Combining SAT solvers with heuristic ideas for solving RCPSP with logical constraints

An exploration of variable ordering heuristics impact on solving RCPSP-log

Iarina Maria Tudor[1]

Supervisor(s): Emir Demirovic[1]**, Konstantin Sidorov**[1]**, Maarten Flippo**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Iarina Maria Tudor
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirovic, Konstantin Sidorov, Maarten Flippo, Jeremie Decouchant

An electronic version of this thesis is available at http://repository.tudelft.nl/.
Part of this research was assisted by ChatGPT.

Abstract

This paper provides a novel method of solving the resource-constrained project scheduling problem (RCPSP) with logical constraints (RCPSP-log) using satisfiability (SAT) solving and integrating variable selection heuristics. The extension provides two additional precedences: OR constraints and bidirectional (BI) relations, making it possible to express more complex dependencies between tasks. OR constraints enable a task to be dependent on multiple preceding tasks, while BI relations do not allow tasks to be executed at the same time. Both problems are known to be NP-hard.

The solution method consists of using a max satisfiability (MaxSAT) solver combined with variable selection heuristics. The paper investigates two heuristics, based on the greedy approach of scheduling each activity as early as possible and variable conflict analysis.

The results on well-known datasets from the literature show that most of the instances can be solved within the designated time limit. The proposed approaches contribute to reducing the makespan, especially when dealing with no logical constraints or only OR relations. For BI constraints, the algorithm is having difficulties in finding solutions, especially when increasing the number of activities, reporting fewer solutions than the baselines. Overall, the results provide insight into the integration of variable selection heuristics for SAT solving, with the potential for more investigation into problem-specific ideas.

1 Introduction

Project scheduling is a complex issue that affects many sectors. For example, the time to perform all sub-tasks has a significant impact on the efficiency of production operations and logistical timetables [5]. As a result, many academics are seeking efficient algorithms that can identify optimal schedules in a fair period of time.

RCPSP stands for Resource-Constrained Project Scheduling Problem, which is a general problem dealing with scheduling a set of activities by determining their start and end times to minimize the total duration, subject to constraints on the availability of resources. Activities are related to each other by a set of precedence constraints, which specify that certain activities cannot start before others are completed. Furthermore, there are constraints on the availability of resources. Each activity requires a specific amount of resources, which are limited and cannot be used simultaneously by multiple activities. The availability of resources can be represented by a set of resource constraints, which specify the maximum amount of each resource that can be used at any given time [2]. RCPSP assigns a start and finish time to each activity, considering its duration and precedence relationships, while aiming to minimize the total makespan. The Resource-Constrained Project Scheduling Problem with logical constraints, known as RCPSP-log, extends the traditional RCPSP by incorporating logical constraints such as OR and BI constraints [15]. By capturing additional precedence rules, this formulation offers a more accurate way of modeling complex relationships and dependencies between project tasks [15]. However, finding optimal solutions efficiently to the above-mentioned problems is very difficult due to their NP-hard nature [14], making them the perfect candidate for further research. This paper is focusing on the Resource-Constrained Project Scheduling Problem with logical constraints.

RCPSP-log has been the subject of an investigation by various researchers in the past, such as Coelho and Vanhouckel [3] and Hartmann and Kolisch [7]. Their work focused on developing specific meta-heuristics and logical constraint modeling techniques. For RCPSP-log, their proposed solution involves a satisfiability solver (SAT) combined with a meta-heuristic genetic algorithm, which is complex and very problem specific. Considering this, there has been little research done on simple heuristic approaches for this problem, as it is unlikely to outperform state-of-the-art methods, namely problem-specific algorithms such as genetic and evolutionary algorithms. Nonetheless, exploring these possibilities remains worth considering, given the consistent and ongoing improvements being made to SAT solvers [8]. Moreover, focusing on a fast and reliable heuristic approach could potentially solve multiple problems instead of having to develop extensive, specific algorithms for each particular case.

This paper aims to answer the research question of *"How can the integration of heuristic ideas into SAT solvers enhance the solution process of RCPSP with logical constraints such as OR, AND, and BI constraints?"*. The study looks into the performance of satisfiability solving enhanced with variable selection heuristics for logical precedences. The ultimate goal is to discover novel optimizations that can advance the state-of-the-art solutions for the RCPSP-log.

The research uses a method to encode RCPSP-log instances into propositional logic so that SAT approaches, specifically MaxSAT solvers, can be used in combination with variable selection heuristics to find optimal solutions. The heuristic approaches originate from the greedy perspective of scheduling activities as soon as possible. The method assigns initial weights based on time information to determine the order of truth value selection. The paper experiments with the greedy heuristic as a stand-alone as well as combined with a variable state independent decaying sum (VSIDS), a known good heuristic for SAT solving [11], based on conflict analysis. The evaluation is done using the datasets from PSPLIB [10], considering instances of 30 and 60 jobs.

