Integration of variable selection heuristics into a MaxSAT solver for solving the multi-mode resource-constrained project scheduling problem

Denis Rangelov Tsvetkov[1]
Supervisors: Emir Demirović[1], Konstantin Sidorov[1], Maarten Flippo[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
July 4, 2023

Name of the student: Denis Rangelov Tsvetkov
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Konstantin Sidorov, Maarten Flippo, Jérémie Decouchant

Abstract

The multi-mode resource-constrained project scheduling problem (MRCPSP) is an extension of the resource-constrained project scheduling problem (RCPSP), which allows activities to be executed in multiple modes. The state-of-the-art solutions for solving this NP-Hard problem are dedicated algorithms and (meta-)heuristics. However, this paper considers a more flexible approach using a MaxSAT solver. The idea is to replace the existing variable selection strategy of the solver, Variable State Independent Decaying Sum (VSIDS), with two scheduling heuristics, Earliest Starting Time (EST) and Shortest Feasible Mode (SFM). We examine that combining the three heuristics results in a more efficient solver. In contrast, scheduling rules alone lead to a solver that performs significantly worse on any of the chosen metrics and benchmarks.

1 Introduction

In literature, a popular way to define schedules is as resource-constrained project scheduling problems (RCPSP). As planning is an irreplaceable part of numerous real-world problems, such as resource allocation or risk management, the RCPSP has been a hot topic of research for many years now [1]. The projects in the RCPSP are represented as sets of activities (tasks) that are defined by their starting time, processing time (duration), resource requirements, and precedence relations. In addition, tasks can be processed in parallel as long as the precedence constraints are respected and the resources needed at any point in time, do not exceed some predefined limit. The objective of the problem in most cases is to minimize the makespan (the time required to complete all activities).

The multi-mode resource-constrained project scheduling problem (MRCPSP) is a generalization of the RCPSP. It allows for activities to be executed in different modes while using two types of resources, non-renewable and renewable [27]. Non-renewable resources are limited for the whole makespan, e.g., budget. In contrast, renewable resources are limited only at deterministic points in time, e.g., number of people/machines available [22]. Finally, the different modes for each task are characterized by distinct processing times and/or resource requirements.

MRCPSP is known to be NP-hard [5]. Despite the problem's complexity, several algorithms [24, 3, 25, 12, 28] and heuristics [11, 7, 6, 14, 19, 16] have been proposed throughout the years. However, even though these dedicated algorithms typically yield good solutions, a substantial amount of expert time is required to study and reproduce them [21].

This is why we consider a more generic approach to solve the MRCPSP using propositional logic and modeling the problem as a maximum satisfiability problem (MaxSAT). This approach is more flexible than dedicated algorithms because MaxSAT solvers can be used on any problem that can be encoded into MaxSAT, without the need for the solver to be modified.

Research on employing SAT/MaxSAT solvers for solving the MRCPSP has already been done before [8, 18]. In [8], the MRCPSP is split into two parts - mode assignment and scheduling of single-mode activities. However, in the proposed solution, the SAT solver is only utilized to find a feasible mode assignment, and a dedicated algorithm is used to schedule the activities based on the selected modes. Alternatively, in [18], the MRCPSP is encoded entirely into MaxSAT. On the other hand, the research is focused on reducing the size of the MaxSAT encoding of the MRCPSP and examining how the solver reacts to an initial solution that it can exploit to optimize the search.

This paper expands upon previous work by incorporating scheduling heuristics into the existing variable selection strategy of the solver, Variable State Independent Decaying Sum (VSIDS) [15]. The two heuristics we consider are Earliest Starting Time (EST) and Shortest Feasible Mode (SFM). These heuristics are fitted for MRCPSP and are therefore not used in SAT/MaxSAT solvers. However, given that EST and SFM are identified as one of "the most effective scheduling rules" [6], we expect that a solver that utilizes them should outperform an off-the-shelf MaxSAT solver when solving the MRCPSP.

The focus of this research is to investigate the impact the new heuristics have on the performance of the solver, e.g., the time required to find an optimal solution. To accomplish this, multiple tests are run on more than 3000 problem instances varying in terms of the number of activities, modes, etc. The results that we obtain tell us that combining VSIDS and the scheduling results leads to a solver that finds more optimal solutions and takes 5% to 42% less CPU time. On the other hand, using just EST and SFM proves to be inefficient compared to VSIDS.

The rest of the paper is structured as follows. Firstly, Section 2 provides a formal definition of the MRCPSP. Secondly, an outline of SAT, MaxSAT, and VSIDS is given in Section 3. After that, Section 4 provides the encoding of the problem into MaxSAT, and further details about EST, SFM, and how they are incorporated into the solver. Then, Section 5 presents the results from the performed experiments. Following that, in Section 6 we examine potential ethical issues, related to the research. Lastly, conclusions and ideas for future work are discussed in Section 7.

2 MRCPSP Model Formulation

The MRCPSP is characterized by a set of activities that can be executed in multiple modes with different processing times and/or resource requirements. Furthermore, the tasks use two types of resources, non-renewable and renewable. The goal of the problem is to minimize the makespan of the projects.

