Exploring Heuristic Methods for the Resource-Constrained Project Scheduling Problem with Logical Constraints

Melle Schoenmaker
Supervisor: Emir Demirović
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

This paper presents the use of a heuristic solution method to improve the process of creating a conjunctive normal form (CNF) encoding of and finding optimal solutions to instances of the resource-constrained project scheduling problem with logical constraints (RCPSP-Log).

The RCPSP is an optimisation problem consisting of a set of resources with constant availability per time period, a set of activities with a duration and resource consumption rate, and a set of AND precedence relation pairs (i, j); activity j can start only if activity i has finished. The goal is to construct a schedule that minimises the total duration (makespan) of the project while satisfying all resource and precedence constraints. RCPSP-Log extends RCPSP by introducing two types of precedence relations OR and BI. OR relations allow a successor to be scheduled when at least one of its predecessors has finished. The BI relation prevents two jobs from running in parallel. The RCPSP is known to be **NP**-hard, thus the same holds for this extension.

To find solutions to the RCPSP-Log, activities can be assigned a priority value based on a heuristic function. The heuristic function makes use of the critical path method to calculate the latest finishing time (LFT) of an activity. By using the heuristic greedily a feasible sub-optimal solution is generated for a problem instance, which is then used to reduce the size of the instance's CNF encoding. A maximum satisfiability (MaxSAT) solving algorithm is used to search for a variable assignment for the CNF encoded problem and to prove whether it has an optimal makespan while satisfying all precedence and resource constraints.

The computational results of encoding and solving RCPSP instances from a well-known dataset with both the standard and the reduced encoding show that there is a clear advantage to using a heuristic algorithm, in terms of the time needed to find optimal solutions, to provide a starting point to the SAT solver.

1 Introduction

Project scheduling is a complicated problem that applies to various industries. For instance, the efficiency of factory processes and logistical schedules are heavily dependent on the time it takes to complete all sub-tasks. This has lead to various researchers looking for efficient algorithms that are able to find optimal schedules within a reasonable amount of time [1], [2], [3].

To find optimal solutions to problem instances it should be clear what is and is not allowed when constructing a schedule, which is where a problem definition is used to clarify all of a problem's rules. The resource constrained project scheduling problem (RCPSP) is a generalised problem statement that can be used to describe simple projects. It consists of a set of resources with constant availability and a set of activities with a duration, resource consumption and a set of predecessors. An activity is only allowed to be scheduled if there are enough resources available and all predecessors have finished executing, and once an activity is scheduled it must be finished. The RCPSP can already be used to model complicated processes, but the downside of this problem definition is that it limits the model to only have so-called AND precedence constraints, where all of an activity's predecessors need to be finished before it can be scheduled. Realistically, there are cases where the RCPSP does not suffice, which is where variations that extend the problem definition are needed. RCPSP with Logical constraints (RCPSP-Log) is a variation of the RCPSP, which extends the type of precedence constraints with two new precedence relations: OR and BI.

The main reference article [3] of this paper originally introduced the problem definition of the RCPSP-Log and a meta-heuristic algorithm to find solutions to instances of the

RCPSP-Log. To our knowledge [3] is the only publication to propose a complete method of solving the RCPSP with either OR or BI precedence relations. However, the use of a meta-heuristic algorithm as proposed by Vanhoucke and Coelho is not in line with the goal of our research, as we look to improve the exact satisfiability (SAT) solving algorithm using a heuristic algorithm. Another piece of related work defines an encoding for the RCPSP with no overlap BI constraints into conjunctive normal form (CNF), and solving it with the Z3 satisfiability modulo theories (SMT) solver [4]. This thesis has overlap with our paper as it discusses a method of encoding RCPSP into a boolean satisfiability encoding, but does not cover the use of heuristics to improve the SAT solving process.

From these previous works there is a clear gap in research on the inclusion of simple heuristics into the SAT solving process. In this paper we will reiterate and slightly alter the problem definition, clarifying the notation that will be used throughout the rest of the paper. We introduce a different solving approach, which greedily uses a priority heuristic based on critical paths to find upper bounds to the makespan of problem instances of varying sizes. The aim of our research is to study the use of these simple heuristic methods to create a SAT encoding of reduced size for the RCPSP-Log to study the increase in performance compared to a standard encoding when applied to large problem instances. The main research question that is answered in this paper is: "May SAT solving algorithms be augmented with domain-specific information to improve the solving of the RCPSP-Log?".

