A heuristic-guided constraint programming approach to PRCPSP-ST

Using priority-rules to guide constraint solvers

Codrin Ogreanu[1]
Supervisor(s): Emir Demirović[1]**, Imko Marijnissen**[1]**, Maarten Flippo**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

Abstract

This paper introduces a new approach to the Preemptive Resource Constrained Project Scheduling Problem with setup times. The method makes use of a Constraint Optimization Problem solver, which has been modified to use priority-rule-based heuristics in its variable and value selection procedures. An alternative implementation which uses a combination of a priority-rule heuristic and a domain-independent solver heuristic has also been investigated. Both methods were tested on four well-known problem sets against a problem-independent solver configuration. Experimental results show a significant reduction in the time needed to find optimal solutions for the new methods.

1 Introduction

The Resource Constrained Project Scheduling Problem (RCPSP) is a well-known problem in Operations Research (OR). It involves scheduling a set of activities, subject to resource and precedence constraints, in order to minimize an objective function, such as the duration of the project from the start of the first activity to the end of the last, commonly referred to as the makespan. Due to the problem being NP-Hard [1], and having applications in project management and industrial processes, it has been widely studied [2].

There are many variants to RCPSP, all representing different constraints or liberties that are allowed during the scheduling process. One particular extension to RCPSP is preemptive RCPSP with setup times, or PRCPSP-ST [3, 4]. In this version, activities can be stopped and resumed at a later point. This can result in more efficient final schedules, with the caveat that a setup time is required before a preempted activity can be resumed. An example of pre-emptability allowing for better schedules is splitting up long activities to fill gaps in the schedule. This extension has largely been ignored until recently, despite pre-emtability being a feature of many project management tools [5]. It has been argued that accounting for pre-emption is inevitable, and indeed beneficial in areas such as the textile industry, or multiprocessor scheduling [6].

Over the past few decades, there have been multiple approaches to the base problem and its extensions. These include exhaustive search methods, such as boolean satisfiability solvers [7, 8], Constraint Programming (CP) [9], or branch-and-bound methods [6, 10]. Others focus on heuristic and meta-heuristic algorithms, such as schedule generation schemes with priority rules [11], genetic algorithms [12, 13], simulated annealing [14], and tabu search [11]. The state-of-the-art approach to PRCPSP-ST involves alternating between a boolean satisfiability solver and a genetic scheduling algorithm [8].

However, one approach that has not been fully explored for PRCPSP-ST specifically is using exhaustive search methods, such as Constraint Programming. This can be attributed to the significantly larger search space that pre-emptability causes. As a result, hybrid methods that use both exhaustive search and metaheuristics are often preferred [8]. While for this purpose, techniques such as genetic algorithms have been the preferred approach, Ruiz and Stützle argue that even simple heuristics can outperform complex metaheuristics [15]. In this paper, we show that such heuristics can help quickly guide solvers towards satisfiable and even optimal solutions, mitigating the issues caused by the increased search space.

To this end, this paper presents two main contributions. Firstly, the inclusion of domain-specific information into a constraint solver is investigated. To achieve this, two priority rule heuristics, Greatest Resource Demand (GRD) and dynamic Earliest Start Time (dEST), were used. Both are already established in the context of RCPSP [16, 11]. They are used in the value and variable selection procedures of the constraint solver. They are used to guide the solver into scheduling activities which are more expensive to pre-empt first, which allows smaller activities to fill in gaps in the schedule. Secondly, the resulting method is applied to the PRCPSP-ST problem.

We compared our method against a baseline solver configuration, which uses two problem-independent CP branching heuristics, Variable State Independent Decaying Sum (VSIDS) [17] and solution-guided Phase Saving [18]. We also introduce an additional method, which uses VSIDS and dEST as its selection procedures, and which we compare against our two previous methods. We show that the constraint solver configurations which incorporate domain-specific information outperform the baseline by multiple metrics.

The outline of the paper is as follows. In section 2, the formulation for the RCPSP and PRCPSP-ST problems is provided. Section 3 gives an overview of the previous work on PRCPSP-ST. In section 4, any preliminaries required for the main method are covered. Section 5 presents the algorithmic details of the heuristics, with the experimental setup and results being discussed in section 6. The ethical aspects of this research are covered in section 7. Finally, section 8 contains the concluding remarks of the paper and potential future work.

2 Problem Description

In section 2.1, we provide a mathematical formulation of RCPSP and PRCPSP-ST. The latter is discussed in more detail in section 2.2, where the methods by which pre-emptability can be formulated are covered.

2.1 Formal Description

RCPSP can be formulated as an activity-on-node problem: there is a set of activities $J = \{j_0, j_1, j_2, \ldots, j_n, j_n + 1\}$ which need to be scheduled. Activities j_0 and $j_n + 1$ are fake activities that act as the source and sink of the project. There is also a set of activity pairs P that represents the precedence constraints between activities. Together these sets form an acyclic graph $G(J, P)$. Finally, there is a global pool of renewable resources $R = \{r_1, r_2, \ldots, r_k\}$.

Each resource has a maximum capacity a_r. Every activity j uses u_{jr} resources for the duration d_j it is being processed, for every $r \in R$. For every activity pair $(a, b) \in P$, the end-time of the first activity must come before the start time of the second activity, i.e. $s_a + d_a \leq s_b$. Finally, during every time unit of the project, for any given resource, the total use must not exceed the capacity:

$$\forall t \in M, \forall r \in R : \sum_{j \in J} \sum_{i=s_j}^{i<s_j+d_j} \delta_{ti} \cdot u_{jr} \leq a_r$$

, where M is the makespan of the project, and δ_{ti} is the Kronecker delta. The goal is to schedule every activity such that the project makespan is minimized. This is equivalent to minimizing the finish time of j_{n+1}. A glossary of all of the acronyms used to formally describe the problem can be seen in table 1.

An example of an RCPSP instance can be seen in fig. 1. The problem has 4 activities and one resource type with capacity 2. The precedence constraints, activity durations and resource consumptions are illustrated in fig. 1a. An optimal schedule is constructed in

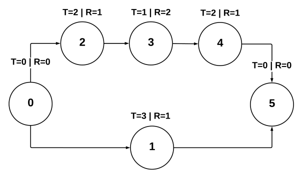
Table 1: Glossary of used acronyms.