The set V of N activities is defined as:

$$V = \{v \mid 0 \leq v < N\}, \text{ where}$$

0 and $N - 1$ represent dummy start and end tasks. Furthermore, the sets R^r and R^n represent the renewable and non-renewable resources, respectively. Additionally, for each resource $k \in R^r$, we have a capacity c_k^r fixed at each deterministic point in time. In contrast, the capacity c_l^n for all $l \in R^n$ is constant for the entire planning period. Then, to model the execution modes m we use N sets M_v for each activity $v \in V$, such that:

$$M_v = \{(d_{v,m}, r^r_{v,m,k}, r^n_{v,m,l}) \mid \forall k \in R^r, \forall l \in R^n\}, \text{ where}$$

- $d_{v,m}$ - the processing time of time activity v when executed in mode m;
- $r^r_{v,m,k}$ - the required units of renewable resource k of activity v when executed in mode m;
- $r^n_{v,m,l}$ - the required units of non-renewable resource l of activity v when executed in mode m.

The dummy start and end activities can only be executed in one mode with duration and resource requirements equal to 0. Finally, we introduce two sets of auxiliary variables, defined as follows:

$$s_{v,m,t} = \begin{cases} 1, & \text{if activity } v \text{ is executed in mode } m \\ & \text{and starts at time } t, \\ 0, & \text{otherwise,} \end{cases}$$

$$x_{v,m,t} = \begin{cases} 1, & \text{if activity } v \text{ is executed in mode } m \\ & \text{and is being processed at time } t^1, \\ 0, & \text{otherwise.} \end{cases}$$

[1] If activity v starts in mode m at time t, then it is being processed from t to $t + d_{v,m}$.

These variables help formulate the problem's constraints and objective [26]. First, we need the following constraints to ensure that the precedence relations are respected:

$$\sum_{p}^{M_i}\sum_{t=0}^{T}(t+d_{i,p})\cdot s_{i,p,t} \leq \sum_{q}^{M_j}\sum_{t=0}^{T} t\cdot s_{j,q,t},\ \forall (i,j)\in E.$$

With these, if activity i starts in any mode $p \in M_i$ at time t, and j is a successor of i, then j can only start after $t + d_{i,p}$, inclusive as time-lag of zero is considered. Then, we need to prevent activities from starting more than once or in multiple modes:

$$\sum_{m}^{M_v}\sum_{t=0}^{T} s_{v,m,t} = 1,\ \forall v \in V.$$

The upcoming two inequalities assure that the demands for renewable and non-renewable resources, respectively, do not exceed the available capacity for each resource:

$$\sum_{v}^{V}\sum_{m}^{M_v} x_{v,m,t}\cdot r^r_{v,m,k} \leq c^r_k,\ \forall k \in R^r,\ t = 0,1,...,T,$$

$$\sum_{v}^{V}\sum_{m}^{M_v} r^n_{v,m,l}\cdot \sum_{t=0}^{T} s_{v,m,t} \leq c^n_l,\ \forall l \in R^n.$$

Lastly, the equation

$$\min \sum_{t=0}^{T} s_{N-1,m_N,t}$$

is used to minimize the start time of the dummy end activity, which coincides with minimizing the makespan.

The MRCPSP can be modeled as a directed graph $G(V, E)$, in which each node $v \in V$ represents an activity and each edge $e_{i,j} \in E$ represents a precedence relation between activities i and j (see Figure 1). It is important to note that the graph needs to be acyclic, otherwise the problem is not satisfiable (see Theorem 1).

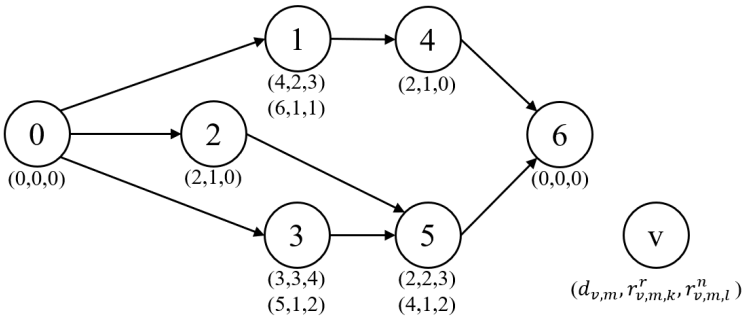

Figure 1: An example graph representation of an MRCPSP instance with seven activities, one renewable resource, and one non-renewable resource [14]. Each edge refers to a precedence relation between two activities. Each label below a node corresponds to an execution mode for that activity, defined as a triplet of the mode's processing time, and its demand for renewable and non-renewable resources.

Theorem 1. *The MRCPSP has no satisfiable solutions if its directed graph representation contains a cycle.*

Proof. Let $G(V, E)$ be a directed graph of any MRCPSP problem, and let $C = (v_1, v_2, ..., v_n)$ be a cycle in that graph. From the definition of a cycle, it follows that the edges $e_{1,2}, e_{2,3}, ..., e_{n,1}$ must exist. Therefore, we say that v_2 is a successor of v_1, v_3 is a successor of v_2, ..., and v_1 is a successor of v_n. However, this means that none of these activities can be started, as the precedence relations do not allow for that. Thus, since there are activities that cannot be started and completed, we conclude that a satisfiable schedule for the problem does not exist. □

Consider the MRCPSP example, shown in Figure 1. The problem consists of seven activities, one renewable resource k with capacity $c_k^r = 4$, and one non-renewable resource l with capacity $c_l^n = 8$. An optimal schedule of the start time and execution mode for each of the five non-dummy activities is provided in Figure 2.