This research has two main contributions. First it demonstrates how the solutions found through a heuristic method are able to be used to significantly reduce the size of a problem's encoding and how reducing encodings speeds up the SAT solving process, allowing it to find better solutions when ran with a time bound. Secondly we provide a method to encode RCPSP-Log instances into propositional logic so that MaxSAT solvers can be used to find optimal schedules. The use of SAT solvers instead of problem specific algorithms is promising due to the consistent improvements that are being made to SAT solvers [5]. Writing an algorithm to produce the CNF encoding of a problem only has to be done once, allowing the progressive improvements in SAT solvers to speed up the solving of RCPSP-Log problem instances.

The main conclusions that are we have drawn is that reducing the encoding constructed by using the upper bound found with heuristics can provide a significant advantage to the SAT solver, especially for larger problem instances that have a high amount of added OR or BI logical constraints. The results have shown a significant decrease in both the size of the encoding and the time needed to encode and solve the problem instance after limiting it with the heuristic upper bound. Lastly the results show that for larger problem instances with 90 and 120 activities more instances can be solved to optimality after reducing the encoding.

The structure of this paper is as follows: First of all, section 2 provides a definition of the RCPSP and the RCPSP-Log extension. Then, section 3 illustrates the main ideas that are used for the heuristic algorithm and the SAT encoding. Section 4 states the experimental setup and presents the results, alongside a discussion of the results. Next, section 5 reflects on the ethical aspects of this research and the reproducibility of results. Finally section 6 covers the conclusion and potential future work.

2 Problem Description

As understanding a problem and clearly defining it is necessary to propose a solution, this section defines the RCPSP and how RCPSP-Log differs. An illustrative example problem

instance is explained with a figure to showcase all elements of the RCPSP-Log.

The Resource Constrained Project Scheduling Problem (RCPSP)

The RCPSP is an **NP**-Hard problem, with the goal of scheduling each activity at a certain time, so that the total makespan of the project is minimised. The makespan of a project is defined as the finish time of the *end* activity, which can only be scheduled once all other activities have finished.

A project consists of a set of jobs N that all need to be executed to finish the project and a set of resources R with a constant availability a_k. Every job $i \in N$ has: a non-negative integer duration d_i and a non-negative integer resource requirement $r_{i,k}$ for every resource $k \in R$. 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 [3, p. 578]. For a job to be scheduled the following requirements have to hold: all of its predecessors have to be finished at or before the start time of the job and each resource's availability minus the resource consumption of active jobs has to be greater or equal to the job's resource requirements.

The Resource Constrained Project Scheduling Problem with Logical Constraints (RCPSP-Log)

The RCPSP-Log introduces two new types of precedence constraints, OR and BI. It still has the AND constraint that is imposed in the original RCPSP, which allows a job to be scheduled if and only if all of its predecessors have finished.

The OR constraint is similar to AND as it also has a set of predecessor jobs. However, unlike the AND constraint, only one of these jobs needs to finish before being able to schedule the successor job. Coelho and Vanhoucke add an additional constraint in their problem definition that says that successors of an activity with OR precedence can only start once all predecessors of the activity have also been finished [3, pp. 578-579]. This is a necessity for their solution method of converting the problem to one with only AND constraints. However, this adds an additional level of complexity to the OR constraint which is undesirable. Instead this paper interprets OR precedence as an activity requiring at least 1 predecessor to finish with the only additional constraint being that all activities need to finish before the *end* activity can start.

The BI constraint, also called a no-overlap constraint, can only exist between a pair of jobs, and prevents these two jobs from being executed in parallel. In addition a changeover time is specified for each BI constraint as the resource needs to be transferred between the two activities. This changeover time is often modelled as an additional job that is executed between the two jobs specified in a BI constraint. To reduce the time needed to encode and solve instances, this paper analyses only the BI constraint with a changeover time of 0, since each additional value for the changeover time would require 16 instance sets to be solved. It is possible, however, to make slight modifications to the encoder to include support for a variable changeover time.

An example project and schedule can be seen in Figure 1. This example shows the durations above and resource requirements below the activities. The traditional AND constraints are represented with solid arrows between two jobs, OR constraints as dotted arrows from one job to another, and BI constraints as a pair of dotted arrows, as is present between jobs E and F. The top schedule is one that regards the OR relations as AND, and disregards

