Utrecht University

Minimising Total Tardiness for Stochastic Resource and Machine Scheduling Problems

Jan Posthoorn

Master's Thesis
ICA-4145666

Supervisors:
dr. J.A. Hoogeveen
dr. ir. J.M. van den Akker
R.W. van den Broek MSc

Department of Information and Computing Sciences
Utrecht University
Netherlands
August 2019

Contents

1 Introduction

Scheduling problems often become difficult for humans to solve when there is too much to plan at once, or when there are too many constraints involved. This is one of the reasons why research into scheduling problems is important. Also, scheduling problems are not limited to one industry, but are very common in many places around the world. One such scheduling problem is the project scheduling problem. This is a very common scheduling problem with applications in many industries like for example, product development, construction, software development, factories, production processes, renovation and many more. Also, outside of industry there are applications, like coaching a team or planning a vacation or a wedding. A well-studied project scheduling problem is the resource constrained project scheduling problem (RCPSP). Here the tasks require some resources to be performed, like machines, or manpower, or tools for example. These resources have a limited capacity and are renewable, so once a task finishes using some resources, they are free to be used by any other task. An example of such a resource constrained project scheduling problem is a software project. Our resources are the different kinds of employees that we need for the project. These can be for example, programmers, testers and designers. Our tasks are things like designing/programming/testing the user interface of the software. Clearly there are some constraints between these tasks, for example, a tester cannot test a piece of software before it has been created by the programmer. The project manager may assign some due dates to some milestones of the project like the first working prototype. If someone proposes a schedule for the software project, then we can compare the planned completion times of the milestones to the due dates that the project manager made in order to get a feel for the quality of the schedule. We can then see if we expect to make all our due dates and if not, how much tardiness we have.

We will be looking at the resource constrained project scheduling problem with a focus on the stochastic variant of the problem. We look at the stochastic variant of RCPSP because it has not been extensively studied like the traditional RCPSP and because it has practical applicability. The best studied variant is the deterministic variant. The most well-studied objective for machine- or project scheduling is the maximum completion time, also called makespan. The makespan is defined as the finishing time of the last task in the schedule. In the general and stochastic settings it is common to look at the (expected) makespan as an objective. For the stochastic setting robustness is something that is also often considered, but there is no consensus on what a good definition for robustness precisely is.

As an objective we will be looking at the total tardiness which is the sum of the tardiness of all the tasks in our schedule. Objective functions based on tardiness have been studied before but not to the extent that makespan objectives have. We feel that tardiness is a very useful objective because in many cases we do not only care about the completion time of the whole schedule but we also care about the completion time of the individual parts of the schedule. For example, if we have a project that consists of multiple milestones, we do not only care about the final project completion, but also about competing these milestones on time.

The RCPSP (resource constrained project scheduling problem) is in its basic form already a hard problem. Blazewicz et al. proved that the RCPSP is $\mathcal{NP}$-hard in the strong sense [2]. This gives the indication that the stochastic variant will also be quite difficult to solve efficiently.

We will focus on stochastic resource constrained project scheduling with robustness in mind. We will also look at the traditional RCPSP because many ideas and concepts concerning this problem might generalise to the stochastic variant of the problem.

The rest of this article has the following layout. In Section 2 we will talk about the formal definition of the problem we study. After that we give a small illustrative example in Section 3 about project scheduling. Then a literature overview of related works will be given in Section 4.

Then we talk about the research questions we want to answer in Section 5. Continuing on in Section 6, we talk about the work we did on machine scheduling and in Section 7 we talk about what we did for the resource constrained project scheduling problem. Then in Section 8 we give our final conclusions about the research and in we finish with some remarks about further research in Section 9.

Variable	Description
n	Number of tasks
R	Number of resources
p_j	Processing time of task j
P_j	Probability distribution of the processing times of task j
d_j	Due date of task j
r_{ij}	Resource requirement of resource i by task j
R_i	Resource availability of resource i
$G = (V, A)$	Precedence graph
C_j	Completion time of task j
T_j	Tardiness of task j

Table 1: The variables and parameters of the RCPSP

2 Formal Definition

There are many variants of the RCPSP and to get a better overview of what people are studying some notation was introduced. Using the adaptation made by Brucker et al. [5] to the original Graham notation [15] for resource constrained project scheduling problems the problem we are looking at is denoted as $PS|prec, stochasticprocessing|\sum T_i$, which means we are looking at project scheduling with precedence constrains and stochastic processing times and our objective is to minimise the total tardiness. We will now discuss what all these elements really entail. There is a project with n tasks and we must complete each task in order to complete the project. There is also a dummy task 0 which represents the start of the project. Each task has a given processing time of p_j and a given due date d_j. The dummy task has a processing time of 0 and an infinite due date. There are R types of resources and each task j has a value r_{ij} which represents how many units task j requires of resource i during its execution. All resources are renewable and will be released by a task once it completes. A task requires all its resource requirements to be met during the whole duration of its execution. At each moment in time there are R_i units of resource i available. There are also precedence constraints between the tasks. These constraints are given in the form of a precedence graph $G = (V, A)$, where V is the set of all tasks and A represents the precedence constraints. If G contains an arc from task j to task k this means that we can only start task k when we have finished task j. These are finish-start zero-time lag precedence constraints. G contains an arc from the dummy task 0 to each task, which represents that we cannot start any project task before we start the project. The main objective is to minimise the total tardiness over all tasks. We define the tardiness T_j of task j using its completion time C_j in a given schedule, and its due date d_j. The lateness L_j of task j is defined as $C_j - d_j$ and the tardiness, T_j of task j is defined as $\max(0, L_j)$. This combined gives us the following objective: $minimise : \sum_{j=1}^{n} \max(0, C_j - d_j)$, or shortened: $minimise : \sum_{j=1}^{n} T_j$

The stochastic resource constrained project scheduling problem (SRCPSP) is an extension of the RCPSP. It is the same as the RCPSP described above but there are some key differences. We now associate a probability distribution with the processing time of each task instead of using a deterministic value. The processing time is defined by a set $P_j \subset \mathbb{R}^+$ which contains all possible realisations of the task processing time. Another difference is that we now want to optimise the expected total tardiness with respect to the processing time. Other than these extensions it is the same as the traditional RCPSP. Table 1 contains a summary of all variables and their descriptions.

Task	A	B	C	D	E	F	G
Probabilities	(0.9: 2) (0.1: 3)	(0.9: 2) (0.1: 3)	(0.75: 1) (0.2: 2) (0.05: 3)	(0.75: 1) (0.2: 2) (0.05: 3)	(0.9: 2) (0.1: 3)	(0.75: 1) (0.2: 2) (0.05: 3)	(1.0: 3)

Table 2: The probability distributions of the processing times of the tasks in the example

3 Small Example

Here we will give a small example project to illustrate stochastic resource constrained project scheduling. We will first describe an example problem and then show an example schedule we can create.

In Figure 1 you can see 7 non-dummy tasks in an example project about making paint. The goal of the project is to make magenta paint from blue and red paint and to verify that all the paints that we make conform to our standards. Each task has a name represented by a single letter and each task has a short description of what it represents. Lastly each task contains a list of numbers. The bottom left number defines the due date (d_j), the one after that defines the resource requirement for resource type 1 (r_{1j}), the last one is the resource requirement for resource type 2 (r_{2j}). The first resource type represents paint machines and the second one represents workers. An arc in this figure represents a precedence relation, we can see for example, that we need to complete task A before we can start executing task C, in other words, we need to make paint before we can test that paint. All tasks can only be started after the dummy *Start* task, and all tasks must be completed before we can perform the dummy *End* task. These dummy tasks have an infinite due date and a processing time of 0. We will now look at the problem of finding a schedule for this problem instance. We have 2 machines available and 3 workers. The full probability distributions for the processing times for all the tasks can be found in Table 2.

A possible schedule can be that we start with task A, then start tasks B and C in parallel and after task B we perform task D. Next, we start task E, and lastly, we start tasks F and G in parallel. The resource profile for workers and paint machines can be found in Figures 2 and 3 respectively. These figures assume that all the processing times turn out to be equal to their most likely value. It can be easily verified that we do not violate any of the precedence constraints and using the figures it can be seen that we also do not violate any of the resource constraints. The colours in these figures have no meaning for the scheduling problem, they are just there because this example is about coloured paint.

If we have the total tardiness as our objective, we can see that B, D and G finish later than their due date with the processing times as seen in example Figures 2 and 3, which gives them a positive tardiness. The total tardiness for this example schedule becomes: $1 + 2 + 2 = 5$. If we evaluate all the 216 possible combinations of processing times, we can compute that the expected total tardiness of this schedule is 7.74675. We can also see that in the worst-case, we get a total tardiness value of 30.

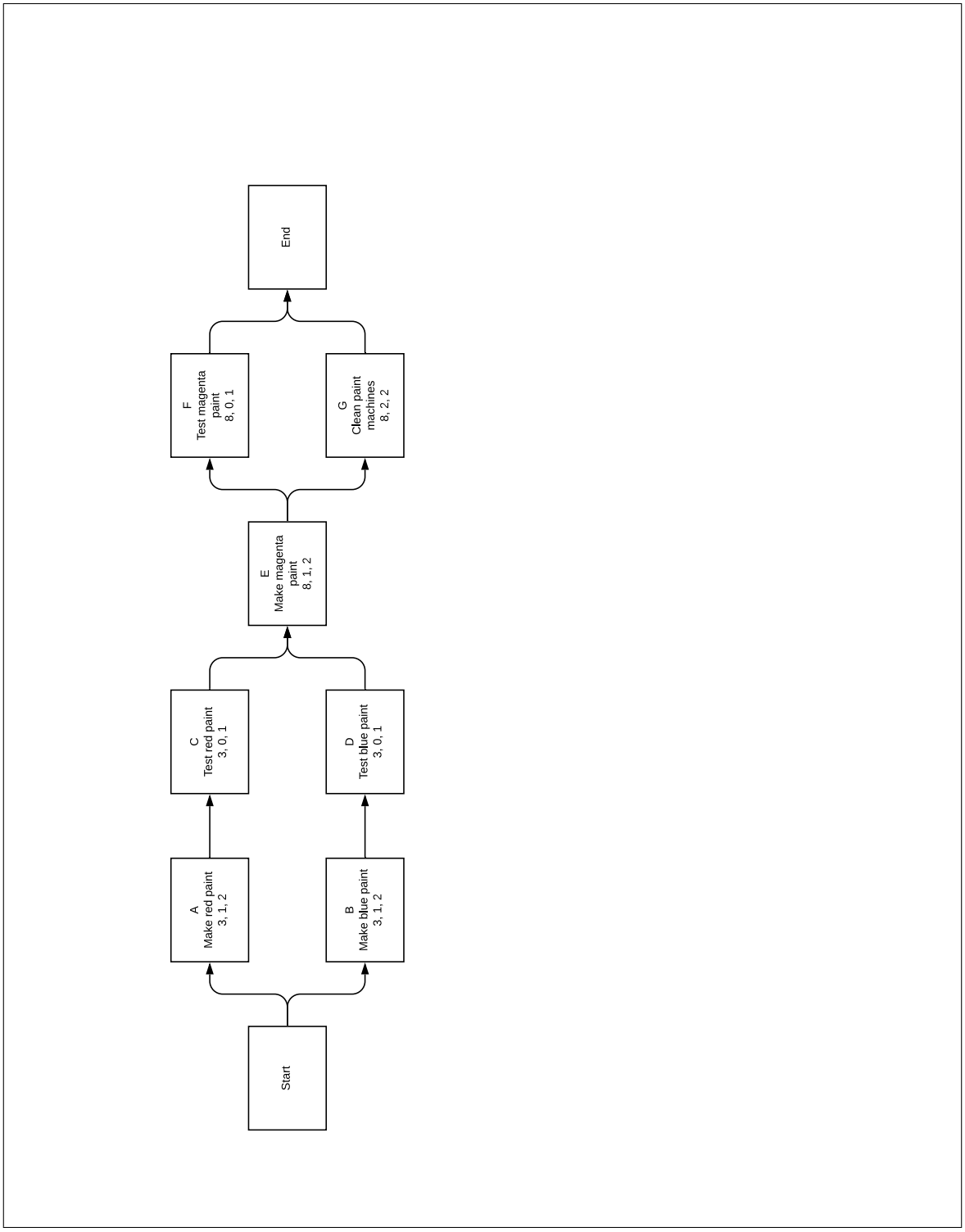

Figure 1: A small example about making paint

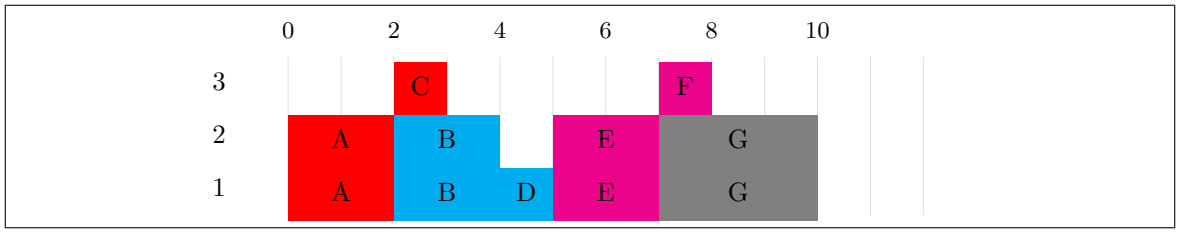

Figure 2: The worker resource usage of the schedule

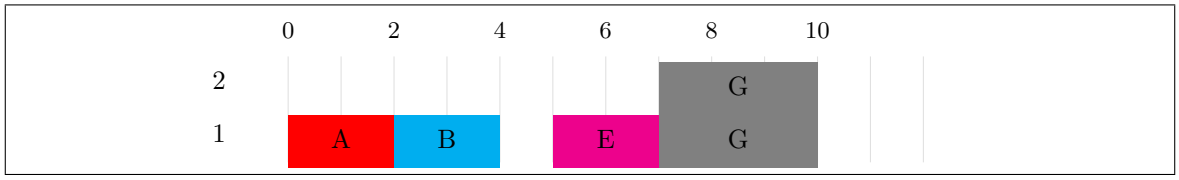

Figure 3: The paint machine resource usage of the schedule

4 Related Work

RCPSP is a well-studied scheduling problem, just like machine scheduling. There are some variants like the stochastic variant of machine scheduling or RCPSP that are less well-studied but have great practical applicability. RCPSP is a generalisation of machine scheduling, we can not only schedule machines, but more general resources in the RCPSP setting.

The rest of this section will be structured in the following manner. First in Section 4.1 we will look at approaches to solve the stochastic RCPSP. Then in Section 4.2 we will look at also taking robustness into account. Continuing on in Section 4.3 we will look at other approaches and related settings. And lastly in Section 4.3.5 we talk about some surveys that have been done on machine scheduling and RCPSP.

4.1 Stochasticity

There are many approaches on how to deal with stochasticity within SRCPSP. Below we will discuss some of these approaches. We will start by looking at the deterministic RCPSP in Section 4.1.1. Next, we will look at an approach called baseline scheduling in Section 4.1.2 and lastly we will look at more general approaches for stochastic scheduling in Section 4.1.3.

4.1.1 Deterministic RCPSP

Before we consider the stochastic variant of the problem, we will first look at the deterministic case. Many of the ideas of the non-stochastic variant of the problem might still be usable in the stochastic variant, that is why it is worthwhile to study this.

Vanhoucke et al. [30] looked at the deterministic RCPSP with minimising weighted earliness-tardiness penalty costs as objective. They develop a branch-and-bound algorithm. They generate lower bounds by solving the same problem, but without the resource constraints exactly, we call this the subproblem. If the solution to the subproblem does not violate any resource constraints, they have found the optimal solution. Otherwise they look at the first resource conflict in the schedule and add new precedence constraints between the tasks in the conflict as a branching strategy. They branch on the set of *minimal delaying alternatives* which are a minimal subset of tasks which resolve

the resource conflict when delayed. They compute a solution to the subproblem in the following way. First, they set the finishing time C_j for each task to be its due date d_j or later. This is done based on the precedence relations of the tasks and their d_j. Next, they look for sets of tasks, which can be shifted backwards in time and decrease the earliness-tardiness penalty. They continue shifting until no single shift will result in an improvement. They use this subproblem for a lower bound for their branch and bound algorithm.

Shadrokh and Kianfar [28] study the Resource Investment Problem (RIP), which is an extension of RCPSP where the number of resources available R_i during the project is given but we can buy more resource units of each resource for a cost. The objective function they look at is weighted tardiness plus weighted resource cost. They use a genetic algorithm with crossover, mutation, immigration and local search. The local search works roughly as follows. For each type of resource, look at the tasks which are being executed at the time we use the maximum number of units of this resource type. Then we check if we can shift some of these activities and whether it results in a better solution.

Luciano et al. [23] look at an application of RCPSP. Their case involves the docking of ships at a port. Here the docking spaces are the resources. They study multi-mode resource constrained project scheduling problem, which means that tasks can have multiple modes of execution. For example, a task can be completed faster, but this takes more resources. The question they want to answer is the following: What are the ideal arriving times for a set of ships by minimising the earliness and tardiness cost. They use a genetic algorithm to solve the problem. They use crossover and mutation. They evaluate their solutions using simulation. They look at random arrival dates multiple times and evaluate their individuals on multiple instances to get a more realistic view of the proposed solution. They also apply some form of local search to the individuals where they try to change the modes of the tasks and see if this gives better solutions.

4.1.2 Baseline Scheduling

A common approach is to first generate a baseline schedule (also called preschedule or predictive schedule); this is a schedule in which we ignore the stochastic part and create a deterministic schedule and afterwards or during execution changes can be made to this schedule. A possible strategy is setting all the processing times p_j equal to the expected value of the stochastic processing times $E(P_j)$ and using these new processing times in the creation of the schedule.

