Comparing schedule generation of VSIDS against CPRU for RCPSP-t solvers

Wouter Breedveld
Supervisor(s): Emir Demirović, Maarten Flippo, Imko Marijnissen
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 23, 2024

Name of the student: Wouter Breedveld
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Maarten Flippo, Imko Marijnissen, Julia Olkhovskaia

Abstract

This paper looks at the different parts of the Critical Path and Resource Utilization (CPRU) heuristic for use in the Resource Constraint Project Scheduling Problem, with variable resources (RCPSP-t programming problem). RCPSP-t has many real-world instances such as in hospitals or manufacturing. Optimizing the solution generation for these instances may improve the efficiency of these industries.
CPRU was split into parts and different versions were compared against VSIDS a more modern heuristic to determine if problem-specific heuristics perform better than generally good ones. A new adaptation of CPRU that adapts the research utilisation score by calculating the fraction used based on the amount of resources available given scheduled activities instead of looking at the resources of the problem instance. From the results, we concluded that CPRU may perform better for large instances of RCPSP-t compared to VSIDS in terms of generating good schedules within a few iterations. We also found that CPRU generates better schedules given a small time limit. We did not find evidence that the CPRU adaptation improved performance compared to CPRU in terms of schedule and time limits.

1 Introduction

RCPSP (Resource Constraint Project Scheduling Problem) is a class of problems where some tasks, such as car repairs, must be scheduled. Ideally, this happens in a way that minimizes the difference between the start time of the earliest starting task and the end time of the latest finishing task. Tasks each have an amount of resources they take, for our car example, imagine the time of specific employees or specific equipment, as being the required resources. Tasks may have different durations and require different resources. RCPSP is a well-known problem in project scheduling with many variants [1]. With all these variants RCPSP can present a whole scale of real-world planning problems, making the efficient solving of these problems useful for many people.

RCPSP-t, the variant studied in this paper, is a different variation of RCPSP. In this variant, a task may even require a different number of resources per time step; the repair may need a carjack in the beginning and a wheel gun later.

Solving RCPSP-t problems has many modern-day use cases. From scheduling support tickets in an IT company [2], to scheduling manufacturing jobs [3], to scheduling aircraft repairs [4] going into the Line Maintenance Scheduling problem and also recently, medical research [5], [6].

Many papers have been devoted to exploring different heuristics for solving RCPSP problems. Selection based on placement of tasks, such as scheduling the last possible task first, were the first to be explored [7]. Some research has also already been done into RCPSP-t specific heuristics [6], [8]. One such notable heuristic is CPRU [6] which takes into account resource utilisation and precedence relations. It also introduces so-called tournaments, where for variable selection, instead of considering the heuristic value for all variables to be chosen from, it instead looks at a random subset of a certain size and selects the variable with the highest heuristic score out of these. Tournaments were introduced to generate multiple schedules and select the best one instead of having one deterministic schedule.

CPRU has however to our knowledge, never been considered as a heuristic in a constraint programming (CP) solver, where instead of re-running the algorithm multiple times, it is stored what combinations of start times do (not) work together and the solver uses this information to try and work towards a schedule. The effect of the tournaments on CPRU in such a CP solver has also not been considered.

This paper looks at CPRU and tries to find out if the tournament used with CPRU influences performance by implementing a version with and without tournaments into a CP solver. This paper also introduces a variation of CPRU, where CPRU always calculates resource usage based on the initial instance, CPRU-a dynamic calculates resource usage to try and improve the accuracy of the heuristic at the cost of some performance. CPRU-a does this by changing the amount of available resources used to calculate the CPRU heuristic score after a start time gets set for a task as shown in figure 3. CPRU-a was also implemented into the same CP solver. Finally, a comparison was made with all three CPRU versions against VSIDS, a modern variable selection heuristic for CP solvers focused on satisfying recent conflicts.

The results show, that for large instances of RCPSP-t, using CPRU as the variable selection heuristic may result in generating better schedules within fewer iterations compared to using VSIDS. We think this is due to CPRU having a clearly defined starting point, in prioritising tasks which start early and as such decreasing problem size quickly, whereas VSIDS is not able to decrease the size of its problem as much as quickly. Different versions of CPRU, with and without tournaments, were found not to significantly influence the result in terms of generating better schedules within fewer iterations. The suggested variation CPRU-a did not show any improvement relative to the performance of CPRU. This we deem most likely because the adaptation primarily improves the heuristic once a large number of tasks has already been assigned, but at that point, the order of selection for the variables is less important as the remaining instance has become very simple.

In section 2 we give a formal definition of the problem. Section 3 shows past work that relates to this problem. Section 4 provides details about CPRU and the RCPSP-t solver used in this paper. In section 5 we introduce a new heuristic based on CPRU, CPRU-a. Section 6 explains the setup of the experiment and analyses its results. Ethical implications and reproducibility of this research are considered in section 7. Finally, in section 8 conclusions are made based on the results of the experiments and proposals are made for future work.

2 Problem definition

To formally define this problem we use the definition by [9]. In this formulation we define a tuple (V, UB, p, E, R, B, b) where:

- $V = \{0, 1, \ldots, n, n+1\}$ is the set of activities. In this set activities 0 and $n+1$ are considered dummy activities, so they don't represent actual activities. This will help us create a schedule because we can now place all activities that are not dummies before and after an activity. The set of non-dummy activities is thus defined as $A = \{1, \ldots, n\}$
- UB is the upper bound on the time from start to finish from the earliest starting to the latest finishing task. This is needed in the definition of B to represent the different resource consumption of each time instant, meaning relative to the starting time, in $0 \ldots UB - 1$.
- $p \in \mathbb{N}^{n+2}$ is a vector of natural numbers, where p_i is the duration of activity i. For the dummy activities $p_0 = p_{n+1} = 0$ and for all non-dummy activities $p_i > 0$.
- E is a set of pairs of activities. Each of these pairs is a precedence relation. So if $(i, j) \in E$ then activity i must be finished before j can start. It is assumed the precedence graph $G = (V, E)$ contains no cycle, as otherwise the activities in the cycle could never be scheduled. By convention, activity 0 has an edge to all other activities and activity $n+1$ has an edge coming from all other activities.
- $R = \{1, \ldots, v\}$ is a set of the resources.