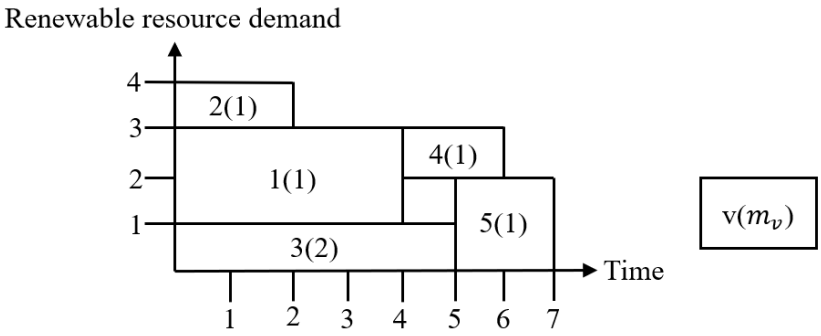

Figure 2: Optimal schedule with makespan of 7 units of time for the problem in Figure 1. Each rectangle is labeled with a corresponding activity number and the execution mode that is picked.

3 Overview of MaxSAT and VSIDS

This section provides formal definitions of the SAT/MaxSAT problems and the Variable State Independent Decaying Sum (VSIDS) heuristic [15]. This heuristic is utilized by the solver provided for this research by the Faculty of Electrical Engineering, Mathematics, and Computer Science of the Delft University of Technology. Additionally, VSIDS serves as a baseline for comparison against the MRCPSP scheduling rules, introduced in Section 4.

3.1 SAT/MaxSAT

Given a propositional formula F, SAT problems aim to assign a truth value to all variables, such that the provided formula is satisfied. The formula F is defined as the conjunction of a set of clauses:

$$F := C_1 \wedge C_2 \wedge ... \wedge C_n, \text{ where}$$

C_i is a disjunction of literals:

$$C_i := L_1 \vee L_2 \vee ... \vee L_m.$$

Each literal L_i can either be a propositional variable (x_i) or its negation ($\neg x_i$). Thus, F is said to be in conjunctive normal form (CNF).

In this paper, we focus on an extension of SAT, namely MaxSAT. The difference between the two problems is that in MaxSAT we try to find an assignment that maximizes the number of satisfied clauses. Furthermore, by introducing non-negative weights to MaxSAT we can use two types of clauses, hard and soft. The hard clauses must all evaluate to True, otherwise the problem is considered unsatisfiable. In contrast, soft clauses can be False, but the goal of a MaxSAT solver is to minimize the cost (the sum of the weights of the unsatisfied soft clauses). Finally, this encoding allows us to maximize the value of a function, which in the case of MRCPSP means that we can minimize the makespan [18].

3.2 VSIDS

VSIDS is a heuristic that dynamically ranks the variables during the solver's execution. It is considered more efficient than other known alternatives due to the low computational overhead [15].

The main idea behind the heuristic is that each variable is assigned a floating-point value and the one with the largest value is selected for branching. These values are updated dynamically as clauses are learned, in such a way that recently learned clauses are preferred. To clarify, clause learning happens when a conflict occurs. Then, the solver performs a conflict analysis [17], which yields a new more concise clause without any redundant literals [4]. Furthermore, when the solver learns a clause, the values of the variables that are part of the conflict are increased by a fixed value (bump). Additionally, all variables' values are multiplied by a decay factor at regular intervals.

In the solver used in this paper, each variable is initialized with a value equal to 0. Furthermore, the bump is 1, and the decay factor is 0.95.

4 Integration of Heuristics into a MaxSAT Solver

In this section, we go over the integration of MRCPSP-specific heuristics into a MaxSAT solver. First, Section 4.1 presents the encoding of the MRCPSP into weighted CNF (WCNF), in which state the solver can use it. Then, Section 4.2 provides details about the chosen heuristics and how they are incorporated into the existing algorithm.

4.1 Encoding MRCPSP into MaxSAT

To be used by the solver, the MRCPSP must first be encoded into WCNF. The following propositional variables are introduced:

$$\begin{aligned} s_{v,t}, &\forall v \in V,\ t = 0, 1, ..., T, \\ x_{v,m,t}, &\forall v \in V,\ \forall m \in M_v, t = 0, 1, ..., T, \\ y_{v,m}, &\forall v \in V,\ \forall m \in M_v. \end{aligned}$$

The first two variables have the same meaning as the ones shown in Section 2. The last one is defined as:

$$y_{v,m} = \begin{cases} 1, & \text{if activity } v \text{ is executed in mode } m, \\ 0, & \text{otherwise.} \end{cases}$$

To ensure that only one mode is selected, the constraints

$$\sum_{m}^{M_v} y_{v,m} = 1,\ \forall v \in V$$

must be satisfied. When translated into WCNF, we get the following hard clauses:

$$(y_{v,m_1} \vee y_{v,m_2} \vee ... \vee y_{v,m_n}) \wedge (\neg y_{v,m_1} \vee \neg y_{v,m_2}) \wedge (\neg y_{v,m_1} \vee \neg y_{v,m_3}) \wedge ... \wedge (\neg y_{v,m_{n-1}} \vee \neg y_{v,m_n}),\ \forall v \in V,$$

where $M_v := \{m_1, m_2, ..., m_n\}$. The first clause ensures that at least one mode is selected for an activity, and the rest ensure that at most one mode is selected.