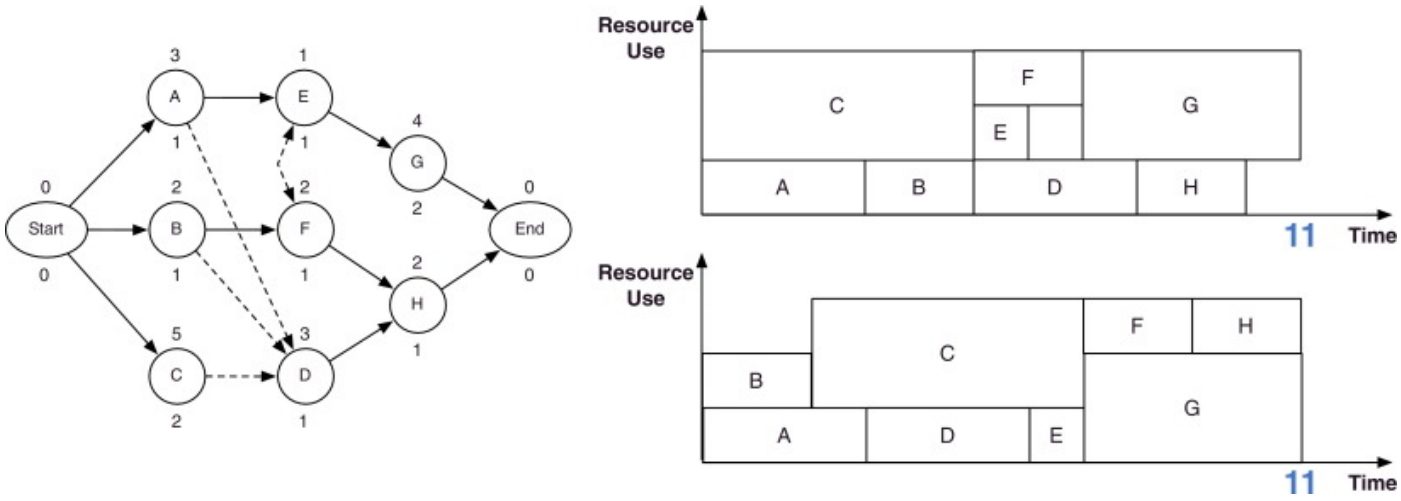

Figure 1: An example project with eight jobs and 2 dummy jobs Start and End, and two possible schedules (Above: without OR and BI constraints, Below: with OR and BI constraints)
Source: [3, Fig. 1]

the BI constraint. The bottom schedule is made while respecting the AND, OR and BI precedence relations.

3 Heuristic augmentation of SAT solvers

Experimental work

Multiple methods have been employed to solve the RCPSP-Log. The use of SAT solvers is interesting due to the continuous improvements that are made to SAT solving algorithms [5]. A problem encoding has to be made only once, but can be solved with progressively faster SAT solver algorithms.

Aside from improvements to the SAT solver, the encoding can also be optimised, by reducing redundancy and unnecessary clauses, to further improve the speed at which problem instances can be solved. Heuristics can also play a role in reducing the runtime of both the encoding and the SAT solver. The size of the encoding depends on the planning horizon T[1] of a project, which can be much larger than the optimal makespan of the project. A heuristic solver can be used to find a sub-optimal solution to the problem instance, the makespan of this solution can then be used as an upper bound of the makespan for the encoding. The heuristically found solution can also be converted into a variable assignment to provide the SAT solver an initial solution to start with. Lastly, a heuristic can be included in the SAT solver to adapt its variable selection procedure, this problem specific information can help the SAT solver make better decisions towards the optimal solution.

[1] Sum of the durations of all activities

Heuristic: Latest Finishing Time

The heuristic that was implemented to help find solutions for the RCPSP-Log is based on the latest finishing time principle, which is part of the Critical Path Method [6]. The heuristic iterates through all paths from an activity to the end activity while summing up the duration of each activity. The heuristic provides the priority with which an activity should be finished to not cause delays to the makespan of the project. If an activity has a lower latest finishing time than another activity it means that this low priority activity can be scheduled at a later time without impacting the makespan.

By using this heuristic a feasible sub-optimal solution can be found for the problem instance within a single search, this is a quick way of getting an upper bound on the minimal makespan of the project. An exact solution would have to look through a set of permutations that grows exponentially with the number of activities in the project. The downside of the heuristic approach is that the solution is not guaranteed to be optimal, which is due to the additional resource and logical precedence constraints that are part of the RCPSP-Log. Therefore optimal solutions still have to be found using exact solving methods.

CNF Encoding

To find optimal solutions using the results from the heuristic method the problem can be transformed into a CNF encoding adapted from [4]. This encoding consists of a set of literals and a set of clauses. For instance, literals are created for each activity i at each time period t where:

$$y_{i,t} = \begin{cases} 1 & \text{If activity i starts in period t.} \\ 0 & \text{If activity i does not start in period t.} \end{cases}$$

This set of literals is used to construct clauses that represent completion, precedence, consistency and resource constraints.