Acronyms	Definitions
d_j	Duration of activity j
a_r	Maximum capacity of resource r
u_{jr}	Amount of resource r needed by j
s_j	Scheduled start time of activity j
e_j	Scheduled end time of activity j
J	Set of all activities
P	Set of precedence constraints between activities
R	Set of all resources
S_j	Set of scheduled activities
M	The makespan of the project
T_j	The tuple of task slices assigned to activity j

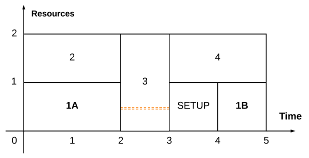
fig. 1c. Without pre-emption, the resource usage cannot be fully optimized, resulting in a project makespan of 6.

PRCPSP-ST includes the following extensions to the original formulation: each activity j_i may be split at any integer point into two sub-activities, j_i^0 and j_i^1. If activity j_i^0 has the duration $d_i^0 < d_i$, then j_i^1 has $d_i^1 = d_i - d_i^0 + t^s$, where t^s is the setup-time needed to resume the preempted activity. Both activities have the same resource requirements as the original activity. Additionally, a new precedence relationship is introduced, with j_i^0 having to precede j_i^1. These sub-activities can, in turn, be split again, as long as the resulting sub-activities have non-unit durations.

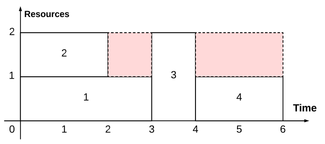
An optimal schedule of the problem in fig. 1 can be seen in fig. 1b. Because activity 1 can be split into activities 1a and 1b, the project makespan can be lowered to 5, even when accounting for setup times.



(a) Activity graph. Activities 0 and 5 are the mock "source" and "sink" activities. Arcs indicate precedence constraints. Duration and resource constraints for each activity are located above the nodes.



(b) Project schedule under PRCPSP-ST. Activity 1 is preempted, which results in a schedule with a makespan of 5.



(c) Project schedule under RCPSP. Without pre-emption, the available resources cannot be fully utilized, resulting in a schedule with a makespan of 6

Figure 1: A project instance with 4 activities and a single resource type with a capacity of 2.

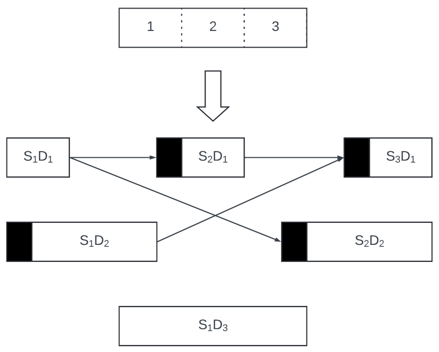
2.2 Representing Pre-emptability

One possible encoding of the pre-emptability of activities is to represent them as *activity segments* [8]. In this representation, every possible sub-activity is enumerated, with setup times being added to every sub-activity not containing the start of the original activity. As these setup times are static during the scheduling process, this representation will be referred to as static encoding. Finally, new precedence relations are drawn between the sub-activities, such that any valid path reconstructs the original activity. An example can be seen in fig. 2a.

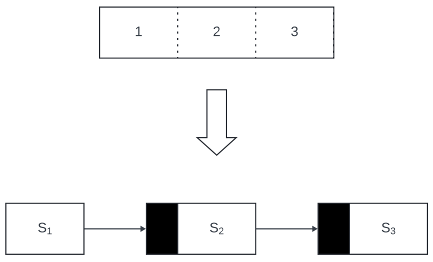
One advantage of this method is its simplicity; after all of the activities are encoded this way, the problem can be treated as an RCPSP instance. The downside is the greatly increased activity count, as every activity generates $\sum_{i=1}^{i \leq d_j} i$ sub-activities. For instances with long activities, this increase in the search space becomes prohibitive.

Another encoding was introduced by Dr. Emir Demirović, Imko Marijnissen and Maarten Flippo, and is implemented in the constraint solver used in the experimental setup. It instead splits every activity into d_j *task slices*, each with a duration of 1, forming a chain using the added precedence constraints. Whether pre-emption occurs under this encoding is instead decided during the runtime of the scheduling algorithm: if two sub-activities are scheduled next to each other, the setup time is ignored. For this reason, the encoding will be referred to as dynamic encoding. An example of this can be seen in fig. 2b.

While this representation incurs a runtime overhead for deciding if the setup time needs to be accounted for, it ultimately leads to a significantly smaller search space. For this reason, this will be the encoding method used for the rest of this paper.



(a) The static activity segment representation [8]



(b) The dynamic activity segment representation

Figure 2: An activity with $d_j = 3$ encoded in static and dynamic forms. The setup times are represented by the filled rectangles preceding activities.

3 Related Work

Blazewicz et al. were the first to prove that the base problem of RCPSP was NP-Hard [1]. Kaplan was the first to discuss the problem of preemptive RCPSP in her doctoral dissertation [3], and later the problem of RCPSP with setup times [4]. She formulated the problem of PRCPSP as a dynamic programming one, which incorporated lower and upper bounds.

Demeulemeester and Herroelen proved that Kaplan had made an incorrect assumption, which caused her algorithm to miss the optimal solution on certain problem instances [10]. They instead proposed a branch-and-bound procedure that made use of five dominance rules. Their experiments on the *Patterson* problem instance set showed a 33-fold increase in computation time compared to the non-preemptive variant, with minimal makespan improvements. Because of this, they concluded that preemption had a negligible effect, even if setup times are not included.

However, it was shown that the Patterson set had two main flaws: the instances were not consistent, and in particular some instances with the same number of activities greatly differed in how difficult they were to solve [19]. This made the set unsuitable for experiments, in particular regarding PRCPSP [5]. This suggested that the previous assumptions about PRCPSP could have been incorrect, and should be re-evaluated. Indeed, Ballestín et al. [5], using schedule generation scheme methods showed that preemption led to a significant decrease in the project makespan, even with a maximum of only one interruption per activity.

Following this, multiple techniques for optimizing PRCPSP were investigated. Peteghem and Vanhoucke [12] developed a bi-population genetic algorithm for the multi-mode extensions of PRCPSP to great results. Moukrim et al. [6] introduce a branch-and-price approach, which includes an antichain linear program [20], a method similar to Constraint Programming. It additionally incorporated an Interval Order Enumeration algorithm, and constraint propagation. They were able to solve multiple datasets to optimality, as well as improve the lower bounds for some unsolved instances.

