Abstract

Scheduling has been subject to much research. The resource-constrained project scheduling problem (RCPSP) is no exception. With the multiple different variations and additions to the standard definition that are possible, many exact, heuristic and meta-heuristic approaches have been proposed. One of those variations is allowing tasks in the project to be split up into smaller subsegments and scheduled at different times. The splitting of tasks is introduced to try and find project schedules that require less overall time to complete.

This paper proposes a satisfiability (SAT) encoding for the RCPSP while also allowing tasks to be split. A SAT solver is used to solve the encoded problem instances and compared with the results of a heuristic algorithm. The comparison shows a lower rate of solved instances but found results are in almost all cases more optimal and proof of optimality can be provided. For applications where the sacrifice of completing tasks without splitting is only worthwhile if the absolute lowest schedule duration can be found this method provides a good alternative to other approaches.

1 Introduction

The problem of scheduling tasks arises in industries all the time. It is not hard to imagine that generating an optimised schedule can be of great profit for production or logistic operations. For example, optimisation can minimise the overall required time or minimise the delay before starting a task. Because this type of problem is so prevalent it has already been subject to much research.

Formally this specific type of problem is known as the resource-constrained project scheduling problem (RCPSP) [1–3]. Problems of this type are considering the allocation of tasks to resources within a project. Tasks are anything that needs to be done during a project. Examples could be work performed by people or a simulation run by a computer. Resources are made available for a project like experts assigned to the project or time reserved on computers to run simulations. Tasks are given a duration required to finish and a required amount of a resource during processing. Resources are limited in availability during the project, eg. only one expert is available at a single moment in time so multiple tasks requiring an expert must be processed consecutive. An important addition to this setup is the order in which pairs of tasks must be processed. Some tasks require another task to be finished before it can be started. This requirement is referred to as a precedence constraint.

It can be required by an application of an algorithm to include additional constraints on schedules. To provide for these additional constraints variations and extensions to the problem definition have been classified over time [4], [5]. More recently, the variations and extensions have also been surveyed and put into a structured overview [6].

For this research, the preemptive resource-constrained project scheduling problem with setup times (PRCPSP-ST) variant is under study. Preemption allows a task to be interrupted during its scheduled time by another task. Each interruption can be seen as splitting the task into multiple smaller activities. The setup times are introduced for each interruption in a task to discourage endless splits resulting in a chaotic schedule. A model for allowing preemption [7] and a model including setup times [8] have already been established. Both models have been combined and a proposed algorithm for it was found to result in a reduction of the makespan compared to the optimal schedule without task preemption [9]. Within this algorithm, the activities are split into all possible integer time segments and a satisfiability (SAT) solver selects a feasible subset from these segments [10]. The resulting list was used to construct a schedule with a genetic algorithm established in earlier research [11].

In the research done on solving RCPSP variants, the recent focus has been on heuristic and meta-heuristic algorithms. These algorithms are usually variants of branch-and-bound algorithm [7] or a form of genetic algorithms [12] that were established for the standard RCPSP.

SAT solvers are a general tool that can be used on any algorithmic problem if it is encoded as the required input for the solver. A SAT solver is a program that tries to solve Boolean satisfiability problems. Boolean satisfiability problems are a set of true or false variables that are set up in a formula with ‘AND’ and ‘OR’ operators (sets of variables and operators are often referred to as clauses). The SAT solver tries to assign values to all variables in such a way that the outcome of the overall formula becomes true (also referred to as being satisfied). If a Boolean formula encoding can be found for a problem, a SAT solver can be used to try and solve it. In addition to finding a solution to the problem, it can include information about the solution it provides like the assignment for the variables it found or a minimal set of unsatisfiable clauses.

SAT solvers have been used as a part of the algorithm but using them as a complete solution to try and solve PRCPSP-ST instances has not been researched before. Because RCPSP is known to be strongly NP-hard [13], a SAT solver might not be efficient enough to outperform the heuristic and meta-heuristic methods. But what a SAT solver does provide is a way to prove if a found solution is optimal and can therefore not be improved further. When looking for a reduction in makespan getting confirmation on optimality might be worth some possible trade-offs a SAT solver introduces.

Because there is room to try and find out if a SAT solver can match or even outperform the heuristic and meta-heuristic algorithms the main research question considered in this paper is *Can a satisfiability encoding of PRCPSP-ST models solved on a SAT solver be used to reduce the average makespan of the resulting schedule compared to a heuristic algorithm when run for an equal amount time?*.

This paper first proposes a heuristic algorithm that can solve the extended RCPSP instances with preemption and setup times. This algorithm is used to produce a baseline result for the instances in three datasets. And then, a satisfiability encoding for the instances is run on a SAT solver for an equal amount of time. The results are compared, and it shows that the SAT solver approach does struggle with coming up with an initial solution in many cases when the time limit is strict. It does make up for it by providing better solutions in almost all cases and providing proof of optimality in over half of its results. For applications of a satisfiability approach, the trade-off between always having a solution compared to knowing when a solution is optimal will have to be considered.