Similar clauses are added for the start times to guarantee that an activity is started only once:

$$(s_{v,0} \vee s_{v,1} \vee ... \vee s_{v,T}) \wedge (\neg s_{v,0} \vee \neg s_{v,1}) \wedge (\neg s_{v,0} \vee \neg s_{v,2}) \wedge ... \wedge (\neg s_{v,T-1} \vee \neg s_{v,T}),\ \forall v \in V.$$

Next, to ensure that any activity $v \in V$ can start in any of its modes $m \in M_v$ with enough time to complete we add the following 'completion' clauses:

$$\neg y_{v,m} \vee s_{v,0} \vee s_{v,1} \vee ... \vee s_{v,T+1-d_{v,m}}, \text{ where}$$

$d_{v,m}$ is the processing time of activity v when executed in mode m. Note that the clauses above are equivalent to the implications

$$y_{v,m} \rightarrow (s_{v,0} \vee s_{v,1} \vee ... \vee s_{v,T+1-d_{v,m}}).$$

As previously stated, activities in the MRCPSP are subject to precedence relations. The upcoming hard clauses are added to assure those relations are respected:

$$\neg s_{j,t} \vee \neg y_{i,m} \vee s_{i,0} \vee s_{i,1} \vee ... \vee s_{i,t+1-d_{i,m}},\ \forall (i,j) \in E,\ t = 0, 1, ..., T.$$

Then, to set the processing time variables x correctly we introduce additional 'consistency' clauses. With these clauses, the solver can directly propagate the truth values for x, based on the starting time and mode, picked for each activity $v \in V$:

$$\neg s_{v,t} \vee \neg y_{v,m} \vee x_{v,m,t'},\ \forall m \in M_v,\ t = 0, 1, ..., T - d_{v,m},\ t' = t, t+1, ..., t + d_{v,m} - 1$$

The last two sets of hard clauses are related to the demand and availability of non-renewable and renewable resources, respectively. The encoding of these constraints into WCNF is done using a Binary Decision Diagram (BDD) [2]. As this approach requires a significant amount of auxiliary variables, only the high-level formulation of the clauses is provided:

$$\sum_{v}^{V} \sum_{m}^{M_v} y_{v,m} \cdot r^n_{v,m,l} \leq c^n_l,\ \forall l \in R^n,$$
$$\sum_{v}^{V} \sum_{m}^{M_v} x_{v,m,t} \cdot r^r_{v,m,k} \leq c^r_k,\ \forall k \in R^r,\ t = 0, 1, ..., T.$$

Finally, using one-hot encoding we can translate the goal of the MRCPSP into WCNF by introducing $T + 1$ soft clauses:

$$\neg s_{N-1,m_N,t},\ t = 0, 1, ..., T, \text{ where}$$

the weight of each clause is equal to $t + 1$. With this encoding, the makespan of each MRCPSP instance is equal to the cost of its MaxSAT solution - 1.

4.2 EST and SFM

As described in the previous section, the MaxSAT encoding of the MRCPSP consists of four types of variables - s, x, y, and additional auxiliary variables. Both x and the auxiliary variables can be propagated from the clauses once the s and y have been assigned. We hypothesize that assigning those first should reduce the search space and increase the efficiency of the solver. To accomplish this efficiently, we incorporate scheduling heuristics into VSIDS.

The first heuristic considered is the Earliest Starting Time (EST). As we are trying to minimize the makespan, it is rational to start processing activities as soon as possible. The benefit of using this greedy approach is that it is static, meaning that we only need to compute it once at the start, and therefore, it does not introduce additional overhead. The following adjustments are made to the VSIDS implementation: for each variable $s_{v,t}$, an initial value of $\frac{1}{t+1}$ is assigned. This way, the s variables are branched on before the rest. In addition, the value approaches 0 as t increases. Thus, this implementation ensures that earlier starting times are preferred.

The second heuristic is the Shortest Feasible Mode (SFM). With SFM we first explore the mode with the shortest processing time. This heuristic is considered "the most effective scheduling rule" for the MRCPSP [6]. To elucidate, in the research done by F. Boctor, the heuristic is compared to two other scheduling rules, namely the Least Criticality Ratio (LCR) and the Least Resource Proportion (LRP). The experiments that he performs are on benchmarks with 50 and 100 activities, and a ranging number of resources. He concludes that on average SFM finds a solution that is 39% worse than the optimal, whereas this number is double for any of the other heuristics. SFM is incorporated into the solver similarly to EST: the initial values for the $y_{v,m}$ variables are set to $1 + \frac{d_{v,m}}{T}$. This way, we guarantee that the mode variables y are selected before the start times s and that the shorter modes are explored first.

We conduct three experiments with distinct configurations of heuristics: VSIDS, VSIDS and scheduling rules (VSIDS++), and just scheduling rules (ESTSFM). The results from the experiments are presented in the upcoming section.

5 Experiments and Results

In this section, we provide evidence that VSIDS++ significantly outperforms both VSIDS and ESTSFM on all chosen metrics and benchmarks. A more detailed analysis is provided in Section 5.2. Before that, in Section 5.1, we provide details about the software and hardware that are used to perform the experiments[2].

5.1 Test setup

The encoding of the MRCPSP instances into MaxSAT is done using Python 3.8.12 and the PySAT library on version 3.0.6 [13]. The experiments are run on the solver, mentioned in Section 4. As the solver is written in Rust [10], its code is compiled using Rust's built-in compiler on version 1.69.0 and is run in release mode.

The experiments are conducted on the standard nodes of TU Delft's supercomputer DelftBlue [9]. These nodes have Intel Xeon Gold 6248R CPUs running at 3.0GHz and 192GB RAM @ 2933Mhz. In addition, these machines use Red Hat Enterprise Linux 8 as their operating system.