- $B \in \mathbb{N}^{v \times UB}$ is a matrix of natural numbers, where $B_{k,t}$ is the amount of resource k available at time t. Note that this definition of B is different from the definition for RCPSP. This is because in RCPSP-t the amount of resource k depends on t.
- b is a three dimensional matrix of natural numbers where $b_{i,k,e}$ represents the amount of resource k consumed by activity i at the time step e in activity is duration. All demands should be non-negative and the dummy activities, 0 and $n+1$, do not have any resource demands.

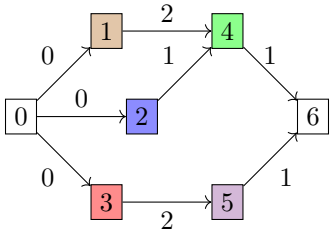
A schedule is defined to be a vector of naturals $S^{n+2} = (S_0, S_1, \ldots, S_n, S_{n+1})$ where $S_i \in \mathbb{N}_0$ is the start time of activity i. Without loss of generality, it can be required that $S_0 = 0$, since the first activity will be the dummy activity, which does not have any (time-specific) resource consumption meaning it can always be placed at time 0.

The goal of RCPSP-t is to minimize the makespan, that is, the finish time of the final activity [6]. Activities must be fully run to completion once started and have predefined durations. Activities also have resource demands that depend on what time instant of their execution they are in. Finally, some activities may be dependent on other activities to have finished before they can start. The resources that the activities consume have a variable amount of availability dependent on the time. At a given time the amount of a resource being consumed by the activities running cannot surpass the amount available at that time.

A solution of RCPSP-t is thus a schedule of S with minimized makespan S_{n+1} such that:
- Every activity starts only after its predecessors have finished.
- The total amount consumed of resource at a given time is lower or equal to the amount of that resource available at that time.

Figure 1 shows an example of an RCPSP-t problem and figure 2 shows an optimal solution for it. Table 1 gives an overview of the terms used in this paper and their meanings.

Precedence graph:



Activity durations and demands:

	0	1	2	3	4	5	6
p_i	0	2	1	2	1	2	0
$b_{i,1,0}$	-	1	3	1	4	3	-
$b_{i,1,1}$	-	2	-	3	-	1	-
$b_{i,1,2}$	-	-	-	-	-	-	-

Resource capacities:

$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{1,4}$	$B_{1,5}$
2	4	5	4	3	3

Figure 1: Example of an RCPSP-t instance with 5 (non-dummy) activities, 1 resource and $UB = 6$

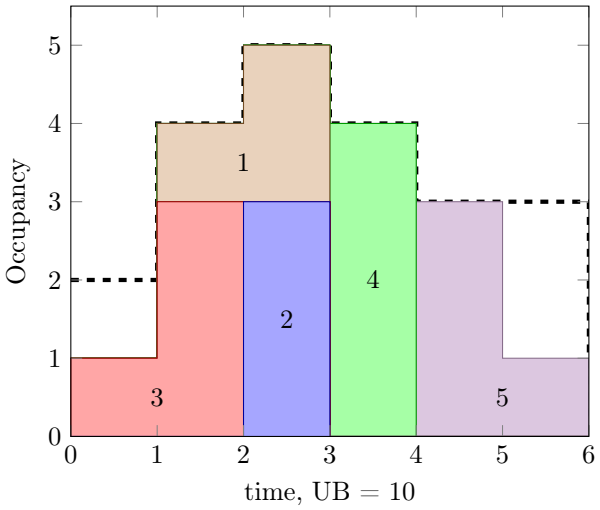


Figure 2: An optimal solution for the RCPSP-t instance of figure 1, with $S = (1, 2, 0, 3, 4)$

Parameters	
p_i	Duration of activity i
$G = (V, E)$	Precedence graph
R	Set of resources
$b_{i,k,e}$	Amount of resource k, activity i consumes at time-instance t
$B_{k,t}$	Amount of resource k available at time t
n	Number of activities
v	Number of resources
$H = \{0, \ldots, UB - 1\}$	Scheduling horizon
Precalculated functions	
$G^* = (V, E^*)$	Extended precedence graph
$(i, j, l_{i,j}) \in E^*$	Extended precedence with lag $l_{i,j}$
$ES(i)$ and $LS(i)$	Earliest and latest start time of activity i
$STW(i)$	Start time window of activity i
$RTW(i)$	Running time window of activity i
Variables	
S_i	Start time of activity i
$x_{i,t}$	Boolean indicating if activity i is running at time t
$z_{i,j}$	Boolean indicating if activity i is running when activity j starts

Table 1: Definition of terms for the constraints

2.1 Preprocessing

To simplify the constraints we will later introduce for the RCPSP-t problem, and to reduce the domain of the variables, we will apply some preprocessing to the input, as derived from [9] and also similar to [6].

We define $G^* = (V, E^*)$ to be the extended precedence graph with E^* being a set of weighted edges, where $(i, j, l_{i,j}) \in E^*$ if there is a path from i to j in E, meaning each node will now be connected to all its successors and predecessors forming a transitive closure. The weight assigned to the edge is a lower bound on the gap, also known as lag, between the start times of the tasks, which can be defined as follows:

$$l_{i,j} = \max(i, \max(l_{i,a} + l_{a,j})) \qquad \forall a \in V \text{ s.t. } \exists x(i, a, x) \in E^* \wedge \exists y(a, j, y) \in E^* \tag{1}$$

If there is an edge from i to j that means j is dependent on i, thus j can only start after i is finished. At the earliest i is finished at $S_i + p_i$ so the minimum distance between the start of time i (S_i) and the start of j time (S_j) is the duration of i.