The results found are promising, showing improvements in different aspects of the solution process. The findings offer perspective on solving more optimal solutions compared to the conventional SAT approach, especially when increasing the number of instances. The heuristic methods find lower average best makespans over time but struggle to quickly evaluate the optimality status and to go from one solution to another, and they perform worse in terms of the total number of solutions found within the time limit. Experiments on more instances or datasets with a higher number of activities could help draw further conclusions. Another finding is that the value selection starting with true assignments contributes to decreasing the number of decisions by almost half.

The structure of this paper is as follows: section 2 provides a formal definition of the RCPSP and the RCPSP-Log followed by an illustrative example. Next, Section 3 provides information on the heuristics ideas and SAT encoding. Section 4 states the experimental setup and presents the results, alongside a discussion of the results. Furthermore, section 5 reflects on the ethical implications of this research together with the repeatability of the setup and results. Finally, section 6 covers the conclusion and future work.

2 Problem Formulation

The Resource Constrained Project Scheduling Problem aims to find a feasible assignment of a set of activities such that the makespan is minimized. The makespan represents the finish time of the end activity, which can only be scheduled once all other activities have finished. RCPSP can be stated as follows, as formulated in [3]:

- a project is composed of a set of activities N and a set of resource types R, with each resource $k \in R$ having a constant availability a_k per period.
- for each $i \in N$ there is a defined duration d_i and a required amount of resources $r_{i,k}$ representing how much of a resource type is needed.
- the project network is represented by a topologically ordered activity-on-the-node (AoN) format where A is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists.
- a schedule S is defined by a vector of activity start times and is said to be feasible if all precedence and renewable resource constraints are satisfied, as defined in [3].
- the precedence constraints for most of the formulations of RCPSP are considered to be AND constraints which allow starting an activity if and only if all of its predecessors have ended.

An example of an instance of this problem is visualized in Figure 1. The project network contains 2 dummy and 6 non-dummy activities with 2 types of resources. The arrows represent the AND constraints.

2.1 Resource Constrained Project Scheduling Problem with Logical Constraints (RCPSP-log)

The Resource Constrained Project Scheduling Problem with Logical Constraints is an extension of the classical RCPSP, adding 2 new logical constraints: OR and BI constraints.

- this paper considers that the OR constraint imposes that only one predecessor of an activity needs to be finished before being able to schedule the successor activity.
- the BI constraint is defined as the constraint enforcing two activities to not be executed in parallel. In addition, it introduces the concept of changeover time, which refers to the minimum time required between the completion of the first activity and the start of the second activity to transfer and potentially prepare the resources for the subsequent activity, as described in [3].

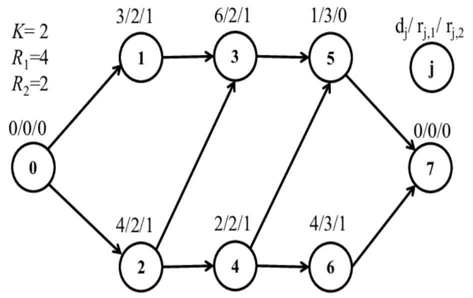

Figure 1: An example project network for RCPSP instance with 6 activities and 2 different resources, found in [9].

2.2 Illustrative example

An instance of the RCPSP-log encoded in a project network in activity-on-the-node format with 8 activities and one renewable resource with a limited availability of 4 units can be visualized in Figure 2. The start and end nodes are dummy activities that mark the start and end of the project, without requiring any resources or duration. They are included in the project network to ensure that all activities are properly linked together and that the project is complete. The logical constraints are displayed in the following manner: AND precedence is done using the classic arcs, OR precedence is indicated using dotted arcs and BI constraints are noted with a bi-directional dotted arc.

The examples has OR constraints between:

- activities A and B are the OR predecessors of C

BI constraints between:

- activities D and E cannot be done in parallel

The optimal schedule is having a makespan of 12 time units.

3 Satisfiablity solving preliminaries

Satisfiability (SAT) solvers have been used to successfully solve the resource-constrained project scheduling problem and variations of it [3]. Therefore, this paper looks into a SAT approach refined with variable selection heuristics for RCPSP with logical constraints.