Deblaere et al. [12] try to create stable baseline schedules that optimise the makespan. They say that a schedule is stable when it can prevent disruptions from propagating. This is not a very precise definition, and again just like robustness, there is no consensus on what a good definition for stability is. Their baseline schedules satisfy all the precedence constraints and all renewable resource constraints, but the schedules are made with deterministic processing times. They develop three integer programming-based heuristics to solve this stability problem. They define stability as solution robustness. Solution robustness is robustness in the starting times of the jobs. This means that we want a low amount of deviation in the starting time of each job. They achieve stability in their baseline schedule by allocating the resources in such a way that processing time disruptions have minimal effects on the rest of the schedule. Their approach is reactive, and it tries to stabilise the baseline schedule. This means that they try to create their baseline schedule in such a way that it will stop disruptions from propagating and thereby influencing the rest of the baseline schedule. The first heuristic tries to minimise the number of extra arcs that are imposed by the resource allocation. They determine for every unit of resource the sequence of tasks that it services. This will give rise to a new set of implied precedence constraints. They solve the resource problem by modelling it as a flow network, where the flow represents the flow of resources. Then they find a feasible resource

flow. Then they use a procedure called *chaining* to construct a *chained Partial Order Schedule* which represents a set of solutions of the RCPSP with a temporal graph. This can be used as a baseline during execution and when disruptions occur the schedule can be changed resulting in the need of a reactive scheduling procedure. They show that all their proposed heuristics work better than existing algorithms on small to medium sizes instances.

4.1.3 Stochastic RCPSP

In all these previously mentioned settings, we assume that all information like processing times p_j of tasks, due dates d_j etc. is known beforehand and is deterministic. Or we just ignored the stochastic part of the problem in the creation of the schedule. A more realistic model in many cases is a stochastic model where for example the p_j of a task is modelled by a random variable P_j. It is also not very realistic that all these distributions are known beforehand, but approximations can be made. Another goal that can be achieved by using stochastic processing times is to model disruptions or other unforeseen circumstances that can occur during the schedule's execution, this will be discussed in Section 4.2.

Chalrabortty et al. [7] use a branch and cut approach to find a good solution for the RCPSP with stochastic processing times. Their objective is minimising the makespan. With this aim in mind, they develop six heuristics for solving the problem. All these heuristics are based on the branch and cut approach. Because it is hard to find a single heuristic that works well in a wide variety of cases, developing multiple heuristics using the same underlying tools has some advantages. They assume that the variability of the input data is bounded by some parametric form they describe and with this form they apply robust optimisation to solve the problem, because they transformed the stochastic problem into a deterministic robustness problem which is easier to solve, and which is more studied. They do this by modelling uncertainty of the processing times in a parametric way. They use the resource and precedence constraints to find some cuts for their branch and cut approach. They generated resource cuts based on the sets of tasks that can never be executed at the same time. They also generated cuts based on precedence relations. Because the P_j of a task has a parametric form, they use this to explicitly enumerate some of the precedence constraints to derive cuts from that. According to their results this approach works better when the instances get large (100 tasks or more) compared to other exact approaches. But their approach does require quite a bit of time.

4.2 Robust Scheduling

In many practical applications we do not only want good schedules (schedules that minimise the objective function), but also want our schedule to be resilient or robust. When robustness is considered it often becomes a second objective next to the normal objective function. There is no general definition of robustness, but many measures have been proposed to measure robustness. A common concept in these is the concept of slack. There are two main slack-based definitions. Total slack is the amount of time by which the C_j of a task can exceed its earliest possible completion time without delaying the project completion time. And free slack is the amount of time by which the C_j of a task could be delayed without affecting the earliest start time of its immediate successors in the project. Another important aspect of robustness is what kind of robustness we want. Do we want solution robustness (deviation in start times of jobs) or quality robustness (deviation in project duration)?

Van de Vonder et al. [29] compared three approaches that all used buffers to make a project scheduling solution more robust. Two of their heuristics are based on the principle of *critical chain*

buffer management by Goldratt [14]. The other heuristic is based on *adapted float factor model* developed by Leus [22]. They evaluated these approaches using simulation and concluded that none of the approaches dominated the other approaches for both solution- and quality robustness under varying circumstances. This shows that it is hard to find a general method for generating robust solutions.

Hazir et al. [18] look at the expected net profit as an objective function, where we lose some money for each time unit we are late for a job in accordance with the tardiness penalty and earn some money for each time unit we are early for a job in accordance with the revenue rate . Their works as follows. They first find a solution that maximises the net profit and look at the objective value this gives. They call this value the 'threshold budget'. Then for the second phase they solve again but with an inflated threshold budget as an extra constraint on the budget, now they optimise for robustness, for them this means maximising the project buffer size, instead of optimising for net profit. The project buffer size is defined as the difference between the project deadline and makespan, divided by the deadline. This comes down to minimizing the makespan. They considered quite a few other measures that all try to measure robustness. Some examples are: average slack, weighted sum of slack, slack combined with a utility function which gives diminishing returns for giving more slack to a single task, dispersion of slack, percentage of potentially critical tasks, project buffer size. The measures were compared by first generating a baseline schedule according to some fixed policy and calculating the value of the measure for this schedule. Then they run a simulation multiple times and compute the correlation with the performance measures they defined. These measures are PM_1 the proportion of runs in which the project finishes on time and PM_2 the average delay in project completion time as percentage of the project deadline. They conclude that the project buffer size is the best measure. They also conclude that the inflation of the budget for the second phase greatly increases the robustness.

Chtourou and Haouari [8] also try to solve the stochastic RCPSP and their focus is on finding robust solutions. They use a two-stage priority rule-based algorithm. They develop a heuristic and in the first stage they run this heuristic many times and because of the randomness in some parts of the heuristic, many different solutions are generated. They look for the smallest makespan value they encounter, and this will be the threshold for the next stage. In the next stage they use a different heuristic that must find a solution with at most the threshold makespan and which is as robust as possible. They tried many combinations from 16 heuristics and 12 robustness measures all found in literature, to find the best combination for their algorithm. They used simulation as a validation tool. Their robustness measures also use slack, but they also sometimes include the number of immediate successors of a task. The heuristics consist of some method of destruction and reconstruction of the schedule and a priority rule for selecting tasks to add to the schedule. Their experiments show that, when comparing robust and non-robust schedules with that same makespan value according to the first phase, the robust schedules outperformed the non-robust scheduled in terms of makespan most of the time according to the simulated results.

Artigues and Leus [1] also try to solve the stochastic project scheduling problem with robustness in mind. Their primary objective is minimising the makespan. They define robust as performing well across all scenarios, they translate this into optimising for the minimax absolute regret. This means that they want the worst-case to be as good as possible. They looked at a scenario relaxation approach which solves the problem for only a subset of all possible scenarios. They choose to only look at the extreme scenarios. Each scenario is a realisation of the processing times P_j. They use a MILP to get the best earliest-start policy for a given scenario. These policies will be explained in Section 4.3.1. They start with one scenario and keep on computing a lower and upper bound for the absolute regret until the lower bound is equal to the upper bound. They improve these bounds by adding more scenarios which are the worst-case scenarios for the optimal earliest-start policy from

the previous round. Doing this until the lower- and upper bound are equal is their exact approach, which is very slow according to the authors. Then they propose a heuristic that solves the problem in the same manner but now they stop when the lower and upper bound are reasonably close, which is the case when the difference between the upper- and lower bound divided by the lower bound is smaller than a fixed epsilon.

Van den Broek et al. [4] looked at the related problem of generating robust shunting plans. These shunting plans are also subject to resource and precedence constraints. They generate a baseline schedule and a simple scheduling policy which can reschedule parts of the baseline schedule if disruptions occur. The baseline schedule is a partial order schedule and the scheduling policy is the earliest start time policy. The goal is to find a robust baseline schedule. They define robustness of a schedule as the likelihood that all trains depart on time from the shunting yard. The main topic of this paper is about how to measure robustness in this setting. They compare multiple measures like, sum of total slacks, sum of free slacks, makespan, and some more complex measures like the probability that the makespan is smaller than a certain value or an approach based on $E(P_j)$, which can be computed by solving a linear program. They make the observation that if a path is longer then the likelihood of a disruption appearing on a path is higher. They also propose 2 measures that use normal distributions to approximate processing times in combination with paths. One of these measures is the minimum probability that a path completes before its deadline, over all paths. They compare 13 different measures in total for approximating the robustness. The evaluation is done using simulation of the shunting plans for 2 real world locations. They show that measures that use normal distributions as approximations of P_j have a high correlation with robustness.

4.3 Other approaches and problem settings

The next four subsections will be about other approaches to deal with stochastic scheduling or other related problems to RCPSP and possible approaches to solve them.

4.3.1 Scheduling Policies

Another way of solving a stochastic scheduling problem is by not creating a schedule but by formulating a policy. Using this policy at every decision point we can decide which task(s) we will execute next. The most common policy is called the elementary policy, which can only use the information obtained by executing the schedule up to this point.

Emmons and Pinedo [13] showed that for the stochastic machine scheduling case with stochastic processing times P_j, due dates d_j and weights on identical parallel machines we can still do well. They minimise total weighted tardy jobs and prove for multiple different problem settings what the optimal simple scheduling policy is when assuming that all processing times are independent. These scenarios include pre-emption for example.

Creemers [9] showed that we can still get good results with fewer assumptions about the distributions of the processing times. He uses phase type distributions to approximate the underlying distribution. With these new distributions he can now create an optimal algorithm using stochastic dynamic programming and Markov chains, but this approach does not work well on large instances, and it only finds the optimal simple policy. Because we use multiple distributions to approximate a distribution this can give us some extra decision points to make for example pre-emption choices.

4.3.2 Online Setting

Another related setting is the online setting. In this setting, the jobs that need to be scheduled are unknown in advance and during project execution they are revealed, and we need to schedule them.

Gupta et al. [16] looked at stochastic online machine scheduling, with minimising the expected total weighted completion time as the objective. They show that their online algorithm is an $(144+72\Delta)$-competitive algorithm and that this bound is tight. Here Δ is the squared coefficient of variation of a random variable. The algorithm schedules tasks based on the *instantaneous expected increase*, which is the cost for a machine m if we would schedule all the jobs currently scheduled on m in decreasing order of weight over expected processing time. They schedule the new job on the machine with the lowest *instantaneous expected increase*.

4.3.3 Machine Scheduling

Machine scheduling is a well-studied special case of RCPSP. Here we have only one type of resource, namely machines, and all our tasks must be executed on these machines. In the RCPSP setting all units of a resource are equivalent (each person/machine/tool/...), but there are some variants of machine scheduling where not all machines are the same. In some cases when we make assumptions about the distributions in the stochastic variants of the problem we can get some optimality results. If we look at the multiple identical machine scheduling problem with pre-emptions, Bruno et al. [6] assumed that the processing times P_j followed an exponential distribution. Using the memorylessness property of this distribution they proved that a SEPT (Shortest expected processing time first) policy is optimal when optimising for total expected flow time or weighted completion time, and that a LEPT (Largest expected processing time first) policy is optimal in the case of makespan optimisation. This was later extended by Kampke [21]. He determines the optimal policies for more general cases where SEPT and LEPT are not always optimal. He generalises the objective function and under some restrictions adds resource and precedence constraints while still being able to find the optimal scheduling policy.

Wu and Zhou [31] make no assumptions about the underlying distribution of stochastic variables, but they only look at the single machine case. In their problem settings the processing times and the due dates are stochastic variables. The objective function they consider is the expected maximum lateness. They first develop a way to solve the deterministic case and then extend it to the stochastic variant. They use dynamic programming to solve the problem. This exact approach becomes slow when the instances get large, because of this they also develop 3 heuristics. The first heuristic is based on the *earliest due date* first rule. They assume that the due dates are exponentially distributed for the heuristics, and they schedule the tasks with the smallest exponential parameter λ first. Their other two heuristics work in a similar way, but they also take the processing time into account next to the parameter λ.

Passage et al. [26] used a local search approach for solving the machine scheduling problem. They used approximations based on normal distributions to find good estimates of the expected makespan to guide the local search effectively, because using simulation is time too consuming. They show that the maximum of two normal distributions can be approximated quite well by another normal distribution in many cases. The accuracy they achieve with this approach is comparable with running 300 simulations per iteration of local search, but their approach is much faster. They also compare multiple local search approaches, namely ILS and VND.

4.3.4 Related Problems

Another related scheduling problem is the job shop scheduling. Here each job has a list of operations that need to be completed in a specific order and each operation needs to be performed on a specific machine. Job shop scheduling is a special case of RCPSP. If we say that n is the number of jobs and m is the number of machines, we can model it by adding $n + m$ resource types all with a capacity of 1. Here the first m resource types correspond to the machines and the other n resource types

represent the jobs and these are needed to enforce that multiple operations of the same job cannot be scheduled at the same time. One approach to solve this problem based on a decomposition was described by De Bontridder [11]. He wanted to minimise the total weighted tardiness in a job shop scheduling setting. He used a tabu search approach to determine the order of the jobs on each machine and with that given he can solve a maximal cost flow problem to find the optimal starting times for all jobs given their ordering on the machines.

A common project modelling method is PERT (Project Evaluation and Review Technique). Using the PERT method, a PERT network can be created which represents the project with all its individual tasks and precedence relations between them. If the processing times of tasks are independent and exponentially distributed, we get a Markovian PERT network. Creemers et al. [10] tried to tackle the problem of creating schedules for Markovian PERT networks. Their objective was the *net present value*, where each task generates a cash flow c_i, positive or negative at its start. To account for change of value of money over time we define a variable r as the applicable continuous discount rate. The present value of the cash flow c at time t equals ce^{-rt}. They compute all sets of tasks that can be performed in parallel and use recursion to find the optimal solution.

4.3.5 Surveys

For a recent survey on stochastic machine scheduling, we refer to Pinedo [27]. For surveys on RCPSP we refer to Herroelen et al. [19], Herroelen and Leus [20], Hartmann and Briskorn [17] and Brčić et al. [3].

5 Research Questions

All these questions that are formulated below are about SRCPSP with total tardiness as objective, but we will also answer these questions for the simpler machine scheduling case. We want to see what answers to our questions might change for this related problem setting. In Section 6 we will give some more insights about the machine scheduling problem and how it is related to the RCPSP.

The first question is: *How to best assess the expected total tardiness?* Because of the stochastic and complex nature of the problem it is very difficult and time consuming to determine the expected value of the objective exactly. Therefore we will look at alternative methods to get an assessment of the expected total tardiness. Our next question is *How to best use local search to solve our problem?* In this case *our problem* refers to the SRCPSP or the machine scheduling problem. The problem instances quickly get too large to be solved in an exact manner, that is why we will focus on a local search-based approach. We want to answer these two questions for both the machine scheduling and the SRCPSP setting.

6 Machine Scheduling

Before we look at the full resource constrained project scheduling problem, we will first look at a simpler variant of the problem: the machine scheduling problem. The machine scheduling problem is the same as the RCPSP where we have a single resource type and each task requires one unit of this resource type during execution. Other than that, they are the same. This single resource type represents the machines where we need to schedule our tasks on. We can have multiple units of this resource, or in other words, multiple machines.

6.1 Problem Setting

We will start with the problem definition of the machine scheduling problem. There are n tasks, and we must complete each task. There is also a dummy task 0 which represents the start of the project. Each task has a given processing time of p_j and a given due date d_j. The dummy task has a processing time of 0 and an infinite due date. We also have m machines, and each machine can process one task at a time. There are also precedence constraints between the tasks. These constraints are given in the form of a precedence graph $G = (V, A)$. Here V is the set of all tasks and the set A represents the precedence constraints. If A contains an arc from task j to task k this means that we can only start task k when we have finished task j. These are finish-start zero-time lag precedence constraints. G contains an arc from the dummy task 0 to each task, which represents that we cannot start any project task before we start the project. We use the term direct precedence constraints, for these precedence constraints. The main objective is to minimise the total tardiness over all tasks. The tardiness of a task is the penalty we assign to the task if we complete it after its due date . We define the tardiness T_j of task j using its completion time C_j in a given schedule, and its due date d_j. The lateness L_j of task j is defined as $C_j - d_j$ and the tardiness, T_j of task j is defined as $\max(0, L_j)$. This combined gives us the following objective: $minimise : \sum_{j=1}^{n} \max(0, C_j - d_j)$, or shortened: $minimise : \sum_{j=1}^{n} T_j$.

We will look at the stochastic extension of the machine scheduling problem. In this variant of the problem we associate a probability distribution with the processing time of each task instead of using a deterministic value. We assume that all processing times are independently distributed and follow the same type of distribution with expected processing time p_j and variance $\sigma_j^2 = \sigma^2(D)p_j^2$. Here $\sigma^2(D)$ is a given constant. For task j, we denote the stochastic processing time by P_j. Another difference is that we now want to optimise the expected total tardiness with respect to the processing time. Other than these extensions it is the same as the machine scheduling problem.

6.2 Solution Representation

We will not consider unforced idle time in the solution, because it is not useful for this problem. This means that a machine can only be idle if there are no more tasks planned on the machine or if the next task planned on the machine is not yet available. The reason for this is the following, if we have a schedule with some unforced idle time and we remove this idle time the schedule will never get worse. The completion times of some tasks might decrease, and this can't have a negative impact on the total tardiness. Because of this requirement we can represent our solution/schedule as a Partial Order Schedule (POS). This means that we can represent our solution as directed acyclic graph. In this graph the vertices represent the tasks and the arcs represent the order in which we need to perform tasks. Between each pair of consecutive tasks on a machine we add an arc. This graph contains m chains which are the chain of tasks on each machine. These chains consist of all the tasks which are scheduled on a specific machine. There are also arcs between tasks for which

we have precedence constraints. This means that we can also have arcs between the chains of the different machines. This graph must clearly be acyclic otherwise we could never execute the schedule while honouring all precedence constraints.

We also add an extra dummy task at the start of each machines schedule. These dummy tasks are used by the local search operators and have no other purpose. We call this set of tasks the 'dummy tasks'. This set of dummy tasks does not include the dummy precedence task which is used as a starting point for the precedence constraints. We will call the set of all tasks, including the dummy tasks, 'all tasks'. And lastly, we will call the set of all tasks excluding the dummy tasks the 'real tasks'.