Using G^* we can determine the earliest and latest start time ($ES(i)$ and $LS(i)$, respectively) of each activity. We get $ES(i)$ by calculating the cost of the critical path (the path with the highest cost) from task 0 to i. The duration of this path is the minimum time all predecessors of i need to finish, thus also a lower bound on when i can start. We can also get $LS(i)$ by instead calculating the critical path from task i to $n + 1$. Since $n + 1$ is the final task all tasks need to finish before it can start. And, because for any task j which is a successor i the latest start time i can have is $S_j - p_i$, since otherwise, it would not finish before j starts. Thus the cost of the critical path from i to $n + 1$ is the latest start time of i ($LS(i)$).

Using the earliest and latest start times we can define the start time window as $STW(i) = [ES(i), LS(i)]$ which gives all possible starting times for a given task and the running time window as $RTW(i) = [ES(i), LS(i) + p_i - 1]$ which contains all possible times when the tasks may be running.

2.2 Constraint Formulation

The variables used are:

- S_i: Integer denoting the start time of activity i for $\forall i \in V$.
- $y_{i,t}$: Boolean variable denoting the whether activity i starts at time t.

We use the following constraints:

$$S_0 = 0 \tag{2}$$

$$S_i \geq ES(i) \qquad \forall i \in V \backslash \{0\} \tag{3}$$

$$S_i \leq LS(i) \qquad \forall i \in V \backslash \{0\} \tag{4}$$

$$y_{i,t} \leftrightarrow S_i = t \qquad \forall i \in A, \forall t \in STW(i) \tag{5}$$

$$\sum_{\substack{i \in A, \text{ s.t.} \\ t \in RTW(i)}} \sum_{\substack{e \in 0 \ldots p_i - 1 \text{ s.t.} \\ t - e \in STW(i)}} b_{i,k,e} \cdot y_{i,t-e} \leq B_{k,t} \qquad \forall k \in R, \forall t \in H \tag{6}$$

Constraint (2) enforces the dummy task to start at time 0. Constraints (3) and (4) enforce the tasks to start within the lower- and upper-bound respectively as calculated in

the preprocessing step. Constraint (5) enforces $y_{i,t}$ to be correctly assigned. Constraint (6) makes sure all tasks running at a given time do not consume more of any resource than available at that time.

3 Related Work

In [9] a general modelling technique is explained for RCPSP problems and an adaption to this model for RCPSP-t is shown. In the presented model an SMT solver is used to check if constraints are upheld. If all contains are upheld a new variable is selected randomly. The algorithm that determines which variable gets selected next is referred to as the selection heuristic. [9] also analyses the performance differences between some selection heuristics. We use the work by [9] as a basis for the constraints for the solver we will present later.

Over time many (selection) heuristics have been proposed. Marques [7] compared some simple heuristics including Bohm's [10], which prioritizes solving small clauses before large ones. It also looks at the MOM's (Maximum Occurrences on clauses of Minimum size) [11], which tries to solve variables that are in a lot of clauses first, before moving to the less common ones. Marques also compares the heuristics by Jeroslow and Wang [12] which selects the variable that roughly appears in the largest clauses. These papers show the change in heuristics over time, where there is a primary focus on general constraint programming and RCPSP heuristics and not much specialisation for RCPSP-t.

Davis and Patterson [13] create the minimum slack rule (MSLK) which prefers activities with the smallest difference between their possible earliest start time and latest start time. This forms the basis for what will later be seen in the CPRU heuristic as the critical path as explained in section 4.2.

Alvarez-Valdes and Tamarit [14] define and compare a variety of heuristics including the latest start time rule (LST), which selects the activity with the latest possible starting time. This research shows the interest in heuristics focused on the instance and also shows that focussing on starting times when scheduling, may yield good results. In CPRU we will later see a focus on earliest starting time (EST), a slight variation on LST, aswell.

Moskewicz et al. [15] introduce a Variable State Independent Decaying Sum (VSIDS) Decision Heuristic. This heuristic keeps track of how recently variables have been added to a clause and updates a score where recently used variables get a higher score. To prevent some variables from getting a very high score early it implements a system where periodically all variables are divided by a set value. This is still are relevant standard to which we will be comparing in this paper.

Hartmann [6] proposed a tournament-style heuristic in which only a random subset of the variables is considered and the one with the highest heuristic out of these is selected. For this, Hartmann uses a heuristic in which tasks that can start earliest and have large resource consumptions, are prioritised (*critical path and resource utilization*, CPRU). Hartmann compares this tournament heuristic with CPRU against random selection (RND), LST and MSLK and shows the tournament heuristic with CPRU performs better than all others, followed by LST. This paper will focus on CPRU and try to adapt it to improve its performance even more.

In a later work, Hartmann [8] showed a slight improvement in run-time over the tournament CPRU heuristic, using a new heuristic based on the genetic algorithm in [16]. The proposed heuristic allows for delaying of the start time, expanding the number of options explored. However, Hartmann notes that this new heuristic only has better results in small

instances of the problem. Though this does show improvements on CPRU may still be possible.

4 Preliminaries

In this section, we start by introducing the basics of constraint programming in section 4.1. Next, we explain the workings of the Critical Path and Resource Utilization (RCPU) heuristic in section 4.2 and its tournament adaptation in section 4.3. In sections 4.4 and 4.5 we explain some features of the Pumpkin constraint programming solver that will later be used and adapted to do our experiments. Section 4.4 goes into Conflict-Driven Clause Learning, a method of adding new clauses and section 4.5 goes into Variable State Dependent Sum, a heuristic for variable- and value-selection.