3.1 SAT and MaxSAT solvers

The method consists of encoding the instances of RCPSP-log into propositional logic in conjunctive normal form (CNF), which further becomes the input for the solving algorithm. More specifically, the solution is found using a SAT solver, a tool that takes the encoded input and assigns values to the boolean literals considered variables by satisfying, if possible, all clauses involved [6]. The algorithm exhaustively explores the search space, represented by the literals and their values, to check which combinations satisfy the given CNF

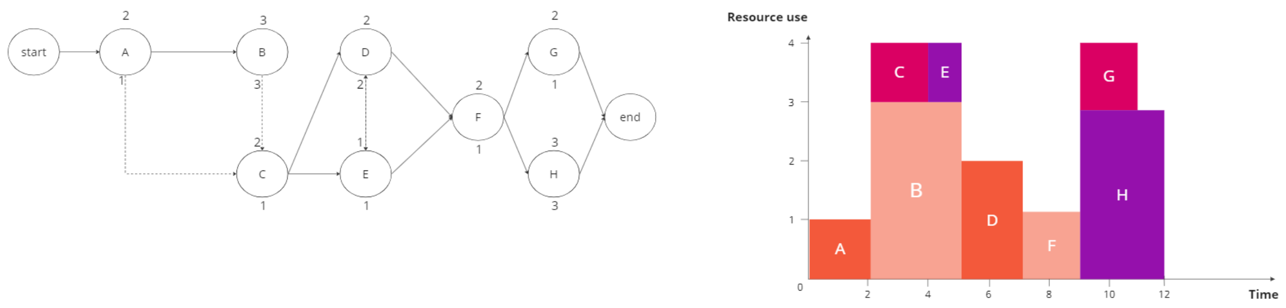

Figure 2: An example project network and the optimal schedule

formula. The exploration is based on a heuristic, which influences the order of the variables and prunes the search tree.

The MaxSAT solver is a variation of SAT solvers that considers additional soft clauses to represent the objective function, wherein each clause is assigned a cost to pay in case of not satisfying it [6]. The aim of the solver is to find an assignment that minimizes the total cost. The algorithm tries to iteratively search for a better assignment until no improvements can be made or time runs out. The results generated by this particular solver are the variable assignment, the optimality of the assignment, and the penalty incurred by violating the soft clauses.

3.2 CNF Encoding of the RCPSP-log problem

A propositional logic formulation of the problem is required to identify an optimal solution utilizing the SAT technique supplemented with variable selection heuristics. The model expresses the RCPSP-log instance in CNF.

The time-indexed formulation for the RCPSP problem using the binary variable $x_{i,t}$ equal to 1 if activity i starts at time t, and 0 otherwise as proposed by Pritsker, Watters, Wolfe in 1969 [16] is transformed into CNF as adapted from [4] and [18] as follows:

The literal $x_{i,t}$ is used to express the constraints for completion, precedence, consistency, and resource clauses.

$$x_{i,t} = \begin{cases} 1, & \text{if activity i starts at time t} \\ 0, & \text{otherwise} \end{cases}$$

Completion Clauses

A completion clause S_i forces an activity to be scheduled early enough to finish before reaching the horizon, meaning that it should be scheduled at least d_i units of time before the deadline. It can be stated as a disjunction of literals $x_{i,t}$ where $t \in \{0....T - d_i\}$ and d_i is the duration of activity i:

$$S_i = \vee_{t=0}^{T-d_i} x_{i,t}$$

Furthermore, each activity can have only one starting time. The clauses F_i are formulated as pseudo-boolean constraints, which are transformed into CNF form using PySAT [1] cardinality constraints. This constraint is an adaptation from the original work of Melle [18], which was allowing multiple start times. It can be stated as follows:

$$F_i = \sum_{i=1}^{n} x_{i,t} = 1 \; \forall t \in \{0....T - d_i\}$$

Resource Clauses

As known from the time-indexed formulation [16], the problem requires a resource constraint stating that at each moment in time, for each type of resource, the capacity is not exceeded. In order to create the resource clauses for this constraint, a new literal has been introduced, where $y_{i,t}$ represents if activity i is active at time $t \in \{0....T - d_i\}$ as follows:

$$y_{i,t} = \begin{cases} 1, & \text{if activity i is active at time t} \\ 0, & \text{otherwise} \end{cases}$$

Taking everything into consideration, new consistency clauses C_i can be derived to ensure that the literal $x_{i,t}$ for activity i is consistent with the time when i is considered active:

$$C_i = \wedge_{t=0}^{T-d_i} \wedge_{u=t}^{t+d_i-1} \neg x_{i,t} \vee y_{i,u}$$

The resource constraint is formulated as a pseudo-boolean constraint, which is further transformed into CNF using PySAT library and the Binary Decision Diagram (BDD) method [1]. No other encoding methods were tested. The constraint for each resource type k, where $r_{i,k}$ is the amount of resource per type needed by activity i and a_k is the maximum availability of that resource type, is stated as:

$$R_{k,t} = \sum_{i=1}^{n} y_{i,t} \dot{r}_{i,k} \leq a_k$$

Precedence Clauses

Precedence clauses ensure that the precedence constraints are satisfied, namely the OR, AND and BI constraints. For this purpose, a new literal $z_{i,t}$ captures if activity i is finished by time t.