The following sections of this paper can be summarized as follows. Section 2 gives the formal definition of the RCPSP and PRCPSP-ST problems followed by an overview of research done in the field of solving the PRCPSP-ST problem. Section 3 describes the heuristic algorithm used to generate baseline results and the Boolean logic encoding for the PRCPSP-ST problems and the SAT solver that will be used during testing. Next, section 4 outlines the experiments that are going to be run to test the performance of both algorithms and it presents the result data from the experiments. A short reflection on the ethical implications of competition for the next state-of-the-art solution is given in section 5. The results are discussed in section 6. Section 7 concludes the research and motivates where future research could be focused on.

2 Problem Formulation

The resource-constrained project scheduling problem (RCPSP) is a strongly NP-hard algorithmic problem [13] where the objective is to minimise makespan (overall required time to finish all tasks).

RCPSP is about a project consisting of a set of tasks N which all have to be completed to finish the project. Each task i has a duration d_i and requires an amount $r_{i,k}$ of a resource type k. A project provides a set of limited resource types R to process tasks each with an availability a_k constant throughout the project horizon. Tasks can be scheduled in timeslots as long as the overall resource type requirement does not exceed the provided amount at any time. Furthermore, a (possible empty) set of task pairs (i, j) defines precedence relations A where i has precedence over j. The task pairs have a finish-start type precedence meaning that a task must be completed entirely before its successor can be started. Some additional assumptions are that each resource type required by any task is provided $k \in R$, no single task will require more of a resource type than provided $r_{i,k} \leq a_k$ and a worst-case scenario makespan called the horizon T is given by the sum of all task durations.

This project structure can be modelled as an activity-on-the-node network (where activity also means task) $G = (N, A)$. The network is extended with 2 dummy tasks that model the start and finish of the project. These dummy tasks have a duration of 0 and no resource requirement. The makespan can now be defined as the starting time of the finish dummy task. An example project network can be seen at the top of figure 1.

2.1 PRCPSP-ST

The RCPSP definition can be extended in different ways including task preemption and setup times. All previous proposed and researched extensions have been surveyed and summarised in a paper [6, 14].

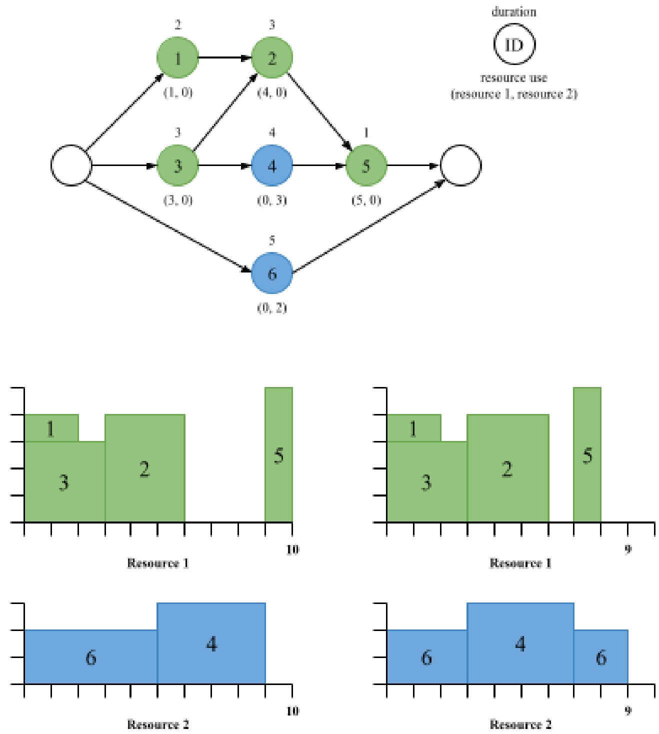

Figure 1: An example project network (top), non-preemptive schedule (left) and preemptive schedule with makespan reduction (right)

Preemption allows an activity to be paused after it has been started by the project. A preempted task in a schedule is like multiple individual tasks that each represent a segment of the original task. Preemption is only allowed at integer points of the task duration. Tasks might be preempted at any fraction of the duration but the infinite ways to split a task make defining an algorithm much harder. A solution to approximate fractional preemption is rescaling the time units used project. When hours are scaled down to minutes for example, a task can be preempted on each minute instead of on the hour approaching a possible required granularity. Figure 1 shows a non-preemptive and preemptive schedule for an example project.

Setup time s is introduced to try and prevent tasks from being split into many impractical segments. This prevention is done by adding additional processing time (setup time) to task segments that start after a previous segment has been preempted. During the setup time, the same amount and type of resources are required as the task itself. By penalizing preemption in this way algorithms will only introduce split tasks when the makespan can be improved in a meaningful way.