Constraint Programming based approaches have been largely foregone in the context of PRCPSP. This can be attributed to the large increase in the total search space which activity pre-emption introduces. Because of this, "pure" CP methods have been primarily applied to the base problem of RCPSP. As an example, Schutt et al. [9] use a constraint solver implementing the "cumulative" propagator and VSIDS variable selection heuristic in the context of RCPSP. They conclude that the obtained results show that their method can "compete with highly specialised Rcpsp solving approaches".

Heuristic-based approaches to RCPSP come primarily in the form of priority-rule-based heuristics. These are combined with schedule generation schemes (SGSs) in order to construct efficient schedules. Kolisch and Hartmann [11] provide an overview of different schedule generation schemes, while Klein [16] discusses 25 common priority-rule heuristics used in RCPSP. However, experimental results show that, by themselves, priority-rule heuristics cannot compete with approaches such as simulated annealing [11].

As far as the author is aware, the current state of the art for PRPCPS-ST was introduced by Vanhoucke and Coelho [8]. In their paper, they first introduce five different setup time types. They then propose an algorithm that, broadly, splits PRCPSP-ST into 2 subproblems: the *activity selection* problem, and the *activity scheduling* problem. The former uses a genetic algorithm to select the optimal activity to schedule at every iteration. In contrast, the latter uses a boolean satisfiability solver to generate a feasible schedule using the newly chosen activity. They also conclude that, contrary to prior beliefs, allowing preemption did not result in unrealistic schedules with many interruptions, due to the inclusion of setup times. Their technique has shown that combining metaheuristic and exhaustive search methods can be used to blend the problem-specific search performance of the former with the ease of modelling of the latter.

To summarize, both PRCPSP-ST and RCPSP have seen many approaches since they were originally introduced. While both purely heuristic and exhaustive search-based methods have been applied to the problems, combining both currently appears to be the way of moving forward. This lays the foundation of our contribution, described in section 5, where we introduce elements of two priority-rule heuristics, GRD and dEST, into a Constraint Programming solver.

4 Preliminaries

This section introduces the preliminaries necessary for understanding the main method of this paper. It is structured as follows: In section 4.1, priority rule-based heuristics are discussed, and the most common ones are presented. Section 4.2 introduces concepts of constraint optimisation problems and solvers. Finally, how PRCPSP-ST is modelled as a constraint optimisation problem is covered in section 4.3.

4.1 Priority Rule Heuristics

Priority rules-based heuristics are commonly used in combination with schedule generation schemes (SGSs). They assign a value $p(j)$ for each activity $j \in J$. If there are multiple eligible activities during any iteration of the SGS, these heuristics are used to select the one that maximizes (or minimizes) $p(j)$.

They can primarily be classified into static and dynamic heuristics. For static heuristics, the value $p(j)$ is only calculated at the start of the algorithm and therefore remains unchanged during the runtime. In the case of dynamic heuristics, $p(j)$ is re-computed after every iteration. The former are typically very inexpensive to evaluate, while the latter generally provide a more accurate approximation of the quality of scheduling each activity.

Priority rules can broadly be broken down into four further categories [21, 16]:

- Network-based rules use the information within the overall network, such as activity durations, or the number of successors and predecessors.
- Critical path-based rules primarily use the earliest or latest start/finish times of activities.
- Resource-based rules take the resource consumption and availability of activities into account.
- Composite rules make use of multiple information types, in order to overcome their individual weaknesses.

We introduce some common priority rules, and how their $p(j)$ value is computed in table 2. Klein [16], and Kolisch and Hartmann [11] provide a more complete overview of other priority rules used with SGSs.

4.2 Constraint Programming

Constraint Programming (CP) is an exhaustive search technique. It involves modelling the problem as a set of *decision variables*, whose values represent the state of the search. Each decision variable has a domain, which contains what values it can be assigned. *Constraints* are then placed on these variables, which limit the set of acceptable solutions. During every

Table 2: Some common priority rules.

Priority Rule	Category	Priority Value $p(j)$
Greatest Rank Positional Weight (GRPW)	Network-Based	$d_j + \sum_{k \forall (j,k) \in P} d_k$
Shortest Processing Time (SPT)	Network-Based	min d_j
Earliest Start Time (EST)	Path-Based	min ES_j
dynamic Earliest Start Time (dEST)	Path-Based	min $ES_j(PS)$
Latest Finish Time (LFT)	Path-Based	min LF_j
Greatest Resource Demand (GRD)	Resource-Based	max $d_j * \sum_{r \in R} u_{jr}$

iteration of the solver, the domain of one or multiple decision variables is shrunk, until all of the domains only contain one possible value, representing a satisfiable assignment. This solution is recorded, and the solver continues searching for an optimal solution by re-expanding some of the domains. Finally, the optimality of the assignments is evaluated using a given *objective function* over the decision variables.

An important component of CP solvers is the branching strategy. It defines how the search space of the problem is explored during each iteration of the search procedure. Branching strategies are composed of *variable* and *value* selection strategies. The former represents which of the unassigned variables should be considered next. The latter instead decides which value in the variable's domain should be assigned to it, or how the domain should be shrunk. These strategies usually utilize problem-independent information, such as the size of the variable's domains. This is one strength of CP solvers, as they can easily be applied to different problem types.

4.3 Constraint Solver Formulation

The decision variables used to encode PRCPSP-ST are as follows:

- The resource usage of each activity is represented as a vector of decision variables, one for each resource: $[R_j^1, R_j^2, \ldots, R_j^n]$. As the resource usage remains static, this is done primarily for ease of use with the cumulative constraint [9].
- Each task slice $s_x \in T_j$ is represented as three decision variables: S_j^x, D_j^x, E_j^x. They represent the start time, duration, and end time of the slice.
- The task slices $s_x \in T_j^{-1}$, where T_j^{-1} refers to the tuple containing every slice of j except for the last, are assigned another decision variable, C_j^x. It is used to encode whether slices s_x and s_{x+1} have been scheduled such that they are next to each other, therefore not incurring any setup penalty.
- The makespan is represented by one additional decision variable M.