The benchmarks that we use to compare the algorithms are taken from PSPLIB [20] and MMLIB [27]. Table 1 provides an overview of each benchmark regarding the number of test instances and the number of activities, the number of modes per activity, and the number of renewable and non-renewable resources for each instance.

[2] All of the scripts and collected data can be found on GitHub: https://github.com/Revirator/MRCPSP.

Table 1: Number of test instances, number of activities (excluding dummy start and end), number of modes per activity, number of renewable resources, and number of non-renewable resources for each benchmark.

Benchmark	#Instances	N	$\|\|M\|\|$	$\|\|R^r\|\|$	$\|\|R^n\|\|$
j20	554	20	3	2	2
j30	631	30	3	2	2
m5	558	16	5	2	2
n3	600	16	3	2	3
r5	546	16	3	5	2
MMLIB50	475	50	3	2	2

5.2 Results

In this paper, we compare the three heuristic configurations on four distinct metrics - CPU time, the division of solutions into optimal, satisfiable, and timeouts, the number of decisions made, and the area under the time-objective curve (AUC). Furthermore, the experiments are split into two groups:

- small benchmarks (j20, j30, m5, n3, r5), which are run with 5 seconds, 20 seconds, or 60 seconds timeouts and 1GB or 2GB of available RAM;
- big benchmarks (MMLIB50), which are run with 60 seconds, 180 seconds, or 600 seconds timeouts and 8GB of available RAM.

Finally, it is important to note that the CPU time metric does not include the time it takes to encode the problem into MaxSAT, as this is not the purpose of the experiments.

Table 2: Average CPU time required to solve a single instance (find an optimal solution or timeout).

Benchmark	Timeout / Memory								
	5s / 1GB			20s / 1GB			60s / 2GB		
j20	3.31s	1.93s	3.83s	4.81s	3.26s	8.17s	6.15s	4.55s	15.76s
j30	4.67s	3.61s	5.19s	12.13s	7.87s	14.82s	20.7s	16.13s	32.93s
m5	3.48s	2.24s	4s	5.76s	4.45s	9.48s	9.11s	7.67s	20.25s
n3	1.6s	0.95s	2.42s	1.94s	1.28s	4.03s	2.03s	1.31s	6.42s
r5	3.38s	1.85s	3.4s	4.44s	2.92s	6.65s	5.38s	3.95s	11.31s

[1] - VSIDS
[2] - VSIDS + EST + SFM (VSIDS++)
[3] - EST + SFM (ESTSFM)

In Table 2, we can see for each benchmark the average CPU time that the solver needs to solve a single instance. The experiments show that VSIDS++ takes between 22% and 42% less time compared to VSIDS. In contrast, ESTSFM takes more than double the amount of time compared to the other two alternatives. Therefore, we conclude that VSIDS is required for the solver to perform efficiently. However, incorporating domain-specific knowledge into it proves to improve performance when solving the MRCPSP.

Then, Table 3 presents the division of the found solutions into optimal, satisfiable, and timeouts. Timeouts also include instances that run out of memory during the solver's execution. We notice that VSIDS++ finds more optimal and satisiable solutions compared to VSIDS when run with a 5 seconds timeout. However, for most benchmarks, this difference disappears when the timeout is 20 or 60 seconds. On the other hand, using just scheduling rules to select variables performs worse for all configurations compared to the other two alternatives. Given the results of the 5 seconds experiment, we conclude that VSIDS++ finds an optimal/satisfiable solution quicker than VSIDS. The findings in Table 2 further support this statement.

Table 3: Division of the found solutions into optimal, satisfiable, and timeouts (including executions that run out of memory).

Benchmark	Timeout / Memory		
	5s / 1GB	20s / 1GB	60s / 2GB
j20	392, 161, 1 / 481, 73, 0 / 310, 216, 28	519, 35, 0 / 526, 28, 0 / 427, 115, 12	539, 15, 0 / 537, 17, 0 / 479, 73, 2
j30	38, 445, 148 / 285, 216, 130 / 60, 278, 293	293, 188, 150 / 360, 141, 130 / 251, 123, 257	371, 109, 151 / 430, 121, 80 / 355, 122, 154
m5	367, 191, 0 / 437, 120, 1 / 272, 257, 29	501, 57, 0 / 499, 59, 0 / 377, 171, 10	525, 33, 0 / 525, 33, 0 / 435, 122, 1
n3	573, 27, 0 / 581, 19, 0 / 476, 115, 9	593, 7, 0 / 596, 4, 0 / 557, 39, 4	599, 1, 0 / 599, 1, 0 / 579, 19, 2
r5	376, 170, 0 / 495, 51, 0 / 344, 159, 43	519, 27, 0 / 512, 25, 0 / 457, 60, 29	536, 10, 0 / 536, 10, 0 / 496, 38, 12

[1] - VSIDS
[2] - VSIDS + EST + SFM (VSIDS++)
[3] - EST + SFM (ESTSFM)

In Table 4, we present the average number of decisions the solver makes in order to find an optimal solution. The solver makes a decision, when it needs to select a variable to branch on. We see that VSIDS++ makes 10 times fewer decisions compared to VSIDS. This happens because we introduce a new value selection strategy that initially assigns a truth value equal to True when it selects a variable, as opposed to the old one, which assigns False first. The assumption is that this should help guide the search in the 'right' direction, and thus, the solver should make fewer decisions in the end. However, we apply the same idea when using ESTSFM, but because the dynamic ranking of VSIDS is not present, it makes more than four times the amount of decisions that VSIDS makes.