4.1 Constraint Programming

Constraint Programming (CP) modelling strategy for combinational optimisation problems. That is, a problem where there is some finite set of objects, and the challenge is to find the optimal one [17]. In CP three main concepts are used to represent such a problem. Firstly, all decisions that need to be made are encoded as variables, i.e. at what time should task X start? Each task would get its own start time variable. Second, constraints are created to model and enforce the relation between different variables. An example would be the sum less than or equal constraint, enforcing that the sum of a given set of variables is lower than some number, which in our case could be the resource capacity. Finally, there is an objective function, this is what the CP model is optimizing for. In our case, this would be the minimization of the makespan.

4.2 Critical Path and Resource Utilization heuristic

The Critical Path and Resource Utilization (CPRU) heuristic is a variable- and value-selection heuristic designed by Hartmann [6]. CPRU was specifically designed to solve RCPSP-t instances. It consists of two different parts, CP_i a score on the critical path and RU_i a score on the resource utilisation, which are multiplied together to produce a final score (higher is better) to rank the variables.

Firstly there is the critical path part (CP_i). CP_i is defined as $CP_i = (UB - 1) - LS(i)$, thus activities which have a long critical path, that is tasks whose duration of their successors are longer, are generally prioritised. This makes sense because if these kinds of tasks were planned at a later time, it may leave very little wiggle room for its successors.

Secondly, there is the resource utilization part (RU_i). This factor prioritizes tasks which take a large amount of resources compared to the amount of resources available in their time window. This resource utilisation can be defined as:

$$RU_i = \frac{1}{v} \sum_{k=1}^{v} \frac{\sum_{e=1}^{p_i} b_{i,k,e}}{\sum_{t=ES(i)}^{LC(i)} B_{k,t}} \tag{7}$$

However, in CPRU the resource utilization also takes into account the successors of a task. Thus an extended resource utilization $RU_i^{'}$ is used instead, which takes into account the resource utilization of a task and its successors. Weighting their own usage with weight

ω_1 and the successors with ω_2 as follows:

$$RU_i^{'} = \omega_1 \cdot |D_i| \cdot RU_i + \omega_2 \cdot \sum_{j \in D_i} RU_j \qquad \text{where } j \in D_i \leftrightarrow \exists x(i,j,x) \in E^* \tag{8}$$

Finally, both parts can be multiplied together to reach a final heuristic score that takes into account both the critical path and the resource consumption of tasks.

$$CPRU_i = CP_i \cdot RU_i^{'} \tag{9}$$

4.3 Tournament heuristics with CPRU

In the paper by Hartmann [6] which also introduced CPRU, tournaments are introduced as a way to increase diversity in the selection of variables.

Tournaments work by, instead of evaluating the heuristic for all selectable variables, only selecting a subset of variables from which the variable to be set will be chosen. Hartmann mentions in the paper that the amount of variables that are selected should be a percentage of the amount of selectable variables. This makes sense, as otherwise the heuristic may perform very differently between large and small instances and in particular may become very selective for large instances. The factor which determines the number of selected variables is aptly named the tournament factor $\phi \in [0, 1]$. The number of evaluated variables Z, given K eligible variables is then calculated as follows: $Z = \max\{\phi \cdot |K|, 2\}$, with Z being rounded to an integer. The original paper does not define which type of rounding, but in this paper we presume rounding happens to the nearest whole number.

4.4 Conflict-Driven Clause Learning

Conflict-Driven Clause Learning (CDCL) is a technique to solve SAT problems by adding new implied constraints. CDCL maintains an implication graph based on the constraints. This graph encodes disjunctions between variables as pairs of implications and each implication forms an edge in the graph between two variables. CDCL can be described using the following four-step repeating process:

1. Select a variable and assign it a value
2. Apply propagation to the constraints to remove illegal assignments from the domains.
3. Adapt the implication graph.
4. If there is a conflict, do conflict analysis:
    - Find the variables that caused the conflict.
    - From these variables create new implied constraints. There are various methods to do this, such as 1-UIP or ReLSat [18].

Note that in the first step of finding the conflict, where the variables that caused the conflict are found, this can be done by looking at the incoming edges for this variable in the implication graph. Each source of such an edge is a conflict-causing variable.

4.5 Variable State Independent Decaying Sum

Variable State Independent Decaying Sum (VSIDS) is a variable- and value-selection heuristic, originally created by Moskewicz [15] for the Chaff SAT solver. VSIDS is based on the idea that satisfying (recent) conflicts in the constraints leads to a better selection strategy. That is, if a variable seems to appear in a lot of conflicts, it must be more important than a

variable which rarely appears. VSIDS is often combined with CDCL as mentioned in section 4.4 as it allows for a very low overhead when combining the clauses from CDCL with VSIDS [15]. VSIDS can be described with the following five steps:

1. Each value in each variable and each variable is given a counter, initialised at 0.
2. When an implied constraint is added, all counters of variable values which appeared in the conflict analysis are incremented, as well as their respective variable counters.
3. Whenever an unassigned variable has to be set, the variable with the highest counter will be selected and its value set to the value of this variable with the highest counter.
4. Ties are broken randomly, though this may differ per implementation.
5. Occasionally, all counters are divided by a constant.

5 CPRU-a

In this section we introduce CPRU-a, an adapted version of CPRU that takes into account the shift in resource availability as starting times are assigned. CPRU-a replaces the $B_{k,t}$ term in the $RU_i^{'}$ with a dynamically updating available resource term $C_{k,t}$ where $C_{k,t}$ is the amount of resource k available at time t after removing resources from activities for which the domain of the start time has a size of one, meaning the activity was either set by the variable selector previously or decreased in size by propagation. The formula for RU_i under CPRU-a thus becomes as follows:

$$RU_i = \frac{1}{v} \sum_{k=1}^{v} \frac{\sum_{e=1}^{p_i} b_{i,k,e}}{\sum_{t=ES(i)}^{LC(i)} \mathbf{C_{k,t}}} \tag{10}$$

We believe CPRU-a may lead to a quicker decrease in makespan as it enables the resource utilisation part of CPRU to make a more accurate estimation of how good the resource utilisation is and it may make the estimation more reliable over time since in CPRU, the RU score may not be fully representative anymore once large chunks of resources are allocated to tasks.