Before the constraints used to model the problem can be discussed, one special constraint must be introduced: $Cumulative(S, D, RR, c)$. This constraint was introduced specifically for scheduling problems [9]. It takes three vectors S, D, RR representing the start times, durations, and resource requirements of each activity, and a scalar c representing the total capacity. It enforces that for any time unit, the total resource use does not exceed the capacity. This constraint allows for a large part of PRCPSP-ST, and RCPSP in general, to be easily modelled.

The necessary constraints can then be represented as follows:

$$S_j^x + D_j^x = E_j^x \qquad \forall j \in J, \forall x \in T_j \tag{1}$$

$$E_j^x \leq S_j^{x+1} \qquad \forall j \in J, \forall x \in T_j^{-1} \tag{2}$$

$$C_j^x \leftrightarrow E_j^x = S_j^{x+1} \qquad \forall j \in J, \forall x \in T_j^{-1} \tag{3}$$

$$\neg C_j^x \leftrightarrow D_j^{x+1} \geq 1 \qquad \forall j \in J, \forall x \in T_j^{-1} \tag{4}$$

$$Cumulative(S, D, RR, a_r) \qquad \forall r \in R \tag{5}$$

$$E_j^n \leq S_k^0 \qquad \forall (j,k) \in P \tag{6}$$

$$M \geq E_j^n \qquad \forall j \in J \tag{7}$$

Finally, the objective function of the formulation is simply the minimization of the makespan:

$$Obj = \min(M) \tag{8}$$

5 Main Contributions

We present the method we applied to the preemptive resource-constrained project scheduling problem with setup times. It involves integrating two priority-rule heuristics, Greatest Resource Demand and dynamic Earliest Starting Time, into the variable and value selection processes of the solver. In section 5.1 the motivation behind using the two heuristics is discussed. How these priority rules are implemented in the solver is covered in section 5.2. Finally, the combination of the VSIDS heuristic with dEST is examined in section 5.3

5.1 GRD/dEST heuristic

The reasoning behind the composite GRD/dEST heuristic is to exploit the pre-emptability of activities, specific to PRCPSP-ST. After the most demanding activities are scheduled, there will inevitably be gaps in the schedule. While under normal RCPSP these gaps can generally not always be filled due to activity durations being either too long or too short, activity pre-emption can allow us to fit in activities exactly.

Both GRD and dEST, and their priority values, have been shown in table 2. GRD works by scheduling activities which require the most resources first. dEST prioritizes activities which can be scheduled first. This changes as activities are scheduled, which makes this heuristic dynamic.

Our use of dEST differs from ordinary priority-rule-based heuristics in that it is used in the value selection process, instead of the variable selection one. Instead of selecting which activity should be next inserted into the schedule, it decides where this activity should be placed.

The GRD heuristic was included in the variable selection process. By prioritizing the scheduling of variables with high resource demands, we benefit twofold: the first advantage is that the gaps left in the schedule can be filled afterwards by "smaller" activities. In ordinary RCPSP, this is not usually possible, but pre-emptability allows for more flexibility when completing these gaps. Secondly, more demanding activities are more expensive to pre-empt, due to the setup time scaling linearly with resource requirements. By prioritizing them, we avoid this cost as much as possible.

One alternative way to prioritize filling in gaps in the schedule, which had been initially considered instead of dEST, is to compute a map of the remaining resources available throughout the project. It would then schedule activities such that the remaining resources would be minimized. Mathematically, its priority value could be described as $p(j) = \min_{t \in D} rr_t$, where D is the set of all timeslots when j can be scheduled, and rr_t represents the total amount of resources still available in the time interval $t, t+1$. The value rr_t could be computed in the following way, where δ_{ti} is the Kronecker delta:

$$rr_t = \sum_{r \in R} a_r - \sum_{j \in S_j} \sum_{i=s_j}^{i<s_j+d_j} \delta_{ti} \cdot u_{jr}$$

In practice, however, this method would incur a large computational overhead, as the resource map would be re-computed for every value selection step. dEST achieves a similar effect of filling in as many gaps in the schedule as possible while being considerably less expensive to evaluate.

5.2 Implementation in the constraint solver

As mentioned in section 4.2, constraint solvers use a branching strategy to define how the search process progresses. In our implementation, the two main components, namely the variable and value selection strategies, work independently.

In the implementation of the GRD heuristic as the variable selection strategy, only two decision variable types are considered: the start and end times of the task slices, S_j^x, E_j^x. This is because once both the start and end of a slice is assigned, the duration can be inferred by the solver automatically. During the selection process, we consider the unassigned variables and select the one containing the largest total resource requirement. While this incurs a runtime overhead, it is lessened by pre-computing the total resource requirements $\sum_{r \in R} u_{jr}$ for each task slice.

The value selection process using dEST is dynamic. One advantage provided by the use of CP solvers is that the lower and upper bounds of the decision variables are updated regardless, and so selecting the smallest value can be done with almost no overhead. By prioritizing the smallest available values for the start and end times, the number of preemptions is also kept to a minimum.

5.3 Integration with VSIDS

VSIDS is a recent problem-independent variable selection heuristic, normally used in SAT solvers [22]. The main idea is to maintain counters of many times each variable was assigned, while periodically dividing them by a constant. Then, variables with the highest counters are selected. Essentially, VSIDS can infer which variables are the most important to assign first due to the large number of conflicts they have recently generated.

Because of the ease with which variable and value selection strategies can be combined into different branching strategies, we incorporated VSIDS with dEST into the VS/dEST strategy. The rationale was to utilize the performance of VSIDS, while also making use of domain-specific information.

Table 3: Datasets statistics. #Inst, #Act, #Res and $\overline{\text{Dur}}$ refer to the number of instances, activities, types of resources and the median duration of the activities, respectively.

Dataset	#Inst	#Act	#Res	$\overline{\text{Dur}}$	Makespan Bounds	
					Lower	Upper
J30	480	30	4	5	34	129
J60	480	60	4	6	44	154
J90	480	90	4	6	69	174
J120	600	120	4	5	66	234

6 Experimental Results

In section 6.1, the main characteristics of our experimental setup are discussed, such as the solver configurations, and the metrics used to quantify their performance. The obtained results are covered in section 6.2 .