Table 4: Average number of decisions required to solve a single instance (find an optimal solution or timeout).

Benchmark	Timeout / Memory		
	5s / 1GB	20s / 1GB	60s / 2GB
j20	3147943 282595 2223519	3217309 290042 4715018	3223309 296625 4995760
j30	7063709 536387 10056199	9053227 639097 28793461	9186846 680804 39235787
m5	3394564 213558 3970509	3458203 228599 5161972	3468427 239493 5615743
n3	1434594 169965 1878360	1436585 171991 2018564	1437923 172598 2041166
r5	4319163 382064 4003984	4449898 389197 4677351	4454136 393482 4863584

1 - VSIDS
2 - VSIDS + EST + SFM (VSIDS++)
3 - EST + SFM (ESTSFM)

Next, in Table 5, we look at the results from the average AUC. An example of the time-objective curves for the three heuristics run on a single instance can be found in Figure 3. We examine that VSIDS++ outperforms the other two heuristics as its area is two to three times smaller. This means that VSIDS++ takes less time to find a 'good' solution, and thus, less time to find the optimal solution. Furthermore, ESTSFM also outperforms VSIDS on some of the run configurations. Therefore, we conclude that ESTSFM finds a 'good' solution quicker than VSIDS, but given the results in Tables 2 and 3, it seems to struggle to find the optimal solution. Additionally, the large number of timeouts for ESTSFM further decreases the average AUC.

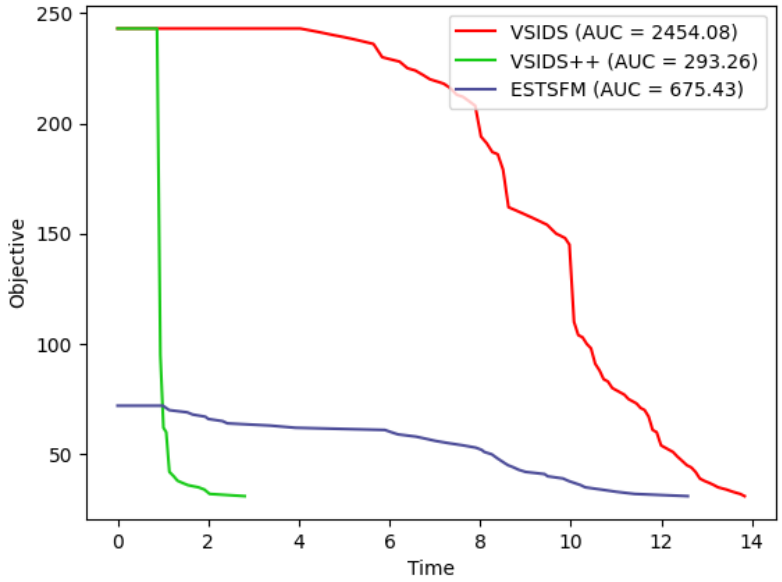

Figure 3: Time-objective curves and the areas below them for VSIDS, VSIDS++, and ESTSFM run on a single instance from the j30 benchmark set.

Table 5: Average area under the time-objective curve.

Benchmark	Timeout / Memory		
	5s / 1GB	20s / 1GB	60s / 2GB
j20	287.03 98.74 169.87	294.61 124.70 332.24	353.24 185.79 553.86
j30	989.55 347.31 288.54	1600.39 496.91 768.41	1983.22 712.59 1626.91
m5	293.65 79.15 149.16	268.17 106.41 342.27	299.39 155.74 582.89
n3	137.29 48.12 100.68	100.51 57.53 147.46	127.28 88.01 200.68
r5	264.89 98.03 127.94	256.73 130.16 225.49	314.01 191.55 435.54

[1] - VSIDS
[2] - VSIDS + EST + SFM (VSIDS++)
[3] - EST + SFM (ESTSFM)

Similar results are obtained for the big benchmarks and are presented in Table 6. We see that VSIDS++ finds more optimal solutions than VSIDS while making fewer decisions. However, the improvement in CPU time is not as significant as for the small benchmarks at a mere 5% to 21%. Conversely, for VSIDS++, the AUC is three times smaller than VSIDS. Therefore, even though the CPU time difference is not as significant, we can still conclude that VSIDS++ finds a 'good' solution quicker.

The more interesting result from the experiment can be found in the metrics obtained for ESTSFM. Even though with ESTSFM the solver makes three to nine times more decisions, it still takes roughly the same amount of CPU time on average. Furthermore, it appears that ESTSFM's AUC score is better compared to the other heuristics when the timeout is 60 seconds. However, this happens because we do not consider executions that time out when calculating the AUC. Additionally, given the AUC scores when the timeout is 180 and 600 seconds, we conclude that ESTSFM takes more time to find a 'good' solution. Therefore, we think that VSIDS is irreplaceable for big benchmarks too.

Table 6: Division of solutions into optimal, satisfiable, and timeouts, CPU time, number of decisions, and area under the time-objective curve (AUC) for the MMLIB50 benchmark.

