these observations can reduce the amounts of instances to check even further than the reductions of 3.3.5.

These characteristics do not imply that an instance has to contain jobs with different processing times to get a ratio strictly greater than 1. For example, the instance in Table 3.6 also has a ratio greater than 1.

j	1	2	3	4	5
p_j	3	3	3	3	3
r_j	1	1	3	5	5

Table 3.6: Instance found with ratio > 1 for $n = 5$ with all processing times equal

6 jobs

As it is not feasible to check for all instances with release times and processing times up to 5, I chose to generate all instances with 6 jobs with processing times up to 3. For these 26384 instances, it took 6265,32 seconds to solve to optimality. This means an average of 0,24 seconds per instance. The total time to find all SRPT schedules was 3,071 seconds. One of the worst instances found again is the first instance with a ratio greater than 1. The approximation ratios greater than 1 are plotted in Figure 3.2.

The greatest approximation ratio is $\frac{16}{15}$. I expect that is the range of instances is increased, a similar pattern will show, with more instances with an approximation ratio of $\frac{16}{15}$ at multiples of instance 1 and 2.

Interesting to see is that these instances 1 and 2 for $n = 6$ are similar to instance 1 from the analysis for $n = 5$. The difference is the extra job that was added at release time 3. Both instances produce the same optimal and SRPT schedules, with the same approximation ratio.

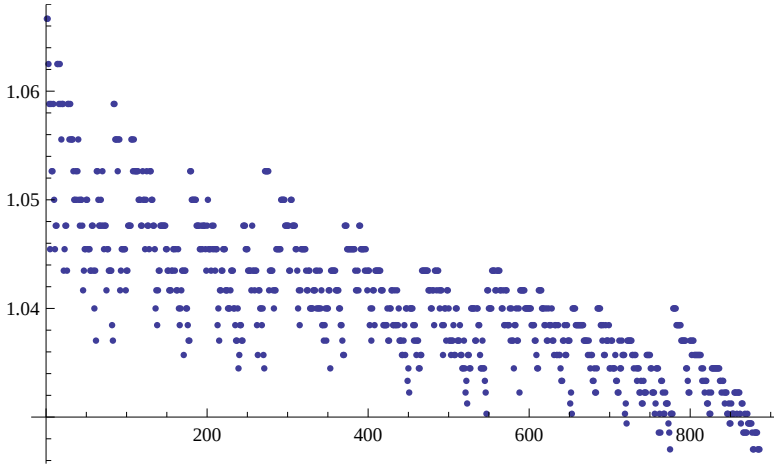


Figure 3.2: First 153 approximation ratios > 1 for $n = 6$

j	1	2	3	4	5	6
p_j	1	1	2	1	1	1
r_j	1	1	1	3	3	3

(a) Instance 1

j	1	2	3	4	5	6
p_j	1	1	2	1	1	1
r_j	1	1	1	3	3	4

(b) Instance 2

Table 3.7: Instance found for $n = 6$ with greatest approximation ratio

7 jobs

To check instances of 7 jobs is interesting as the greatest known approximation ratio, $\frac{21}{19}$ comes from an instance with 7 jobs. For 7 jobs, I decided to do the analysis for a maximum processing time per job of 3. This decreases the amount of instances to analyze from 1971775 to 84456. In total, 4254 instances were found with a ratio more than 1. The total running time to find the optimal schedules was 41361,8 seconds, an average of 0,49 seconds per instance. The time to find the SRPT schedules was 7,162 seconds, meaning an average less than a millisecond per instance.

Again, I plot the approximation ratios in the order they were found to visualize them in Figure 3.3.

The worst ratio found is the $\frac{21}{19}$ from the [Chung et al. (2010)] instance, which was found as instance 7, and two variations of it. The variations are shown in Table 3.8.