$$z_{i,t} = \vee_{u=0}^{t-d_i} x_{i,u} \; \forall t \in \{0...T\}$$

The AND constraint is defined as for each pair of activities $(i, j) \in A^{AND}$ $z_{i,t} = 1$ must hold. This can be formulated into a precedence clause $P_{i,j}^{AND}$ as follows:

[1]PYSAT: A Python package for satellite data analysis. Available at https://github.com/pysat/pysat.

$$P_{i,j}^{AND} = \wedge_{t=0}^{T-d_j} \neg x_{j,t} \vee z_{i,t}$$

For OR constraints, only for one predecessor i where $(i, j) \in A^{OR}$ $z_{i,t} = 1$ must hold. To capture this relation, the precedence clause $P_{i,j}^{OR}$ is constructed as follows, where n represents the number of predecessors of j:

$$P_{i,j}^{OR} = \wedge_{t=0}^{T-d_j} \neg x_{j,t} \vee_{n=1}^{|n|} z_{i_n,t}$$

The last logical constraint BI forces $(i, j) \in A^{BI}$ to not be active at the same time. For this research, the changeover time defined in Section 2.1 is considered 0 for simplicity. All past work experiments for RCPSP-log were also considered 0 ([3], [18]). The precedence clause $P_{i,j}^{BI}$ is stated as:

$$P_{i,j}^{BI} = \wedge_{t=0}^{T} \neg y_{i,t} \vee \neg y_{j,t}$$

Finally, all clauses are combined together in conjunction for the CNF form.

Soft Clauses

The MaxSAT solver accepts as input additional soft clauses that can be violated at a cost. The soft clauses W of this problem represent the total makespan, which we aim to minimize and have a cost of 1 for each violation. It is formulated as:

$$W = \sum_{t=0}^{T} y_{n+1,t} \times 1$$

The final CNF clause is represented by the conjunction of all hard and soft clauses.

4 Heuristic Solution Method

Variable selection heuristics are used to guide the search of the solver to prune the search space. The rules for ordering help find feasible solutions in fewer iterations, focusing on promising variables first. This paper is investigating two approaches based on the intuition of minimizing the makespan by trying to schedule each activity closer to the start of the project and variable activity and conflict analysis, known as good practice in SAT solving.

4.1 Exploration of greedy ideas

Considering the limited research for domain-specific knowledge for RCPSP-log, no insightful heuristics were found in the literature. Therefore, scheduling each activity as early as possible (EST), which is a very basic and intuitive approach, serves as an entry point for further research. For this heuristic, each variable $x_{i,t}$, representing the start time, has an initial weight depending on time t. The score is computed using the formula:

$$weight = \frac{1}{t+1} * 1000 \, \forall t \in \{0...T\}$$

The weights determine the fixed order in which the variables are explored since the selection is made using a maximum heap. All other variables have a weight of 0, which indicated that they should be assigned after all others, in any specific order. On top of that, since the goal is to find a start time closer to the lower bound, the heuristic changes the value ordering of the baseline, starting with a true assignment.

4.2 EST + VSIDS

The greedy heuristic was also tested in combination with the Variable State Independent Decaying Sum (VSIDS) [11] used by default by the solver. This heuristic is based on how active a variable is, which is encoded in a score. Moreover, the algorithm incorporates a decaying mechanism that takes into consideration how long ago a variable was active. The main assumption behind VSIDS is that an active variable will lead to conflicts faster and will influence the outcome. Therefore, the heuristics prioritize high-scoring variables [11]. For the algorithm proposed, the VSIDS scores are initialized with the EST weights, guiding the start of the search. After each step, the scores are modified with respect to the number of conflicts, following the variable-state independent decaying sum decaying mechanism. Considering the contribution of VSIDS in SAT solving, the EST + VSIDS approach could incorporate more problem-specific knowledge and keep the advantages of the best-practice method.

5 Experimental Setup and Results

The experiments were done using RCPSP datasets with 30 and 60 jobs from PSPLIB [10], which were transformed to accommodate RCPSP-log, as discussed in Section 5.1. The parser and encoder algorithms were implemented in Python 3.9 using the PySAT library, and run on an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz with 16GB RAM. The output was run on the pumpkin MaxSAT solver, which was provided by the supervising team. The heuristic ideas were implemented in Rust on top of the pumpkin MaxSAT solver [2]. The evaluation is considering the run time of the solver in seconds, without the time to process and encode the instances. The difference in the number of instances tested per number of jobs is related to the computational and memory limitations of the computer performing the tests. The number of instances per dataset is visualized in Table 1.