Metric	Timeout / Memory		
	60s / 8GB	180s / 8GB	600s / 8GB
Division of solutions	147, 310, 18 [1] 202, 253, 20 [2] 163, 44, 268 [3]	212, 252, 11 [1] 214, 253, 8 [2] 190, 78, 207 [3]	225, 248, 2 [1] 223, 251, 1 [2] 204, 119, 152 [3]
CPU time	52.9s [1] 41.36s [2] 46.92s [3]	126.83s [1] 108.63s [2] 121.87s [3]	350.88s [1] 332.71s [2] 364.42s [3]
#Decisions	62310872 [1] 1627725 [2] 185708949 [3]	64844994 [1] 1879326 [2] 366601392 [3]	65906527 [1] 2198776 [2] 544396261 [3]
AUC	12221.99 [1] 4457.22 [2] 3095.68 [3]	14730.82 [1] 6373.3 [2] 14247.29 [3]	20824.49 [1] 10737.45 [2] 34896.76 [3]

[1] - VSIDS
[2] - VSIDS + EST + SFM (VSIDS++)
[3] - EST + SFM (ESTSFM)

6 Responsible Research

In this section, we address the two main concerns about this research: the reproducibility of the experiments and the fair comparison of the algorithms.

To ensure that the experiments can be reproduced by anyone, in Section 5.1 we provide details about all the software, hardware, and benchmarks that are used to conduct the experiments. Furthermore, all scripts and results are available on the linked GitHub page.

The second and more important issue is the fairness of the results obtained. To assure this, we perform an 'apple-to-apple' comparison between the heuristics [23]. To elaborate, each experiment is run on the same hardware and under the same timeout and memory conditions. In addition, the outcomes of the experiments are not compared to the results of algorithms, discussed in other papers. The reason for this

is that the test setups used to perform experiments in other research are different than ours, hence, the comparison is not going to be adequate.

7 Conclusions and Future Work

In this paper, we have looked at the multi-mode resource-constrained project scheduling problem (MRCPSP). The MRCPSP is a generalization of the resource-constrained project scheduling problem (RCPSP) and is known to be NP-Hard [5]. The problem allows for activities to be executed in multiple modes and to use two types of resources, renewable and non-renewable. This paper has provided a new approach to solving the problem by using a MaxSAT solver. This approach utilizes scheduling rules to select variables to be explored first. This is accomplished by encoding the MRCPSP as a Boolean formula in weighted conjunctive normal form (WCNF), which can thereafter be used by the solver. In addition, the existing heuristic in the solver, Variable State Independent Decaying Sum (VSIDS), is updated to incorporate two MRCPSP-specific heuristics, Earliest Starting Time (EST) and Shortest Feasible Mode (SFM).

We have compared three heuristic configurations VSIDS, VSIDS + EST + SFM (VSIDS++), and EST + SFM (ESTSFM) on four different metrics and more than 3000 test instances. The results obtained from the experiments yield two important findings:

- ESTSFM performs significantly worse on every metric and benchmark compared to the other two alternatives. Our hypothesis is that this is due to the static nature of the heuristic and the fact that the variables' order does not change during the solver's execution;
- VSIDS++ takes between 5% and 42% less CPU time and finds more optimal solutions compared to VSIDS. Furthermore, based on the area under the time-objective curve (AUC), we have seen that VSIDS++ discovers a 'good' solution quicker than VSIDS.

Therefore, we conclude that VSIDS, which is widely used in SAT/MaxSAT solvers [15], cannot be completely replaced by domain-specific heuristics. However, based on the results obtained, we can say that propositional logic algorithms can be combined with domain-specific knowledge to obtain an algorithm that performs better when solving the MRCPSP.

However, research on this topic is not complete. Here, we propose ideas for future work that can be done on this topic:

- More heuristics should be compared. In this paper, we use two MRCPSP heuristics that are considered to be the most effective [6], however, the research by F. Boctor proposes a total of 10 different heuristics that could be combined in 21 different ways;
- Other benchmarks exist for the MRCPSP that are not considered in this paper [27]. These are big test instances with 100 jobs and up to 9 possible modes per activity. It will be beneficial, to see if the same results will be obtained for these instances as for the ones in this work;
- Another idea is to incorporate previous work into the current solver. For instance, in [18], a more compact encoding of the MRCPSP is discussed, as well as ideas for providing the MaxSAT solver with an initial solution that it can exploit to guide the search;
- Finally, we have chosen to use a MaxSAT solver as it is more flexible than dedicated algorithms. However, it would be insightful to examine the trade-off between flexibility and efficiency. For this, we propose that the improved solver that uses the VSIDS++ heuristic is compared to the state-of-the-art approaches for solving the MRCPSP.

References

[1] Mohammad Abdolshah. "A review of resource-constrained project scheduling problems (RCPSP) approaches and solutions". In: *International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies* 5.4 (2014), pp. 253–286.

[2] Ignasi Abıo et al. "A new look at BDDs for pseudo-Boolean constraints". In: *Journal of Artificial Intelligence Research* 45 (2012), pp. 443–480.

[3] Sprecher Arno. *Resource Constrained Project scheduling exact methods for the multi-mode case*. Vol. 409. Springer, 1994. DOI: 10.1007/978-3-642-48397-4. URL: https://doi.org/10.1007/978-3-642-48397-4.

[4] Armin Biere et al. "Conflict-driven clause learning sat solvers". In: *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications* (2009), pp. 131–153.

[5] Jacek Blazewicz, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. "Scheduling subject to resource constraints: classification and complexity". In: *Discrete Applied Mathematics* 5.1 (1983), pp. 11–24. ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(83)90012-4. URL: https://www.sciencedirect.com/science/article/pii/0166218X83900124.

