Resource-constraint project scheduling with task preemption and setup times by heuristically augmented SAT solver

Jasper Vermeulen[1]
Supervisor: Emir Demirović[1]
[1]EEMCS, Delft University of Technology, The Netherlands
J.Vermeulen-1@student.tudelft.nl

Abstract

This will only be written once all experiments have done and the conclusion is on paper.

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 overal 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). The standard RCPSP has a set of tasks that each require a specified resource for a given amount of time and precedence restrictions are given for the tasks. On its own, this problem definition for standard RCPSP is limited and of little use to realistic applications where additional constraints must be followed or more options are available. To make sure the researched algorithms solving the scheduling problem would have a wider use case, many variations and extensions to the problem definition have been classified over time [1], [2]. More recently, the variations and extensions have also been surveyed and put into a structured overview [3].

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 an task to discourage endless splits resulting in a chaotic schedule. Both a model for allowing preemption [4] and including setup times [5] 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 [6]. Within this algorithm, the activities are split into all possible integer time segments and a SAT solver makes a selection from these segments [7]. The resulting list was used to construct a schedule with a genetic algorithm established in earlier research [8].

In the research done on solving RCPSP variants, the focus has been on heuristic and meta-heuristic algorithms. These algorithms are usually variants of branch-and-bound algorithm [4] or a form of genetic algorithms [9] that were established for the standard RCPSP.

SAT solvers are a very general tool they can be used on any algorithmic problem as long as it is encoded as the required input for the solver. SAT solvers have been used as a part of the algorithm but using a SAT solver as a complete solution to try and solve PRCPSP-ST instances has not been researched before. Heuristic algorithms by comparison are specialised by knowledge or an insight into the problem variation translated into a heuristic rule for the algorithm to use. Because the RCPSP is known to be strongly NP-hard [10], the SAT solver might not be more efficient than the heuristic and meta-heuristic at first. But, a SAT solver can also be modified to include a heuristic for the PRCPSP-ST variant. In a way, the addition of a heuristic can be seen as an augmentation of the SAT solver. The augmentation is making the solver more specialised for the PRCPSP-ST variant and could result in a better performance than other algorithms.

Because there is room to try and find out if a SAT solver can outperform the heuristic and meta-heuristic algorithms the main research question is: *Can the addition of a simple heuristic to a SAT solver algorithm used to solve to PRCPSPST models reduce the average makespan of the resulting schedule when run for an equal amount time?*. The expectation for this research is to first show that a SAT solver can be used to solve PRCPSP-ST instances. And next, show that the heuristically augmented version of the SAT solver algorithm will result in a lower makespan than a heuristic algorithm when running an equal amount of time.

Report section structure goes here

2 Related Work

3 Problem Formulation

The resource-constrained project scheduling problem is a strongly NP-hard algorithmic problem [10] with the objective is to minimize makespan (overall required time to finish all tasks).

3.1 RCPSP

RCPSP is about a project consisting of a set of tasks N which all have to completed to finish the project. Each task i has a duration d_i and a requires an amount $r_{i,k}$ of a resource type k. A project provides a set of limited resources 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 difined as the starting time of the finish dummy task.

3.2 PRCPSP-ST

The RCPSP definition can be extended in different ways including task preemption and setup times [3], [11].

Preemption allows and 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. In reality tasks might be preempted at any fraction of the duration but the infinite ways to split a task makes an algorithm much harder to define. 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.

Setup time s is introduced as a way to try and prevent task 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 resource 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.

4 Heuristics and augmentation of SAT solvers

To evaluate the performance of a heuristically augmented SAT solver on PRCPSP-ST instances comparative data needs to be generated. To make sure the variables between algorithms are limited they were all designed, implemented and run on the same hardware for this research. This way the at least coding skills and hardware improvements over time are not introduced in the data. Improved performance of new algorithms only as a result of the use of more recent hardware has shown up in the past as described in section 6. Three algorithms are used to solve the same problem instances for an equal amount of time: a heuristic algorithm, SAT solver, heuristically augmented SAT solver.

The heuristic algorithm is an adapted version of the iterated greedy algorithm [12]. 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 has to be encoded into conjunctive normal form boolean logic. When the encoding is made any SAT solver is able to provide feasible schedules. Because there is a clear objective to reduce the makespan a more advanced MAX-SAT solver is used. This will create feasible schedules while also trying to optimize to an objective function.

To augment a SAT solver for a specific problem the code of the solver has to be changed. In the case of this research a heuristic function will be added that improves the selection of a possible variable. The heuristic function will use knowledge about PRCPSP-ST to try and select a variable that will reduce the resulting schedule makespan.