Completion Clauses

Completion clauses ensure that each activity starts early enough to finish before reaching the horizon T of the project. A completion clause for an activity i with duration d_i is constructed as follows:

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

These clauses are then combined into a set of completion clauses:

$$S = \wedge_{i=0}^{n+1} S_i$$

Resource Constraint Clauses

The RCPSP-Log, like the RCPSP, has resource constraints. These constraints are defined as:

$$\sum_{i=0}^{n+1} x_{it} r_{i,k} \leq a_k \quad \forall t \in [0, ..., T], \forall k \in R$$

The equation states that for all time periods and for all resources, the sum of every active activity's resource consumption is less than or equal to the available amount of that resource.

To construct clauses that capture resource constraints new literals have to be created corresponding to x_{it}.

$$x_{it} = \begin{cases} 1 & \text{If activity i is active in period t.} \\ 0 & \text{If activity i is not active in period t.} \end{cases}$$

It is important that x_{it} is assigned a value that is consistent with the start time of the activity i, which is guaranteed by consistency clauses. Note that these literals are not necessary for the dummy activities 0 and $n+1$, as these have duration 0 and therefore are never active. The consistency clauses for one activity i are constructed as:

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

These are combined into the set of all consistency clauses as:

$$C = \wedge_{i=1}^{n} C_i$$

Resource constraints can now be formulated using the new literals x_{it}. The resource constraint of resource k at time t is formulated as the pseudo-boolean constraint:

$$R_{kt} = \sum_{i=1}^{n} x_{it} r_{i,k} \leq a_k$$

This pseudo-boolean can be converted into CNF to allow a standard SAT solver to handle it. The complete set of resource constraint clauses is constructed as:

$$R = \wedge_{k \in R} \wedge_{t=0}^{T} R_{kt}$$

Precedence Clauses

To ensure that no activity is started before its precedence constraints are satisfied, precedence clauses are constructed. Since there are three different types of precedence relations, there are different encodings for each. To simplify the notation, we define z_{it}, which represents whether activity i is finished by time t.

$$z_{it} = \vee_{u=0}^{t-d_i} y_{iu}$$

For an AND precedence relation $(i,j) \in A^{And}$, to schedule j it must hold that $z_{it} = 1$. This implication $y_{jt} \implies z_{it}$ can be rewritten to the logical equivalent $\neg y_{jt} \vee z_{it}$. Putting these two together the precedence clause is thus formulated as:

$$P_{ij}^{And} = \wedge_{t=0}^{T-d_j} \neg y_{jt} \vee z_{it}$$

And each of these clauses are again combined into a set of AND precedence clauses:

$$P^{And} = \wedge_{(i,j) \in A^{And}} P_{ij}^{And}$$

For the OR precedence relations $(i_1, j) \in A^{Or}$, optionally $(i_2, j), (i_3, j) \in A^{Or}$, to schedule j it must hold for one predecessor $i_{1,2,3}$ $z_{i_x t} = 1$. This implication $y_{jt} \implies z_{i_x t}$ can be rewritten to the logical equivalent $\neg y_{jt} \vee_{x=1}^{|x|} z_{i_x t}$. Putting these two together the precedence clause is thus formulated as:

$$P_{ij}^{Or} = \wedge_{t=0}^{T-d_j} \neg y_{jt} \vee_{x=1}^{|x|} z_{i_x t}$$

And each of these clauses are combined into a set of OR precedence clauses:

$$P^{Or} = \wedge_{(i,j)\in A^{Or}} P_{ij}^{Or}$$

For a BI precedence relation $i, j \in A^{Bi}$, to schedule j it must hold that

$$\forall t \in [0, ..., T](x_{it} \wedge x_{jt} \neq 1)$$

which is equivalent to

$$\forall t \in [0, ..., T](\neg x_{it} \vee \neg x_{jt} = 1)$$

By using the aforementioned literals x_{it} used for resource constraints, a single BI relation can be encoded as follows:

$$P_{ij}^{Bi} = \wedge_{t=0}^{T} \neg x_{it} \vee \neg x_{jt}$$

These are then combined into a set of BI precedence clauses:

$$P^{Bi} = \wedge_{(i,j)\in A^{Bi}} P_{ij}^{Bi}$$

Finally, the three sets of clauses corresponding to each precedence type are combined into one set of precedence clauses:

$$P = P^{And} \wedge P^{Or} \wedge P^{Bi}$$

Soft Clauses

For the SAT solver to find an optimal solution, soft clauses are added corresponding to the makespan of the project schedule. A soft clause is a clause that can be violated without making an assignment unsatisfiable, by increasing the cost of the solution by the weight of the clause. The makespan of the project is translated into a cost by adding a soft clause for every $y_{n+1,t}$ with weight 1.