6.3 Overview

We will use a local search approach based on simulated annealing to try and solve this problem. We will first describe the components of the local search approach in a high-level manner and then each of the important components will be discussed separately in the upcoming sections. First, we will generate an initial solution as described in Section 6.4. Then we will use simulated annealing, as described in Section 6.7, with the neighbourhoods described in Section 6.5 to improve that solution, in a first improvement manner. After each step we will evaluate our new solutions with one of the methods described in Section 6.8. Then we use simulated annealing to determine if we should accept the new solution or not. In each step we use the method described in Section 6.6 to select which neighbourhood we will apply. We repeat this until we failed to improve the solution 5000 times in a row and then we check all possible neighbours in all our neighbourhoods to check if we are indeed locally optimal. If we are, we stop, if we are not, we apply the improvement we just found and continue the procedure of trying random neighbours from random neighbourhoods. We also stop the search after half an hour because of time constraints. How the parameters were determined for all these methods can be found in Section 6.11. How all the instances for the parameter tuning and other experiments were generated can be found in Section 6.9. The setup that was used for all the experiments is described in Section 6.10.

We choose to use an approach based on local search because we choose the problem setting for its practical applicability, and exact methods are not feasible for problem instances of practical size.

6.4 Initial Solution

Before we can start our local search, we will need to have some starting point. In this section we describe how we generate our initial solution. We start out by dividing the set of all real tasks into three parts: the available tasks, the unavailable tasks and the planned tasks. The planned tasks are all tasks which are assigned to some spot in the schedule. The available tasks are all tasks for which we have planned all their predecessors but not themselves. And lastly, the unavailable tasks are all tasks for which not all predecessors have been planned yet. After initialising all the task sets, we repeat the following steps until we have scheduled all tasks. We first pick a random available task and schedule it at the end of the already scheduled sequence on a random machine. Next, we move the newly planned task to the set of planned tasks. Then we look at all the successors of the newly planned tasks and check if we can move them from the unavailable set to the available set. Continuing in this fashion gives us an initial solution.

6.5 Neighbourhoods

We define 2 neighbourhood/local search operators: the shift and the swap. In the following sections we will discuss how these operators work, how we can check if a specific shift/swap is allowed and how we can randomly generate shifts/swaps. For explaining the next sections, we will need to define some terms.

First, we will define the direct- and implied predecessors/successors. The direct predecessors of a task j are all the precedence predecessors of task j. The set of implied predecessors of task j contains all the precedence predecessors of task j, this also includes the predecessors of its predecessors. There is more, we also include all the resource predecessors and their implied predecessors. The resource predecessors are the machine predecessors, but we define it in the more generic way to make sure that we can use the same definition when we talk about the RCPSP. In other words: The implied predecessors of task j, are all its direct predecessors and their implied predecessors, combined with its resource/machine predecessor and the implied predecessors of the resource/machine predecessor. The direct- and implied successors are defined in a similar manner. An important note is that the direct predecessors/successors are not dependent on the schedule, but the implied predecessors are, because they depend on the resource/machine predecessors/successors. Because of this, they can't be determined in advance before the search.

6.5.1 Shift

The shift operator shifts a single task to a different place in the schedule. This can also be to a different machine. We define the shift by a pair of tasks $(T1, T2)$. We will call $T1$ the place task and $T2$ the shift task. We will shift the shift task to the spot after the place task. We can select $T1$ randomly from the set of all tasks (including the dummy tasks). We can then select $T2$ from the set of real tasks. In this manner we can select all possible shifts, but not all shifts are always allowed. What is allowed cannot be totally determined in advance because it depends on the current schedule. We can check if we can place task $T2$ after task $T1$ by checking if the set of implied predecessors of task $T2$ placed after task $T1$ does not have any overlap with the set of implied successors of task $T2$ placed after task $T1$. If this is the case, the shift is feasible. We then perform the shift by moving task $T1$ directly behind task $T2$.

6.5.2 Swap

The swap operator swaps the places of 2 different tasks in the schedule, which can be placed on the same or on different machines. We define the swap by a pair of tasks $(T1, T2)$. We can select $T1$ and $T2$ randomly from the set of real tasks. In this manner we can select all possible swaps, but not all swaps are always allowed. What is allowed cannot be totally determined in advance because it depends on the current schedule. In a similar manner as we did for the shift, we can compute the set of implied- predecessors and successors for both tasks and check that there is no overlap between the set of predecessors and successors for both tasks. We can then perform the swap by switching the placing of task $T1$ and task $T2$ in the schedule.

6.6 Neighbourhood Selection

Because we have multiple neighbourhoods that can be used during the search, we need a method to decide which method to use at what time. Currently we only have two neighbourhoods, but later on there will be more. We use a method for this that computes the expected score decrease for each neighbourhood and uses that information to determine a selection probability for each

neighbourhood. The idea behind this approach is that the 'optimal' probabilities for each operator can be different during, for example, the start of the search compared to the end of the search. This method should hopefully solve this problem by determining the probabilities dynamically during the search.

The advantage of this method is that it is adaptive to the search and will encourage the selection of neighbourhoods that are currently performing well. To prevent overfitting the probabilities associated with the neighbourhoods we introduce some parameters. We introduce 3 parameters:

- expectation adaptation A_E
- probability adaptation A_P
- minimum probability m_P

We want to prevent that a neighbourhood is no longer selected because it is currently not very effective compared to the other that is why we introduce a minimum probability for each neighbourhood. The adaptation parameters are there to prevent overfitting. We want to prevent that if we make a single very good move in a neighbourhood that that neighbourhood immediately starts dominating the other neighbourhoods.

We initialise all operators with an equal probability and set the expected value of applying each operator to zero.

If we call the expected decrease of neighbourhood i: $E(i)$ we update this expected decrease in the following way: $E(i) = (1 - A_E) * E(i) + A_E * \Delta$, where Δ is the difference in score between the old and new schedule. Then we determine the minimum value of all the expected decreases $min_{E(i)}$. We then compute the *adjusted total* as $ET = \sum_{i=1}^{n}(E(i) - min_{E(i)})$ where n is the number of neighbourhoods. Next, we compute the new probability for operator i as $P_{new}(i) = (1 - n * m_P) * ((E(i) - min_{E(i)})/ET) + m_p$. Then we set the probability $P(i) = (1 - A_P) * P(i) + A_P * P_{new}(i)$.

I will now explain why the result is always a proper probability distribution again. If we look at $(1 - n * m_P)$ in formula for $P_{new}(i)$ we see that we first compute the fraction of the distribution that we need to distribute over all the operators, this is most likely smaller than 1 because of the minimum probability we assign to each operator. We defined ET as the sum over all $(E(i) - min_{E(i)})$, this means that the sum over all operators for the second part $((E(i) - min_{E(i)})/ET)$ would equal 1. And lastly, we add m_P to each $P_{new}(i)$ to make it a proper probability distribution. There is one edge case for this approach. If all the $E(i)$'s are equal, ET is equal to 0, and in the formula for $P_{new}(i)$ we would divide by 0. If we detect this problem, we can randomly slightly perturb all the $E(i)$'s and try again. It is very unlikely that after this step all the $E(i)$'s will be equal again.

In Section 6.11 we give some information about which parameter values were used for the experiments. When choosing values for the parameters there are some restrictions. The the first restriction is that A_P and $A_P \in [0, 1]$. A higher value for any of these parameters will cause the probabilities to change more quickly and fit the currently observed fitness landscape better. A too high value for these parameters will make this method vulnerable to outliers and a too low value might cause the method too not adequately adapt its probabilities to the current operator performance. The m_P must be chosen between $[0, u]$, where u is the reciprocal of the number of operators that we are using.

6.7 Simulated Annealing

To increase the effectiveness of the local search we will use simulated annealing instead of just hill climbing. It works like this. If we find a new solution we look at its estimated total tardiness according to our current evaluation method, see Section 6.8 for more details on this, and we compare

this with the estimated total tardiness of the previous solution. If the score of the new solution is lower, we always accept it as our new solution. If the score is higher, we will accept the new solution with a certain chance. This chance of accepting the new, worse solution is described in Equation 1. In this equation, e is Euler's constant, $eval()$ is the evaluation function we are using, x is the previous solution y is the new solution and T is the temperature parameter.

$$p = e^{\frac{eval(x)-eval(y)}{T}} \tag{1}$$

Simulated annealing has three parameters. The first one is the starting temperature T_{start}. The second one is Q which represents the frequency with which we reduce the temperature. The third one is α, the factor with which we reduce the temperature we have the restriction that $0 \leq \alpha \leq 1$. The cooling schedule works like this. The temperature T starts at the starting temperature T_{start} and after we complete Q local search iterations we multiply T by α. In Section 6.11 we give some insight about how the parameters of this approach are selected.

6.8 Evaluation

After we apply one of the local search operators we get a new schedule and we want to be able to evaluate this schedule in some manner to compare it to the original schedule. We need this evaluation to decide which schedule is 'better' according to some function. We distinguish between three evaluation methods. These will be described in the sections below.

6.8.1 Percentile Evaluation

A simple approach is just taking a fixed percentile of the processing time distribution for each task and then computing the starting- and completion times for each task using the schedule.

Given a solution and problem definition, we get a partial ordering of the tasks which we can use to generate an ordering in which we can compute the starting- and completion time of each task. The starting time of a task is the maximum of the completion time of all its predecessors (implied and direct). The completion time of a task is its starting time plus its processing time, which will just be a fixed percentile of its processing time distribution. We can then compute the total tardiness, by computing the tardiness for each task. This approach just ignores all the stochasticity of the problem. Instead of using the expected processing time for each task we can also use a certain fixed percentile of the processing time. If we choose a high percentile we will be pessimistic with respect to the processing times, but this might give a buffering effect for the schedule which might turn out to work better.

Many distributions have closed form equations to compute these percentile values and in case this equation does not exist one can always find good approximations for these values.

6.8.2 Simulation Evaluation

The simulation approach is very similar to the percentile-based approach. But this time we sample the processing times of each task according to its distribution. With this sample of the processing time we can evaluate the solution like we explained in Section 6.8.1. This sampling can easily be done with using inverse transform sampling. If we do this sampling and evaluating multiple times, we can get a good approximation of the expected total tardiness by averaging the results. We use a set of samples that are re-used. The idea behind this is the following: by not changing the samples every iteration we give the local search some time to optimise for a fixed set of samples. If we would change the samples every iteration it might be the case that the local search would not properly

converge. This will be tested and you can read about the results of these tests in Section 6.11. This evaluation method can be controlled with two parameters. The first is the simulation count. This represents the number of simulations that we will do to get an estimate of the expected total tardiness. The next parameter is the replace percentage. This is the percentage of processing time samples that are replaced each time we replace part of the samples. After $|V|^2$ local search steps we replace a certain percentage of the processing time samples. Here $|V|$ is the number of real tasks. This set of samples is reused for each evaluation we do. When we sample, we sample all the processing times of all the tasks and use this set of p_j's as one sample. The p_j's of different samples are not mixed. If we use a certain sample, we use a fixed set of p_j's.

6.8.3 Normal Approximation Evaluation

The starting time of each task depends on the completion time of its machine predecessor and the completion time of its direct precedence predecessors. The starting time of a task is in fact the maximum of the completion times of all these tasks. This is because non-forced idle time is not allowed in the schedule. We will assume the completion times of all predecessors are normally distributed and compute the maximum over all these completion times and assume that that is normally distributed again. Using equations 2, 3 and 4, we can approximate $X = \max(N_1, N_2)$, where N_1 is a normal distribution with parameters (μ_1, σ_1^2) and N_2 is a normal distribution with parameters (μ_2, σ_2^2). We then assume that X is normally distributed again. ρ is the correlation coefficient of N_1 and N_2 which we will assume to be 0 during all our experiments. In the equations $\Phi(x)$ denotes the cumulative distribution function of the standard normal distribution and $\phi(x)$ denotes the probability distribution function of the standard normal distribution. These formulas are enough to approximate the new distribution because we know that $\sigma^2(X) = E(X^2) - E(X)^2$.

We use a specific ordering of these tasks while computing the maximum. First, we take the immediate machine predecessor on the same machine into account. Then we take the remaining direct precedence constraints into account. We ignore the direct precedence constraints which come from tasks that are scheduled on the same machine, because their influence is already taken into account by the immediate machine predecessor. When we have a distribution for the starting time of a task we get the distribution of the completion time of the task by adding the means of the starting time and processing time together and the variance of the starting time and processing time together. This approach gives us a normal distribution for the starting- and completion times for each task and we can use the expected values of these distributions to approximate the expected total tardiness of the schedule. For more details about this approach we refer to [25], [26] and [24].

$$\theta = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho\sigma_1\sigma_2} \tag{2}$$

$$E(X) = \mu_1\Phi(\frac{\mu_1 - \mu_2}{\theta}) + \mu_2\Phi(\frac{\mu_2 - \mu_1}{\theta}) + \theta\phi(\frac{\mu_1 - \mu_2}{\theta}) \tag{3}$$

$$E(X^2) = (\sigma_1^2 + \mu_1^2)\Phi(\frac{\mu_1 - \mu_2}{\theta}) + (\sigma_2^2 + \mu_2^2)\Phi(\frac{\mu_2 - \mu_1}{\theta}) + (\mu_1 + \mu_2)\theta\phi(\frac{\mu_1 - \mu_2}{\theta}) \tag{4}$$

6.9 Generating Instances

We generate the instances in the following way. First, we select the number of real tasks, $|V|$, machines m, precedence constraints $|A|$, and the scaling factor $\sigma^2(D)$ between the processing time and the variance. We select the minimal and maximal mean for the normal distributions. We also compute the average of these numbers and call it p. We then set the minimal due date to 0 and the

maximal due date to $|V| * p/m$. Now that we have all the ingredients we need for generating the instances we can start. For each task we sample the mean and due date uniformly between their minimum and maximum. For each of the precedence constraints we select 2 different tasks (T_1, T_2) at random. We then need to check that this new constraint does not conflict with the current set of constraints. A new precedence constraint (T_1, T_2) is in conflict with the current set of precedence constraints $G(V, A)$ if there exists a path in G from T_2 to T_1. This would create a cycle in the precedence constraints, and this makes the resulting problem unsolvable. If we select such a pair of tasks, we just select a new pair of tasks and try again. We repeat this until we have generated the number of constraints that was asked for.

In Table 3 and Table 4 you can see some settings that were used to generate some of the instances that were used for the machine scheduling experiments. The processing times for these instances are normally distributed.

6.10 Experimental Setup

There are two goals with these experiments. The first goal is optimising the parameters for the evaluation methods, the second goal is to compare the three different evaluation methods. We will first need to find the proper parameter settings for each evaluation method before we can compare them. For a given problem instance, evaluation method and set of parameters we run the algorithms as described in Section 6.3. When we terminate the search, we perform a million simulations to get a good approximation of the expected total tardiness of the final solution. For each experiment we measured this final score and the time used in the local search. For each combination of evaluation method and parameter we repeated the experiment ten times. All experiments were run on a 64-bits Windows system with an i7-6500U CPU and 8GB of DDR4 RAM.

6.11 Parameter Tuning

In this section we will describe the setup of the parameter tuning experiments. We will describe what settings were used and how the experiments were run. We first did some experiments to determine the effect of the simulation count and replace percentage on the simulation evaluation method. More details about this can be found in Section 6.11.1. After that we did some experiments to determine a good starting temperature for the simulated annealing, and we looked at different percentiles for the percentile evaluation method. In Section 6.11.2 we go into more details about this. We also repeated some of the experiments from Section 6.11.1 too see if the simulation parameters and the starting temperature influenced each other. These sections are split up like this because they use different sets of instances for their experiments. We increased the number of instances for the parameter experiments in Section 6.11.2 too get more representative results.

For all parameter tuning experiments we used an α of 0.975, and every $8 * |N|$ iterations we multiplied the temperature with α. Here $|N|$ is the average of the size of all neighbourhoods. The size of the neighbourhoods is roughly the number of tasks squared. For the selection of the neighbourhoods, as described in Section 6.6, we used a A_E and A_P of 0.1 and a m_P of 0.2. We did not conduct proper experiments to determine what values work good for these parameters. The values were determined based on intuition and some small amount of tuning by hand.

The stopping criterion we used is the same as we described in Section 6.3 but we add one more additional constraint. We also stop after we have made 1 000 000 improvements. This is because these tests are mostly for the tuning of the parameters so we can afford to maybe sometimes stop a little earlier.

Instance	Task Count	Machine Count	Precedence Count	$\sigma^2(D)$
P1-0	20	3	0	0.5
P1-1	20	4	10	0.25
P1-2	20	3	10	0.1
P1-3	20	1	5	0.1
P1-4	20	2	5	0.33

Table 3: Parameter Tuning Instances (set 1)

Instance	Task Count	Machine Count	Precedence Count	$\sigma^2(D)$
P2-1	50	2	25	0.75
P2-2	50	3	25	0.5
P2-3	50	2	50	0.5
P2-4	50	3	50	0.75
P2-5	50	4	25	0.3

Table 4: Parameter Tuning Instances (set 2)

6.11.1 Parameter Tuning Run 1

We tried 5 different values for the replace percentage parameter: $\{0\%, 25\%, 50\%, 75\%, 100\%\}$. We tried 9 different values for the simulation count parameter: $\{50, 100, 150, 200, 250, 500, 1000, 2000, 4000\}$. The results of these experiments can be found in Section 6.12.1. For this experiment to determine good values for the simulation evaluation method we used 5 instances. All these instances were generated with the method described in Section 6.9. The settings that were used to generate these instances can be found in Table 3. During these experiments we used a starting temperature of 1. This seemed like a low starting temperature but because of the small number of tasks in these first experiments, it seemed to work okay. The small number of tasks makes it so that the total search space is not extremely large which is why it is more likely that we will converge earlier and that is why a quite low starting temperature works well we think.