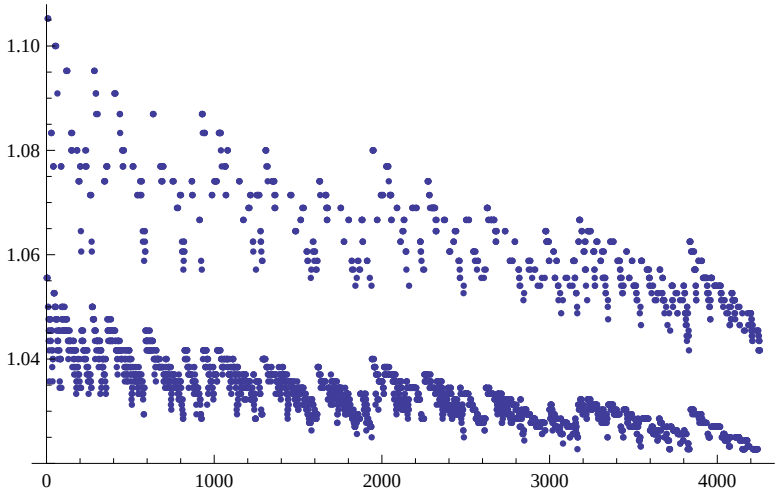


Figure 3.3: Approximation ratios found > 1 for $n = 7$

j	1	2	3	4	5	6	7
p_j	1	1	2	1	1	1	1
r_j	1	1	1	3	3	3	4

(a) Instance 8

j	1	2	3	4	5	6	7
p_j	1	1	2	1	1	1	1
r_j	1	1	1	3	3	4	4

(b) Instance 10

Table 3.8: Instances found for $n = 7$ with greatest approximation ratio other than the Chung instance

3.4.2 Large instances

In addition to analyzing certain ranges of small instances, this paper wants to show how SRPT performs in comparison to solving to optimality for larger instances. The first approach was to generate 100 random instances with 100 jobs. The parameters were a maximum processing time per job of 5 and a maximum release time of 10. The instances had 5 machines. To generate SRPT schedules for these instances is no problem and only takes 0,001 or 0,002 seconds per instance. To find an optimal schedule using the linear program described in 3.3.1 generates a branch and bound tree of many gigabytes, giving a memory error within 10 minutes of starting the CPLEX algorithm.

To alleviate the high memory use, I first reduced the time horizon of the linear program, T. The original time horizon gives CPLEX a lot of decision variables that are not necessarily needed. The number of machines reduces

time horizon needed. I chose to take T as the time needed by the SRPT schedule plus the maximum processing time. This time T should be enough as for all small instances analyzed, the time needed by the optimal schedule was less or equal to the amount of time needed by the SRPT schedule. In addition, I set the parameter "MemoryEmphasis" of CPLEX to 1. This enables the CPLEX solver to use hard disk space to store parts of the branch and bound tree instead of only being able to use the RAM.

Even with this adaption, trying to solve an instance with 100 jobs is too ambitious with the hardware I have available. After some trial and error, I settled for an instance with 25 jobs, maximum processing time of 4, maximum release time of 9 and 4 machines. Finding an SRPT schedule took 0,007 seconds while the linear program took 36175,1 seconds, over 10 hours. Both schedules found have an objective value of 209, meaning SRPT gave an optimal schedule in this case.

As solving to optimality takes so long, I only did the analysis for one instance but the difference in time needed should be evident. For larger instances, solving to optimality will take even longer while finding an SRPT schedule should still prove very easy.

3.5 Trying to find a worse instance

The worst instances found for 5, 6 and 7 jobs all have the same form. In each of these worst instances, there are three jobs released at time 1, with one longer than the other two. Then at a later time, which is the first time where the first three jobs can all be completed, more jobs are released that are all of the same length as the remaining processing time at that time of the longest job in an SRPT schedule. For the simplest instances in this class, these jobs are released in time slot 3 and have a length of 1. The general form is given in Table 3.9. As b is the double of a number of jobs, it must be a positive real number.

j	1	2	3	4	...	$3+2b$
p_j	1	1	2	1	...	1
r_j	1	1	1	3	...	3

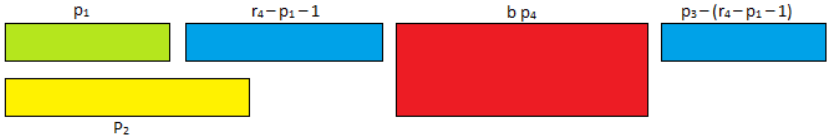
Table 3.9: General form of worst instances of 5 and 7 jobs ($b \in \mathbb{N}$)

Now, it would be interesting to see what the maximum ratio is that can be achieved with instances like those in Table 3.9. Perhaps a slight change to a large multiple of one of the instances will give a greater ratio than $\frac{21}{19}$. To allow for these variations, consider the instances in Table 3.10.

j	1	2	3	4	...	$3+2b$
p_j	p_1	p_2	p_3	p_4	...	p_4
r_j	1	1	1	r_4	...	r_4

Table 3.10: Modified form of multiples of 3.1 and 3.4(a) ($b \in \mathbb{N}$)

The form of the SRPT and optimal schedules for these worst instances all have the same form. In the SRPT schedule, the two shorter jobs are started at time slot 1. After one of these jobs is finished, the longer job is started. Before the longer job is finished, the other jobs are released such that the completion of job 3 is delayed until the end of the schedule. In the optimal schedule, jobs 1, 2 and 3 are done before the jobs are released. The form is displayed generally in Figure 3.4. Here, it is assumed without loss of generality that $p_1 \leq p_2$.