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

Complete CNF Encoding

The complete CNF encoding ϕ can now be constructed by taking all of the aforementioned sets of clauses and combining them:

$$\phi = S \wedge C \wedge R \wedge P \wedge W$$

SAT solver

A SAT solver is an algorithm that is able to efficiently assign values to the boolean literals to find an assignment where all clauses are satisfied. The MaxSAT solver is a variation of standard solvers that also takes in a set of soft clauses, each having a cost. It solves this weighted CNF while also minimising the sum of costs of violated soft clauses by iteratively looking for assignments with $totalcost = bestknowncost - 1$. The encoded problem is given as input to the MaxSAT solver, which then reports the variable assignment, cost of the soft clauses and the time taken to find the solution. Most SAT solving algorithms are also able to take additional inputs such as a timeout, to halt the solver after a certain amount of time, and a list of assumptions which can be used to provide the solver with variable assignments representing an initial or partial solution.

4 Experimental Setup and Results

The problem instances that were used to compare the standard encoding to the reduced encoding are the single mode problems with 30, 60, 90 and 120 jobs from PSPLIB [7]. The heuristic solving algorithm is coded in C++, which was run using Visual Studio with the C++ ISO version C++17 on an Intel® Core™ i7-8750H CPU with 8GB RAM. The SAT encoder was coded in Python version 3.8.10 64-bit using the PySAT library [8], and its output was run on the pumpkin MaxSAT solver which was made available to us by our supervisor. The runtime of the heuristic solving algorithm was added to the encoding and the solving time of the reduced encoding when calculating the difference with the standard encoding. Both the encoder and the pumpkin solver were run on compute nodes of the Delftse High Performance Computing (DHPC) DelftBlue Cluster [9], each instance was allocated 1 core of an Intel® XEON E5-6248R 24C 3.0GHz and 16 GB RAM. To speed up the process of encoding and solving each variation, each set of instances was broken up into batches of 40 problem instances, which could then be run in parallel.

Creating RCPSP-Log instances from RCPSP

There are no pre-existing datasets for the RCPSP-Log, making it necessary to define a procedure to convert RCPSP instances into RCPSP-Log. As proposed in [3, p. 585] this is done by defining two variables κ_1 and κ_2, that determine the starting activity and percentage of activities whose precedences are converted. For activity i the AND relations with all of their predecessors will be converted into OR relations if $(i + \kappa_1) \mod \kappa_2 = 0$. Similarly, for BI relations, activities are selected with the same formula, but only the AND relation with the immediate predecessor with the lowest activity number is converted into a BI relation while the others do not change.

Experimental Findings

Each set of instances, j30.sm, j60.sm, j90.sm, j120.sm was transformed into instances with OR and instances with BI relations with $\kappa_2 \in \{0, 1, 2, 5, 10\}$. Results of encoding and solving these datasets have been summarised in Table 1, and the individual results can be found in our repository[2]. The results are shown for both the standard and the heuristically reduced encoding as $standard|reduced$ for the following measured values: the number of proven optimal solutions $\#OPT$, the number of timeouts $\#TIMEOUT$, the number of instances without an initial solution $\#NOSOL$. Alongside those values, the average percentage difference between the standard and reduced encoding is presented for: the number of variables $\%Diff_{nv}$, the number of clauses $\%Diff_{nc}$, the time to encode $\%Diff_{tEnc}$, the time to prove optimality (excluding timeouts) $\%Diff_{tSolve}$, the total time to encode and prove optimality $\%Diff_{tTotal}$, the found makespan $\%Diff_{makespan}$. The quality of the results is determined by $\#OPT$, $\#TIMEOUT$, $\#NOSOL$ and $\%Diff_{makespan}$.