6.11.2 Parameter Tuning Run 2

For the starting temperature we tested the following values: $\{1, 2, 3, 4, 5, 6, 7, 25, 50\}$. We then also tested multiple percentiles for the percentile evaluation method. The values we tested were $\{40, 50, 55, 60, 65, 70\}$. We also tested the influence of the starting temperature on the parameters for the simulation evaluation method. For these parameter tuning experiments, we used a different set of instances compared to the first parameter tuning experiments described in Section 6.11.1. We generated 3 instances for each of the categories described in Table 4. These instances were used to do the parameter tuning runs. The results of these experiments can be found in Section 6.12.2.

6.12 Results Parameter Tuning

We will now discuss the results of the tests that were done for the parameter tuning. The parameter tuning was done in multiple steps that is why there are multiple sections coming up about parameter tuning.

		Simulation Count								
		50	100	150	200	250	500	1000	2000	4000
Instance	P1-0	3.64	3.51	3.41	3.26	3.30	3.21	3.12	3.02	3.02
	P1-1	22.27	22.22	22.17	22.16	22.16	22.14	22.12	22.08	22.07
	P1-2	76.65	76.32	75.78	76.54	75.58	75.36	75.34	75.11	75.45
	P1-3	221.26	221.81	219.42	217.62	219.85	216.66	217.44	219.05	216.44
	P1-4	3.75	3.57	3.58	3.52	3.52	3.48	3.43	3.43	3.41

Table 5: Average score per instance and simulation count

6.12.1 Parameter Tuning Results Run 1

The score metrics in all the results are the average total tardiness of the final solutions and the time metrics are in milliseconds. All combinations for the simulation counts and replace percentages were tested and the tables where only the simulation counts are shown take the average over all replace percentage runs with the specified simulation count. The same goes for the tables about the replace percentages. The tables that will be discussed in the upcoming Sections are 5, 6, 7, 8, 9, 32, 33, 34 and 35.

6.12.1.1 Simulation Count

From Table 5 we can see that as we increase the simulation count the final score also slowly decreases. In the simulation counts tested so far, we cannot yet see the score approaching any plateau, so further experiments are required to find the point at which the score decrease starts to taper off. But we can already see the diminishing results from doubling the simulation count in the results.

From Table 6 we can also deduce that the time needed roughly doubles as the simulation count doubled which means that the simulation count quickly becomes the dominant factor in determining the total running time.

6.12.1.2 Replace Percentage

We see from Table 7 that the replace percentage of 0% seems to work best for all instances, which is quite remarkable. We expect that the replace percentage is not important if the simulation count becomes sufficiently large. At some point we have enough simulations to be sure that the set is quite representative and will give a good approximation of the expected total tardiness.

We can see in Table 8 that a higher replace percentage increases the time used. We expect that this is because replacing more samples will delay the convergence of the schedule. We expect that the time required for replacing more samples will not have a significant impact.

6.12.1.3 Simulation Count and Replace Percentage

We also tested the interaction between the simulation count and the replace percentage as you can see in Table 9. Table 9 only shows the results for instance P1-0. The tables for the other instances, Table 32, 33, 34 and 35 can be found in Section 10. We can see that there are no clear interactions between the parameters, we still see the same patterns that we saw for each parameter individually, and we see no other apparent patterns.

		Simulation Count								
		50	100	150	200	250	500	1000	2000	4000
Instance	P1-0	361	656	964	1284	1553	2675	5479	10986	22128
	P1-1	298	577	891	1129	1627	3592	5599	12298	24813
	P1-2	322	592	828	1134	1347	2458	5032	8934	18751
	P1-3	318	560	821	1168	1579	3417	6491	12358	22450
	P1-4	308	593	918	1217	1653	3535	7169	14289	28006

Table 6: Average time (ms) per instance and simulation count

		Replace Percentage				
		100	75	50	25	0
Instance	P1-0	3.31	3.32	3.36	3.40	3.01
	P1-1	22.19	22.20	22.16	22.18	22.04
	P1-2	76.21	75.99	75.48	75.74	75.22
	P1-3	219.78	219.67	220.14	218.75	216.35
	P1-4	3.59	3.57	3.52	3.52	3.41

Table 7: Average score per instance and replace percentage

		Replace Percentage				
		100	75	50	25	0
Instance	P1-0	5261	4690	5163	4778	3927
	P1-1	7318	6497	3726	3558	2569
	P1-2	4645	4999	4485	4361	3139
	P1-3	5705	5445	5223	4951	4047
	P1-4	6869	5895	6350	5706	4850

Table 8: Average time (ms) per instance and replace percentage

		Simulation Count				
		50	100	150	200	250
Replace Percentage	100	3.67	3.43	3.36	3.3	3.36
	75	3.67	3.64	3.31	3.53	3.31
	50	3.8	3.86	3.55	3.18	3.49
	25	4	3.61	3.75	3.35	3.28
	0	3.07	3.03	3.07	2.96	3.05

Table 9: Average score for instance P1-0 for different Simulation Count and replace percentage settings

6.12.2 Parameter Tuning Results Run 2

Here we show and discuss the results of the second set of parameter tuning experiments that were performed. More details about the setup can be found in Section 6.11.2. The tables that will be discussed in the upcoming Sections are 10, 11, 12, 13, 10 and 15.

6.12.2.1 Starting Temperature

For the starting temperature we tested the following values: $\{1, 2, 3, 4, 5, 6, 7, 25, 50\}$ with the normal approximation method. This was done to get an idea of the range of starting temperatures that would work best. The results of these tests can be found in Table 10. Here we see that a starting temperature of in the range 1 till 5 seems to work best for the instances shown here.

6.12.2.2 Simulation Evaluation

Using the knowledge from Section 6.12.2.1, we performed some more tests of the simulation evaluation method. Because we were unsure if the starting temperature would impact the other parameters of the simulation approach we tested multiple parameter settings again. For the simulation count we tested all values in $\{50, 100, 150, 200, 250, 500, 1000\}$, for the replace percentage we used $\{0\%, 25\%, 50\%, 75\%, 100\%\}$ and lastly for the starting temperature we used $\{1, 3, 5, 7\}$.

The results of these tests can be found in the Tables 11, 12 and 13. Each table shows the results spread over 1 of the parameters and is the average over the other parameters. As seen before in Section 6.12.1 a higher simulation count is better for the end result and it also lowers the standard deviation of the final score. This is all as expected. We also observed the same time scaling effect as we saw in Section 6.12.1, where the time needed was roughly doubled as we doubled the simulation count. These results are not shown because they contained no new insights.

6.12.2.3 Replace Percentage

For the replace percentage we also see the same effect as before, a replace of 0% is best. This can be seen in Table 12. A replace percentage of 50% seems to perform the worst in most cases, but higher or lower percentages only perform marginally better.

6.12.2.4 Starting Temperature with Different Evaluation Methods

If we compare Table 10, 13 and 15 we can see that results for the starting temperature are slightly different for the different evaluation methods. The simulation method works better with a slightly higher starting temperature. We can also see that a higher starting temperature reduces the standard deviation for the results for the simulation approach. This effect was not present for the normal approximation. The normal approximation approach performs better under the best settings than the simulation approach for every instance. From this we can conclude that we would need more simulation per evaluation to perform on a similar level to the normal approximation method.

6.12.2.5 Percentile Evaluation

In Table 14 we can see that a slightly pessimistic approach with respect to the processing time performs best. On average the 55th percentile gave the best results. For the starting temperature, shown in Table 15, we see similar results, as we saw for the normal approximation; a starting temperature of roughly 3 seems to work best. If we compare the best results for the normal distribution as seen in Table 10, and the best results from the simulation as seen in Table 11, and the best

Start Temperature		1	2	3	4	5
Instance	P2-1	1461.38 (55.46)	1413.60 (99.92)	1401.83 (69.52)	1439.25 (115.43)	1487.04 (125.99)
	P2-2	2916.06 (155.85)	2926.74 (163.51)	2884.96 (183.24)	2785.04 (173.91)	2840.47 (128.09)
	P2-3	1937.49 (37.84)	1927.40 (33.01)	1909.44 (33.49)	1949.78 (31.91)	1909.44 (26.71)
	P2-4	2971.48 (57.97)	2983.49 (93.11)	2974.93 (65.04)	2952.29 (54.56)	2952.3 (77.36)
	P2-5	1022.42 (18.15)	1016.76 (36.00)	1031.16 (29.11)	1052.31 (33.04)	1046.27 (43.72)
Start Temperature		6	7	25	50	
Instance	P2-1	1444.51 (145.56)	1431.23 (66.70)	1791.76 (138.82)	2209.22 (178.87)	
	P2-2	2716.24 (54.47)	2835.75 (159.79)	3111.00 (127.28)	3512.05 (209.22)	
	P2-3	1951.03 (29.02)	1969.28 (41.28)	2312.02 (76.22)	2758.28 (203.05)	
	P2-4	3035.01 (155.37)	2983.63 (81.28)	3256.15 (63.35)	3738.74 (140.09)	
	P2-5	1055.13 (31.48)	1084.15 (37.00)	1349.70 (72.96)	1803.08 (110.68)	

Table 10: Score: average (standard deviation) per instance and starting temperature. The evaluation method is normal approximation

		Simulation Count				
		50	100	150	200	250
Instances	P2-1	1591.29 (173.46)	1530.17 (159.86)	1495.59 (116.63)	1455.31 (105.93)	1456.16 (103.61)
	P2-2	3010.68 (226.28)	2914.47 (218.93)	2886.61 (196.74)	2842.73 (179.60)	2819.53 (163.77)
	P2-3	2043.60 (115.54)	1990.39 (88.36)	1953.33 (66.60)	1943.96 (65.35)	1940.34 (60.94)
	P2-4	3181.72 (202.45)	3087.86 (146.45)	3051.08 (137.63)	3046.87 (116.17)	3032.58 (122.98)
	P2-5	1096.75 (93.66)	1053.82 (51.14)	1044.11 (49.47)	1039.52 (45.78)	1038.35 (39.39)

Table 11: Score: average (standard deviation) per instance and simulation count. The evaluation method is simulation

results for the percentile evaluation as seen in Table 14 we see that is all cases except for instance $P2-2$, the normal approximation approach performs the best. What is remarkable about this is that the best results for the percentile evaluation are achieved when we use the 40th percentile of the processing time. We can also see that in all cases except for instance $P2-4$, the simulation performs the worst. For instance $P2-4$, the percentile evaluation approach performs the worst.

		Replace Percentage				
		0	25	50	75	100
Instances	P2-1	1373.78 (63.36)	1523.97 (118.86)	1532.69 (129.04)	1531.64 (144.54)	1566.43 (160.13)
	P2-2	2681.24 (92.48)	2906.40 (188.98)	2961.99 (183.33)	2966.06 (207.89)	2958.32 (196.13)
	P2-3	1880.03 (38.49)	1983.71 (80.16)	2015.03 (91.46)	2003.40 (88.30)	1996.00 (78.01)
	P2-4	2927.48 (63.19)	3102.89 (141.17)	3127.32 (160.99)	3120.08 (145.19)	3122.35 (155.37)
	P2-5	1011.14 (37.27)	1056.31 (58.04)	1066.81 (55.35)	1068.97 (67.98)	1069.31 (70.39)

Table 12: Score: average (standard deviation) per instance and replace percentage. The evaluation method is simulation

Average Score		Start Temperature		
		1	3	5
Instances	P2-1	1527.82 (161.52)	1504.78 (132.59)	1484.51 (133.43)
	P2-2	2929.23 (214.22)	2892.43 (206.27)	2862.75 (201.43)
	P2-3	1983.41 (100.18)	1973.54 (96.12)	1970.21 (76.38)
	P2-4	3102.69 (172.06)	3075.70 (152.59)	3061.68 (144.39)
	P2-5	1058.17 (77.15)	1046.13 (59.27)	1059.22 (48.19)

Table 13: Score: average (standard deviation) per instance and starting temperature. The evaluation method is simulation

		Percentile					
		40	50	55	60	65	70
Instance	P2-1	1504.02 (80.54)	1448.78 (103.14)	1440.90 (78.80)	1468.54 (66.61)	1502.85 (71.70)	1531.12 (92.10)
	P2-2	2765.50 (108.16)	2825.76 (135.14)	2784.88 (131.32)	2828.95 (134.63)	2887.29 (157.52)	2889.5 (123.19)
	P2-3	1968.14 (47.55)	1934.98 (40.95)	1930.51 (54.73)	1928.74 (41.42)	1955.15 (48.32)	1974.94 (39.73)
	P2-4	3098.77 (73.57)	3057.42 (89.13)	3070.46 (100.86)	3075.92 (125.41)	3061.24 (81.62)	3093.64 (77.09)
	P2-5	1045.08 (35.03)	1051.12 (45.56)	1032.04 (35.80)	1038.83 (61.71)	1065.04 (51.61)	1083.20 (91.19)

Table 14: Score: average (standard deviation) per instance and processing time percentile. The evaluation method is percentile

		starting temperature		
		1	3	5
Instance	P2-1	1486.83 (92.01)	1475.36 (78.59)	1485.92 (93.95)
	P2-2	2851.18 (131.09)	2822.58 (151.35)	2817.18 (132.77)
	P2-3	1941.77 (49.47)	1944.76 (45.58)	1959.36 (49.83)
	P2-4	3085.74 (100.79)	3078.14 (95.56)	3065.06 (81.43)
	P2-5	1038.10 (73.72)	1050.12 (46.69)	1069.44 (47.91)

Table 15: Score: average (standard deviation) per instance and starting temperature. The evaluation method is percentile

6.13 Final comparison

After the parameter tuning experiments, we used our approach in combination with each of the evaluation methods with the best settings found in the previous experiments to compare the different evaluation methods. For the percentile approach we used the 55th percentile and for the simulation we used a simulation count of 500 and 1000. The results of this comparison can be found in Table 16.

We can see that the normal approximation and percentile-based approach perform very similarly with respect to score and time. We can also see that both simulation-based approaches perform very similar with respect to score, but as expected the one with the lower simulation count does much better with respect to time. This means that we are nearing a point where an increase in the simulation count does not seem to give much score improvement. Both the simulation-based approaches are considerably slower than the normal approximation or the percentile approaches but on the other hand the simulation-based approaches score better on 2/3 of the instances. Overall, we can see that the normal approximation and percentile-based approaches perform quite similar and are quite quick compared to the simulation-based approach, but the simulation-based approach performs a bit better with respect to the final score.

Because of a lack of time we only tried instances where the processing times were normally distributed. This might give the normal approximation-based approach an advantage because this makes it a better approximation. This is something to take into account when looking at the results.

We expected the normal approximation-based approach to perform a bit better given the results we saw by Passage et al. [26] but this was not the case. An important difference might be the objective function. We are looking at expected total tardiness and not expected makespan. We expect that the normal approximation approach works well for the expected makespan because the makespan is defined by path of tasks which makes it the sum of many processing time distributions and because of the central limit theorem this becomes close to normally distributed which is our assumption at each step in the normal approximation approach. In case of total tardiness, we have to determine the tardiness of all the tasks and not only the tasks at the end of the schedule which might make this approach less suitable for our setting.

I	Normal		Percentile		Simulation-500		Simulation-1000	
	Score avg(dev)	Time avg	Score avg(dev)	Time avg	Score avg(dev)	Time avg	Score avg(dev)	Time avg
P2-1	1419.09 (79.39)	5686.92	1458.47 (107.35)	8905.62	1401.67 (66.38)	62401.77	1396.41 (60.83)	121704.1
P2-2	2192.01 (116.04)	6088.64	2152.36 (111.90)	16554.57	2054.88 (90.40)	54245.54	2057.88 (91.24)	109338.3
P2-3	885.77 (60.09)	5003.84	882.08 (57.21)	10178.42	839.71 (44.76)	52127.16	850.39 (44.50)	97919.38
P2-4	2868.68 (145.65)	5285.62	2814.64 (139.77)	4050	2690.33 (108.49)	51800.06	2687.17 (97.57)	124235.1
P2-5	1996.67 (146.96)	6275.62	1939.97 (150.90)	9925.95	1836.89 (103.25)	48922.44	1827.54 (100.19)	98593.92
P2-6	4059.84 (211.21)	9820.22	3872.61 (214.44)	15920.88	3760.97 (210.20)	56793.76	3750.26 (221.77)	96232.22
P2-7	1918.86 (41.98)	7790.96	1908.49 (46.04)	3288.48	1908.84 (38.58)	58564.86	1909.71 (36.79)	107433.3
P2-8	1951.04 (89.55)	10340.56	1985.37 (122.13)	6431.2	1903.20 (48.20)	66440.34	1913.67 (54.95)	125749.8
P2-9	1175.49 (42.85)	5342.68	1178.96 (56.93)	4225.82	1185.95 (57.45)	53543.26	1174.56 (49.03)	117897.9
P2-10	2955.35 (58.98)	12584.68	3074.87 (103.15)	23835.65	2980.28 (86.14)	51801.48	2970.70 (81.16)	99899.77
P2-11	2589.56 (97.75)	20464.14	2629.59 (87.96)	28939.6	2461.71 (63.83)	69097.55	2484.65 (97.95)	135860.8
P2-12	2974.41 (113.80)	10232.98	3019.92 (138.60)	9885.92	2824.91 (103.82)	60866.38	2808.18 (116.74)	133483.2
P2-13	1023.73 (21.86)	4966.14	1021.24 (28.26)	8218.08	1041.11 (56.75)	57945.71	1034.03 (43.30)	123520.7
P2-14	869.27 (30.60)	5921.4	891.56 (38.73)	4425.4	904.13 (31.33)	68560.7	908.36 (38.65)	140048.1
P2-15	1268.61 (78.03)	3758.94	1281.96 (69.79)	5189.3	1275.49 (85.73)	56815.02	1266.95 (80.82)	107999.9

Table 16: The average score, standard deviation of the score and time required per instance and evaluation method.

7 Resource Constrained Project Scheduling

We will now look at the full problem, the Resource Constrained Project Scheduling Problem (RCPSP).

7.1 Problem Definition

We already gave the formal definition of the RCPSP and the SRCPSP in Section 2, but in this section we will talk about how we can relate these problems to the (stochastic) machine scheduling problem.