6.1 Experimental Setup

Datasets: Our algorithm was tested against various instances from four well-known datasets: J30, J60, J90 and J120 [23]. The number of activities in the instances ranges from 30 to 120. These datasets have all been previously used in recent approaches to PRCPSP-ST [8, 6]. The characteristics of these datasets are shown in table 3.

Metrics: Multiple metrics were used to evaluate the performance of the solver. The first, %DEV, measures the percentage deviation of the makespan our solution found from the makespan of the optimal solution for the problem instance. Since the used optimals are for non-preemptive RCPSP, it is possible to obtain makespans that are lower than the known optimals. It was computed according to $\%DEV = (M_{opt} - M_{obs})/M_{opt} * 100$, where M_{obs} is the recorded makespan, and M_{opt} is the optimal makespan. This means that positive values of %DEV show an improvement over the optimal makespan. Negative values occur when the solver could only find a satisfiable assignment within the set timeout. A related metric is #Imp, which measures the number of instances for which a positive %DEV was found.

Another metric, %RU, represents the resource utilization of the schedule. That is, for every time step we divide the used resources by the total resources, and sum up the results. Because our method is designed to minimize unused resources, this metric can help verify that there exists a correlation between maximized resource usage and lowered project makespans.

The number of pre-emptions #PRE was used to quantify how efficiently the different methods made use of pre-emptability, as this number is directly proportional to the setup cost incurred by the schedule. It is also desirable in practice to maintain a realistic number of pre-emptions, such as in the case of project management.

Finally, we measure how quickly each method reached either the optimal solution or the best satisfiable assignment it could find. This is done through two metrics: the time T spent in the solver, measured in seconds, and the number of decisions made by the constraint solver, #DC. The latter allows for a more objective evaluation of how efficiently the solver explores the search space, independent of the quality of the heuristics' actual implementations.

Missing Optimal Solutions: Due to the hardness of RCPSP, some of the used instances do not have any known optimal solutions, only known upper and lower bounds for the makespan. These cases were ignored from the main analysis, due to the possibility of skewing data. Tables for where the upper and lower bounds were instead considered as optimals can be found in appendix A.

It is also possible for the different methods to not find optimal or satisfiable assignments for the same instances. The main analysis considers the results from all instances, as it provides a more complete overview of the performance of the different methods. Tables where the experiments had the same outcomes for the same instances can be found in appendix A.

Runtime: Due to the nature of PRCPSP-ST, the search space is many times larger when compared to the base problem of RCPSP. In order to gather experimental data even for prohibitively large instances, a timeout of 10 minutes was added to the solver. Metrics were collected every time a new solution was found by the solver.

All computational experiments were run using the DelftBlue supercomputer [24], on an Intel Xeon E5-6248R 24C 3.0GHz processor. The Constraint Programming solver the heuristics were implemented on, **Pumpkin**, was developed by Dr. Emir Demirović, Imko Marijnissen and Maarten Flippo.

Setup times: For all experiments, a static setup time of 1 was used. While previous works on PRCPSP-ST use non-integer setup times [8], most constraint solvers, including Pumpkin, do not directly allow the use of continuous variables. Larger setup times were not considered, as the ratio between the average activity duration and the setup penalty generally made pre-emption impractical.

Algorithms: Three main constraint solver configurations were tested: the first, serving as the baseline, is the unmodified solver, which uses VSIDS and solution-guided phase saving (VS/SG). The second replaces both heuristics with the GRD/dEST composite heuristic method introduced in section 5.1. Finally, the third configuration uses VS/dEST, introduced in section 5.3.

6.2 Results

The metrics were computed separately for the cases where the optimal solution was found, and those where only a satisfiable assignment was reached. In both cases, the means for the %DEV, %RU, #Imp, T, #PRE and #DC metrics can be seen in table 4 and table 5, respectively. The tables also include the number of instances for which an optimal / satisfiable assignment was reached.

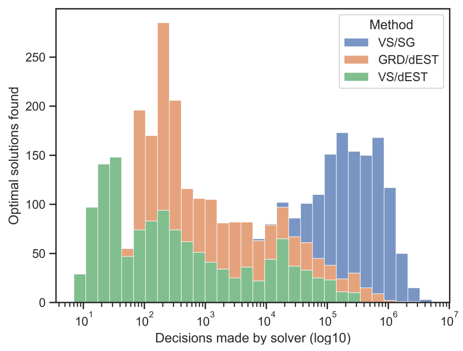
Only the final results for each instance were considered in the metric mean calculations, in order to prevent skewed results due to one of the methods finding more intermediary solutions compared to the rest. The complete intermediary results can be accessed at the code repository associated with this paper [25].

An overview of the number of decisions each method required to find either an optimal solution or the first satisfiable assignment, can be seen in fig. 3a and fig. 3b. Similar histograms plotted against the time needed to find these assignments yielded similar results and can be seen in appendix B.

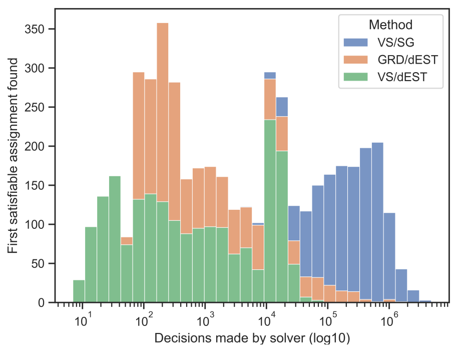
Table 4: The means of the metrics when an optimal solution was found, considered per dataset and over all datasets. The final column contains the total number of optimal solutions found for each group. Significant differences between methods are marked.