2.2 Related Work

Algorithms for both preemption and the inclusion of setup times have been proposed and researched in earlier work.

For pre-emption, a branch-and-bound procedure was used to show that task preemption did not reduce the makespan of projects by a significant amount [7]. A zero-one integer programming approach was used to solve projects with preemption under multiple objectives [15]. Colouring techniques were used to produce optimal preempted schedule results [16]. Later, meta-heuristic approaches were shown to be effective on preemption-based scheduling with linear programming algorithms [17, 18]. The branch-and-bound was researched again but this time allowing parts of tasks to be scheduled in parallel called fast-tracking [19]. This fast-tracking did show a more significant reduction in makespan at the cost of the problem becoming more complicated. The introduction of preemptive scheduling to the multi-mode variant of the RCPSP has also been studied [20–22].

Setup times have been introduced for task segments after preemption before [8]. The setup times can be modelled in different ways and are summarized in an overview paper [23]. The fast-tracking method used for preemptive scheduling also included setup times after a task is split [19].

Newer research showed that even without fast-tracking preempted tasks can sometimes lead to a makespan reduction contrary to previous research [9]. It also showed that when this reduction is possible it does not require many splits in the project tasks. This motivates further research into the PRCPSP-ST problem variant if having a lower makespan is profitable enough to have tasks not be finished once it is started. The algorithm used to show that makespan reductions are possible is meta-heuristic and does not provide proven optimal solutions. When having a lower makespan is critical enough to sacrifice tasks being completed in one go it might be reasonable to require the schedule to be the most optimal it can be. This research will provide a satisfiability encoding of the problem that when solved is proven to be optimal.

3 Heuristic and SAT solver

Two algorithms were implemented for this research to be compared: a heuristic algorithm and a Boolean satisfiability encoding run on a SAT solver.

The heuristic algorithm is an adapted version of the iterated greedy algorithm [24]. It was designed for flow-shop scheduling but with a few tweaks, it can also be applied to RCPSP.

For a SAT solver to be used the PRCPSP-ST must be encoded into conjunctive normal form Boolean logic. When the encoding is made any SAT solver can provide feasible schedules. Because there is a clear objective to reduce the makespan a more advanced MAX-SAT solver is used. A MAX-SAT solver expands the features of a SAT solver by adding native support for an objective function. This objective function gives a score to each solution found and when running a MAX-SAT solver it continues to maximize this score until no further improvements can be found.

3.1 Heuristic

As a heuristic solution, a tweaked version of the iterated greedy algorithm is implemented [24]. This algorithm requires an activity list representation of the project. It starts with a setup of an initial schedule and then iterates over a destruction phase and a construction phase until a time limit or a given number of iterations limit is reached.

An activity list representation allows a serial generation scheme to construct a feasible schedule [25]. The activity list represents a project as a permutation vector of all the tasks. It is required that no task appears in the list after any of its successors. The serial generation scheme schedules all the tasks in the order of the list at the earliest possible start time that does not break precedence or resource restrictions. Because the generation scheme uses the tasks in order of the list tasks close to the front of the list can be seen as having a higher priority and are scheduled sooner by the algorithm. When the algorithm has scheduled all tasks, the result is a left-justified schedule.

The initial schedule is generated with the use of a greedy heuristic. Firstly, a resource utility rate u_i is calculated for each task

$$u_i = \frac{d_i \times r_{i,k}}{a_k} \tag{1}$$

For each task, its resource requirement is divided by the availability of the resource. The result is multiplied by the task duration. Next, all tasks are put into a list and ordered by non-increasing resource utility rate. After ordering each task is moved directly in front of the first successor in the list. The result is an activity list representation, and the serial generation scheme is run to create the initial (left-justified) schedule.

After the initial list is generated the main iterative part of the algorithm starts with the destruction phase. During this phase, a copy is made of the initial schedule and next $d = \lceil \frac{|N|}{4} \rceil$ tasks are removed from the activity list at random. These are picked one by one and are kept separately in the order they were removed.

The second step of the main iteration is the construction phase. From the removed tasks the first is picked and placed at any index in the remaining activity list that doesn't break the precedence order. For each possible index, the makespan of the left-justified schedule made with the serial generation scheme is calculated. The index with the lowest makespan is chosen and the task is inserted at the index. This process is repeated for each removed task until all tasks are in the activity list added to the schedule. At this point the resulting schedule makespan is compared to the initial schedule makespan and when an improved makespan has been found the initial schedule is overwritten by the new schedule.

This heuristic solution can be run for any number of destruction and construction phases until either an iteration limit is reached, or a time limit is reached. At that point, it will return the most optimal schedule it has found.

3.2 CNF Encoding

The conjunctive normal form encoding used for this research is based on existing work used to solve the RCPSP with SAT [26]. This encoding was altered to include preemption.