The RCPSP includes everything from the machine scheduling problem we looked at in Section 6. As the name implies, we also have resource constraints. These are a generalisation of the machine constraints we had. In other words, the resources we now have can represent machines, like in the machine scheduling problem, but we can use them for more. We can now have multiple different types of resources. A task can require multiple units of a resource for its execution, a task can also require different types of resources at the same time. Just like in the machine scheduling problem we have a fixed capacity during the project execution. The resource requirements for all the tasks are given beforehand and are deterministic. Just like in the machine scheduling case, the processing times of the tasks are the only stochastic part of the problem. For the problem to be feasible the resource requirement for each resource, of each task must be less than or equal to the resource capacity for that resource. Writing this down all more formally gives us the following addition to the problem definition. There are R types of resources and each task j has a value r_{ij} which represents how many units task j requires of resource i during its execution. All resources are renewable and will be released by a task once it completes. A task requires all its resource requirements to be met during the whole duration of its execution. At each moment in time there are R_i units of resource i available.

7.2 Schedule Representation and Execution

It is not immediately clear what the best, or most appropriate, way is to represent the schedule for the RCPSP. We will look at 2 options for representing the schedule. It is not only a matter of 'best' representation however. It might be that one of the representations fits a certain application better than the other. We will look at 2 approaches. In the first approach we will schedule the tasks based on their resource requirements and in the second we approach we will ignore the resources during the planning and solve the resource problem later.

7.2.1 Fixed Flow Representation

The usage of the machines was clear in the machine scheduling variant of the problem because of our choice of schedule representation. We could just schedule a sequence of tasks on each machine and these tasks would remain bound to the machine they were scheduled upon. In our first approach we do the same for each separate unit of each resource.

We model all the resource flow explicitly and define a partial order schedule in terms of each resource unit instead of each task. From every resource unit we can deduce a series of precedence constraints which give us a scheduling of the tasks. This leads to a clear and well-defined schedule, but we might lose some flexibility during the execution of the schedule, because every part of it is fixed. The advantage of this is that we can for example, guarantee that resources are reserved for a certain important task, and that it will not be preceded by some much less important task chosen on the fly; we have more control over the schedule.

This representation might be more suited for applications in which the type of resources we are modelling are less flexible in nature. For example, an engineer might be less flexible than a litre of water.

Similar to the machine scheduling representation we also add a dummy task for each resource we have. These dummy tasks have as resource requirement the capacity of the resource for which it is the dummy task, and zero for all other resources. We use this dummy task for our local search operators. This way we can easily define all possible shifts. These dummy tasks have a processing time of 0, so they don't influence the schedule.

7.2.2 Partial Order Schedule Representation

Another option is that we will not model the flow of resource units explicitly, but we will only create a POS of the tasks. We now model the schedule as a directed acyclic graph(DAG) of tasks where an arc from task j to task i means that we need to complete task j before we can start task i.

In this graph there will be arcs representing the precedence relations that are given as part of the problem definition and these can't be removed. More arcs can be added to the graph as long as they do not create any cycles. These arcs can be used to force that a task will be completed after its predecessor. These arcs that are added can also be removed again, but the original precedence arcs can't be removed.

Using this approach, it is not immediately clear how the resources will flow between the tasks, but that is what might make this method more flexible, the flow of resources will dynamically be determined. There are multiple ways to do this. One approach is placing each task for which each of its predecessors in the POS are completed in a queue. This queue is just a fifo-queue, and we start the first task in the queue when enough resources are available to start the task. We can also use a priority-queue instead of a normal queue. Here we can, for example, add the tasks to the queue based on their due date.

With the addition of the queuing process and a sample of the processing times we are back to the deterministic case where we can easily compute the total tardiness. Because we assign no fixed start times to a task and use this queue to determine the order of execution, we will always have a resource feasible solution. Another note we need to make with this approach is that is not deterministic. It can happen that 2 tasks are added to the priority queue with the same priority and then we need to make an arbitrary decision on which one we will place in front of the other. Or when 2 tasks have the same completion time and we are using a fifo-queue, we get the same problem.

For this representation to work well we need a good priority rule. We propose the recursive priority rule that can be seen in Equation 5. In this equation the set $Suc(Task_i)$ contains all the tasks in our POS that have an incoming arc from the task $Task_i$. Because our POS is always a DAG, there are no cycles so we can compute this priority value for each task. We can compute this priority value in a bottom-up manner where we start with the tasks for which the set $Suc(Task_i)$ is empty.

The idea behind this priority rule is that $d_j - E(P_j)$ estimates how late we can start and still expect to finish before the due date. We also want to look at the tasks that come after a certain task because if their due date is low or if there are many tasks after the current task we can get quite a big tardiness penalty.

$$Prio(Task_j) = -(d_j - E(P_j)) - \sum_{Task_i \in Suc(Task_j)} Prio(Task_i) \tag{5}$$

7.3 Evaluation

The evaluation methods do not change drastically compared to the ones described for the machine scheduling in Section 6.8. We will now describe the changes that we need to make to the evaluation methods for each type of solution representation in order to make it work.

7.3.1 Percentile Evaluation

Given the representations explained in the previous section the percentile evaluation method does not really change. We still replace the stochastic processing times with the deterministic processing times that are set to a certain percentile of the processing time distribution. The rest of the evaluation is straightforward.

7.3.2 Simulation Evaluation

The case is similar for the simulation evaluation method. We can again just sample the processing times and then solve the deterministic case without any problems.

7.3.3 Normal Approximation Evaluation

Lastly the normal approximation method. How this method works is dependent on our solution representation because this approach uses all the predecessors of a task. The precedence predecessors are not the problem, but the resource predecessors can be a problem.

7.3.3.1 Normal Approximation Evaluation - Flow

If we model all the flows explicitly this approach remains unchanged in essence, because we model all the flows explicitly we know what the resource predecessors are for each task. We can just use the same approach as we described in Section 6.8.3.

7.3.3.2 Normal Approximation Evaluation - POS

For the POS representation there are multiple approaches to solve this problem of not knowing what the resource predecessors are. We could compute all the possible combinations of resource predecessors, but the number of combinations would be exponential, which makes this approach unfeasible. It is important for the local search to be effective, to have a relatively quick evaluation method, and this approach would not achieve that.

The approach that we propose works as follows. We start by computing the priority value for each task like we described in Section 7.2.2. We start with all the tasks without any predecessors in our priority queue. We start by scheduling the first task from our priority queue and approximate its completion time like we described in Section 6.8.3. We continue the same way by scheduling according to our priority queue and approximating the completion time in the same manner. This is not always enough to determine what the set of resource predecessors for the next task will be, and we need to know this to approximate the completion time. If there are multiple resource predecessors available we determine for each resource the set of predecessors that combined provide enough resource capacity and have the earliest completion time. So we start by selecting the task with the earliest completion time that has no resource successor yet and set it as our resource predecessor. We continue with this process until all our resources are covered.

This approach takes a lot less time, but it is also a worse approximation. It is important to realise that the goal of the evaluation is not to approximate the total tardiness as accurately as possible,

	A(0)	B(1)	C(2)	D(3)	E(4)	F(5)
r_{1j}	1	1	1	1	2	0
r_{2j}	1	1	1	2	0	2
$E(P_j)$	2	1	2	1	4	1
d_j	5	3	3	11	8	4
$Prio(j)$	8	2	3	-7	-4	-3

Table 17: The task definitions for the example partial order schedule

but to be able to decide between two schedules, which one has the lowest total tardiness. As we said before in Section 7.2.2, when we have multiple tasks with the same priority, we choose one arbitrarily as the resource predecessor. This approach only looks at a likely predecessor for each resource unit and only takes them into account. When we take the maximum over multiple predecessors, we start with the resource predecessors and sort them in descending order based on their expected completion time. This means we take the resource predecessor with the latest completion time into account first. Next we do the same thing for the predecessors from the POS that we have not yet taken into account and lastly we look at the precedence predecessors that we have not yet taken into account and incorporate them into the maximum as well.

7.3.3.3 Normal Approximation Evaluation POS Example

To better illustrate the workings of the evaluation method described in Section 7.3.3.2, we will apply the evaluation method to an example. The POS for the example can be found in Figure 4 and the relevant parts of the task definitions for this example can be found in Table 17. In this example we have two resource types where $R_1 = 2$ and $R_2 = 2$.

We will start by calculating the Priority values for all the tasks given the current POS. We do this by starting with the task without any successors, task E and F. For these two tasks their priority equals $-(d_j - E(P_j))$ because they have no successors. Next we can compute the priority for task D, B and C. For task D we get a priority of $-(d_3 - E(P_3)) - (-(d_5 - E(P_5)))$, for task B we get $-(d_1 - E(P_1)) - (-(d_4 - E(P_4)))$ and similarly for task C we get $-(d_2 - E(P_2)) - (-(d_4 - E(P_4)))$. Lastly the priority for task A becomes $-(d_0 - E(P_0)) - (-(d_3 - E(P_3)) - (-(d_5 - E(P_5))) + (-(d_4 - E(P_4))))$. The results of all these computations can be found in Table 17.

With these priority values we can now start with the evaluation method as we described in Section 7.3.3.2. We start with the available tasks A, B and C. We start with task A because it has the highest priority. There are still resources available for task C so we will start that one as well, because it has the highest priority of the available tasks. Then after A and C finish we have resources to start on task B or task D. We start with task B because it has the highest priority between the two. Task B has task A or C as its resource predecessor, we choose one randomly. After B finished we start with E because it has a higher priority than D. Task E gets B as its resource predecessor and A or C as its resource predecessor. Task E gets the one from A and C that was not randomly chosen by B as its resource predecessor. If we assigned A to be the resource predecessor of B then C will be the resource predecessor of E and vice versa. We then continue on with task D which gets E and B as its resource predecessors and again the one from A and C that was not randomly chosen by B. After D finished we start task F which only has task D as its resource predecessor.

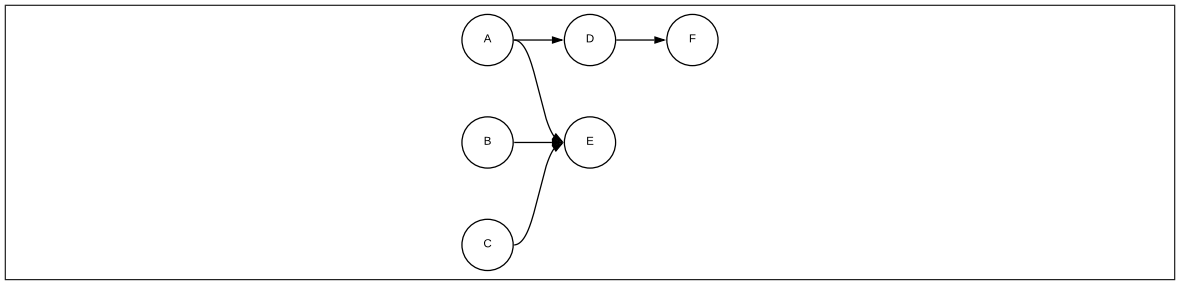

Figure 4: An example partial order schedule

7.4 Local Search

The local search structure is still the same as we used for the machine scheduling problem. Details about this can be found in Section 6.3. In the next sections we will describe the local search operators for the schedule representations and how they work. We will start with describing the local search operators for the POS representation because they are more similar in how they work to the operators we defined for the machine scheduling problem. After we explain how the operators work for both representations we will further illustrate their workings with an example in Section 7.4.3.

7.4.1 Local Search: POS Representation

We define four local search operators for this representation, shift, swap, AddOrdering and RemoveOrdering. We will now describe these 4 operators in detail below.

7.4.1.1 Shift

First the shift. The shift works very similar to the machine scheduling shift we defined in Section 6.5.1. We select the tasks in the same manner and we compute the feasibility in the same manner. After we found a feasible combination for $T1$ and $T2$ we will perform the shift. We do this by adding a new arc from $T2$ to $T1$ in our schedule. Next we add an arc from each non-precedence predecessor of $T1$ to each non-precedence successor of $T1$. Lastly, we remove all the non-precedence incoming- and outgoings arcs from $T1$ except for the incoming arc from $T2$. These steps remove all the old arcs from and to $T1$ and they connect all the *loose ends* that $T1$ leaves behind at its old position in the schedule. Our schedule also includes some dummy tasks at the start to facilitate the shifting of tasks to the front. This way we can shift a task to any point in the schedule.

7.4.1.2 Swap

The swap operator swaps the places of 2 different tasks in the schedule. The swap works the same as the machine scheduling swap we defined in Section 6.5.2. The swap is generated in the same way, and the feasibility is checked in the same way. When we find a feasible pair, we connect all the non-precedence predecessors of $T1$ to $T2$ and vice versa. We also connect $T1$ to all the non-precedence successors of $T2$ and vice versa. Then we remove all the original non precedence- predecessors and successors for both $T1$ and $T2$.

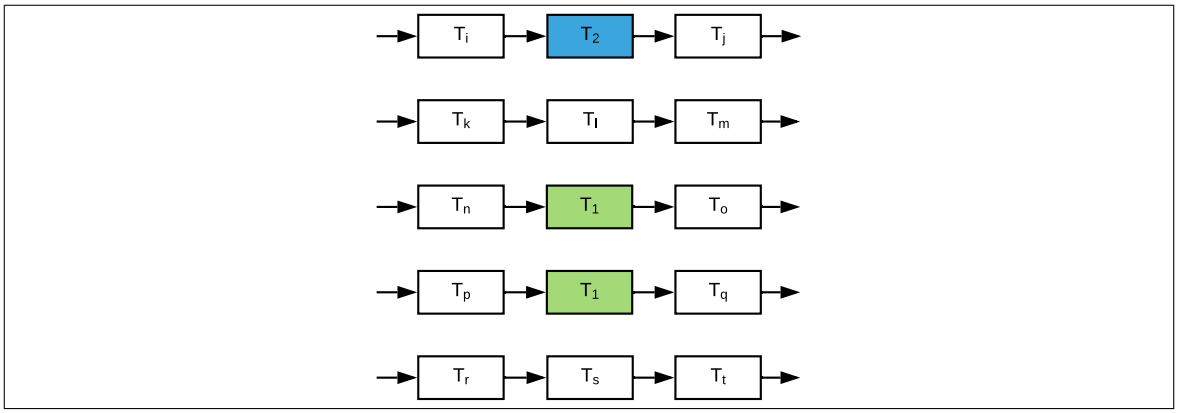

Figure 5: Part of a fixed flow schedule representation where five resource flows are shown for a single resource type. In this example task $T1$ (shown as T_1) requires two units of this resource type and task $T2$ (shown as T_2) requires one unit of this resource type.

7.4.1.3 AddOrdering and RemoveOrdering

AddOrdering adds an arc to the POS and RemoveOrdering removes an arc from the POS. For the AddOrdering to be feasible the new arc we add needs to not create a cycle in the POS. As we said before, the RemoveOrdering can be done on any arc that is not a precedence arc in the POS and we do not allow the addition of an arc that is already present.

7.4.2 Local Search: Fixed Flow Representation

For the flow representation we only have the shift and the swap like in the machine scheduling problem. These operations do become a bit more involved compared to the machine scheduling variants.

7.4.2.1 Shift

In Figure 5 you can see a setting in which task $T1$ requires two units of the resource type shown and task $T2$ requires one unit. In this figure only a portion of the tasks in each resource flow is shown and the use of five resource flows is just for illustrative purposes, any other positive number of flows would work as well.

The shift is again defined by two tasks, we will call them task $T1$ and task $T2$. We will shift $T1$ behind $T2$ for all their overlapping resources. The shift is now different from the shift definition we gave in Section 6.5.1, this is because it is very likely that the resources required by task $T1$ are not a subset of the resources required by task $T2$. Because of this we need to select extra predecessors for the resources that task $T2$ cannot cover, that are required for task $T1$. First compute the intersection between the resource requirement for both tasks, and within each resource we randomly match resource flows from task $T1$ with resource flows from task $T2$. We only allow shifts where this intersection of resources is non-empty. In other words, it does not seem logical to shift a task behind another task if they have no overlapping resource requirements.

Figure 6 shows that we matched one of the resource units of task $T1$ with the unit of task $T2$ and we moved the unit of task $T1$ behind the unit of task $T2$.

Now that we determined the position for the resource units from $T1$ that overlap with $T2$ we will have to plan the remaining resource units. For each resource flow, we again randomly match

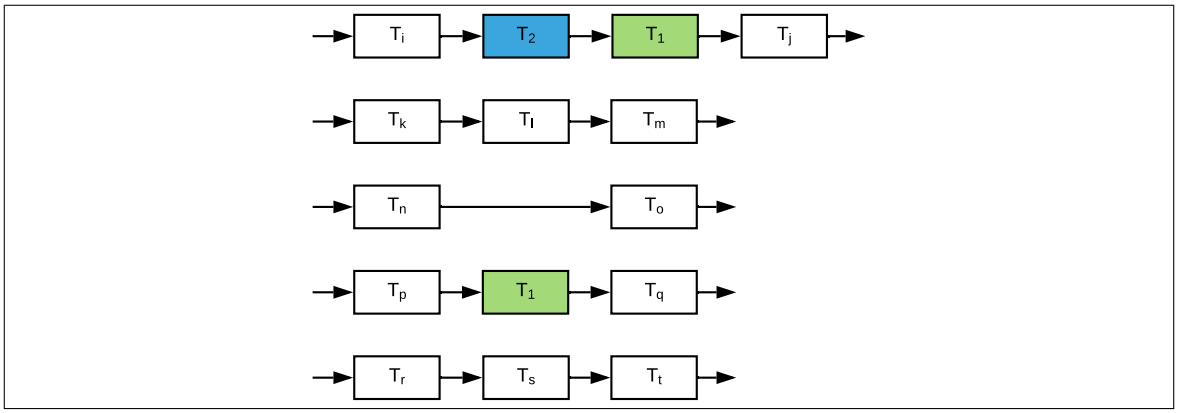

Figure 6: Part of a fixed flow schedule representation where five resource flows are shown for a single resource type. We shifted one of the resource units of task $T1$ behind the unit of task $T2$.

unmatched resource flows of task $T1$ with resource flows that we have not selected yet for this shift operation. For these matches the position of task $T1$ within this flow is not yet determined. Not all positions for task $T1$ may lead to feasible solutions. We use the flow matches between task $T1$ and task $T2$ and the positions we determined for them to compute the restrictions for the positions in the flows that we have not yet determined.