Dataset	Method	Metrics						
		%DEV	%RU	#Imp	#DC	T	#PRE	#Ins
J30	VS/SG	0.65	48.81	107	161050	23.87	12	404
	GRD/dEST	0.54	44.52	79	30676	14.10	2	366
	VS/dEST	0.64	46.53	106	11685	6.42	2	407
J60	VS/SG	0.31	48.40	38	618776	156.44	28	342
	GRD/dEST	0.28	44.16	31	26202	13.09	5	333
	VS/dEST	0.30	46.24	39	6279	6.61	7	362
J90	VS/SG	0.11	46.47	10	890968	286.16	42	207
	GRD/dEST	0.09	43.28	13	17385	16.54	8	313
	VS/dEST	0.12	46.52	21	8079	13.06	13	361
J120	VS/SG	**0.40**	56.66	4	975005	366.55	65	40
	GRD/dEST	**0.06**	51.43	2	65982	50.99	22	113
	VS/dEST	**0.18**	57.42	5	22998	45.07	35	176
All	VS/SG	0.42	48.49	159	**503642**	**138.01**	26	993
	GRD/dEST	0.31	44.76	125	**29200**	**18.19**	6	1125
	VS/dEST	0.36	47.91	171	**10714**	**13.52**	11	1306

In general, from the results obtained for %DEV, it can be observed that pre-emptability does lead to more efficient schedules. While the improvement is smaller compared to that obtained in previous studies [8], this can be attributed to the larger setup time relative to the average activity durations shown in table 3. VS/SG is able to reach larger makespan improvements, especially for the J120 instances, despite finding fewer optimal solutions. VS/SG also generated more resource-efficient schedules.



(a) Number of decisions to find an optimal solution.



(b) Number of decisions to find a satisfiable assignment.

Figure 3: An overview of the methods' progress over the

Table 5: The means of the metrics when only a satisfiable assignment was found, considered per dataset and over all datasets. The final column contains the total number of satisfiable assignments found for each group.

Dataset	Method	Metrics						
		%DEV	%RU	#Imp	#DC	T	#PRE	#Ins
J30	VS/SG	-1.24	66.19	19	512743	233.61	6	76
	GRD/dEST	-5.00	62.11	11	247915	171.53	4	95
	VS/dEST	-1.47	66.54	12	290754	259.14	5	73
J60	VS/SG	-39.24	66.40	12	951112	461.58	35	103
	GRD/dEST	-8.58	64.25	2	215589	226.93	13	73
	VS/dEST	-6.17	70.32	2	203985	347.98	21	118
J90	VS/SG	-304.06	38.64	4	1169319	541.88	69	84
	GRD/dEST	-7.10	61.96	1	219438	278.74	22	46
	VS/dEST	-7.62	72.29	2	141516	367.29	43	119
J120	VS/SG	-617.67	30.02	0	1554786	566.68	90	61
	GRD/dEST	-9.82	61.05	2	274184	294.86	32	84
	VS/dEST	-6.41	72.37	0	113001	363.67	66	424
All	VS/SG	**-148.01**	52.30	35	**1018511**	448.71	48	324
	GRD/dEST	**-7.01**	62.31	16	**243005**	236.42	17	298
	VS/dEST	**-4.82**	71.45	16	**149929**	351.34	49	734

Three significant improvements of both GRD/dEST and VS/dEST over the baseline can be identified: #DC and T are significantly lower, and the number of instances solved to optimality is higher. This is also the case for finding the first satisfiable assignment, as can be seen in table 5. Notably, VS/dEST was able to find at least one satisfiable assignment within the time limit for all 2040 instances.

Finally, the histogram plots in fig. 4 show that both GRD/dEST and VS/SG find optimal, or satisfiable, solutions relatively early into the solving process. They then continue to find solutions at a relatively uniform pace. In contrast, VS/SG makes $\approx 10^5$ decisions before reaching any optimal or satisfiable assignments.

7 Responsible Research

We present the steps taken to ensure that the research carried out for this paper was conducted responsibly. In section 7.1, information about the datasets used in the Experimental Results section is presented, while in section 7.2 the steps taken to ensure the reproducibility of the results are laid out.

7.1 Dataset Availability

All four of the problem instances used in the Experimental Results section are publicly available to download from PSPLIB [26]. They do not contain any personal information, as they were automatically generated using the standard project generator ProGen [19]. The raw data collected from the solver used to compute the metrics in section 6.2 are available at the GitHub repository associated with this paper [25].

7.2 Reproducibility

The constraint programming solver used in this paper, Pumpkin, is not publicly available at the time of writing. Once it is published, the Pumpkin code containing the GRD and LW selection strategies used in this paper will then be available on the "prcpsp_rp" branch of the repository containing the solver. The scripts used to parse and process the data collected from the solver are available at the previously mentioned GitHub repository.

8 Conclusion and Future Work

In this paper, the inclusion of PRCPSP-ST-specific information into a Constraint Programming solver was examined. This was done through the use of two simple heuristics, GRD and dEST. They were compared against two state-of-the-art problem-independent heuristics for CP solvers, VSIDS and Solution-guided phase saving. Our results have shown that, especially during the value selection process, scheduling activities at the earliest possible time can lead to a significantly more efficient search procedure. This can be attributed to two main factors: by its nature, PRCPSP-ST has a larger time window during which activities may be scheduled (horizon) compared to ordinary RCPSP. This is because the solver assumes that, in the worst case, all activities are constantly pre-empted. Secondly, as activities can be pre-empted, it becomes much more unlikely for large gaps in the schedule to form.

There are many potential avenues for future work:

- As mentioned earlier in this paper, five different types of setup times which are common in practice have been identified [8]. Our recommendation for modelling these fractional setup times is to scale both the activity durations and the setup times by a factor of 5. This should allow for how dynamic setup time durations influence the search process, while not making the search space prohibitively large.

- Examining fig. 3a, it can be seen that VS/SG takes longer to begin finding satisfiable assignments, but following this, it is quickly able to begin finding optimal solutions. This may indicate that VS/SG may be more suitable if a small number of instances can be evaluated with longer timeouts. This could be explored by running the three methods on a smaller subset of instances, with a timeout of 20 to 30 minutes.

- More priority-rule-based heuristics could be considered. In his paper on bidirectional planning [16], Klein identified 23 other priority rules commonly used in RCPSP. It would be worthwhile to investigate if a combination of these can guide the search process of CP solvers more efficiently.

References

[1] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, January 1983. URL: `https://www.sciencedirect.com/science/article/pii/0166218X83900124`, `doi:10.1016/0166-218X(83)90012-4`.

[2] Robert Pellerin, Nathalie Perrier, and François Berthaut. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European*

Journal of Operational Research, 280(2):395–416, January 2020. URL: https://www.sciencedirect.com/science/article/pii/S0377221719300980, doi:10.1016/j.ejor.2019.01.063.