(a) SRPT

(b) optimal

Figure 3.4: General SRPT and optimal schedules for instances of Table 3.10

From the schedules in Figure 3.4 and the general form in Table 3.10, we can deduce that:

1. In the SRPT schedule, for the last part of job 3 to be processed last, $p_4 \leq p_3 - (r_4 - p_1 - 1)$ needs to hold. Otherwise SRPT rules dictate that job 3 must be processed before the other jobs. We can say $p_4 = p_3 - (r_4 - p_1 - 1) - c$ for some $c \geq 0$.

2. From the optimal schedule, as the block can only start after all first 3 jobs are finished, $r_4 > p_1 + p_2$ and $r_4 > p_3$.

3. $p_3 \geq p_2 \geq p_1$, otherwise job 3 would have to be selected before job 1 or 2 in the SRPT schedule.

Now, we want to find the maximum ratio that can be achieved with an instance of the form specified in Table 3.10. First, I will specify the objective values achieved by both schedules.

$$SRPT = p_1 + p_2 + 2\sum_{i=1}^{b}(r_4 + ip_4 - 1) + r_4 + bp_4 - 1 + p_3 - (r_4 - p_1 - 1) \tag{3.1}$$

$$OPT = p_1 + (p_1 + p_2) + p_3 + 2\sum_{i=1}^{b}(r_4 + ip_4 - 1) \tag{3.2}$$

In both expressions, the sum can be simplified by:

$$\begin{aligned} 2\sum_{i=1}^{b}(r_4 + ip_4 - 1) &= 2br_4 - 2b + 2p_4\sum_{i=1}^{b} i \\ &= 2br_4 - 2b + p_4(b^2 + b) \end{aligned}$$

Substituting this in both expressions and simplification of these expressions lead to:

$$\begin{aligned} SRPT &= p_1 + p_2 + 2br_4 - 2b + p_4(b^2 + b) + r_4 + bp_4 - 1 + p_3 - (r_4 - p_1 - 1) \\ &= 2p_1 + p_2 + p_3 + (b^2 + 2b)p_4 + 2br_4 - 2b \\ OPT &= p_1 + (p_1 + p_2) + p_3 + 2br_4 - 2b + p_4(b^2 + b) \\ &= 2p_1 + p_2 + p_3 + (b^2 + b)p_4 + 2br_4 - 2b \end{aligned}$$

The ratio is:

$$\begin{aligned} \frac{SRPT}{OPT} &= \frac{2p_1 + p_2 + p_3 + (b^2 + 2b)p_4 + 2br_4 - 2b}{2p_1 + p_2 + p_3 + (b^2 + b)p_4 + 2br_4 - 2b} \\ &= 1 + \frac{bp_4}{2p_1 + p_2 + p_3 + (b^2 + b)p_4 + 2br_4 - 2b} \end{aligned} \tag{3.3}$$

The goal is to maximize expression 3.3. As p_2 is only in the denominator with positive coefficient, expression 3.3 decreases as p_2 increases. As the

goal is to maximize the expression and the minimum value of p_2 is p_1, the first conclusion is that $p_2 = p_1$ in the maximum ratio instance of this form.

$$\frac{SRPT}{OPT} = 1 + \frac{bp_4}{3p_1 + p_3 + (b^2 + b)p_4 + 2br_4 - 2b} \tag{3.4}$$

We said that $p_4 = p_3 - (r_4 - p_1 - 1) - c$ for some $c \geq 0$. Expression 3.4 becomes:

$$\begin{aligned}\frac{SRPT}{OPT} &= 1 + \frac{b(p_3 - (r_4 - p_1 - 1) - c)}{3p_1 + p_3 + (b^2 + b)(p_3 - (r_4 - p_1 - 1) - c) + 2br_4 - 2b} \\ &= 1 + \frac{bp_1 + bp_3 - br_4 - bc + b}{(b^2 + b + 3)p_1 + (b^2 + b + 1)p_3 + (-b^2 + b)r_4 + (-b^2 - b)c + b^2 - b}\end{aligned} \tag{3.5}$$