We compute these restrictions in the following manner. For each flow where the position is determined we compute all the implied predecessors and -successors. These predecessors cannot be successors in all the other flows and the successors cannot be predecessors in all the other flows. This gives us an interval for each flow in which we can select a position for task $T1$ to shift to.

We handle the flows for which the position has not yet been determined in random order to prevent bias. We select the position for each flow randomly within the feasible interval and after we select a position, we again compute the implied predecessors and -successors and update the restrictions for the other tasks.

After we have determined the position for each flow, we execute the shift by removing task $T1$ from all its original positions and adding task $T1$ to all the newly selected positions.

In Figure 7 we can see that we matched the remaining unit of task $T1$ with another resource flow, this could have been any of the flows except the uppermost flow because that one already had a unit of task $T1$ in it. Normally the position on the unit within the flow depends on the restrictions that we calculated, but in this case, we don't know the restrictions so it was placed randomly. We moved it to the bottom flow in this example but the second or third flow would also have worked and keeping it in the same flow would also have been an option.

7.4.2.2 Swap

The swap is also again defined by 2 tasks $T1$ and $T2$. Because the swap is in fact a combination of two shifts, we can use a very similar approach as described above to implement the swap. We start again with computing the intersection between the resource requirements and again we only allow swaps where this intersection is non-empty. We randomly match resource flows and for both tasks we randomly select flows for the remaining unmatched resource flows. We do this in the same manner as we described for the shift and we also compute the restrictions in the same manner. After we have determined the position for each flow for both tasks we execute the swap by removing task $T1$ and $T2$ from all their original positions and adding them back to the newly selected positions.

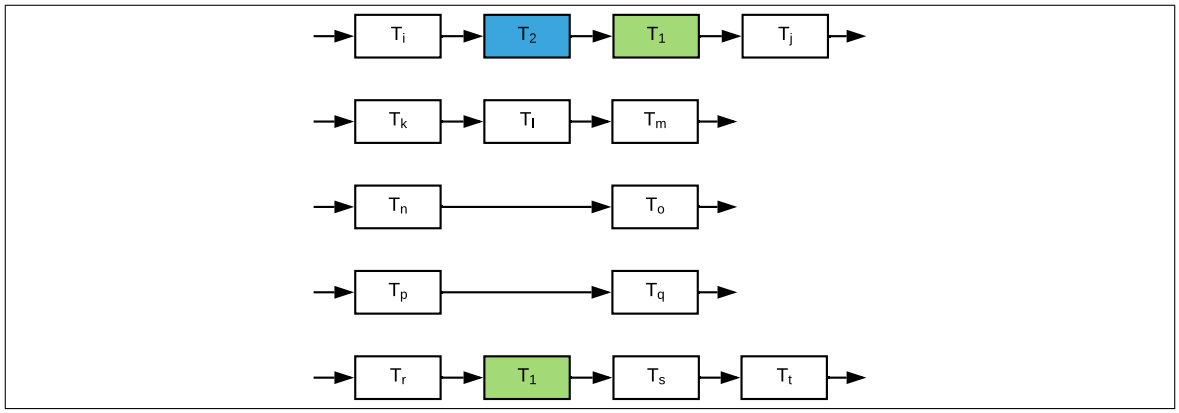

Figure 7: Part of a fixed flow schedule representation where five resource flows are shown for a single resource type. We moved the other resource unit of task $T1$ to a different resource flow.

7.4.3 Local Search: Example

We will now give a small example to further illustrate how the local search for both representations works. We will start with describing the example setting. There are 2 types of resources. Resource R_0 has a capacity of 2, and resource R_1 has a capacity of 3. There are 5 tasks: $\{A(1,1), B(2,3), C(1,2), D(1,1), E(1,1)\}$. Between the brackets behind each task you can see their resource requirement for resource R_0 and R_1 respectively. There is one precedence constraint. We require that task C is completed before we start with task E. The due dates and processing times distributions are not relevant for this example.

7.4.3.1 Local Search: Example flow

We will start out with the fixed flow representation. We will use the problem setting as described in Section 7.4.3. An example schedule can be found in Figure 8 and 9. The colours have no meaning once again, they are just there to visually differentiate the tasks. On the vertical axis you can see the numbering of the resource flows. On the horizontal axis you can see the time.

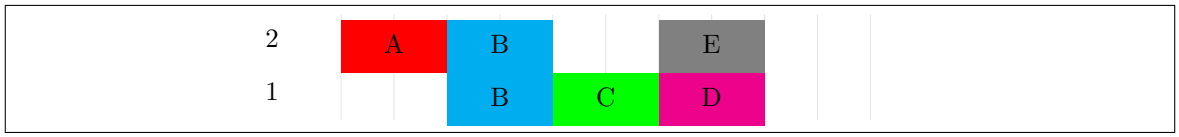

Figure 8: The R_0 resource usage of the example flow schedule

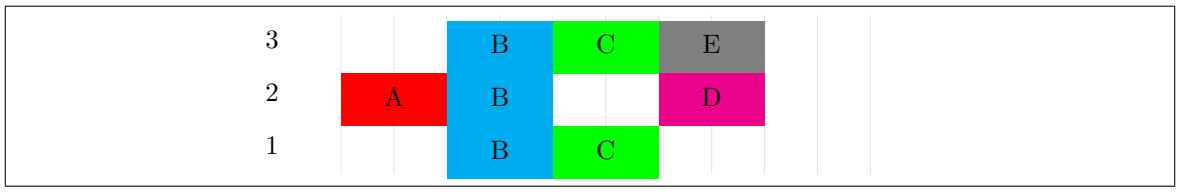

Figure 9: The R_1 resource usage of the example flow schedule

Now we will look at an example shift. We want to shift task C behind task A. For resource R_0

we see there is a 1:1 matching between the task C and task A because they both require a single unit of resource R_0. For resource R_1 task A once supplies one unit of this resource and task C requires 2 units. This means that we randomly select one of the resource flows where task C is currently scheduled in for resource R_1, flow 1 or flow 3. Let's say we select flow 3 and match it with the supplied unit of R_1 from task A. We then need to select a random flow to place the remaining unit from task C, we can choose from flow 1 and flow 3. Let's say we select flow 3. Now for each of the resource units from task C that we could match with a resource unit from task A the place in the new schedule is determined, because we place the unit directly behind the matched unit from task A. From these matches we compute the restrictions for the resource flows on which we haven't placed the remaining units yet. For this example this means that we need to place our last remaining unit of resource R_1 before task B because our matched resource units have task B as a successor, and we can't have task B be a successor of one unit, and a predecessor of another unit. This fixes the position of this unit. Now we shift all the units to the places we just determined and the result can be seen in Figure 10 and 11. The resulting schedule is most likely not optimal, but we wanted to just give an example of a shift. For the sake of clarity we did not include any dummy tasks in this example, but normally they are used to facilitate shifting tasks to the start of the schedule. These dummies can for example be used to shift task C in such a way that it can be performed in parallel with task A, resulting in a more compact schedule.

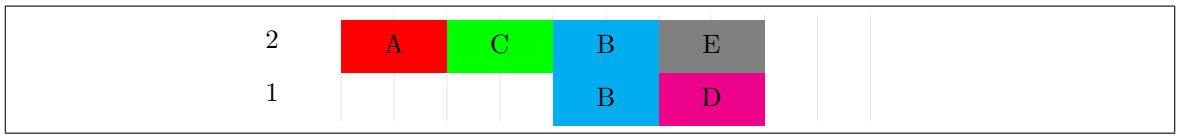

Figure 10: The R_0 resource usage of the example flow schedule after the shift

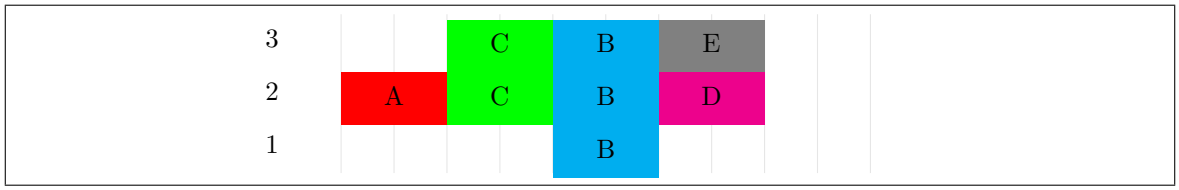

Figure 11: The R_1 resource usage of the example flow schedule after the shift

Next up is the swap. We will look at performing a swap between task B and task D. Just like with the shift we will first look at the resources we can match. For resource R_0 that means we randomly choose between the units of task B to match with the unit from task D, let's say we choose to match flow 1 of task B with flow 1 of task D. For the other unit of task B in flow 2 there is only one flow left to match with, which is flow 2, so we select that flow. For resource R_1 we do something similar. We first randomly match flow 2 of task B with flow 2 of task D. Then we match flow 1 of task B with flow 3, and lastly we match flow 3 of task B with the last remaining flow, flow 1.

Then we compute all the restrictions for the flows where we need to place units that follow from our current choices. For flow 2 of resource R_0 we need to place the unit somewhere after task A. We can deduce this from our matching we made for resource R_1. For resource flow 1 and 3 of rFesource R_1 we can see that we need to place our unit of task B somewhere behind task C because of the matching we made for resource R_0.

Now we select the spots for each of the 3 remaining units in a random order. We start with

resource R_1, flow 3. We can choose between 2 positions: behind C or behind E. Let's say, we choose behind E. Now we update the restrictions for the other flows as well. This means that for resource R_0, flow 2, we will need to place our task behind E. This fixes the positions of the units in the other 2 flows. We are done now. The results can be seen in Figure 10 and 11. A small note we want to make is that it might appear to be the case that the resulting schedule can be compacted, but this is not the case in its current form. It can be compacted by the use of the operators but to see why the resulting schedule looks this way you need to look at both Figure 10 and 11 at the same time. From Figure 10 we can deduce that in this schedule task C always starts after task D finishes, then the same needs to be true in Figure 11.

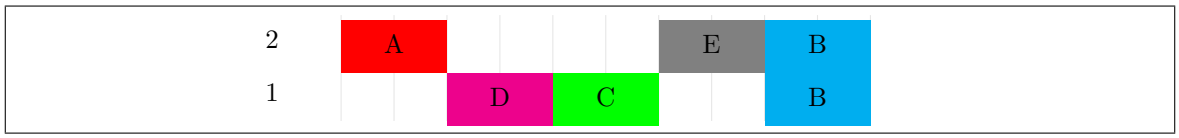

Figure 12: The R_0 resource usage of the example flow schedule after the swap

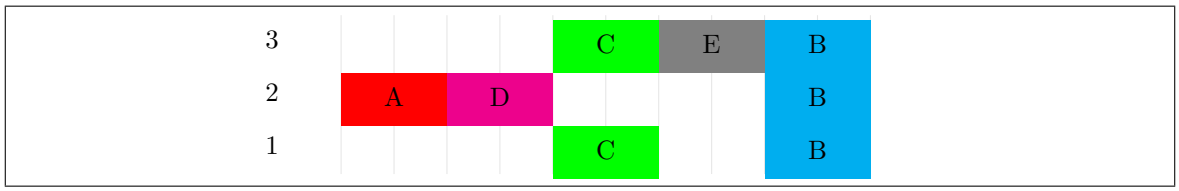

Figure 13: The R_1 resource usage of the example flow schedule after the swap

7.4.3.2 Local Search: Example POS

We will now look at local search for the POS. We will use the problem setting as described in Section 7.4.3. An example schedule can be found in Figure 14. The colours have no meaning once again, they are just there to visually differentiate the tasks. For consistency, the same tasks do have the same colours as in Figure 8 and 9. The arc from C to E in Figure 14 is from the precedence constraint and the other arcs where added to create this schedule.

Let's start with the shift. We look at shifting task E behind task D. We first remove all non-precedence incoming arcs of task E. In this case that means we remove the arc between B and E

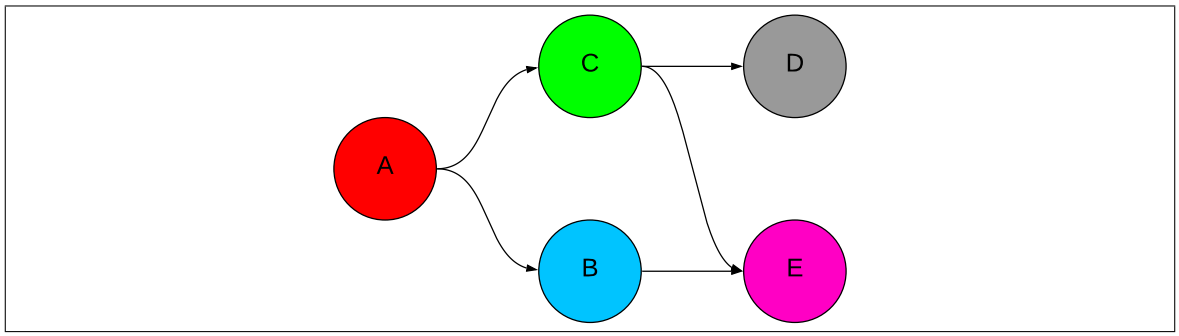

Figure 14: An example partial order schedule

and not the arc between C and E. We then add an arc between D and E, because that is the shift we wanted to perform and we are done. The results can be seen in Figure 15.

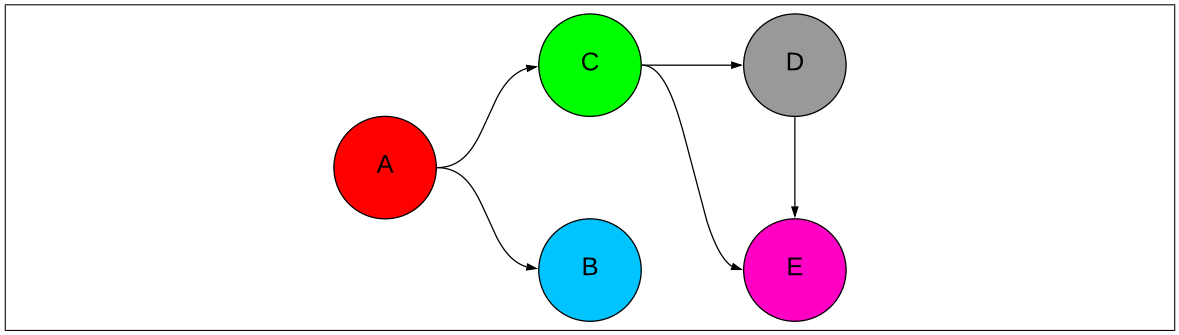

Figure 15: The example partial order schedule after shifting task E behind task D

Now we will look a swap. We will swap task A with task C. We will remove the arc from A to C, the arc from A to B and the arc from C to D. We then add an arc from C to A, from C to B and from A to D. The results can be seen in Figure 16.

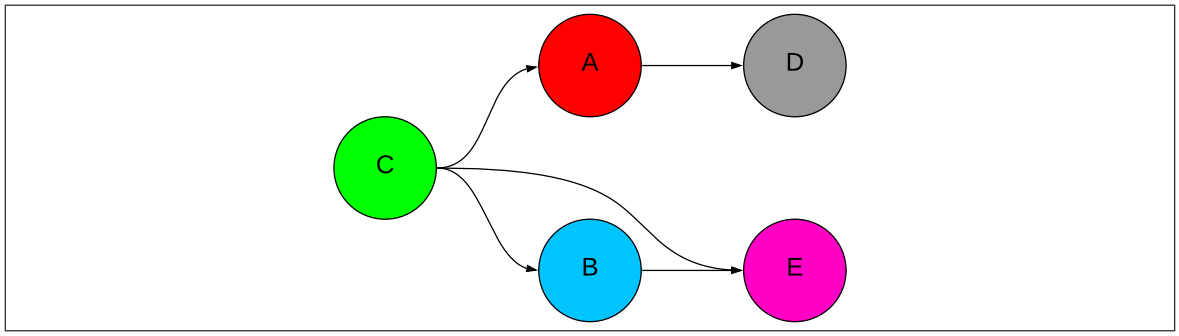

Figure 16: The example partial order schedule after swapping tasks A and C

Now we will look at the AddOrdering. We will add an arc between B and D, that is all. The results can be seen in Figure 17.

Now we will look at the RemoveOrdering. We will remove an arc between C and D. The resulting schedule consists of 2 components. The results can be seen in Figure 18.