Table 1: Number of instances per data set and type of logical constraint, for all frequencies.

	-	OR	BI
J30	229	229	229
J60	70	70	70

5.1 RCPSP-Log instances for experimentation

Coelho and Vanhouckel [3] proposed a method for augmenting the single-mode RCPSP instances with 30, and 60 jobs from PSPLIB [10]. This research is using the same method. Two new variables k1 and k2 are added to control the amount of AND constraints transformed into OR and BI constraints. The OR precedence relations are generated using the formula $(i + k1) \mod k2 = 0$ in which i will become the activity involved in an OR with its predecessors. The selection of BI constraints is using the same formula considering i one of the involved activities and its immediate predecessor with the

[2]pumpkin MaxSAT solver: private satisfiability solver developed by the algorithmic department from TU Delft.

lowest activity number. The modified data set using the aforementioned method was also used in past work ([3], [18]), making the results of this research comparable with other contributions.

5.2 Experimental Results

The heuristic earliest start time first (EST) was evaluated alone and combined with VSIDS against the benchmark performance for each type of logical constraint with different frequencies and a timeout of 15 seconds. The parameters used for experiments are k1 and k2, which control the generation of logical constraints; Percent Log which represents the actual percentage of these constraints in the instance. The number of solutions found is presented using the parameter *solutions*, which illustrates the optimal, satisfiable, and unknown solutions found within the time limit of 15s. Moreover, *time* represents the average wall-time, in seconds, for finding the optimal solution. The average includes the optimal solution times together with the timeouts for satisfactory or unknown. The distance between CPU and wall-time [3] was very minimal, so only the latter is included. The last parameter is *decisions*, representing the average number of decisions taken by the solver until the final solution is reached, rounded to thousands. The calculation considered only the instances where an optimal or satisfiable solution was found. Furthermore, the average best makespan was computed for j30 for no logical constraints, 100% OR, and BI relations.

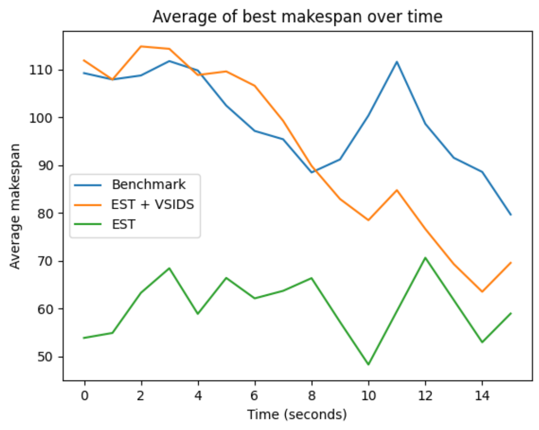

Figure 3: Average of best makespan over time (seconds) for j30 instances with no additional logical constraints

5.3 Discussion and interpretation of outcomes

In general, the heuristic approaches showed a potential impact on the solving process of the RCPSP-log compared to the baselines. Table 2 is illustrating the performance of the benchmark, Table 3 represents the EST + VSIDS and Table 4 shows the findings for the EST heuristic.

The results for instances with 30 activities were better for EST + VSIDS in terms of the number of optimal solutions

[3]Wall-time or wall-clock time: the elapsed time or the time a program takes to run from start to finish.

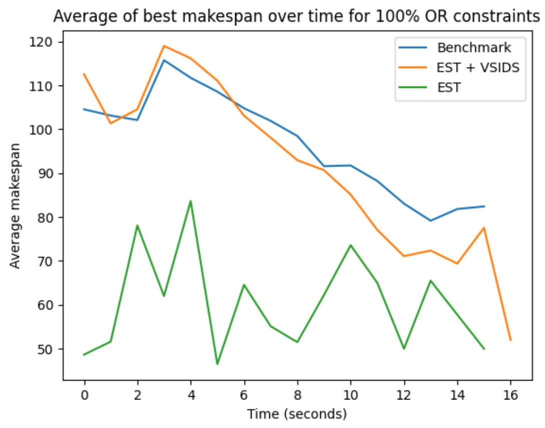

Figure 4: Average of best makespan over time (seconds) for j30 instances with 100% OR constraints