For the new encoding, a project must be extended into a new network N^* by replacing each task with a new network of tasks that represents all possible ways the task could be preempted. This extension has already been documented but a summary will be given [9]. A task will be split into a set of all possible integer segments. From this new set of segments, all chains of segments are generated that represent the original task in its entirety. These chains can now replace the original task in the task network. All segments are added as new tasks with precedence relations representing the segment chains. All predecessors of the task get an additional successor for each segment that contains the first integer part of the original task. Each of the segments that contain the last integer part of the original task should contain the original successors of the task.

With the extended network the earliest es_i and its latest ls_i start times, its earliest ef_i and its latest lf_i finish times are calculated using the critical-part method by the Floyd-Warshall algorithm [27]. With these values two Boolean variables will be defined and used in the SAT clauses. For each task $i \in N$ and $t \in \{es_i, ..., ls_i\}$ there is a start variable $s_{i,t}$ which is true if activity i start at time t and for east task $i \in N$ and $t \in \{es_i, ..., lf_i\}$ a process variable $u_{i,t}$ which is true if activity i is in process at time t.

Now the complete encoding can be made, and it includes five types of clauses. The completion, consistency, precedence, resource and objective clauses. The first four are defined as hard clauses meaning the SAT solver must satisfy them. For the objective, a set of soft clauses is used. The SAT solver will try and maximize the number of soft clauses it can satisfy.

Completion clauses make sure that each task segment is processed once and therefore make sure that all the work in the project is done. New subsets $C^*_{i,l} \subseteq N^*$ are required to define the completion clauses. These subsets each have as elements all task segments that contain time segment l of task i. Equation 2 gives the mathematical definition of the completion clauses.

$$\bigvee_{t \in es_i, ..., ls_i} s_{it} \qquad j \in N; l \in \{0, ..., d_j\}; i \in C^*_{j,l} \tag{2}$$

When a start variable of a task is set to true the consistency clauses given in equation 3 ensure that the required process variables of the task are also set to true.

$$\neg s_{i,t} \vee u_{i,l} \quad i \in N^*; t \in \{es_i, ..., ls_i\}; l \in \{t, ..., t + d_i - 1\} \tag{3}$$

A set of precedence clauses is introduced to satisfy the required precedence constraints. This is done by only allowing a task to have a start variable set to true if all predecessors started early enough to be finished by that time. This clause is given in equation 4.

$$\neg s_{i,t} \bigvee_{l = es_j, ..., es_i - d_j} s_{j,l} \quad (j, i) \in A; t \in \{es_i, ..., ls_i\} \tag{4}$$

The resource clauses are defined as a pseudo-Boolean function that is converted into true CNF. The conversion is done by first building binary decision diagrams from the pseudo-Boolean function. And next, the binary decision diagrams are converted to a set of CNF clauses that represent the same pseudo-Boolean function. The process of converting pseudo-Boolean into SAT is known and researched [28]. The used pseudo-Boolean function is given in equation 5.

$$\sum_{i=1}^{n} u_{i,t} r_{i,k} \leq a_k \qquad t \in \{1, ..., T\}; k \in R \tag{5}$$

Table 1: Summary of datasets used in the experiments

Name	# inst	# tasks	subset size	# tasks in subset
DC1	1800	10 - 50	480	10 - 20
J30	480	30	480	30
RG30	1800	30	480	30

Lastly there are the objective clauses in equation 6. These are soft clauses, and the SAT solver tries to satisfy as many of these clauses as possible.

$$s_{i,t} \qquad t \in \{1, ..., T\}; i = \text{dummy finish task} \tag{6}$$

When this encoding is done the result is run on the Pumpkin MAX-SAT solver that was provided to the research by supervisor Emir Demirović.

4 Experimental Setup and Results

To test the performance of the different algorithmic approaches to solving PRCPSP-ST several experiments have been carried out. Each experiment is run on the high-performance computing cluster at the Delft University of Technology. The algorithms have access to 8GB RAM and 1 core of the Intel(R) Xeon(R) Gold 6248R CPU running at a base frequency of 3.00 GHz and a max turbo frequency of 4.00 GHz.

4.1 Project data

To test the difference between the algorithmic approaches to solve instances with activity preemption several tests are performed using three different datasets. The complete datasets contain a different amount of problem instances (# inst) and projects ranging from 10 to 50 tasks (# tasks). The J30 from the PSPLIB [29] and RG30 datasets contain projects with 30 tasks and have 480 and 1800 instances respectively and the DC1 dataset has projects ranging from 10 to 50 tasks also containing 1800 instances. For the experiments, the subset of the first 480 instances has been taken from all three datasets. This reduces the size of the projects in the DC1 dataset to a range from 10 to 20 tasks. The information about the datasets is summarized in table 1.