Figure 3 shows an example of the different resource availabilities and their adjustment after a task is scheduled. We have that 3 tasks need to be scheduled, for which, for simplicity's sake, the start time has already been narrowed down to a single number (normally this would mean the tasks are already scheduled so we wouldn't need to select any of these tasks). When looking at the resource usage we can see that task 1 takes 6 out of the available 8 resources during its possible time window. Task 2 takes 3 out of 10 and task 3 takes 2 out of 4. Thus RU values would be 0.75, 0.3 and 0.5 respectively for tasks 1, 2 and 3. Thus task 1 would get scheduled first. After this scheduling, is where CPRU-a now updates the availability of the resource. This leads to the RU values staying the same for CPRU whilst for CPRU-a the values now update and task 2 now gets a value of 0.75 indicating that the bounds on the resource are very tight. Thus with CPRU task 3 would be scheduled first, whilst with CPRU-a task 2 would be scheduled first.

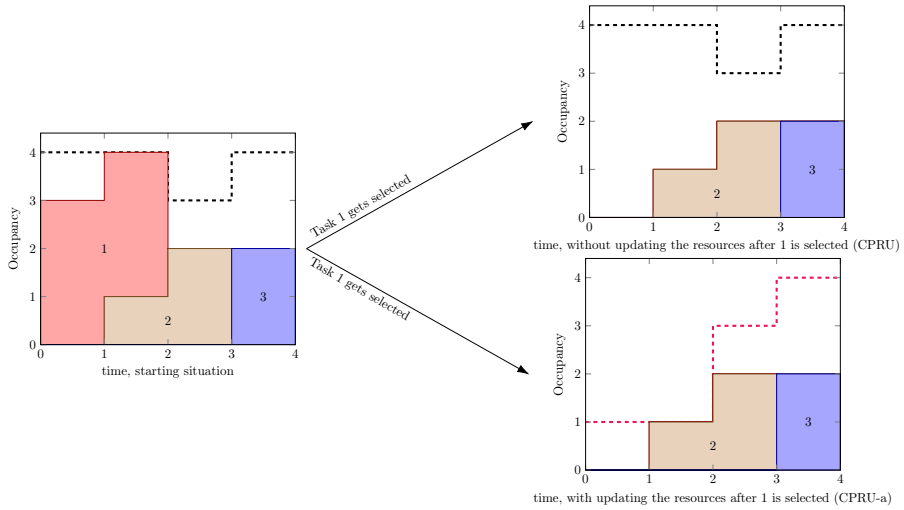


Figure 3: Selection difference between CPRU and CPRU-a

6 Experimental Setup and Results

We will compare three different versions of CPRU, namely CPRU without tournaments, CPRU with tournaments and CPRU-a against each other and VSIDS. We are mainly interested in the ability of each of these heuristics to generate optimal schedules, which we will test for each of these heuristics in two ways. Firstly we allow each heuristic to generate up to a maximum of 10 generated schedules with a time limit of 9 hours for the 10 schedules combined. We then evaluate the makespan of the best schedule generated. This allows us to compare the ability of the different heuristics to generate good schedules. Secondly, we give the heuristics 60 seconds to generate a schedule with an as small as possible makespan. This should give an indication to the performance in terms of time for generating schedules. 60 seconds was chosen as the time limit since from preliminary testing it was concluded that a large amount of test instances could generate some solution within 60 seconds without too many instances getting into an optimal solution.

6.1 Test generation

For these tests, we use the test sets J30 and J120 by Kolisch and Sprecher [19] which were adapted by Hartmann [6] for RCPSP-t.

Hartmann adapted the J30 and J120 test sets for RCPSP-t by first replacing each resource r_{jk} with a list of resources $r_{jk1}, \ldots, r_{jkp_j}$. Each resource capacity was also replaced by a list of capacities $R_{k1}, \ldots, R_{kT}$ where $T = \sum_j pj$ is the sum of all durations.

After these replacements for each period in $t = 1, \ldots, T$ the availability R_{kt} of the resource $k = 1, \ldots, K$ has a probability of P^R of being set to $R_k \cdot F^R$ and a probability of $1 - P^R$ being set to R_k, where P^R and F^R are parameters for setting the probabilities and strengths of the reductions on the resources, respectively.

For the activities, an approach similar to that of the resource availabilities is used. For each activity $j = 1, \ldots J$ and each period $t = 1, \ldots, p_j$ the request r_{jkt} for resource k has a P^r probability of being assigned $r_{jk} \cdot F^r$ and a $1 - P^r$ probability of being assigned r_{jk},

where, again, P^r and F^r are parameters for setting the probabilities and strengths of the reductions on the activities, respectively.

Each of the test sets (J30, J120) has 6 different subsets (denoted j30t1, ..., j30t6 and j120t1 ...j120t6) with varying reduction factors on resource capacities and usages, and different probabilities of resource and usage reductions. Table 2 from [6] shows the parameters used for the 6 different subsets. The J30 subsets each have 480 instances, whilst the J120 subsets each have 600 instances. The instances in the J30 subsets contain 30 tasks to be scheduled and the instances in the J120 subsets contain 120 tasks to be scheduled.

Set no.	1	2	3	4	5	6
$P^R = P^r$	0.05	0.1	0.2	0.05	0.1	0.2
$F^R = F^r$	0	0	0	0.5	0.5	0.5

Table 2: Parameter settings for generation of test sets. Adapted from [6].

6.2 Test setup

For CPRU we use the same parameters as proposed by [6], namely setting the tournament factor $\phi = 0.5$ and setting $\omega_1 = 0.4$ and $\omega_2 = 0.6$. For variable selection, we select the smallest value in the domain. For CPRU-a we re-compute the amount of available resources based on the already assigned variables, each time a variable selection needs to be made.