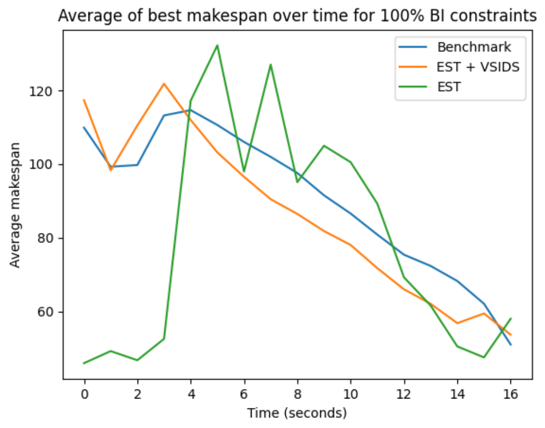

Figure 5: Average of best makespan over time (seconds) for j30 instances with 100% BI constraints

and overall solved instances. The same heuristic achieves a lower number of decisions, half as the baseline, which is due to the value ordering. Since the greedy approach does not find solutions for all instances, the average number of decisions is not comparable to the other approaches. In terms of time, there is no significant impact from a heuristic perspective. Moreover, the EST is performing worse in terms of solutions found; most of the satisfied solutions found in the baseline are transformed into unknowns. When analyzing it deeper, those instances seem to cause more problems for the heuristic, even if the optimal ones are solved faster. What is interesting to mention is that even with this drawback, the average best makespan found by EST is better for no logical constraints or 100% OR precedences, as shown in Figures 3 and 4, for the instances that are solvable in 15 seconds. The greedy order provides good insight into initial solutions, but it takes more time to go from one solution to another and verify the optimality. The worst performance is found for 100% BI constraints, which seem to create the most difficulties for all approaches for j30. EST is having the most issues with

	Benchmark		OR				BI			
	k1	_	1	1	1	1	1	1	1	1
	k2	_	1	2	5	10	1	2	5	10
	Percent Log		100	50	20	10	100	50	20	10
j30										
	solutions	196|32|0	184|44|0	192|36|0	195|33|0	195|33|0	162|66|0	179|49|0	190|38|0	191|37|0
	time (s)	3.60	4.56	4.18	4.31	4.15	6.61	5.26	4.44	4.17
	decisions	805k	1069k	917k	952k	944k	1691k	1208k	1075k	1000k
j60										
	solutions	45|25|0	39|31|0	36|34|0	40|30|0	43|27|0	18|52|0	25|45|0	38|32|0	42|28|0
	time (s)	10.12	10.71	11.52	10.84	10.54	14.37	13.44	11.37	10.85
	decisions	3786k	3464k	4096k	3638k	3684k	5642k	4969k	3956k	3629k

Table 2: Benchmark VSIDS results of logical constraints for the single-mode PSPLIB 30 and 60 activities instances, where k1 and k2 control the percentage of logical constraints, calculated as percent log for both OR and BI constraints, and solutions are presented as optimal|satisfiable|unkown.

	EST + VSIDS		OR				BI			
	k1	_	1	1	1	1	1	1	1	1
	k2	_	1	2	5	10	1	2	5	10
	Percent Log		100	50	20	10	100	50	20	10
j30										
	solutions	200|28|0	187|41|0	192|36|0	193|35|0	196|32|0	162|66|0	181|47|0	192|36|0	195|33|0
	time (s)	3.32	4.33	4.05	4.05	3.78	6.38	5.23	4.13	3.82
	decisions	454k	677k	536k	649k	567k	1142k	920k	674k	554k
j60										
	solutions	45|25|0	34|36|0	35|35|0	38|32|0	42|28|0	9|61|0	22|48|0	37|33|0	46|24|0
	time (s)	9.67	11.92	11.87	11.14	10.44	16.10	14.02	11.15	10.48
	decisions	1957k	2658k	2629k	2273k	1966k	4574k	3721k	2501k	2389k

Table 3: Heuristic EST + VSIDS results of logical constraints for the single-mode PSPLIB 30 and 60 activities instances, where k1 and k2 control the percentage of logical constraints, calculated as percent log for both OR and BI constraints, and solutions are presented as optimal|satisfiable|unkown.

	EST		OR				BI			
	k1	_	1	1	1	1	1	1	1	1
	k2	_	1	2	5	10	1	2	5	10
	Percent Log		100	50	20	10	100	50	20	10
j30										
	solutions	187|10|31	173|9|46	178|13|37	180|9|39	184|8|36	148|15|65	166|14|48	181|11|36	184|13|31
	time (s)	4.31	5.01	4.82	4.71	4.50	6.77	5.82	4.99	4.46
	decisions	216k	183k	200k	185k	199k	312k	253k	253k	192k
j60										
	solutions	48|5|17	41|11|17	45|7|18	48|7|15	46|9|15	33|12|25	41|9|20	42|8|20	49|7|14
	time (s)	8.69	10.07	9.63	9.28	9.26	11.55	10.44	9.96	9.09
	decisions	457k	619k	513k	558k	551k	1072k	843k	638k	617k

Table 4: Heuristic EST results of logical constraints for the single-mode PSPLIB 30 and 60 activities instances, where k1 and k2 control the percentage of logical constraints, calculated as percent log for both OR and BI constraints, and solutions are presented as optimal|satisfiable|unkown.

these instances; the average makespan is larger most of the time and is the only approach that cannot find solutions for all tests, as illustrated in Table 4.