The network structures of the datasets have been subject to earlier research. One thing of note is the serial/parallel indicator I_2 (later renamed to SP) of the datasets [30,31]. This indicator measures how close the network structure of a project is to a complete serial or parallel network. The indicator shows that both the DC1 and J30 have a more limited SP indicator range and the RG30 has the largest range. This means that the RG30 dataset has the biggest variance in network structures for its project instances.

The setup time penalty s is set to 1, 2 and 5 time units to test the impact on the overall makespan. These values are chosen to be around .1, .2 and .5 times the length of the longer tasks in the datasets that are around 10 time units.

To solve the instances the tweaked version of the iterated greedy heuristic is used to calculate a baseline and the CNF encoding run on the Pumpkin MAX-SAT solver is used to calculate data to compare to the baseline. Each algorithm is run for 60 seconds of CPU time on each instance.

Table 2: Heuristic and SAT algorithm percentage of makespans reduced by allowing preemption

Dataset	s	%Imp by heuristic	%Imp by SAT
DC1	1	13 %	12 %
	2	4.5 %	14 %
	5	0.34 %	11 %
J30	1	3.5 %	5.8 %
	2	1.4 %	6.7 %
	5	0 %	4.3 %
RG30	1	0.84 %	1.9 %
	2	0.63 %	1.5 %
	5	0 %	1.5 %

4.2 Performance indicators

The percentage of schedules that can be reduced below the known optimal solution by allowing preemption is calculated to motivate why introducing preemption can be beneficial for certain projects. This value is then calculated by taking the percentage of instances within a dataset for a certain setup time that is lower than the known optimal makespan. Additionally, the deviation percentage of the lower makespan can indicate the amount of time saved and is also calculated.

To assess the performance two test values are calculated. First, the average percentage deviation from either the known optimal makespan or the lower bound makespan without activity preemption is calculated. This is used to check if both algorithms can come up with schedules that are reasonably close to existing solutions. The second test value is the percentage of makespans that is improved by the CNF encoding run on the Pumpkin MAX-SAT solver compared to the iterated greedy heuristic solution. This measure will show if using a CNF encoding instead of a heuristic approach without any further optimization could be used to find more optimal schedules.

During the assessment of the two test values, it must be considered that the CNF encoding solved by a SAT solver might not produce a single schedule within the given time limit. So, a percentage of instances where the SAT solver finds a solution is also calculated. This number will indicate the trade-off for finding a proven optimal solution when the time to create a schedule is limited.

4.3 Results

The experiments described have been performed and the data is collected and summarized.

Table 2 shows the percentage of instances for which the algorithms could find a lower makespan when preemption is allowed (%Imp). For the lower setup times of 1 and 2 time units, the heuristic approach could find reduced makespans in 0.8 to 13% of instances. This number drops down to almost 0% for the high setup time of 5 time units. The SAT solver approach shows a similar range of 1.9 to 13.6% for the lower setup times but this number stays more consistent at a range of 1.5 to 10.8% when the setup times are increased to 5 time units.

For the improved instances, the deviation percentage (%Dev) from the known optimal solutions is shown in table 3. When the heuristic algorithm finds an improved schedule the deviation percentage from the known optimal solution is around -3.8% for the DC1 dataset, -1.2% for the RG30 dataset and -2.5% for the J30 dataset. The SAT solver algorithm has values of -4.4%, -3.4% and -1.6% for those datasets respectively. These are the averaged values over the different setup times.

Table 3: Heuristic and SAT algorithm %Dev of optimal makespan for improved instances

Dataset	s	%Dev heuristic	%Dev SAT
DC1	1	-3.6 %	-4.8 %
	2	-3.4 %	-4.2 %
	5	-4.5 %	-4.1 %
J30	1	-2.9 %	-3.4 %
	2	-2.1 %	-3.7 %
	5	-	-3.2 %
RG30	1	-1.2 %	-3.0 %
	2	-1.2 %	-3.5 %
	5	-	-2.6 %

Table 4: Deviations from known optimal or lower bound makespan solutions

Dataset	s	%Dev heuristic	%Dev SAT
DC1	1	-0.16 %	-0.18 %
	2	-0.16 %	0.08 %
	5	0.71 %	-0.20 %
J30	1	1.9 %	1.4 %
	2	2.4 %	0.32 %
	5	2.9 %	1.83 %
RG30	1	5.3 %	12 %
	2	5.8 %	10 %
	5	5.6 %	9.5 %

Table 4 shows the average deviation percentage (%Dev) of all instances compared to the known optimal solutions. For the DC1 dataset, all deviations for both algorithms are close to 0. The results for dataset J30 show a higher deviation of around 2 to 3% for the heuristic algorithm and around 0.5 to 1.5% for the SAT solver algorithm. The higher number indicates the algorithms could not find the known optimal solutions on average. For the RG30 dataset, these numbers go up the highest at 5% for the heuristic and 10% for the SAT solver approach.

