# 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

TBD in final 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 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).

### 2.1 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 defines precedence relations $A$. 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 $k \in R$ and no single task will require more of a resource type than provided $r_{i,k} \leq a_k$.

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.

### 2.2 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 times are 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.

## 3 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 5. 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.

### 3.1 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.

### 3.2 CNF Encoding

### 3.3 SAT solver

## 4 Experimental Setup and Results

## 5 Responsible Research

## 6 Discussion

## 7 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.