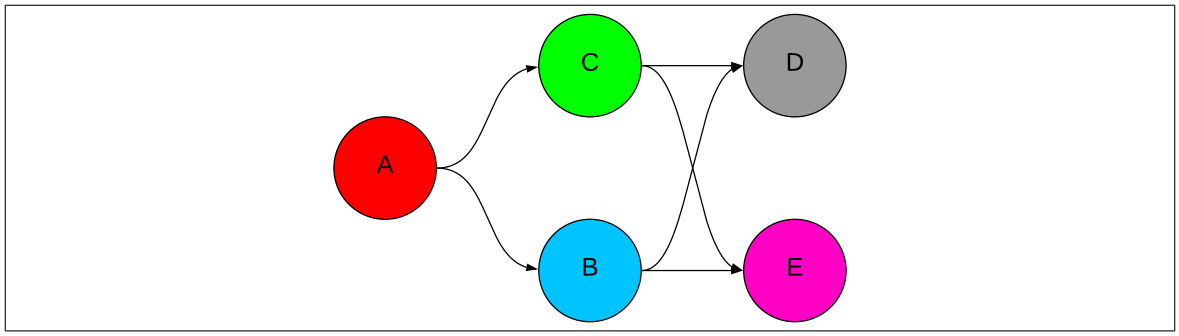

Figure 17: The example partial order schedule after a AddOrdering operation, we added an arc from B to D

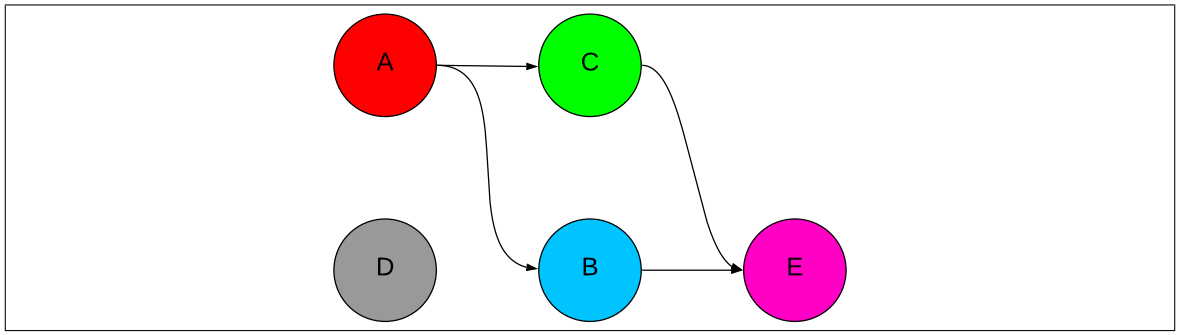

Figure 18: The example partial order schedule after a RemoveOrdering operation, we removed an arc between C and D

7.5 Experiments

In our experiments we wanted to compare the following things based on multiple criteria: final total tardiness score, time. We will also look at the standard deviation of the tardiness because the stability of the results is also important to consider. The things we want to compare are:

- How the different schedule representations compared
- How the different evaluation methods performed with each schedule representation

Before we could run the experiments we first needed to tune the settings for all the approaches to prevent bias from the selected parameters. In Section 7.5.1 we go into more detail about these experiments.

7.5.1 Parameter Tuning

We used the same setup for this set of parameter experiments as we did for the machine scheduling case, described in Section 6.11. We experimented with the starting temperature, simulation count and percentiles. All the parameter settings that are not being explicitly tested are the same as in 6.11. For all the tests involving the POS, we changed the m_P to 0.1, because we now have 4 operators to choose from instead of 2. The instances that we used for the parameter tuning can be found in Table 18. All instances use a different distribution for the processing times. The first

Instance	TaskCount	PrecedenceCount	ResourceCount	$\sigma^2(D)$
RCPSP-P0	50	25	4	1
RCPSP-P1	50	25	4	1
RCPSP-P2	50	25	4	1

Table 18: Parameter Tuning Instances RCPSP

instance uses an exponential distribution, the second instance uses an Erlang distribution with a shape parameter of 4, and the third parameter follows a normal distribution. The capacities for all the resources are randomly chosen between 1 and 4 inclusively. We refer to Section 6.9 for more information about how the instances were generated.

7.5.1.1 Starting temperature

The results for the starting temperature experiments with the flow representation can be found in Table 19 and with the POS representation can be found in Table 20.

The starting temperature for the POS representation is not very conclusive, for one instance a temperature of 1 works the best, and for another one 7 works the best. For the Flow representation we see similar results. For the machine scheduling instances we already saw that a low starting temperature seemed to work well and because the local search for the RCPSP takes more time I expected the same results, or an even lower starting temperature to work well, because with a starting temperature that is too high the method has no time to converge to a good solution.

7.5.1.2 Percentiles

The results for the percentile experiments with the flow representation can be found in Table 21 and with the POS representation can be found in Table 22.

For the flow representation the 50th percentile seemed to give the best results. For the POS representation the winner was not so clear, but percentiles in the range [0.5, 0.6] seemed to work well for these instances. These results are quite similar to what we saw for the machine scheduling problem and we expect that also for the RCPSP a slightly pessimistic approach might give a buffer-like effect which can lead to better solutions.

7.5.1.3 Simulation Count

The results for the simulation count experiments with the flow representation can be found in Table 23 and with the POS representation can be found in Table 24. For these runs a replace percentage of 0% was used because this seemed to give the best results for the machine scheduling problem.

For the flow representation and the POS representation the results are not very conclusive. We expected that higher simulation counts would give better results, but this does not seem to be the case, especially for the POS representation. We expect that this is because of the half an hour time restriction that we put on the experiments. If we increase the simulation count, we can make more informed decisions within our local search, which makes them more effective per step. But increasing the simulation count also has the effect of increasing the duration of each local search step. We expect that these effects are the cause of the inconclusive results. We did not see this effect for the machine scheduling problem, but this is because the evaluation of the machine scheduling schedules is less complicated and can be done much faster we did not run into the 30 minute barrier that often compared to these experiments.

	Starting Temperature			
Instance	1	3	5	7
RCPSP-P0	92375.81 (7374.37)	93440.77 (7917.28)	90968.53 (7645.14)	92492.58 (7994.05)
RCPSP-P1	55.25 (59.26)	66.93 (50.76)	88.60 (61.33)	78.28 (51.79)
RCPSP-P2	46878.57 (4426.52)	47293.16 (3978.30)	46165.02 (5850.45)	47604.08 (4468.62)

Table 19: Score: average (standard deviation) per instance and starting temperature. The flow representation was used and the results are the average over the normal approximation, 55th percentile and 500 simulation evaluation methods.

	Starting Temperature			
Instance	1	3	5	7
RCPSP-P0	17318.31 (5292.27)	17107.07 (4897.65)	16846.48 (4007.29)	16811.02 (4200.04)
RCPSP-P1	3.54 (5.32)	13.55 (12.27)	13.40 (10.73)	16.77 (10.40)
RCPSP-P2	9049.06 (5592.34)	8912.25 (5818.30)	8996.90 (5793.21)	9239.96 (5817.89)

Table 20: Score: average (standard deviation) per instance and starting temperature. The POS representation was used and the results are the average over the normal approximation, 55th percentile and 500 simulation evaluation methods.

	Percentiles				
Instance	0.45	0.5	0.55	0.6	0.65
RCPSP-P0	92347.91 (1412.82)	87835.46 (546.11)	94337.38 (1421.95)	91495.16 (4184.00)	91597.8 (4767.37)
RCPSP-P1	72.62 (58.95)	56.12 (34.31)	100.95 (31.37)	106.30 (28.22)	113.90 (4.45)
RCPSP-P2	46958.72 (2199.67)	45088.95 (2583.2)	48415.61 (2794.97)	46063.32 (3496.96)	46476.34 (2673.84)

Table 21: Score: average (standard deviation) per instance and processing time percentile. The evaluation method is percentile and the flow representation was used.

	Percentiles				
Instance	0.45	0.5	0.55	0.6	0.65
RCPSP-P0	15550.9 (708.74)	14804.69 (653.07)	14700.98 (486.51)	14726.93 (403.20)	14421.94 (455.01)
RCPSP-P1	4.16 (2.95)	4.01 (2.38)	4.04 (3.44)	3.75 (2.90)	5.02 (3.05)
RCPSP-P2	5519.88 (387.80)	5434.35 (469.71)	5623.29 (380.40)	5665.41 (318.94)	5730.40 (679.87)

Table 22: Score: average (standard deviation) per instance and processing time percentile. The evaluation method is percentile and the POS representation was used.

	Simulation Count				
Instance	50	100	250	500	1000
RCPSP-P0	13071.5 (171.84)	12779.52 (128.07)	12627.84 (73.43)	12632.5 (267.46)	13050.65 (216.04)
RCPSP-P1	2.31 (1.36)	2.37 (1.80)	1.86 (1.06)	1.40 (0.90)	4.80 (6.12)
RCPSP-P2	4494.74 (434.13)	4269.46 (97.33)	4527.40 (493.60)	4708.46 (376.08)	4595.20 (260.82)

Table 23: Score: average (standard deviation) per instance and simulation count. The evaluation method is simulation and the flow representation was used.

	Simulation Count				
Instance	50	100	250	500	1000
RCPSP-P0	13127.78 (473.66)	12679.89 (360.20)	12623.05 (130.32)	12862.33 (228.93)	12456.54 (170.26)
RCPSP-P1	2.25 (2.04)	4.49 (5.08)	7.34 (7.36)	6.61 (3.93)	3.81 (5.59)
RCPSP-P2	4706.49 (256.48)	4595.95 (219.97)	4887.80 (191.31)	4452.88 (127.83)	4775.02 (252.88)

Table 24: Score: average (standard deviation) per instance and simulation count. The evaluation method is simulation and the POS representation was used.

7.5.2 Results

Now that we have a good set of parameters, we can do perform experiments to compare the different representations and evaluation methods.

7.5.2.1 Instances

We generated 21 instances for this set of experiments. They were once again generated like we described in Section 6.9, with the same additions as described in Section 7.5.1 because we want RCPSP instances and not machine scheduling instances.

In Table 25 we described the settings that were used to generate each of the instances. We varied the number of tasks between 40, 60 or 80 tasks. We set the number of precedence relations as a factor times the task count, the factors we used are 0, $\frac{1}{2}$ and 1. The number of different resource types was set to 1, 3 or 5, and the capacity of each of these resources was chosen randomly between 1 and 4 inclusively. In each instance the processing times of all the tasks followed the same type of distribution. The options we tried were exponential-, normal- and Erlang distributions. We used a $\sigma(D)$ of 0.5.

For each experiment we used a starting temperature of 3. For the experiments that used the percentile evaluation method we used the 50th percentile for the POS representation and the 55th percentile for the flow representation. For the simulation evaluation method we used a simulation count of 100.

As before we ran the experiment for each combination of representation, evaluation method and instance 10 times.

7.5.2.2 Fixed Flow Schedule Results

We will now compare the different evaluation methods for the experiments that all used the flow schedule representation. The results for the percentile evaluation can be found in Table 26, the result for the simulation evaluation in Table 27, and lastly, the results for the normal approximation evaluation can be found in Table 28.

All approaches got a score of 0 on instance 21 on all runs. This is the best score that can be achieved, so with respect to total tardiness the methods performed equally good on this instance. For the rest of the instances, the simulation approach is clearly the best as can be seen in Table 27. The simulation approach gets the lowest score on 16 out of the 21 other instances. On the instances where the simulation-based approach does not get the best score, it gets the second-best score 4 out of 5 times. This makes it the clear winner for the flow representation. If we look at the time used however, the simulation-based approach is not the clear winner. We already saw this for the machine scheduling experiments, so this came as no surprise. On 16 out of the 21 instances the

Instance	Task Count	Precedence Count	Resource Type Count	Distribution
1	40	20	3	Erlang
2	40	20	3	Normal
3	40	20	3	Exponential
4	60	0	3	Erlang
5	60	0	3	Normal
6	60	0	3	Exponential
7	60	30	1	Erlang
8	60	30	1	Normal
9	60	30	1	Exponential
10	60	30	3	Erlang
11	60	30	3	Normal
12	60	30	3	Exponential
13	60	30	5	Erlang
14	60	30	5	Normal
15	60	30	5	Exponential
16	60	60	3	Erlang
17	60	60	3	Normal
18	60	60	3	Exponential
19	80	40	3	Erlang
20	80	40	3	Normal
21	80	40	3	Exponential

Table 25: The instances used for the RCPSP experiments

maximum allowed time of half an hour was consistently used in each run. And for the remaining instances it gets a second place with respect to average time.

The second-best evaluation method for the flow representation is the percentile-based approach as can be seen in Table 26. On the instances 2, 3, 5 and 7 and 21 it gets the best score and on 11 out of 21 instances it gets the second-best score. With respect to time the percentile approach is the best. For all instances it gets the lowest average time, and only for instance 7, 10, 14 and 17 it consistently used half an hour, which is much better than the other approaches.

Lastly, we have the normal approximation-based approach which gets the best score for the first instance and the second-best score 5 out of 21 times as can be seen in Table 28. Also, with respect to time the normal approximation-based approach gets the last place. On 19 out of the 21 instances it consistently used half an hour. For the remaining 2 instances it gets a second place for instance 6 and a last place for instance 15. We expect that the increased number of interdependencies between the tasks compared to the machine scheduling problem worsens the quality of the approximation. One of the reasons for this is the false assumption of a correlation coefficient of 0. Another reason is because we approximate the maximum of many distributions by a concatenation of many pairwise maxima, and this approximation gets worse if the size of the set of distributions we want the maximum over increases. What was also surprising is that the normal approximation-based approach did not perform better on the instances where the processing times were normally distributed. It seemed to do the best on instances with Erlang distributions, but this can also be a coincidence.

Overall we can say that with respect to score the simulation-based approach is the clear winner and with respect to time the percentile-based approach is the clear winner. The normal approximation-based approach clearly performed the worst with respect to score and time.

7.5.2.3 POS Schedule Results

We will continue by comparing the different evaluation methods for the experiments that all used the POS schedule representation. The results for the percentile evaluation can be found in Table 29, the result for the simulation evaluation in Table 30, and lastly, the results for the normal approximation evaluation can be found in Table 31. Just like with the flow-based representation, all approaches got a score of 0 on instance 21 on all runs. It is clear that instance 21 is not very useful for this comparison as it is just too easy to get a score of 0 it seems.

Just like with the flow representation the simulation-based approach is the best with respect to score as can be seen in Table 30. For the POS representation the superiority of the simulation-based approach is even more clear. The simulation-based approach gets the best score for all instances. This time the simulation-based approach gets last place with respect to time. On 14 out of the 21 instances the maximum amount of time is consistently used. For all the instances the simulation-based approach uses the most amount of time by far.

The percentile-based approach gets the second place with respect to score again, just like we saw for the flow representation as can be seen in Table 29. It gets the second place with respect to score for all instances. With respect to time this approach is the clear winner. On average it uses the least amount of time for all of the instances.

And again, the normal approximation-based approach performs the worst as can be seen in Table 31. For all the instances it performs the worst. With respect to time the normal approximation-based approach gets second place on all instances. We expect that for the same reasons as we gave in Section 7.5.2.2 the normal approximation-based approach performs the worst. Also for the POS representation the normal approximation-based approach did not seem to benefit from the instances with normally distributed processing times.

Overall we can say that with respect to score the simulation-based approach is the clear winner

Instance	Average Score (StdDev Score)	Average Time (StdDev Time)
1	7209.58 (987.05)	422689.1 (51253.61)
2	42469.01 (3303.73)	1311458 (117814.2)
3	92569.3 (10023.41)	1798870 (3810.01)
4	34279.76 (2215.18)	524320.9 (16745.91)
5	91514.41 (9647.38)	1734839 (206165.7)
6	20336.14 (281.34)	127978.3 (11395.58)
7	96298.73 (5797.49)	1800030 (46.52)
8	14058.6 (780.79)	313987.2 (50685.9)
9	103975.2 (5290.49)	1504530 (95688.23)
10	182351.6 (5864.15)	1800050 (42.36)
11	43291.99 (3192.88)	570084.8 (17695.13)
12	53998.07 (2786.63)	1151393 (212508.3)
13	31411.44 (1198.79)	429755.2 (25967.81)
14	128130.2 (6648.70)	1800017 (18.38)
15	224.17 (79.98)	208388.4 (29576.19)
16	91.73 (32.19)	676770.8 (74979.16)
17	176.70 (134.09)	1800033 (24.71)
18	110.55 (27.29)	779643.2 (13925.15)
19	70.27 (31.87)	471418.3 (58207.71)
20	84.51 (30.30)	723809.7 (52340.08)
21	0 (0)	1745989 (170850.9)

Table 26: Score and time: average (standard deviation) per instance. The flow representation was used and the percentile evaluation method was used.

Instance	Average Score (StdDev Score)	Average Time (StdDev Time)
1	7463.27 (1387.22)	1800009 (8.78)
2	43202.74 (1851.64)	1800024 (18.02)
3	100114.8 (7569.13)	1800021 (14.37)
4	33873.98 (2997.08)	1800008 (5.89)
5	95379.04 (7190.61)	1800028 (30.84)
6	20280.58 (125.76)	1678885 (118697.9)
7	99165.05 (5674.28)	1800038 (26.90)
8	13324.29 (652.37)	1800010 (7.17)
9	94054.38 (4343.17)	1800032 (21.28)
10	153316.7 (5905.02)	1800049 (41.79)
11	25090.61 (523.95)	1800011 (6.22)
12	52338.37 (2607.32)	1800013 (10.58)
13	21957.39 (580.05)	1800010 (7.20)
14	121574.9 (5952.96)	1800024 (11.89)
15	6.26 (1.75)	806187.2 (11950.69)
16	12.10 (6.55)	1756746 (49819.72)
17	1.23 (1.63)	1800040 (27.16)
18	0.62 (1.28)	1800011 (9.65)
19	0.13 (0.22)	1799875 (465.80)
20	11.74 (5.51)	1730583 (53692.64)
21	0 (0)	1800020 (18.16)

Table 27: Score and time: average (standard deviation) per instance. The flow representation was used and the simulation evaluation method was used.

Instance	Average Score (StdDev Score)	Average Time (StdDev Time)
1	6961.39 (1087.12)	1800007 (8.19)
2	50740.35 (3861.91)	1800015 (11.11)
3	113299.7 (6765.49)	1800026 (25.61)
4	41660.16 (2911.22)	1800004 (4.20)
5	98976.35 (6608.08)	1800020 (23.04)
6	20890.38 (361.72)	951963.3 (409614.8)
7	104374 (7246.46)	1800023 (20.39)
8	15793.11 (724.20)	1800002 (1.95)
9	117284.5 (5950.30)	1800012 (11.52)
10	197512.1 (10348.93)	1800040 (51.80)
11	35705.81 (2022.85)	1800005 (6.60)
12	60902.68 (5520.91)	1800011 (8.76)
13	25550.13 (934.38)	1800002 (3.57)
14	140050.9 (6448.24)	1800021 (18.31)
15	310.18 (41.78)	1718157 (258816.7)
16	87.43 (28.76)	1800005 (7.18)
17	515.68 (142.78)	1800034 (18.42)
18	234.99 (76.92)	1800005 (9.13)
19	48.03 (28.63)	1800006 (5.23)
20	68.01 (32.25)	1800014 (12.95)
21	0 (0)	1800035 (24.98)

Table 28: Score and time: average (standard deviation) per instance. The flow representation was used and the normal approximation evaluation method was used.