In table 5 the comparison between the results of the SAT solver compared to the heuristic algorithm is shown. For the DC1 and J30 datasets, the number of instances for which the heuristic solution could find a solution strictly better than the SAT solver is around 9% on average over the different setup times. The RG30 dataset shows that the heuristic approach found lower makespans in around 70% of the instances. These results are calculated for the instances where the SAT solver could find a solution.

Finally, an overview of the SAT algorithm performance is given in table 5. This shows that the percentage of instances for which a schedule could be found (%Satisfied) are 75.2%, 28.8% and 24.0% in the DC1, J30 and RG30 datasets respec-

Table 5: Comparison of SAT solver solutions against heuristic algorithm

Dataset	s	%Imp by SAT	%Equal
DC1	1	13 %	80 %
	2	20 %	60 %
	5	19 %	78 %
J30	1	30 %	58 %
	2	39 %	56 %
	5	38 %	55 %
RG30	1	18 %	7 %
	2	24 %	6 %
	5	27 %	8 %

Table 6: SAT algorithm performance

Dataset	%Satisfied	%Optimality proven
DC1	75.2	67.5
J30	28.8	81.9
RG30	24.0	9.0

tively. For found schedules, the SAT solver also provides if it is proven to be optimal. The percentage of proven optimal solutions (%Optimality proven) are 67.5%, 81.9% and 9.0% for the DC1, J30 and RG30 datasets respectively.

5 Responsible Research

In algorithmic optimization for NP-hard problems, the impossibility of brute-force methods on large problem instances results in researchers competing to improve the current state-of-the-art solutions. Because research is mostly focused on positive results several criteria for optimizing the state-of-the-art are often overlooked or ignored. The result is research papers that look like they indicate more optimal algorithms have been found whereas the truth of the improvement might come from a different place altogether.

When technology evolves a lot of variables involved in algorithm testing are also improving, even without directly being a part of the algorithm. Examples are newer hardware, better compilers, improved coding skills and programming language differences. If a new algorithm is tested and these variables are not kept the same or the old versions are retested with the same new variables the resulting improvement might not be a result of a more optimal algorithm. Additionally, these variables do not cancel each other out. Over time it is to be expected that all the example variables are resulting in better performance as time passes.

To compare the performance of algorithms in a fair way some rules to keep variables equal are required. In this project the following rules have been set to keep the results fairer in comparison:

- Each algorithm is implemented in the same coding language
- The algorithms are implemented by the same researcher to ensure the coding skill is consistent
- Every algorithm is run on the same hardware
- Compiler and operating system are equal for all algorithms
- As a stopping criterion the CPU time is used

To make sure these rules can also be adhered to in the future the source code is provided[1] to test the algorithms on newer hardware when it is released. Other researchers are also encouraged to implement the described algorithms for themselves and set up the experiments alongside their algorithms.

6 Discussion

The deviation percentage from the optimal or best-known solutions in table 4 is important to be discussed first. They show if the algorithms can come up with schedules that are somewhat reasonable compared to the known optimal makespans without activity preemption. When these values become too high the comparison between the two algorithms becomes less valuable. This is because existing approaches without preemption can come up with better schedules already. Looking at the algorithms produced for this research the results for the DC1 and J30 datasets are close enough to compare the approaches further. The percentage deviation from the non-preemptive makespans is within 3% for the J30 and around 0% for the DC1 dataset. This deviation is close enough to continue the comparison between the algorithms. The RG30 dataset on the other hand shows to be much harder to solve with deviations going up to 12%. This could be a result of the high variance in serial/parallel indicators of the network structure for the RG30 dataset.

As for the motivation to allow preemption the percentage of tasks that could be improved in table 2 and by how much these instances are improved in table 3 the statement made in earlier research could be confirmed. In a non-insignificant amount of cases, a reduction of makespan can be found when allowing preemption [9]. The research on this topic can provide valuable insights when the absolute lowest schedule makespan is required.

The SAT algorithm performance in table 5 and the comparison of the SAT algorithm against the heuristic algorithm 6 gives the most important insight into the value and shortcomings of a SAT solver approach. For both the DC1 and J30 datasets the SAT solver matched or even outperformed the heuristic approach in around 90% of the instances where it could find a solution in time. Finding a solution in time is also the biggest downside of using an approach aimed at finding the most optimal solution. For the DC1 dataset over 75% was solved and this is better in most cases. For the J30 dataset with less than 30% of instances getting a result within time, it is harder to decide what the better approach is, and it depends much more on the practical application. When having the absolute minimum makespan is worth the risk of not having a solution or depending on a heuristic approach the SAT solver might still be worth it. In the cases it does provide a solution over 80% is also proven to be optimal. This provides the certainty that no further effort must be spent on trying to reduce the makespan.