For VSIDS we use a solution-guided value selector [20], a relatively new value selector designed for use with VSIDS.

Each of these heuristics was implemented into the pumpkin solver provided by our supervisors, which uses Conflict-Driven Clause Learning as described in 4.4 with the constraints as mentioned in 2.2. In particular, it uses Lazy Clause Generation (LCG) [21] as described in [22] to generate new clauses. For the conflict analysis, it uses the 1-UIP (Unique Implication Point) learning scheme, which searches for a node in the implication graph such that all paths from the decision variable to it go through it [23].

Since the optimal solutions for the test generation as described in section 6.1 are not known we will compare solutions to the lower bound LB/t as given by $l_{0,n+1}$.

To make the tests as fair as possible, we run them on the high-performance CPU cluster DelftBlue [24]. Specifically on the "compute type-a" nodes. This should help prevent performance loss from other processes interfering with the tests by using a workload manager to compartmentalize resources. An exception is made for the maximum 10 schedules generated for the J30 instances. These we run on an AMD Ryzen 5 3600X 3.80GHz, running Windows Subsystem for Linux (WSL) instead. This choice was made to alleviate some pressure from DelftBlue which was, at the time of writing, very congested. Jobs that were not able to be completed within the 9-hour time limit on the AMD Ryzen 5 were rerun on DelftBlue to prevent performance from being an issue. To confirm if there was no difference between DelftBlue and the AMD Ryzen 5, the makespan result of 5 jobs which were run on both were compared and no difference was found.

6.3 Results

In table 3 we can see the results of running on the adapted J30 test set and in table 4 the results of running on the adapted j120t1 test set for a maximum of 10 iterations or 9 hours. The other J120 test sets were not considered as there was not enough computing capacity

available to run all of the test sets within a reasonable time. In tables 5 and 6 we see the result for running a maximum of 60 seconds. Each table describes the deviation in per cent from the lower bound on the optimal solution, so 0% would mean a result equal to the lower bound, thus a lower result is better. The J30 and J120 test sets contain unsatisfiable instances, these have been removed from the results.

$P^R = P^r$		0.05	0.1	0.2	0.05	0.1	0.2	All
$F^R = F^r$		0.05	0.1	0.2	0.05	0.1	0.2	All
Max. schedules generated	Rule	Average deviation from lower bound LB/t						
10	CPRU-a	37%	52%	77%	25%	31%	39%	43%
	CPRU-no-t.	37%	52%	77%	25%	31%	39%	43%
	CPRU	37%	52%	77%	25%	31%	39%	43%
	VSIDS	89%	93%	98%	89%	91%	92%	92%

Table 3: Deviation from the lower bound for VSIDS and CPRU(-a) with and without tournament heuristic for $J = 30$ when generation max 10 schedules

$P^R = P^r$		0.05
$F^R = F^r$		0.05
Max. schedules generated	Rule	Average deviation from lower bound LB/t
10	CPRU-a	37%
	CPRU-no-t.	37%
	CPRU	37%
	VSIDS	145%

Table 4: Deviation from the lower bound for VSIDS and CPRU(-a) with and without tournament heuristic for $J = 120$ when generation max 10 schedules

In table 3 we see that for the J30 instances, the CPRU variations seem to generate better schedules in 10 iterations than VSIDS. This makes sense as VSIDS does not have any preexisting knowledge about the problem and thus will start very unguided making small improvements, with which it very quickly reaches the 10 schedules limit. The CPRU variations on the other hand are already optimizing by trying to schedule tasks as early as possible and are thus generating schedules with a smaller makespan, though it is possible they need more time.

We see the same happening in 4 where for the j120t1 instances again VSIDS is vastly outperformed by the CPRU variations. For these larger instances, we still think the previous reasoning of VSIDS just trying to find solutions, where CPRU is already looking for optimal ones, is very much applicable.

In both tables, there seems to be little performance difference between the three CPRU variations. This seems to indicate the tournament and adaptive features do not matter much in terms of generating good schedules early on. For the tournament feature, this is most likely because with only 10 schedules being generated very few tournaments are actually happening and thus the influence is very small. For the adaptive feature, since this feature

primarily influences the selection after many start times have already been set, and thus the bound on resources adjusted, it would have little influence on smaller instances like J30 where at the end when there are few variables left, the selection order may matter less. It is interesting however that for large instances CPRU-a also seems to neither improve nor deteriorate performance. This may suggest that the ability of CPRU to predict what variable should be set next is already very good, as it does not benefit from further precision of the heuristic.

$P^R = P^r$		0.05	0.1	0.2	0.05	0.1	0.2	All
$F^R = F^r$		0.05	0.1	0.2	0.05	0.1	0.2	All
Time limit	Rule	Average deviation from lower bound LB/t						
60s	CPRU-a	37%	52%	77%	25%	31%	39%	44%
	CPRU-no-t.	37%	52%	77%	25%	31%	39%	44%
	CPRU	37%	52%	77%	25%	31%	39%	44%
	VSIDS	37%	52%	77%	25%	31%	39%	44%

Table 5: Deviation from the lower bound for VSIDS and CPRU(-a) with and without tournament heuristic for $J = 30$ for generating within 60 seconds

$P^R = P^r$		0.05	0.1	0.2	0.05	0.1	0.2	All
$F^R = F^r$		0.05	0.1	0.2	0.05	0.1	0.2	All
Time limit	Rule	Average deviation from lower bound LB/t						
60s	CPRU-a	45%	57%	84%	36%*	40%	46%	52%
	CPRU-no-t.	45%	57%	84%	37%	40%	46%	52%
	CPRU	45%	57%	84%	37%	40%	46%	52%
	VSIDS	121%*	115%*	109%*	122%*	121%*	115%*	117%

* In these test sets the algorithm failed to generate some schedules within 60 seconds. These instances are not considered for the result of these algorithms.