Discussion

Since the heuristic solving algorithm was run separately from the encoder and solver, its runtime was included in the comparison of both the encoding and the solving time of the standard and upper bounded method. This does mean that the difference in total runtime is

[2] `https://github.com/MelleSch/RP_Code_Results_RCPSP-Log`

Table 1: Comparative results of the PSPLIB j30, j60, j90 and j120 instances with varying amounts of transformed precedence constraints. $\#OPT$, $\#TIMEOUT$ and $\#NOSOL$ denote the number of problem instances, with notation *standard*|*reduced*, for which a proven optimal solution was found, the timeout was reached, and no initial solution was found, respectively. $\%Diff_{nv,nc,tEnc,tSolve,tTotal,makespan}$ each show the average percentage difference between the standard and the reduced encoding for the number of variables, number of clauses, time to encode, time to prove optimality (excluding timeouts), total time to encode and prove optimality, and the makespan.

j30		OR				BI			
κ_1	-	1	1	1	1	1	1	1	1
κ_2	-	10	5	2	1	10	5	2	1
Percent Log	0	10	20	50	100	10	20	50	100
$\#OPT$	465 | 464	457 | 461	450 | 449	424 | 427	328 | 334	461 | 463	458 | 457	434 | 438	383 | 383
$\#TIMEOUT$	15 | 16	23 | 19	30 | 31	56 | 53	152 | 146	19 | 17	22 | 23	46 | 42	97 | 97
$\#NOSOL$	0 | 0	0 | 0	0 | 0	0 | 0	0 | 1	0 | 0	0 | 0	0 | 0	0 | 0
$\%Diff_{nv}$	-63.2	-64.11	-64.92	-66.88	-70.32	-61.23	-60.48	-60.02	-60.42
$\%Diff_{nc}$	-63.43	-64.32	-65.13	-67.12	-70.61	-61.37	-60.6	-60.12	-60.51
$\%Diff_{tEnc}$	-65.32	-67.23	-68.40	-70.63	-73.70	-64.51	-63.61	-62.90	-62.00
$\%Diff_{tSolve}$	-39.75	-37.42	-24.67	-27.13	-20.12	-28.87	-19.46	-31.11	-19.96
$\%Diff_{tTotal}$	-52.42	-51.92	-44.97	-45.61	-37.50	-46.52	-38.92	-43.05	-35.78
$\%Diff_{makespan}$	0.02	0.01	-0.01	-0.08	-0.02	0.01	-0.02	-0.01	0.05
j60									
$\#OPT$	395 | 395	391 | 390	374 | 373	314 | 318	170 | 177	392 | 392	375 | 381	343 | 351	271 | 276
$\#TIMEOUT$	85 | 85	89 | 90	106 | 107	166 | 162	310 | 303	88 | 88	105 | 99	137 | 129	209 | 204
$\#NOSOL$	0 | 9	1 | 6	1 | 5	2 | 8	9 | 31	0 | 0	0 | 0	0 | 1	15 | 7
$\%Diff_{nv}$	-75.38	-76.27	-76.93	-78.79	-81.12	-74.78	-74.47	-74.71	-75.79
$\%Diff_{nc}$	-75.44	-76.31	-76.98	-78.83	-81.18	-74.80	-74.49	-74.71	-75.78
$\%Diff_{tEnc}$	-79.03	-81.06	-81.49	-83.08	-84.96	-79.57	-78.90	-78.74	-79.03
$\%Diff_{tSolve}$	-68.33	-66.56	-75.90	-74.99	-62.16	-64.69	-65.07	-67.77	-71.06
$\%Diff_{tTotal}$	-76.49	-77.48	-80.16	-81.23	-77.93	-75.83	-75.57	-75.99	-77.01
$\%Diff_{makespan}$	-0.30	-0.08	-0.01	-0.67	-1.26	-0.51	-0.26	-0.39	-0.93
j90									
κ_1	-	1	1	1	1	1	1	1	1
κ_2	-	10	5	2	1	10	5	2	1
Percent Log	0	10	20	50	100	10	20	50	100
$\#OPT$	373 | 377	365 | 370	346 | 353	284 | 297	126 | 132	369 | 374	359 | 364	327 | 340	245 | 260
$\#TIMEOUT$	107 | 103	115 | 110	134 | 127	196 | 183	354 | 348	111 | 106	121 | 116	153 | 140	235 | 220
$\#NOSOL$	0 | 26	0 | 31	2 | 32	7 | 45	32 | 101	0 | 4	1 | 1	0 | 1	19 | 13
$\%Diff_{nv}$	-80.68	-81.51	-82.14	-83.79	-85.78	-80.55	-80.53	-80.75	-81.85
$\%Diff_{nc}$	-80.70	-81.52	-82.15	-83.80	-85.79	-80.54	-80.52	-80.73	-81.82
$\%Diff_{tEnc}$	-84.06	-86.06	-86.53	-87.88	-89.52	-85.04	-84.78	-84.54	-85.07
$\%Diff_{tSolve}$	-84.71	-86.54	-84.86	-86.94	-78.93	-85.51	-83.08	-80.76	-80.46
$\%Diff_{tTotal}$	-84.26	-86.21	-86.26	-87.75	-87.22	-85.20	-84.53	-83.81	-84.17
$\%Diff_{makespan}$	-0.95	-1.81	-2.25	-3.77	-9.19	-1.46	-2.18	-4.41	-9.07
j120									
κ_1	-	1	1	1	1	1	1	1	1
κ_2	-	10	5	2	1	10	5	2	1
Percent Log	0	10	20	50	100	10	20	50	100
$\#OPT$	222 | 241	199 | 226	176 | 202	88 | 108	11 | 15	198 | 233	193 | 222	139 | 173	66 | 84
$\#TIMEOUT$	378 | 359	401 | 374	424 | 398	512 | 492	589 | 585	402 | 367	407 | 378	461 | 427	534 | 516
$\#NOSOL$	1 | 172	3 | 141	5 | 146	19 | 175	22 | 266	2 | 47	3 | 17	5 | 4	18 | 17
$\%Diff_{nv}$	-80.26	-80.51	-80.75	-81.32	-82.19	-79.34	-78.82	-78.27	-78.23
$\%Diff_{nc}$	-80.27	-80.52	-80.76	-81.34	-82.22	-79.33	-78.80	-78.25	-78.22
$\%Diff_{tEnc}$	-85.32	-87.19	-87.39	-87.70	-88.34	-86.12	-85.44	-84.60	-83.94
$\%Diff_{tSolve}$	-80.07	-84.92	-86.69	-85.15	-72.53	-81.57	-83.10	-82.14	-84.14
$\%Diff_{tSolve}$	-84.19	-86.75	-87.29	-87.24	-84.26	-85.26	-84.98	-84.13	-84.07
$\%Diff_{makespan}$	-7.91	-9.68	-11.59	-15.38	-20.66	-18.46	-13.66	-19.47	-25.67