[1] https://gitlab.com/itstherealjasper/cse3000

Lastly a note on the impact of the setup times. The percentage of instances for which the makespan can be reduced when allowing preemption drops as the setup time increase as expected as shown in table 2. The makespan deviations for improved instances show too much variance to draw any conclusions. Earlier research already showed the difference in setup times has a much lower impact on the average makespan reduction than allowing for preemption in the first place [9].

7 Conclusions and Future Work

RCPSP has been around for some time now and a lot of research has gone into optimizing algorithms that can solve it efficiently. For the practical applications that require additional or custom constraints, an increasing number of variations of the problem have become the subject of study. In this paper, an exact approach in the form of a satisfiability encoding for the preemption extension with setup times has been proposed and tested against a heuristic solver. When tested both methods were able to produce schedules that were close to optimal non-preemptive schedules or the best-known non-preemptive schedules. After establishing both solutions are capable of producing results close to established results a comparison between the two algorithms could be made. Comparing the approaches showed benefit in the exact SAT solver method if makespan reduction is necessary enough to sacrifice intermediate solutions in case of a timeout. This was shown by the over 90% of equal or improved instance solutions from the SAT solver over the heuristic solutions when the SAT solver found a solution in time for the J30 dataset.

Unfortunately, the results for the RG30 dataset are not close enough to the best-known solutions. Especially the SAT solver struggled to come up with solutions and when it did the results were still unimpressive. A possible explanation is a high variance in how serial/parallel the network structure of the instances is. This could not be confirmed within the time for this research though and could be a start for a follow-up study. Because the heuristic approach also showed worse results on the RG30 dataset it can at least be concluded that some factor other than project size causes the problems to be harder when preemption is allowed.

Future research could focus on comparing different SAT solvers once the problems are encoded into satisfiability clauses. Solvers are often optimized for different objectives, and some might find an intermediate solution quicker or go for the optimal solution directly. A second suggestion is converting the proposed SAT encoding into a more generalized satisfiability modulo theories encoding. This encoding has native support for formulas including integers and makes encoding the resource constraints more straightforward. Another great way to build on the work done in this paper is changing the Pumpkin MAX-SAT solver to be more specialized on the specific problem variation. The size of the CNF encodings is big because of the extended network after each task has been expanded to all its possible subsegments. An example would be to provide the solver with an initial solution created by the first iteration of the heuristic algorithm as a starting point. At the very least an implementation in this way will always produce a result when the encoding can be made within time.

Allowing preemption for project tasks seems to be able to provide lower makespan schedules. If future research can find the optimal solutions within the time limit with higher consistency it seems like it will be well worth the effort.

References

[1] W. S. Herroelen, "Resource-constrained project scheduling — the state of the art," *Journal of the Operational Research Society*, vol. 23, no. 3, pp. 261–275, 1972. [Online]. Available: https://doi.org/10.1057/jors.1972.48

[2] E. W. Davis, "Resource allocation in project network models-a survey," *Journal of Industrial Engineering*, vol. 17, no. 4, pp. 177–188, 1966.

[3] ——, "Project scheduling under resource constraints—historical review and categorization of procedures," *A I I E Transactions*, vol. 5, no. 4, pp. 297–313, 1973. [Online]. Available: https://doi.org/10.1080/05695557308974916

[4] W. S. Herroelen, E. L. Demeulemeester, and B. De Reyck, *A Classification Scheme for Project Scheduling*. Boston, MA: Springer US, 1999, pp. 1–26. [Online]. Available: https://doi.org/10.1007/978-1-4615-5533-9_1

[5] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European Journal of Operational Research*, vol. 112, no. 1, pp. 3–41, 1999. [Online]. Available: https://doi.org/10.1016/S0377-2217(98)00204-5

[6] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, no. 1, pp. 1–14, 2010. [Online]. Available: https://doi.org/10.1016/j.ejor.2009.11.005

[7] E. L. Demeulemeester and W. S. Herroelen, "An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 90, no. 2, pp. 334–348, 1996. [Online]. Available: https://doi.org/10.1016/0377-2217(95)00358-4

[8] L. Kaplan, "Resource-constrained project scheduling with setup times," 1991.

[9] M. Vanhoucke and J. Coelho, "Resource-constrained project scheduling with activity splitting and setup times," *Computers & Operations Research*, vol. 109, pp. 230–249, 2019. [Online]. Available: https://doi.org/10.1016/j.cor.2019.05.004

[10] J. Coelho and M. Vanhoucke, "Multi-mode resource-constrained project scheduling using rcpsp and sat solvers," *European Journal of Operational Research*, vol. 213, no. 1, pp. 73–82, 2011. [Online]. Available: https://doi.org/10.1016/j.ejor.2011.03.019

[11] D. Debels and M. Vanhoucke, "A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem," *Operations Research*, vol. 55, no. 3, pp. 457–469, 2007. [Online]. Available: https://doi.org/10.1287/opre.1060.0358