Taking the partial derivative with respect to c gives:

$$\begin{aligned}&\frac{\partial}{\partial c}\left[1 + \frac{bp_1 + bp_3 - br_4 - bc + b}{(b^2 + b + 3)p_1 + (b^2 + b + 1)p_3 + (-b^2 + b)r_4 + (-b^2 - b)c + b^2 - b}\right] \\ &= \frac{(-1) \cdot (2b^2(r_4 - 1) + 3p_1 + p_3)}{((b^2 + b + 3)p_1 + (b^2 + b + 1)p_3 + (-b^2 + b)r_4 + (-b^2 - b)c + b^2 - b)^2}\end{aligned}$$

As r_4 will always have a value of more than 1 following from $r_4 > 2p_1$, the expression is negative for all values of the parameters. This means, to maximize expression 3.5, we need to set c to its minimum, $c = 0$. This makes $p_4 = p_3 - r_4 + p_1 + 1$.

$$\frac{SRPT}{OPT} = 1 + \frac{bp_1 + bp_3 - br_4 + b}{(b^2 + b + 3)p_1 + (b^2 + b + 1)p_3 + (-b^2 + b)r_4 + b^2 - b} \tag{3.6}$$

Taking the partial derivative with respect to r_4 gives:

$$\begin{aligned}&\frac{\partial}{\partial r_4}\left[1 + \frac{bp_1 + bp_3 - br_4 + b}{(b^2 + b + 3)p_1 + (b^2 + b + 1)p_3 + (-b^2 + b)r_4 + b^2 - b}\right] \\ &\quad = -\frac{(2b^2 + 3b)p_1 + (2b + 1)p_3}{((b^2 + b + 3)p_1 + (b^2 + b + 1)p_3 + (-b^2 + b)r_4 + b^2 - b)^2}\end{aligned}$$

This partial derivative is always negative as p_1 and p_3 are always strictly positive. To maximize the ratio, we can set r_4 to its minimum value. As

$r_4 > p_1 + p_2$ and $r_4 > p_3$, we know the minimum value of $r_4 = \max\{p_1 + p_2 + 1, p_3 + 1\} = \max\{2p_1, p_3\} + 1$.

To further analyze the ratio of the general form, we now distinguish 2 cases. Case 1 where $2p_1 \geq p_3$ and Case 2 where $p_3 \geq 2p_1$

Let us consider Case 1 where $2p_1 \geq p_3$. Now, $r_4 = 2p_1 + 1$. The ratio becomes

$$\frac{SRPT}{OPT} = 1 + \frac{-bp_1 + bp_3}{(-b^2 + 3b + 3)p_1 + (b^2 + b + 1)p_3} \tag{3.7}$$

If we now take the partial derivative with respect to p_1, we get:

$$\frac{\partial}{\partial p_1}\left[1 + \frac{-bp_1 + bp_3}{(-b^2 + 3b + 3)p_1 + (b^2 + b + 1)p_3}\right] = -\frac{(4b^2 + 4b)p_3}{((-b^2 + 3b + 3)p_1 + (b^2 + b + 1)p_3)^2}$$

This partial derivative is again always negative as $p_3 > 0$, which means to maximize expression 3.7, we need to set p_1 to its minimum. We get $2p_1 = p_3$. Substituting $2p_1$ for p_3 in expression 3.7 gives:

$$\begin{aligned}\frac{SRPT}{OPT} &= 1 + \frac{bp_1}{(b^2 + 5b + 5)p_1} \\ &= 1 + \frac{b}{b^2 + 5b + 5}\end{aligned}$$

Now, let us consider Case 2 where $p_3 \geq 2p_1$. Now, $r_4 = p_3 + 1$. Expression 3.6 becomes:

$$\frac{SRPT}{OPT} = 1 + \frac{bp_1}{(b^2 + b + 3)p_1 + (2b + 1)p_3} \tag{3.8}$$

In this expression, p_3 is only in the denominator and has a positive coefficient. To maximize the expression, we have to set p_3 to its minimum, which is $2p_1$ by the case distinction. Substituting $2p_1$ for p_3 in expression 3.8 gives:

$$\frac{SRPT}{OPT} = 1 + \frac{bp_1}{(b^2 + 5b + 5)p_1}$$

$$= 1 + \frac{b}{b^2 + 5b + 5}$$

So, from both cases we get the same ratio of $1 + \frac{b}{b^2+5b+5}$. The conclusions from this maximization so far are that to maximize expression 3.3, the following must be true:

1. $p_1 = p_2$
2. $p_3 = 2p_1$
3. $r_4 = p_3 + 1 = 2p_1 + 1$
4. $p_4 = p_3 - r_4 + p_1 + 1 = p_1$

What is left is to find the maximum over b of the expression $1 + \frac{b}{b^2+5b+5}$. I plot the values of the ratio for values of b from 1 to 50 in Figure 3.5.