Table 6: Deviation from the lower bound for VSIDS and CPRU(-a) with and without tournament heuristic for $J = 120$ for generating within 60 seconds

In tables 5 and 6 we see the result for running the selection algorithm with the solver for 60 seconds and looking at the best schedule generated. For the J30 test sets in table 5 all results are the same. This is mainly because a large percentage of the instances, for the j30t3 test set even 100%, are getting solved to an optimal solution. This means that the extra time the solver has left at that point is wasted. For the J120 result in table 6 however, just like with the generation limit, the lower bound deviation on the CPRU variations is very similar, whilst the deviation on VSIDS is much higher. There were also some test cases for which VSIDS was unable to generate any solution. This again seems to confirm that VSIDS behaves very unguided and does not quite know how to start searching in the search space whilst the CPRU variations are already very focused on building a solution by trying to schedule tasks and thus making the problem smaller instead of trying to do so by solving arbitrary constraints.

Again, we see that the difference between the CPRU variations is very small. This seems to suggest that the benefit gained from the adaptation is very small if present at all. We again think this is largely because CPRU-a primarily becomes useful once many tasks have already been assigned a starting time, but at that point, constraint propagation may already solve the problem.

To try and explain why CPRU and CPRU-a performed similarly, we ran an additional experiment. The j30t1 and j120t1 were rerun for 60 seconds, but this time it was logged which variable was selected. We then compared the selected variables between the CPRU variations and calculated the average number and percentage of selections until the two algorithms made a different choice. Between CPRU and non-tournament CPRU this amounted to less than 1% for both test sets, and on average a change in the first or second variable. For CPRU-a the average was instead after about 40% of total choices, or an average of 25 choices before a different variable was selected, with 136 instances being the exact same as the CPRU version, in the j30t1 set. For j120t1 this was slightly lower at 3% of the total or after on average 17 variables were selected. The percentage is most likely lower, just because more variables were selected in total. However, this still seems to confirm the idea that CPRU-a primarily becomes useful once many tasks have already been assigned a starting time as for small instances it appears to have little influence over the variable selection compared to CPRU. Many of the choices CPRU-a makes, when there are relatively few variables solved, seem to be the same as that of CPRU.

Based on these results we conclude there is no significant difference in performance for CPRU with or without tournament selection. We also conclude CPRU-a may not significantly improve nor deter performance in generating better schedules as its performance is very much in line with all other version of CPRU. Based on the result we also conclude that CPRU may perform better at generating better schedules within fewer iterations or less time than VSIDS, primarily for large instances. We think this is likely because of the way CPRU was specifically designed for RCPSP-t and thus it makes optimal use of the extra knowledge it has to solve the points where problems are going to be generated first, mainly around large resource-consuming tasks, whereas VSIDS does not have any of this knowledge.

7 Responsible Research

The Netherlands Code of Conduct [25] outlines five main principles; honesty, scrupulousness, transparency, independence and responsibility, each of which we deem justifiable in this research.

Once the paper by our supervisors has been released, the model will also be made available to allow for public scrutiny of the tests and the performance of the model. This allows third parties to more easily reproduce this research and increases its transparency. This paper also describes in detail the methods used in the solver, so other researchers can also reproduce it on their own.

To increase transparency the test set and results have also been made available[1]. This together with the previous point allows for a very close replication of this study. This should also help future research in comparing different heuristics by allowing for a homogeneous testing environment.

In light of responsibility, where possible tests were run on local hardware to allow other researchers to also make use of resources, such as the supercomputer used in this project,

[1] `https://github.com/Wouter17/cpru-results`

ensuring the research limits its harm to other researchers.

Great care has also been taken to make the testing as fair as possible. By running on a high-performance CPU cluster provided by the university, inconsistencies due to hardware should be minimized, as the cluster uses SLURM, a workload manager, which compartmentalizes different tests and assigns them their own resources, preventing tests from interfering with each other.

All in all, we believe good ethical principles have been upheld during this research and we do not see any direct unethical or harmful implications of this research.

8 Conclusions and Future Work

In this paper, we looked at the workings of VSIDS and different variations of CPRU and compared their ability to generate good schedules given some time. We also introduced a new variation of CPRU, CPRU-a which tried to improve schedule generation by keeping track of used resources. No significant results affirming an improvement or worsening of CPRU-a over CPRU were found. For large instances, it was concluded however that CPRU may generate schedules with a lower makespan in fewer iterations than VSIDS.

With this in mind, we think future efforts may be put into looking at applying CPRU in other subproblems of RCPSP such as the Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP) where multiple modes could also be considered using the resource utilisation factor. We also think further research is needed into adapting high-performance (meta-)heuristics such as VSIDS for problem-specific instances, like RCPSP-t in this case. With some extra information, these algorithms may keep their good performance in terms of generating schedules in very little time and with additional information the generated schedules may converge to an optimal schedule faster. Finally, we think more testing of CPRU-a may show some usefulness when dealing with very large instances of RCPSP-t.

References

[1] 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, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.05.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221721003982.

[2] V. F. Cavalcante, C. H. Cardonha, and R. G. Herrmann, "A resource constrained project scheduling problem with bounded multitasking", *IFAC Proceedings Volumes*, vol. 46, no. 24, pp. 433–437, 2013, 6th IFAC Conference on Management and Control of Production and Logistics, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20130911-3-BR-3021.00084. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016322273.

[3] H. H. Adelsberger and J. J. Kanet, "The leitstand: A new tool in computer-integrated manufacturing", en, *Production and Inventory Management Journal*, vol. 32, no. 1, pp. 43–48, 1991, ISSN: 0897-8336.