4.1 Heuristic

As a heuristic solution a tweaked version of the iterated greedy algorithm is implemented [12]. This algorithm requires an activity list representation of the project. It start with a setup of an initial schedule and then iterates over a destruction phase and a construction phase until a time limit or number of iterations limit is reached.

An activity list representation allows a serial generation scheme to construct a feasible schedule [13]. The activity list represent a project as 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 task 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 resource availability. 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 on it 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. 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 any number of iterations of the destruction and construction phases until either a iterations limit is reached or a time limit is reached. At that point it will return the most optimal schedule it has found.

4.2 CNF Encoding

The conjunctive normal form encoding used for this research is based on existing work used to solve the RCPSP with SAT [14]. This encoding was altered to include preemption.

For the new encoding a project has to 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 short summary will be given [6]. 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 must also have the original successors of the task.

With the extended network the earliest es_i and its latest ls_i start times, its ealiest ef_i and its latest lf_i finish times are calculated using the critical-part method by the Floyd-Warshall algorithm [15]. 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 amount of soft clauses it can satisfy.

Completion clauses make sure that each task segment is processed once and therefore making sure that all the work in the project is done. New subsets are required to define the completion clauses. $C^*_{i,l} \subseteq N^*$ 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 pseudo-boolean function that are converted into true CNF. The conversion is done by first building binary decision diagrams from the pseudo-boolean function. 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 [16]. The used pseudo-boolean function is given in equation 5.

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

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

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

4.3 SAT solver

Due to difficulties during the CNF encoding my current planning does not allow for any time spent on improving the SAT solver.

5 Experimental Setup and Results

In order to test the performance of the different algorithmic approaches to solving PRCPSP-ST a number of 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 max turbo frequency of 4.00 GHz.

5.1 Project data

To test the difference between the algorithmic approaches to solve instances with activity preemption a number of tests are performed using three different datasets. The complete datasets contain a different amount of problem instances (# inst) and project ranging from 10 to 50 tasks (# tasks). The J30 and RG30 datasets contain projects with 30 tasks and have 480 and 1800 instances respectively and the DC1 dataset

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

has project ranging from 10 to 50 tasks also containing 1800 instances. For the experiments the a subset of the first 480 instances have been taken of 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 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.

5.2 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 the 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 give an idication of the amount of time saved and is also calculated.

To assess the performance two test values are calculated. First the average percentage deviation from the 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 endocing 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 has to be taken into account that the CNF encoding solved by a SAT solver might not produce a single schedule in the given time limit. So a percentage of instances where the SAT solver finds a solution is also calculated. This number will indicate the tradeoff for finding a proven optimal solution when time to create a schedule is limited.

5.3 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 reduces 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.

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 %

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 %

For the improved instances the deviation percentage (%Dev) from the known optimal solutions are 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 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 shows a higher deviation of around 2 to 3% for the heuristic algorithm and around 0.5 to 1.5% for the SAT solve 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 solve 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

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 5: Deviations from known optimal or lower bound makespan solutions

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 %

SAT solver is aroung 9% on average over the different setup times. The RG30 dataset shows that 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 schedules could be found (%Satisfied) are 75.2%, 28.8% and 24.0% in the DC1, J30 and RG30 datasets respectively. 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.

6 Responsible Research

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

Table 6: SAT algorithm performance

Dataset	%Satisfied	%Optimality proven	%Improved
DC1	75.2	67.5	20.6
J30	28.8	81.9	35.9
RG30	24.0	9.0	22.8

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 when 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 more fair 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 to test the algorithms on newer hardware when it is released. Other researchers are also encouraged to implement the described algorithms for themselves and setup the experiments alongside their own algorithms.

7 Discussion

8 Conclusions and Future Work

References

[1] W. Herroelen, E. Demeulemeester, and B. De Reyck, *A Classification Scheme for Project Scheduling*, pp. 1–26. Boston, MA: Springer US, 1999.

[2] 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.

[3] 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.

[4] E. Demeulemeester and W. 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.

[5] L. Kaplan, “Resource-constrained project scheduling with setup times.” 1991.

[6] M. Vanhoucke and J. Coelho, “Resource-constrained project scheduling with activity splitting and setup times,” *Computers & Operations Research*, vol. 109, pp. 230–249, 2019.

[7] 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.

[8] 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.

[9] S. Hartmann, “A competitive genetic algorithm for resource-constrained project scheduling,” *Naval Research Logistics (NRL)*, vol. 45, no. 7, pp. 733–750, 1998.

[10] 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.

[11] 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.

[12] 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.

[13] J. E. Kelley, “The critical-path method : Resource planning and scheduling,” *Industrial Scheduling*, 1963.

[14] 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.

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

[16] N. Eén and N. Sörensson, “Translating pseudo-boolean constraints into sat,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.