[12] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Research Logistics (NRL)*, vol. 45, no. 7, pp. 733–750, 1998. [Online]. Available: https://doi.org/10.1002/(SICI)1520-6750(199810)45:7⟨733::AID-NAV5⟩3.0.CO;2-C

[13] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Scheduling subject to resource constraints: classification and complexity," *Discret. Appl. Math.*, vol. 5, pp. 11–24, 1983. [Online]. Available: https://doi.org/10.1016/0166-218X(83)90012-4

[14] S. Hartmann and D. Briskorn, "An updated survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 297, no. 1, pp. 1–14, 2022. [Online]. Available: https://doi.org/10.1016/j.ejor.2021.05.004

[15] N. Nudtasomboon and S. U. Randhawa, "Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs," *Computers & Industrial Engineering*, vol. 32, no. 1, pp. 227–242, 1997. [Online]. Available: https://doi.org/10.1016/S0360-8352(96)00212-4

[16] L. Bianco, M. Caramia, and P. Dell'Olmo, *Solving a Preemptive Project Scheduling Problem with Coloring Techniques*. Boston, MA: Springer US, 1999, pp. 135–145. [Online]. Available: https://doi.org/10.1007/978-1-4615-5533-9_6

[17] J. Damay, A. Quilliot, and E. Sanlaville, "Linear programming based algorithms for preemptive and non-preemptive rcpsp," *European Journal of Operational Research*, vol. 182, no. 3, pp. 1012–1022, 2007. [Online]. Available: https://doi.org/10.1016/j.ejor.2006.09.052

[18] F. Ballestín, V. Valls, and S. Quintanilla, "Preemption in resource-constrained project scheduling," *European Journal of Operational Research*, vol. 189, no. 3, pp. 1136–1152, 2008. [Online]. Available: https://doi.org/10.1016/j.ejor.2006.07.052

[19] M. Vanhoucke, "Setup times and fast tracking in resource-constrained project scheduling," *Computers & Industrial Engineering*, vol. 54, no. 4, pp. 1062–1070, 2008. [Online]. Available: https://doi.org/10.1016/j.cie.2007.11.008

[20] V. Van Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, 2010. [Online]. Available: https://doi.org/10.1016/j.ejor.2009.03.034

[21] J. Buddhakulsomsiri and D. S. Kim, "Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting," *European Journal of Operational Research*, vol. 178, no. 2, pp. 374–390, 2007. [Online]. Available: https://doi.org/10.1016/j.ejor.2006.02.010

[22] ——, "Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting," *European Journal of Operational Research*, vol. 175, no. 1, pp. 279–295, 2006. [Online]. Available: https://doi.org/10.1016/j.ejor.2005.04.030

[23] M. Mika, G. Waligóra, and J. Weglarz, *Modelling Setup Times in Project Scheduling*. Boston, MA: Springer US, 2006, pp. 131–163. [Online]. Available: https://doi.org/10.1007/978-0-387-33768-5_6

[24] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, 2007. [Online]. Available: https://doi.org/10.1016/j.ejor.2005.12.009

[25] J. E. Kelley, "The critical-path method : Resource planning and scheduling," *Industrial Scheduling*, 1963. [Online]. Available: https://cir.nii.ac.jp/crid/1573105975369889920

[26] A. Horbach, "A boolean satisfiability approach to the resource-constrained project scheduling problem," *Annals of Operations Research*, vol. 181, no. 1, pp. 89–107, 2010. [Online]. Available: https://doi.org/10.1007/s10479-010-0693-2

[27] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, 1st ed. MIT press, 1990.

[28] N. Eén and N. Sörensson, "Translating pseudo-boolean constraints into sat," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006. [Online]. Available: https://doi.org/10.3233/SAT190014

[29] R. Kolisch, A. Sprecher, and A. Drexl, "Characterization and generation of a general class of resource-constrained project scheduling problems," *Management Science*, vol. 41, no. 10, p. 1693, 1995, copyright - Copyright Institute for Operations Research and the Management Sciences Oct 1995 CODEN - MNSCDI. [Online]. Available: https://doi.org/10.1287/mnsc.41.10.1693

[30] L. Valadares Tavares, J. Antunes Ferreira, and J. Silva Coelho, "The risk of delay of a project in terms of the morphology of its network," *European Journal of Operational Research*, vol. 119, no. 2, pp. 510–537, 1999. [Online]. Available: https://doi.org/10.1016/S0377-2217(99)00150-2

[31] M. Vanhoucke, J. Coelho, D. Debels, B. Maenhout, and L. V. Tavares, "An evaluation of the adequacy of project network generators with systematically sampled networks," *European Journal of Operational Research*,

vol. 187, no. 2, pp. 511–524, 2008. [Online]. Available: https://doi.org/10.1016/j.ejor.2007.03.032