Instance	Average Score (StdDev Score)	Average Time (StdDev Time)
1	1012.12 (107.92)	13090 (19947.71)
2	7431.54 (731.86)	6234.9 (1050.91)
3	14487.15 (1151.79)	47651.3 (51799.3)
4	7605.63 (564.68)	12479.8 (17923.27)
5	12663.84 (746.85)	17709.1 (20622.29)
6	20087.05 (66.35)	8859.4 (6681.70)
7	7664.16 (612.48)	17069.4 (18660.92)
8	4306.70 (231.20)	5078.9 (1994.97)
9	17420.02 (732.83)	20874.9 (33456.18)
10	30174.41 (1127.69)	33421 (41839.6)
11	10715.18 (902.68)	9476.1 (10463.89)
12	16899.3 (494.37)	11422.7 (7030.54)
13	11860.42 (1017.02)	16381.7 (22041.05)
14	18000.96 (1043.99)	29574 (40839.46)
15	104.07 (12.34)	6861.4 (5160.52)
16	19.06 (8.48)	10478.5 (11503.58)
17	64.61 (14.56)	18865.9 (28129.71)
18	26.68 (11.14)	4667.6 (3528.32)
19	17.37 (16.55)	6162.5 (375.57)
20	22.65 (8.40)	41229.9 (62773.58)
21	0 (0)	27471.1 (27645.22)

Table 29: Score and time: average (standard deviation) per instance. The POS representation was used and the percentile evaluation method was used.

again. With respect to time we have a clear ordering. The percentile-based approach is clearly the best, then comes the normal approximation and lastly we get the simulation-based approach.

7.5.2.4 Schedule Representation results

Now we will also compare the results for the flow representation with the results for the POS representation. For instances 15 and 19 the flow representation got a better average score if we compare the best score between the 3 evaluation methods for each representation. For the other 19 instances the POS representation got a better final score. Also, the POS scores were more consistent, for almost all cases the standard deviation of the score is lower if we compare the two representations, this holds for all the evaluation methods. The POS representation is also much faster than the flow representation, but due to the increased complexity of the local searcher for the flow representation we expected this.

Instance	Average Score (StdDev Score)	Average Time (StdDev Time)
1	912.39 (93.84)	1800004 (3.68)
2	6219.40 (480.03)	1800010 (7.94)
3	15045.3 (1164.68)	1800013 (7.28)
4	6983.44 (258.69)	1800006 (4.01)
5	11743.59 (676.68)	1800010 (6.83)
6	18650.29 (42.30)	1800007 (4.63)
7	7184.75 (364.49)	1800008 (7.05)
8	3179.37 (96.57)	1800008 (7.89)
9	14497.21 (333.88)	1800009 (4.90)
10	21699.63 (716.74)	1800010 (7.43)
11	6347.52 (297.57)	1800005 (3.35)
12	13452.53 (494.52)	1800008 (5.77)
13	8448.27 (352.44)	1800009 (8.23)
14	14398.39 (418.87)	1800011 (9.14)
15	15.86 (7.14)	1159174 (42031.44)
16	0.41 (0.80)	1037301 (14677.13)
17	0.40 (0.52)	1544779 (21807.53)
18	0.06 (0.14)	1111994 (17949.36)
19	3.06 (3.61)	1165717 (18770.13)
20	0.08 (0.02)	1035662 (15988.72)
21	0 (0)	1039343 (15593.46)

Table 30: Score and time: average (standard deviation) per instance. The POS representation was used and the simulation evaluation method was used.

Instance	Average Score (StdDev Score)	Average Time (StdDev Time)
1	1548.20 (329.46)	389599.2 (370968.8)
2	8927.22 (972.88)	1533501 (408502.9)
3	20948.8 (2576.50)	1560575 (385360.8)
4	12360.22 (1424.88)	488715.9 (206476.7)
5	15228.28 (1266.05)	1096349 (592921.4)
6	21579.61 (493.01)	2604126 (5391606)
7	10953.97 (1230.42)	1668174 (560138.3)
8	5073.66 (460.83)	1804340 (4721539)
9	21782.34 (675.29)	1980009 (1694216)
10	34946.09 (2255.49)	5038050 (5973996)
11	12010.46 (823.85)	57389.93 (24609.94)
12	20986.23 (1370.63)	2021080 (3099365)
13	15977.56 (1493.29)	1953951 (3375619)
14	20491.79 (996.70)	1497676 (968511.5)
15	134.08 (17.87)	913400.7 (2032228)
16	30.38 (4.30)	1099270 (1249184)
17	67.61 (7.56)	2154117 (830160.2)
18	68.81 (15.53)	62751.06 (14343.9)
19	24.10 (10.19)	1334912 (1615765)
20	26.55 (9.61)	1438637 (1155239)
21	0 (0)	1683391 (798348.8)

Table 31: Score and time: average (standard deviation) per instance. The POS representation was used and the normal approximation evaluation method was used.

8 Final Conclusions

Now that we performed all the experiments, we will discuss the combined results for the machine scheduling case and the resource constrained project scheduling case. We will also formulate the answers to the questions we posed in Section 5.

We saw that for both the machine scheduling problem and the RCPSP simulation seemed to give the best results with respect to final total tardiness score. We expected that the simulation-based evaluation method would at least outperform the percentile-based approach in terms of score because the percentile-based approach is a much simpler approach that simplifies the processing time distributions into single numbers. The simulation-based approach does this as well, but it samples the distribution, so the number can be different each time and by averaging these results we do take more of the whole distribution into account. We also think that the flexibility of the simulation approach given by the simulation count parameter is an advantage over, for example, the percentile-based approach. By having this parameter the evaluation method can be tuned to the specific settings of the application it is used in, and the user can make a trade-off between final score and time available. This might be more difficult with the other two evaluation methods.

The percentile-based approach and normal-approximation-based approach perform quite similar with respect to score for the machine scheduling problem. We expect that the more sophisticated normal approximation approach does not work better because of our objective function. We expect that it would have worked better if we would have looked at expected makespan optimisation. For the RCPSP we saw that the percentile-based approach now outperformed the normal approximation-based approach which is partially for the same reasons as we gave for the machine scheduling case and partially because of the increase in the number of predecessors of tasks which makes the approximation worse at each step. Also, for the RCPSP we did not only consider normally distributed processing times for our tasks which also makes the approximations worse in the cases where the processing times were not normally distributed.

In many applications time is of the essence and we saw that our best performing evaluation method, the simulation approach, was the slowest approach for both problems. Because of this it might be advantages in some cases to not use the simulation-based approach but use the percentile-based approach when the amount of time available is very limited. This difference in speed was especially notable for the RCPSP. Because the percentile-based approach performed equally or outperformed the normal approximation-based approach for both problems with respect to score and time we think that the percentile-based approach is the better alternative.

We saw in our experiments that the time needed for evaluation was a big part of the total time needed that is why it comes as no surprise that the simulation-based approach performed the worst with respect to time. We also saw for both problems that an increase in the number of simulations per evaluation started to give diminishing returns at some point. The point at which this happens depends on the problem and the instance being considered. This is an important point to consider because of the linear dependence between the time and the simulation count.

In conclusion, in most cases the simulation approach is the best way to assess the expected total tardiness although if the available time is very limited, a percentile-based approach might be more suitable for the application. This holds for both the machine scheduling problem and the resource constrained project scheduling problem.

We also saw that in terms of score and time the POS representation outperformed the flow representation. This is not a completely fair comparison because the different representations have different applications. The implementation of the local searcher for the POS representation is less complicated which is also an advantage of the POS representation. The flow representation has much more explicit control over the flow of the resources which might be necessary for some applications.

We recommend some form of local search for this machine scheduling problem and for the RCPSP because in practice exact methods are not feasible. A good selection of operators is very important for the success of the operators and a way to select when to use which operator as well. We think that an adaptive operator selector is very useful and believe it outperforms a more static approach. As we said before the simulation-based evaluation seems to work the best in most cases given our local search approach. This holds for both problems. We do think that there are many avenues for improvement and we will talk about them in more detail in the next section.

9 Further Research

There are still many open questions, but we had to stop somewhere with our research. In this section we discuss some of the things that we think are still worth looking into.

For the POS representation we only gave a single priority rule for the normal approximation method. We expect that the specific rule used can have a large impact on the performance and that it might be worth investigating how much potential there still is. The priority rule does not take multiple paths to the same tasks into account which is something that can negatively influence the results.

We propose some more priority rules that might be interesting to look at. As we said before, with our current priority rule we do not take multiple paths to the same tasks into account. We do take this into account in the Equation 6. This priority value can still be easily computed in a bottom-up manner so can still be efficiently computed. In Equation 7 we take a more pessimistic approach in which we also take the standard deviation into account. This will cause the evaluation to give a higher priority to tasks with more uncertainty which might be desirable in some cases. In Equation 8 we take the maximum instead of the sum which causes the evaluation to focus on the earliest tasks that can give us a tardiness penalty. We also think that taking the resource requirements of the tasks into account might be very useful because we might be able to perform some of the tasks in parallel and we could approximate that by taking these into account.

$$Prio_2(Task_j) = -(d_j - E(P_j)) - \sum_{\{Task_i| \text{ there is a path from } Task_j \text{ to } Task_i\}} -(d_i - E(P_i)) \quad (6)$$

$$Prio_3(Task_j) = -(d_j - E(P_j)) - \sigma_j) - \sum_{Task_i \in Suc(Task_j)} Prio_3(Task_i) \quad (7)$$

$$Prio_4(Task_j) = max(-(d_j - E(P_j)), max(Prio_4(Suc(Task_j)))) \quad (8)$$

It might also be intersecting to look at the expected makespan or other optimisation criteria and see how changing them influences the results.

A possible improvement to the local search for the flow representation can be to add more neighbourhoods. We expect that the addition of a neighbourhood that moves the resource units of a task to different resource flows or moves the unit within the same flow can be very beneficial.

We also came up with our own approach for selection which operator to use at what moment in Section 6.6, but we have not compared this approach to other approaches. It would be interesting to see what other methods are available and how they compare.

The simulation-based approach currently works the best, but the main drawback is the amount of time the evaluation requires compared to the other approaches. It is quite likely very beneficial to use an adaptive simulation count where at the start of the search we can get away with a lower count

and as we start to converge, we increase the simulation count such that we can better approximate the expected total tardiness and better fine tune our solution. Another similar option in using the percentile evaluation at the start and after a certain number of iterations we can switch to the simulation-based approach. This could speed up the whole searching process quite a lot.

10 Appendix

		Simulation Count				
		50	100	150	200	250
Replace Percentage	100	22.44	22.21	22.17	22.23	22.15
	75	22.3	22.46	22.18	22.22	22.13
	50	22.26	22.19	22.29	22.16	22.14
	25	22.3	22.18	22.18	22.15	22.16
	0	22.04	22.04	22.04	22.04	22.04

Table 32: Average score for instance P1-1 for different Simulation Count and replace percentage settings

		Simulation Count				
		50	100	150	200	250
Replace Percentage	100	78.77	76.43	75.53	76.27	77.07
	75	78.29	76.99	77.16	76.99	75.15
	50	75.53	77.18	75.5	76.03	75.17
	25	75.68	75.53	75.75	76.59	75.05
	0	74.98	75.45	74.98	76.82	74.91

Table 33: Average score for instance P1-2 for different Simulation Count and replace percentage settings

		Simulation Count				
		50	100	150	200	250
Replace Percentage	100	221.41	224.88	220.29	219.79	219.47
	75	224.28	222.56	220.27	219.72	221.02
	50	220.51	223.2	223.23	218.6	217.11
	25	221.53	219.07	219.02	215.57	220.54
	0	218.55	219.35	214.28	214.43	215.55

Table 34: Average score for instance P1-3 for different Simulation Count and replace percentage settings

		Simulation Count				
		50	100	150	200	250
Replace Percentage	100	4.07	3.6	3.52	3.69	3.71
	75	3.88	3.61	3.9	3.52	3.49
	50	3.73	3.66	3.43	3.52	3.54
	25	3.53	3.6	3.7	3.5	3.64
	0	3.55	3.4	3.36	3.37	3.43

Table 35: Average score for instance P1-4 for different Simulation Count and replace percentage settings

References

[1] C. Artigues, R. Leus, and F. T. Nobibon. "Robust optimization for resource-constrained project scheduling with uncertain activity durations". In: *Flexible Services and Manufacturing Journal* 25.1-2 (2013), pp. 175–205.

[2] J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. "Scheduling subject to resource constraints: classification and complexity". In: *Discrete applied mathematics* 5.1 (1983), pp. 11–24.

[3] M. Brčić, D. Kalpić, and K. Fertalj. "Resource constrained project scheduling under uncertainty: a survey". In: *23rd Central European Conference on Information and Intelligent Systems*. 2012.

[4] R. van den Broek, H. Hoogeveen, and M. van den Akker. "How to Measure the Robustness of Shunting Plans". In: *OASIcs-OpenAccess Series in Informatics*. Vol. 65. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[5] P. Brucker, A. Drexl, R. Möhring, et al. "Resource-constrained project scheduling: Notation, classification, models, and methods". In: *European journal of operational research* 112.1 (1999), pp. 3–41.

[6] J. Bruno, P. Downey, and G.N. Frederickson. "Sequencing tasks with exponential service times to minimize the expected flow time or makespan". In: *Journal of the ACM (JACM)* 28.1 (1981), pp. 100–113.

[7] R.K. Chakrabortty, R.A. Sarker, and D.L. Essam. "Resource constrained project scheduling with uncertain activity durations". In: *Computers & Industrial Engineering* 112 (2017), pp. 537–550.

[8] H. Chtourou and M. Haouari. "A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling". In: *Computers & industrial engineering* 55.1 (2008), pp. 183–194.

[9] S. Creemers. "Minimizing the expected makespan of a project with stochastic activity durations under resource constraints". In: *Journal of Scheduling* 18.3 (2015), pp. 263–273.

[10] S. Creemers, R. Leus, and M. Lambrecht. "Scheduling Markovian PERT networks to maximize the net present value". In: *Operations Research Letters* 38.1 (2010), pp. 51–56.

[11] K.M.J. De Bontridder. "Minimizing total weighted tardiness in a generalized job shop". In: *Journal of Scheduling* 8.6 (2005), pp. 479–496.

[12] F. Deblaere, E. Demeulemeester, W. Herroelen, et al. "Robust Resource Allocation Decisions in Resource-Constrained Projects". In: *Decision Sciences* 38.1 (2007), pp. 5–37.

[13] H. Emmons and M. Pinedo. "Scheduling stochastic jobs with due dates on parallel machines". In: *European Journal of operational research* 47.1 (1990), pp. 49–55.

[14] E.M. Goldratt. "Critical Chain". In: *Great Barrington: The North river press-publishing corporation* (1997).

[15] R.L. Graham, E.L. Lawler, J.K. Lenstra, et al. "Optimization and approximation in deterministic sequencing and scheduling: a survey". In: *Annals of discrete mathematics*. Vol. 5. Elsevier, 1979, pp. 287–326.

[16] V. Gupta, B. Moseley, M. Uetz, et al. "Stochastic online scheduling on unrelated machines". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2017, pp. 228–240.

[17] S. Hartmann and D. Briskorn. "A survey of variants and extensions of the resource-constrained project scheduling problem". In: *European Journal of operational research* 207.1 (2010), pp. 1–14.

[18] Ö. Hazır, M. Haouari, and E. Erel. "Robust scheduling and robustness measures for the discrete time/cost trade-off problem". In: *European Journal of Operational Research* 207.2 (2010), pp. 633–643.

[19] W. Herroelen, B. De Reyck, and E. Demeulemeester. "Resource-constrained project scheduling: a survey of recent developments". In: *Computers & Operations Research* 25.4 (1998), pp. 279–302.

[20] W. Herroelen and R. Leus. "Project scheduling under uncertainty: Survey and research potentials". In: *European journal of operational research* 165.2 (2005), pp. 289–306.

[21] T. Kämpke. "Optimal scheduling of jobs with exponential service times on identical parallel processors". In: *Operations Research* 37.1 (1989), pp. 126–133.

[22] R. Leus. "The generation of stable project plans". In: (2003).

[23] L.L. Lorenzoni, H. Ahonen, and A. G. de Alvarenga. "A multi-mode resource-constrained scheduling problem in the context of port operations". In: *Computers & Industrial Engineering* 50.1-2 (2006), pp. 55–65.

[24] S. Nadarajah and S. Kotz. "Exact distribution of the max/min of two Gaussian random variables". In: *IEEE Transactions on very large scale integration (VLSI) systems* 16.2 (2008), pp. 210–212.

[25] G.J.P.N. Passage. "Combining local search and heuristics for solving robust parallel machine scheduling". https://dspace.library.uu.nl/handle/1874/371568. MA thesis. the Netherlands: Utrecht University, 2016.

[26] G.J.P.N. Passage, J.M. van den Akker, and J.A. Hoogeveen. "Local search for stochastic parallel machine scheduling: improving performance by estimating the makespan". In: *European Conference on Stochastic Optimization*. 2017.

[27] M.L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[28] S. Shadrokh and F. Kianfar. "A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty". In: *European Journal of Operational Research* 181.1 (2007), pp. 86–101.

[29] S. Van de Vonder, E. Demeulemeester, W. Herroelen, et al. "The use of buffers in project management: The trade-off between stability and makespan". In: *International Journal of production economics* 97.2 (2005), pp. 227–240.

[30] M. Vanhoucke, E. Demeulemeester, and W. Herroelen. "An exact procedure for the resource-constrained weighted earliness–tardiness project scheduling problem". In: *Annals of Operations Research* 102.1-4 (2001), pp. 179–196.

[31] X. Wu and X. Zhou. "Stochastic scheduling to minimize expected maximum lateness". In: *European Journal of Operational Research* 190.1 (2008), pp. 103–115.