a little lower than the weighted average of $\%Diff_{tEnc}$ and $\%Diff_{tSolve}$, which was corrected for by calculating the total time separately.

The results in Table 1 show that for instances from j30, the quality of the solutions found

by the SAT solver barely differ, the number of proven optimal solutions found within the 60 second timeout differs by at most 6 for OR %Log=100. The makespans found for the reduced encoding also only have less than 0.1% difference from the standard encoding. Some bigger differences are present in the size of the encoding and the time to encode or solve, with the number of variables and clauses of reduced encodings being around 60% smaller than those of the standard encoding and the time to encode and solve being around 65% and 25% smaller, respectively. The total time to solve was around 40% to 50% faster by using the heuristic.

The j60 instances show differences similar to those present for j30 instances between the quality of the solutions of the two encodings, but with more timeouts and instances without initial solutions being found. The size is around 75% smaller for the reduced encoding, with the time to encode and the time to find optimal solutions being about being 80% and 65% shorter, respectively. For the total time to solve a 75% to 80% speedup was gained.

In the j90 instances a larger increase was found in the number of proven optimal solutions found by reduced encoding compared to the standrard encoding, especially for the instances with high %Log. The number of instances that time out or do not have an initial solution after 60 seconds is again slightly larger. The average difference between makespans is quite a bit larger than they were for j30 and j60 instances, again with larger differences showing up for instances with higher %Log. The encoding is shrunk by 80%, and the time to encode and solve are 85% and 80% less, respectively, for the reduced encoding compared to the standard encoding. In total the process was around 85% faster by calculating the upper bound.

Lastly, the j120 instances show the largest difference between the two encodings, the number of optimal solutions found for the reduced encodings is around 10% higher than that of the standard encoding. Similarly to the pattern seen between the j60 and j90 instances the number of timeouts is larger again. The difference in makespans ranges from 8% to upwards of 25%, also with higher differences for instances with a higher %Log. The reduction in encoding size is slightly smaller than for j90 instances with a difference of about 80%. The times needed to encode and solve and their sum have a similar reduction as is present for j90 instances, with an 85%, 80% and 85% reduction for the reduced encodings, respectively.

5 Responsible Research

The field of project scheduling problems does not directly involve certain ethical concerns. There exist negative effects that bad schedules can potentially have on the people that have to follow the schedule, but these are often related to poor estimations of the duration and resources necessary to complete tasks [10]. There are, however, several concerns that affect research on algorithmic scheduling. These concerns were explained in a seminar by Rubén Ruiz [11], with the main concern being the ability to make fair comparisons between results. To address these concerns we have given detailed descriptions of our methods, results and comparisons.