[6] Fayez F. Boctor. "Heuristics for scheduling projects with resource restrictions and several resource-duration modes". In: *International Journal of Production Research* 31.11 (1993), pp. 2547–2558. DOI: 10.1080/00207549308956882. URL: https://doi.org/10.1080/00207549308956882.

[7] Fayez F. Boctor. "A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes". In: *European Journal of Operational Research* 90.2 (1996), pp. 349–361. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(95)00359-2. URL: https://www.sciencedirect.com/science/article/pii/0377221795003592.

[8] José Coelho and Mario Vanhoucke. "Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers". In: *European Journal of Operational Research* 213.1 (2011), pp. 73–82. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2011.03.019. URL: https://www.sciencedirect.com/science/article/pii/S037722171100230X.

[9] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 1)*. 2023. URL: https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1.

[10] The Rust Project Developers. *The Rust Programming Language*. Online. URL: https://www.rust-lang.org/.

[11] Andreas Drexl and Juergen Gruenewald. "Nonpreemtive Multi-mode Resource-constrained Project Scheduling". In: *IIE Transactions* 25.5 (1993), pp. 74–81. DOI: 10.1080/07408179308964317. URL: https://doi.org/10.1080/07408179308964317.

[12] Sönke Hartmann and Andreas Drexl. "Project scheduling with multiple modes: A comparison of exact algorithms". In: *Networks* 32.4 (1998), pp. 283–297. DOI: https://doi.org/10.1002/(SICI)1097-0037(199812)32:4<283::AID-NET5>3.0.CO;2-I.

[13] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. "PySAT: A Python Toolkit for Prototyping with SAT Oracles". In: *SAT*. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8_26.

[14] Rainer Kolisch and Andreas Drexl. "Local search for nonpreemptive multi-mode resource-constrained project scheduling". In: *IIE Transactions* 29.11 (1997), pp. 987–999. DOI: 10.1080/07408179708966417. URL: https://doi.org/10.1080/07408179708966417.

[15] Jia Hui Liang et al. "Understanding VSIDS Branching Heuristics in Conflict-Driven Clause-Learning SAT Solvers". In: *Hardware and Software: Verification and Testing*. Ed. by Nir Piterman. Cham: Springer International Publishing, 2015, pp. 225–241. ISBN: 978-3-319-26287-1.

[16] Antonio Lova, Pilar Tormos, and Federico Barber. “Multi-mode resource constrained project scheduling: Scheduling schemes, priority rules and mode selection rules”. In: *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial* 10.30 (2006), pp. 69–86.

[17] Joao Marques-Silva. “Search algorithms for satisfiability problems in combinational switching circuits”. PhD thesis. University of Michigan, 1995.

[18] João Miguel Teixeira de Mendonça. “Heuristic Augmentation of SAT Solvers for MRCPSP”. In: (2022).

[19] Linet Özdamar and Gündüz Ulusoy. “A local constraint based analysis approach to project scheduling under general resource constraints”. In: *European Journal of Operational Research* 79.2 (1994), pp. 287–298. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(94)90359-X. URL: https://www.sciencedirect.com/science/article/pii/037722179490359X.

[20] *Project Scheduling Problem Library - PSPLIB*. Mar. 2005. URL: https://www.om-db.wi.tum.de/psplib/main.html.

[21] Stephen M. Remde. “Enhancing the Performance of Search Heuristics. Variable Fitness Functions and other Methods to Enhance Heuristics for Dynamic Workforce Scheduling.” PhD thesis. University of Bradford, 2010. URL: http://hdl.handle.net/10454/4310.

[22] Jerzy Rosłon. “The multi-mode, resource-constrained project scheduling problem in construction: state of art review and research challenges”. In: *Technical Transactions* 114.5 (2017), pp. 67–74. DOI: doi:10.4467/2353737XCT.17.070.6427. URL: https://doi.org/10.4467/2353737XCT.17.070.6427.

[23] Ruben Ruiz. *State-of-the-art flowshop scheduling heuristics*. Dec. 2021. URL: https://www.youtube.com/watch?v=F3Ykma1eqnY.

[24] Roman Słowiński. “Two Approaches to Problems of Resource Allocation Among Project Activities — A Comparative Study”. In: *Journal of the Operational Research Society* 31.8 (1980), pp. 711–723. DOI: 10.1057/jors.1980.134. URL: https://doi.org/10.1057/jors.1980.134.

[25] Arno Sprecher and Andreas Drexl. “Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm”. In: *European Journal of Operational Research* 107.2 (1998), pp. 431–450. ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(97)00348-2. URL: https://www.sciencedirect.com/science/article/pii/S0377221797003482.

[26] Brian F. Talbot. “Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case”. In: *Management Science* 28.10 (1982), pp. 1197–1210. DOI: 10.1287/mnsc.28.10.1197. URL: https://doi.org/10.1287/mnsc.28.10.1197.

[27] Vincent Van Peteghem and Mario Vanhoucke. “An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances”. In: *European Journal of Operational Research* 235.1 (2014), pp. 62–72. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2013.10.012. URL: https://www.sciencedirect.com/science/article/pii/S0377221713008357.

[28] Guidong Zhu, Jonathan F. Bard, and Gang Yu. “A Branch-and-Cut Procedure for the Multimode Resource-Constrained Project-Scheduling Problem”. In: *INFORMS Journal on Computing* 18.3 (2006), pp. 377–390. DOI: 10.1287/ijoc.1040.0121. URL: https://doi.org/10.1287/ijoc.1040.0121.