[3] Lori Ann Kaplan. *Resource-constrained project scheduling with preemption of jobs*. University of Michigan, 1988. URL: https://search.proquest.com/openview/b5c8d82606761df056bf9f8427f9e8b3/1?pq-origsite=gscholar&cbl=18750&diss=y.

[4] Lori Ann Kaplan. Resource-constrained project scheduling with setup times. 1990. ISSN: ,. URL: http://pascal-francis.inist.fr/vibad/index.php?action=getRecordDetail&idt=11890205.

[5] Francisco Ballestín, Vicente Valls, and Sacramento Quintanilla. Pre-emption in resource-constrained project scheduling. *European Journal of Operational Research*, 189(3):1136–1152, September 2008. URL: https://www.sciencedirect.com/science/article/pii/S0377221707005905, doi:10.1016/j.ejor.2006.07.052.

[6] Aziz Moukrim, Alain Quilliot, and Hélène Toussaint. An effective branch-and-price algorithm for the Preemptive Resource Constrained Project Scheduling Problem based on minimal Interval Order Enumeration. *European Journal of Operational Research*, 244(2):360–368, July 2015. URL: https://www.sciencedirect.com/science/article/pii/S0377221714010558, doi:10.1016/j.ejor.2014.12.037.

[7] José Coelho and Mario Vanhoucke. Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *European Journal of Operational Research*, 213(1):73–82, August 2011. URL: https://www.sciencedirect.com/science/article/pii/S037722171100230X, doi:10.1016/j.ejor.2011.03.019.

[8] Mario Vanhoucke and José Coelho. Resource-constrained project scheduling with activity splitting and setup times. *Computers & Operations Research*, 109:230–249, September 2019. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054819301170, doi:10.1016/j.cor.2019.05.004.

[9] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, July 2011. doi:10.1007/s10601-010-9103-2.

[10] Erik L. Demeulemeester and Willy S. Herroelen. An efficient optimal solution procedure for the preemptive resource-constrained project scheduling problem. *European Journal of Operational Research*, 90(2):334–348, April 1996. URL: https://linkinghub.elsevier.com/retrieve/pii/0377221795003584, doi:10.1016/0377-2217(95)00358-4.

[11] Rainer Kolisch and Sönke Hartmann. Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. In Jan Wglarz, editor, *Project Scheduling: Recent Models, Algorithms and Applications*, pages 147–178. Springer US, Boston, MA, 1999. doi:10.1007/978-1-4615-5533-9_7.

[12] Vincent Van Peteghem and Mario Vanhoucke. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 201(2):409–418, March 2010. URL:

https://www.sciencedirect.com/science/article/pii/S037722170900191X, doi:10.1016/j.ejor.2009.03.034.

[13] Sönke Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, 45(7):733–750, 1998. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291520-6750%28199810%2945%3A7%3C733%3A%3AAID-NAV5%3E3.0.CO%3B2-C. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291520-6750%28199810%2945%3A7%3C733%3A%3AAID-NAV5%3E3.0.CO%3B2-C, doi:10.1002/(SICI)1520-6750(199810)45:7<733::AID-NAV5>3.0.CO;2-C.

[14] K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149(2):268–281, September 2003. URL: https://www.sciencedirect.com/science/article/pii/S0377221702007610, doi:10.1016/S0377-2217(02)00761-0.

[15] Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, March 2007. URL: https://www.sciencedirect.com/science/article/pii/S0377221705008507, doi:10.1016/j.ejor.2005.12.009.

[16] Robert Klein. Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects. *European Journal of Operational Research*, 127(3):619–638, December 2000. URL: https://www.sciencedirect.com/science/article/pii/S0377221799003471, doi:10.1016/S0377-2217(99)00347-1.

[17] Joao Marques-Silva, Ines Lynce, and Sharad Malik. Chapter 4. Conflict-Driven Clause Learning SAT Solvers. In Armin Biere, Marijn Heule, Hans Van Maaren, and Toby Walsh, editors, *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2021. URL: http://ebooks.iospress.nl/doi/10.3233/FAIA200987, doi:10.3233/FAIA200987.

[18] Emir Demirovic, Geoffrey Chu, and Peter J. Stuckey. Solution-Based Phase Saving for CP: A Value-Selection Heuristic to Simulate Local Search Behavior in Complete Solvers. In John Hooker, editor, *Principles and Practice of Constraint Programming*, volume 11008, pages 99–108. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science. URL: http://link.springer.com/10.1007/978-3-319-98334-9_7, doi:10.1007/978-3-319-98334-9_7.

[19] Rainer Kolisch, Arno Sprecher, and Andreas Drexl. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science*, 41(10):1693–1703, October 1995. Publisher: INFORMS. URL: https://pubsonline.informs.org/doi/abs/10.1287/mnsc.41.10.1693, doi:10.1287/mnsc.41.10.1693.

[20] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco. An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science*, 44(5):714–729, May 1998. URL: https://pubsonline.informs.org/doi/10.1287/mnsc.44.5.714, doi:10.1287/mnsc.44.5.714.

[21] R. Alvares-Valdes and J. M. Tamarit. Heuristic algorithms for resource-constrained project scheduling: a review and an empirical analysis, 1989. Pages: 113134 Publication Title: Advances in project scheduling.

[22] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 530–535, New York, NY, USA, June 2001. Association for Computing Machinery. URL: `https://dl.acm.org/doi/10.1145/378239.379017`, `doi:10.1145/378239.379017`.

[23] Mario Vanhoucke, José Coelho, and Jordy Batselier. An overview of project data for integrated project management and control. *The Journal of Modern Project Management*, 3(3):158–158, 2016. URL: `https://journalmodernpm.com/manuscript/index.php/jmpm/article/download/218/217`.

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

[25] Codrin Ogreanu. PRCPSP-ST_cp, June 2024. URL: `https://github.com/cogreanu/PRCPSP-ST_CP`.

[26] The Library PSPLIB. URL: `https://www.om-db.wi.tum.de/psplib/library.html`.

A Additional Tables

Table 6: The means of the metrics when an optimal result was found, adjusted to consider only cases where all methods found optimal solutions for the same instances.