When it comes to 60 activities, the results are more interesting, but fewer instances were evaluated. The j60 dataset is successfully solved by both baseline and EST + VSIDS, with differences in the number of satisfied and optimal solutions. The benchmark outperformed the latter approach in terms of optimality. The worst performance of the heuristic only found 9 optimal solutions out of 70 instances for 100% BI constraints. half of the original solution. The time increased for most of the experiments involving logical constraints. In contrast, EST optimality was significantly better. It found more optimal solutions for most tests, doubling the baseline results for BI constraints. On the other hand, this heuristic could not solve all instances and had a notable number of unknown results.

6 Conclusions and Future Work

As discussed in Sections 4.1 and 4.2, the heuristic approaches were compared against each other and the benchmarks in terms of the number of optimal and satisfied solutions found, unknown results, wall-time, and number of decisions. For a deeper look, the average best makespan was investigated for instances with no logical constraints and in the presence of 100% OR and BI relations. All in all, the experiments shown some improvements, as well as drawbacks.

6.1 Conclusions

For 30 activities, the combination of EST and VSIDS performed the best, finding a more optimal solution and reducing the time by an average of 0.2 seconds. The biggest improvement was in the number of decisions, which was almost half for most of the experiments. The motivation behind it may be the value ordering choice, which seems to perform better for this scheduling problem due to the goal of finding a start time as soon as possible. Even if the enhancement was small, Figures 3, 4, and 5 offer perspective in average best makespan reduction. The heuristics find lower makespans faster, especially for no logical constraints and OR precedences.

When it comes to j60, the greedy heuristic optimally solved more instances. The drawback is that, compared to the other two approaches, fewer solutions were found in total. The EST and VSIDS performed poorly, being worse than the baseline. It seemed that a more problem-specific approach has a higher chance of performing better for instances with a large number of activities, but no further conclusions can be drawn before experimenting on larger instances. Depending on the goal and the time constraints, variable ordering heuristics may help the process of solving the RCPSP-log using SAT solving.

6.2 Future Work

The results presented show some potential for improving the solving process of RCPSP-log, but they are very limited. The heuristics selected were very basic in terms of actual knowledge of logical constraints. Therefore, future work could explore more problem-specific domain knowledge.

One idea is to use Critical Path Analysis [12] and precedence rules to estimate the interval of possible start times per activity. This way, the EST weights could be bigger in that interval and smaller outside, in case the estimation is wrong. Moreover, the combination of EST and VSIDS could be improved by bumping the EST scores periodically. This could be a good idea considering the implementation of VSIDS, which seems to balance the scores proportionally and erase the EST information after a couple of iterations. The idea should be supported by visualization of the weight distribution.

Further work could complement the research by evaluating more datasets from PSBLIB, such as j90 and j120. Instances with more activities were not considered in this paper due to time and hardware limitations. This research left out multiple instances from j30 and j60, which could reveal different patterns. Furthermore, multiple settings for timeouts and memory limits could provide additional information. The EST heuristic was performing slower for some instances, which could have been analyzed better with a higher time limit.

7 Responsible Research

When it comes to ethical concerns in terms of project scheduling problems, most issues are related to potential negative effects on the stakeholders involved. Most of these concerns stem from poor estimations of resource usage and duration [13]. That being said, these issues are not impacting the research on scheduling problems. However, there are other limitations that can impact the study, as mentioned by Rubén Ruiz in one of his seminars [17]. Due to rapid advancements in hardware and computational power, the main issues are reproducibility and fair comparison of the results. This section presents the considerations made for these matters.

In order to address these concerns, several measures have been taken for each step in the method. Firstly, all instances evaluated come from the PSBLIB dataset, which is tailored for evaluating solution procedures for single-mode RCPSP. Furthermore, the adaptation of the instances for RCPSP-log followed the method proposed by Coelho and Vanhouke to ensure the validity, reproducibility, and method comparison of the research.

Moving on to the SAT encoding of the problem, the research followed the same model as the one from a past year bachelor thesis [18]. This formulation adapts the definition of OR constraints from the original paper [3]. Moreover, the validity of the model relies on the results from Melle [18] since the benchmarks are generated using similar hardware and software versions, providing similar computational power.

For further research and verification of code errors and results, the code for parsing and encoding, together with the generated files, will be uploaded to a public GitHub repository[4]. The pumpkin repository is private since it is managed by the TU Delft algorithmic team, but the heuristic technique may be replicated by following the procedures and using the equations provided in the paper.