[4] J.-B. Sciau, A. Goyon, A. Sarazin, J. Bascans, C. Prud'homme, and X. Lorca, "Using constraint programming to address the operational aircraft line maintenance scheduling problem", English, *Journal of Air Transport Management*, vol. 115, 2024, ISSN: 09696997. DOI: 10.1016/j.jairtraman.2024.102537. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85183567939&doi=10.1016%2fj.jairtraman.2024.102537&partnerID=40&md5=a753fe70aca8e0bf3cd7ba2f18c4d542.

[5] R. Penha, L. A. d. C. Guerrazzi, D. C. T. d. Andrade, and R. F. Cintra, "Produção científica sobre resource-constrained project scheduling problem: Um estudo bibliométrico e bibliográfico", *Revista De Gestão E Projetos*, vol. 08, pp. 71–86, 02 2017. DOI: 10.5585/gep.v8i2.488.

[6] S. Hartmann, "Project scheduling with resource capacities and requests varying with time: A case study", English, *Flexible Services and Manufacturing Journal*, vol. 25, no. 1-2, pp. 74–93, Jun. 2013. DOI: 10.1007/s10696-012-9141-8. [Online]. Available: https://www.proquest.com/scholarly-journals/project-scheduling-with-resource-capacities/docview/1271881162/se-2.

[7] J. Marques-Silva, "The impact of branching heuristics in propositional satisfiability algorithms", in *Progress in Artificial Intelligence: 9th Portuguese Conference on Artificial Intelligence, EPIA'99 Évora, Portugal, September 21–24, 1999 Proceedings 9*, Springer, 1999, pp. 62–74.

[8] S. Hartmann, "Time-varying resource requirements and capacities", in Oct. 2015, pp. 163–176, ISBN: 978-3-319-05442-1. DOI: 10.1007/978-3-319-05443-8_8.

[9] M. Bofill, J. Coll, J. Suy, and M. Villaret, "Smt encodings for resource-constrained project scheduling problems", *Computers & Industrial Engineering*, vol. 149, p. 106 777, 2020, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106777. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220304873.

[10] M. Buro and H. K. Büning, *Report on a SAT competition*. Fachbereich Math.-Informatik, Univ. Gesamthochschule Paderborn, Germany, 1992.

[11] J. W. Freeman, *Improvements to propositional satisfiability search algorithms*. University of Pennsylvania, 1995.

[12] R. G. Jeroslow and J. Wang, "Solving propositional satisfiability problems", *Annals of mathematics and Artificial Intelligence*, vol. 1, no. 1, pp. 167–187, 1990.

[13] E. W. Davis and J. H. Patterson, "A comparison of heuristic and optimum solutions in resource-constrained project scheduling", *Management Science*, vol. 21, no. 8, pp. 944–955, 1975, ISSN: 00251909, 15265501. [Online]. Available: http://www.jstor.org/stable/2629856 (visited on 05/01/2024).

[14] R. Alvarez-Valdes and J. Tamarit, "Heuristic Algorithms for resource-constrained project scheduling: A review and an Empirical analysis", in *Advances in project scheduling*, Amsterdam, nl: Elsevier, 1989, pp. 113–134. DOI: 10.1016/c2009-0-08770-4. [Online]. Available: https://doi.org/10.1016/c2009-0-08770-4.

[15] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver", in *Proceedings of the 38th Annual Design Automation Conference*, ser. DAC '01, Las Vegas, Nevada, USA: Association for Computing Machinery, 2001, pp. 530–535, ISBN: 1581132972. DOI: 10.1145/378239.379017. [Online]. Available: https://doi.org/10.1145/378239.379017.

[16] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling", *Naval Research Logistics (NRL)*, vol. 45, no. 7, pp. 733–750, 1998. DOI: https://doi.org/10.1002/(SICI)1520-6750(199810)45:7<733::AID-NAV5>3.0.CO;2-C. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291520-6750%28199810%2945%3A7%3C733%3A%3AAID-NAV5%3E3.0.CO%3B2-C. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291520-6750%28199810%2945%3A7%3C733%3A%3AAID-NAV5%3E3.0.CO%3B2-C.

[17] A. Schrijver, *Combinatorial optimization: Polyhedra and Efficiency*. Springer, 2003, vol. A, ISBN: 3-540-44389-4.

[18] A. Nadel, "Understanding and improving a modern sat solver", Ph.D. dissertation, Tel Aviv University, Aug. 2009. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fedce311475ed06ca876190b40a175824a564758.

[19] R. Kolisch and A. Sprecher, "Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program", *European Journal of Operational Research*, vol. 96, no. 1, pp. 205–216, 1997, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(96)00170-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221796001701.

[20] E. Demirović, G. Chu, P. J. Stuckey, S. of Computing, and A. Information Systems University of Melbourne, "Solution-Based Phase Saving for CP: A Value-Selection Heuristic to Simulate Local Search Behavior in Complete Solvers", Tech. Rep., 2017. [Online]. Available: https://people.eng.unimelb.edu.au/pstuckey/papers/lns-restarts.pdf.

[21] O. Ohrimenko, P. J. Stuckey, and M. Codish, "Propagation via lazy clause generation", *Constraints*, vol. 14, pp. 357–391, 3 2009. DOI: 10.1007/s10601-008-9064-x.

[22] T. Feydy and P. J. Stuckey, "Lazy clause generation reengineered", *Principles and Practice of Constraint Programming - CP 2009*, pp. 352–366, 2009. DOI: 10.1007/978-3-642-04244-7_29.

[23] R. Tichy and T. Glase. "Clause learning in SAT". (Apr. 2006), [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall13/cos402/readings/SAT_learning_clauses.pdf.

[24] Delft High Performance Computing Centre (DHPC), *DelftBlue Supercomputer (Phase 2)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[25] N. C. of Conduct for Research Integrity, "Netherlands Code of Conduct for Research Integrity", Tech. Rep., 2018. [Online]. Available: https://www.nwo.nl/sites/nwo/files/documents/Netherlands%2BCode%2Bof%2BConduct%2Bfor%2BResearch%2BIntegrity_2018_UK.pdf.