To ensure reproducibility of the experiments that were performed for this paper we have taken several precautions. The experiments were all performed on the standard single mode RCPSP instances of the PSPLIB dataset. This dataset was specifically generated with the purpose of evaluating solution procedures for single- and multi-mode resource-constrained project scheduling problems [7]. To transform this dataset into RCPSP-Log instances the procedure explained in [3, p. 585] is taken with the same κ_1 and κ_2 as listed in [3, Tab.

2]. Due to a nuance in the definition of OR precedence relations by Coelho and Vanhoucke, which was not in line with the logical definition of OR relations, we do not compare our results to theirs.

Another precaution we took is to publish the code used to generate the results and the results for each individual problem instance to a public repository[3], this allows others to review the code for errors and to rerun the code to verify the validity of the results, or to benchmark the code on a machine with different specifications. Frequently, comparisons between execution times are made based on the results table of previous research and the newly found results, which causes misleading comparisons, since the past benchmarks were run on older hardware and different software versions providing different computational power [11]. Comparisons in execution time were done by, whenever applicable, writing code in the same language and running it on the same machine with the same resources.

Finally all single mode instances of 30, 60, 90 and 120 activities have been solved with the same code version and the results have been listed for each percentage (10%, 20%, 50% and 100%) of logical constraints. The results are compared by calculating the average percentage deviation between the makespans we find using the standard and the reduced encoding. Notably the time needed to solve instances heuristically was added to both the encoding and the solving time, meaning that the total process speeds up slightly more than the weighted average of the two percentages. This does give the reduced encoding a slight disadvantage over the standard encoding, but the advantage is that the encoding and solving time can be compared in a fair manner.

6 Conclusions and Future Work

The goal of this paper is to study whether problem specific heuristics can provide an advantage when using SAT solvers for the RCPSP-Log. As presented in the discussion, using a heuristic to calculate an upper bound to the solution of a problem has proven to significantly reduce the size of and time needed to encode and solve an encoding of any single mode RCPSP instance from PSPLIB. As far as the quality of the results, j30 and j60 instances show no clear advantage to using the reduced encoding compared with the standard encoding, however, the time needed to reach optimal solutions is significantly less when using the reduced encoding compared to the standard encoding. Bigger differences were found for the larger problem instances j90 and j120, where using a reduced encoding allows the SAT solver to find more optimal solutions and the makespan of solutions found for the reduced encoding are below those found for the standard encoding by a decent margin. This margin is most clear for instances of the j120 problem set where makespans for reduced encodings are 8% to 25% smaller after running the solver for one minute.

Possible future work to expand on the findings presented in this paper, would be to use different heuristics to calculate multiple upper bounds to the makespan of an instance, after which the lowest upper bound can be used to limit the size of the encoding. Another extension to this research could be the addition of assumptions, based on the solution found by the greedy heuristic algorithm, that the SAT solver can use to construct an initial solution, this would reduce the time that the SAT solver spends on finding the initial solution which was already constructed using the heuristic solver. Making use of these assumptions can further improve the results found for the reduced encoding compared to the standard encoding. Replacing some of the exact searching performed by the SAT solver with a faster

[3] `https://github.com/MelleSch/RP_Code_Results_RCPSP-Log`

greedy heuristic search would allow the solving time to be shortened even further. Lastly we used a timeout of 60 seconds to limit complex instances from taking up more time than necessary. Despite being able to compare the intermediate solutions, we do miss out on some interesting information about how well the heuristic reduction helps with solving these instances. Running these instances with a greater timeout or until a proven optimal solution is found can add more context to our results.

All in all this paper shows the added value of heuristically computing upper bounds to reduce the size of RCPSP-Log encodings and increase the speed with which optimal solutions can be found for problem instances with 30, 60, 90 and 120 single mode activities from the PSPLIB dataset.

References

[1] D. Debels and M. Vanhoucke. A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55:457–469, 06 2007.

[2] 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.

[3] M. Vanhoucke and J. Coelho. An approach using sat solvers for the rcpsp with logical constraints. *European Journal of Operational Research*, 249(2):577–591, 2016.

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

[6] J.E. Kelley Jr and M.R. Walker. Critical-path planning and scheduling. *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference on - IRE-AIEE-ACM '59 (Eastern)*, page 160, 1959.

[7] 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.

[8] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.

[9] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). `https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1`, 2022.

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

[11] R. Ruiz. State-of-the-art flowshop scheduling heuristics. `https://www.youtube.com/watch?v=F3Ykma1eqnY`, 2021. Accessed:2022-6-8.