Dataset	Method	Metrics						
		%DEV	%RU	#Imp	#DC	T	#PRE	#Ins
J30	VS/SG	0.54	47.15	79	155036	19.83	13	366
	GRD/dEST	0.54	44.52	79	30676	14.10	2	366
	VS/dEST	0.54	44.71	79	2741	1.25	2	366
J60	VS/SG	0.29	46.91	31	621940	150.47	28	318
	GRD/dEST	0.29	43.92	31	27184	13.51	5	318
	VS/dEST	0.29	44.25	31	1548	1.59	6	318
J90	VS/SG	0.12	44.40	10	899965	280.22	42	187
	GRD/dEST	0.12	41.48	10	12925	10.19	7	187
	VS/dEST	0.12	41.85	10	1705	2.14	10	187
J120	VS/SG	0.17	54.01	2	1023394	345.56	65	30
	GRD/dEST	0.17	50.53	2	40543	27.75	22	30
	VS/dEST	0.17	51.54	2	8557	10.90	30	30
All	VS/SG	0.36	46.72	122	503347	130.83	26	901
	GRD/dEST	0.36	43.88	122	26088	13.54	5	901
	VS/dEST	0.36	44.18	122	2298	1.87	6	901

Table 7: The means of the metrics when only satisfiable assignments were found, adjusted to consider only cases where all methods found satisfiable assignments for the same instances.

Dataset	Method	Metrics						
		%DEV	%RU	#Imp	#DC	T	#PRE	#Ins
J30	VS/SG	-1.05	67.24	15	533721	235.13	6	53
	GRD/dEST	-7.00	63.13	2	232997	183.43	4	53
	VS/dEST	-1.27	66.58	10	281736	238.09	4	53
J60	VS/SG	-23.30	67.81	8	990642	442.06	30	46
	GRD/dEST	-9.74	64.90	1	199002	241.39	14	46
	VS/dEST	-4.73	68.04	2	222890	321.13	17	46
J90	VS/SG	-181.16	52.09	3	1300706	500.25	58	14
	GRD/dEST	-9.49	62.92	0	262850	313.45	23	14
	VS/dEST	-3.99	66.41	2	246482	363.47	32	14
J120	VS/SG	-396.72	40.74	0	1707920	551.48	84	13
	GRD/dEST	-14.52	61.04	1	361882	323.17	30	13
	VS/dEST	-5.32	64.25	0	210585	360.55	36	13
All	VS/SG	-45.96	63.03	26	906901	372.77	29	126
	GRD/dEST	-8.58	63.54	4	237201	233.45	13	126
	VS/dEST	-2.97	66.85	14	248995	294.97	15	126

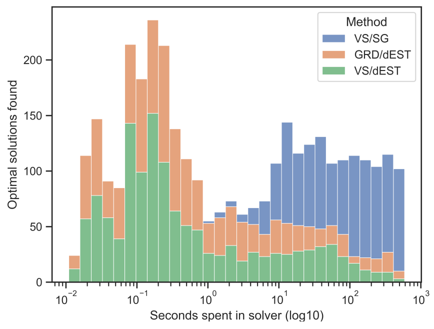
Table 8: The means of the metrics when only satisfiable assignments were found, adjusted to take the upper bound of the makespans as the optimal solution.

Dataset	Method	Metrics						
		%DEV	%RU	#Imp	#DC	T	#PRE	#Ins
J30	VS/SG	-1.24	66.19	19	512743	233.61	6	76
	GRD/dEST	-5.00	62.11	11	247915	171.53	4	95
	VS/dEST	-1.47	66.54	12	290754	259.14	5	73
J60	VS/SG	-42.02	66.40	12	951112	461.58	35	103
	GRD/dEST	-9.12	64.25	2	215589	226.93	13	73
	VS/dEST	-9.13	70.32	2	203985	347.98	21	118
J90	VS/SG	-269.65	38.64	4	1169319	541.88	69	84
	GRD/dEST	-8.29	61.96	1	219438	278.74	22	46
	VS/dEST	-15.48	72.29	2	141516	367.29	43	119
J120	VS/SG	-545.36	30.02	0	1554786	566.68	90	61
	GRD/dEST	-10.83	61.05	2	274184	294.86	32	84
	VS/dEST	-18.36	72.37	0	113001	363.67	66	424
All	VS/SG	-143.39	52.30	35	1018511	448.71	48	324
	GRD/dEST	-7.66	62.31	16	243005	236.42	17	298
	VS/dEST	-12.37	71.45	16	149929	351.34	49	734

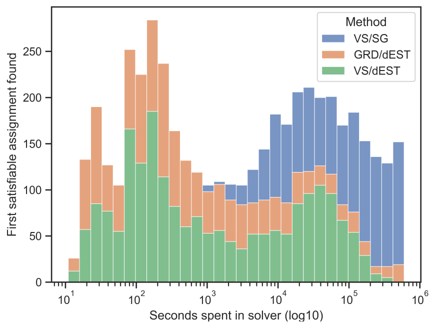
Table 9: The means of the metrics when only satisfiable assignments were found, adjusted to take the lower bound of the makespans as the optimal solution.

Dataset	Method	Metrics						
		%DEV	%RU	#Imp	#DC	T	#PRE	#Ins
J30	VS/SG	-1.24	66.19	19	512743	233.61	6	76
	GRD/dEST	-5.00	62.11	11	247915	171.53	4	95
	VS/dEST	-1.47	66.54	12	290754	259.14	5	73
J60	VS/SG	-44.07	66.40	12	951112	461.58	35	103
	GRD/dEST	-9.69	64.25	2	215589	226.93	13	73
	VS/dEST	-11.33	70.32	2	203985	347.98	21	118
J90	VS/SG	-272.45	38.64	4	1169319	541.88	69	84
	GRD/dEST	-9.34	61.96	1	219438	278.74	22	46
	VS/dEST	-20.34	72.29	2	141516	367.29	43	119
J120	VS/SG	-546.86	30.02	0	1554786	566.68	90	61
	GRD/dEST	-11.74	61.05	2	274184	294.86	32	84
	VS/dEST	-23.28	72.37	0	113001	363.67	66	424
All	VS/SG	-145.02	52.30	35	1018511	448.71	48	324
	GRD/dEST	-8.16	62.31	16	243005	236.42	17	298
	VS/dEST	-15.74	71.45	16	149929	351.34	49	734

B Additional Figures



(a) Time to find an optimal solution.



(b) Time to find a satisfiable assignment.

Figure 4: An overview of the methods' progress over the duration of the solving process.