Finally, all comparisons are made against benchmarks generated and run on the same machine with defined specifications in Section 5, giving a fair evaluation between baseline and heuristic approaches. The time for parsing and encoding was not considered in the results since it depends on the machine and programming language. The runtime specified in the results was strictly derived from the pumpkin solver's performance, which can cause misleading comparisons with other solvers. Therefore, a reproduction of the research with another solver may lead to different base results, but with a similar correspondence between heuristics and benchmarks.

[4]Github repository: The code base for the research can be found at https://github.com/iarinatudor/ResearchProject

7.1 ChatGPT

Another ethical concern is the use of ChatGPT[5] in this research. The AI language model helped check and suggest code related to the PySAT and Matplotlib[6] libraries used for encoding and visualization. The ideas were not coming from ChatGPT. For some specific references, the details of the publication were provided to generate a LaTex bibliography. AI assisted with LaTex formatting issues for tables and images. When it comes to debugging, ChatGPT helped with installing libraries and troubleshooting some errors for Windows. It also assisted with Python error codes for memory limits and dictionary manipulation. All prompts used can be found in Appendix A.

A Prompts used for ChatGPT

- Given reference X, can you give me the LaTex code for overleaf to reference it in references.bib?
- Given table X, how can I format it so that it does Y in LaTex?
- I have some data looking like X and I want a script in python that gathers the files that follow the pattern Y recursively and I want to visualize it like Z.
- I am trying to use PySAT for encoding a constraint saying that exactly one variable has to be true, can you give me the functions for that specific task?
- I use this command *pip install python-sat*$[aiger, approxmc, pblib]$ and the pblib fails, do you have any ideas?
- What is key error X in Python?
- I am working with a dictionary in Python for X, and I have the error Y, do you know why?

[5] ChatGPT: A language model developed by OpenAI. Available at https://chat.openai.com

[6] Matplotlib: A Python library used for data visualization. Available at https://matplotlib.org/

References

[1] I. Abío, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Bdds for pseudo-boolean constraints – revisited. In *Theory and Applications of Satisfiability Testing - SAT 2011*, pages 61–75. Springer Berlin Heidelberg, 2011.

[2] C. Artigues and R. Leus. The resource-constrained project scheduling problem. *Handbook on Project Management and Scheduling*, 1:243–286, 2013.

[3] J. Coelho and M. Vanhoucke. An approach using sat solvers for the rcpsp with logical constraints. *European Journal of Operational Research*, 252(2):431–441, 2016.

[4] M. De Jager. Solving resource-constrained project scheduling problems subject to no-overlap constraints using boolean satisfiability encoding. Master's thesis, University of Groningen, 2021.

[5] E. Demeulemeester, R. Kolisch, and A. Salo. Project management and scheduling. *Flexible Services and Manufacturing Journal*, 25(1):1–5, 2013.

[6] N. Eén and N. Sörensson. An extensible sat-solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518. Springer, 2003.

[7] S. Hartmann and R. Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407, 2000.

[8] M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon. The international sat solver competitions. *AIMag*, 33(1):89–92, 2012.

[9] A. Karam and S. Lazarova-Molnar. Recent trends in solving the deterministic resource constrained project scheduling problem. pages 124–129, 2013.

[10] R. Kolisch and A. Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997.

[11] J. Liang, V. Ganesh, E. Zulkoski, A. Zaman, and K. Czarnecki. Understanding vsids branching heuristics in conflict-driven clause-learning sat solvers, 2015.

[12] M. Lu and H. Li. Resource-activity critical-path method for construction planning. *Journal of Construction Engineering and Management-asce - J CONSTR ENG MANAGE-ASCE*, 129, 2003.

[13] M. Nepal, M. Park, and B. Son. Effects of schedule pressure on construction performance. *Journal of Construction Engineering and Management*, 132(2):182–188, 2006.

[14] N. Nilsson and B. Rolfsson. Complexity results for resource-constrained project scheduling problems with logical constraints. *Journal of Scheduling*, 14(4):317–332, 2011.

[15] Z. Pinedo, P. de Weerdt, J. van der Hoek, and S. Szymaniec. Resource-constrained project scheduling problems with logical constraints: A review. *European Journal of Operational Research*, 276(1):1–13, 2019.

[16] A. Alan B. Pritsker, L. Waiters, and P. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, 1969.

[17] R. Ruiz. State-of-the-art flowshop scheduling heuristics. https://www.youtube.com/watch?v=F3Ykma1eqnY, 2021. Accessed: 2023-6-5.

[18] M. Schoenmaker. Exploring heuristic methods for the resource-constrained project scheduling problem with logical constraints. Bachelor's thesis, Technical University of Delft, 2022. http://resolver.tudelft.nl/uuid:f225ecc7-7360-4f7e-859e-12df9dd974b7.