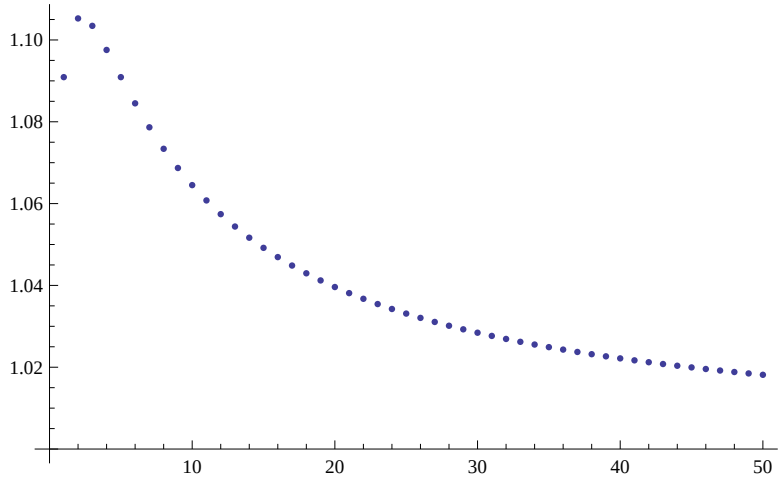


Figure 3.5: $1 + \frac{b}{b^2+5b+5}$ plotted for positive integer $b \leq 50$

The maximum is at $b = 2$, which gives a ratio of $\frac{21}{19}$. If we let $b = 2$ and combine with the other conclusions so far, we end up in a multiple of the [Chung et al. (2010)] instance.

3.6 Conclusions

The analysis showed that all instances with two machines and at most 5 jobs with maximum processing time of 5 and maximum release time of 5 have a

lower ratio than $\frac{21}{19}$ between SRPT and optimal. In addition, all instances with two machines and at most 7 jobs with maximum release time of 5 and maximum processing time of 3 have ratios not more than $\frac{21}{19}$.

Overall, the analysis for small instances showed that on average, SRPT will give an objective value very close to the optimal objective value.

Solving large instances to optimality with the described integer linear program is not feasible on personal computers as even an instance with only 25 jobs took over 10 hours.

The analysis of the general form of the worst instances found showed that this specific form of instances has a maximum ratio of $\frac{21}{19}$. For a greater ratio, a completely different type of instance is needed.

As no instance with a ratio greater than $\frac{21}{19}$ has been found and it is not certain whether this is the greatest ratio possible, the bounds on the approximation ratio of $\frac{21}{19}$ and $\frac{5}{4}$ still stand.

Bibliography

[Baptiste et al. (2007)] Philippe Baptiste, Peter Brucker, Marek Chrobak, Christoph Drr, Svetlana Kravchenko, and Francis Sourd. The complexity of mean flow time scheduling problems with release times. *Journal of Scheduling*, 10:139–146, 2007.

[Chung et al. (2010)] Christine Chung, Tim Nonner, and Alexander Souza. Srpt is 1.86-competitive for completion time scheduling. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1373–1388, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[Du et al. (1990)] Jianzhong Du, Joseph Y.-T. Leung, and Gilbert H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75(3):347 – 355, 1990.

[Graham et al. (1979)] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In E.L. Johnson P.L. Hammer and B.H. Korte, editors, *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979.

[Lu et al. (2003)] X. Lu, R.A. Sitters, and L. Stougie. A class of on-line scheduling algorithms to minimize total completion time. *Operations Research Letters*, 31(3):232 – 236, 2003.

[Philipps et al. (1998)] Cynthia Phillips, Clifford Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

[Sitters (2010)] René Sitters. Efficient algorithms for average completion time scheduling. In Friedrich Eisenbrand and F. Shepherd, editors, *Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 411–423. Springer Berlin / Heidelberg, 2010.

[Vestjens (1997)] A.P.A. Vestjens. *On-line Machine Scheduling*. PhD thesis, Technische Universiteit Eindhoven, 1997.