Enhancing VSIDS with domain-specific information for the MRCPSP

Jarno Berger
Supervisors: Dr. Emir Demirović, Maarten Flippo, Imko Marijnissen
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Jarno Berger
Final project course: CSE3000 Research Project
Thesis committee: Dr. Emir Demirović, Maarten Flippo, Imko Marijnissen, Julia Olkhovskaia

Abstract

The Multi-Mode Resource Constraint Scheduling Problem is an NP-hard optimization problem. It arises in various industries such as construction engineering, transportation, and software development. This paper explores the integration of an adaptation of the Longest Processing Time heuristic to initialize the Variable State Independent Decaying Sum for the MRCPSP. This adaptation prioritizes tasks with a longer duration in all possible modes. Experimental evaluation demonstrates a 10% faster average computation time compared to default VSIDS, and the average deviation to the optimal solution is improved from 0.080% to 0.050%. These two findings combined show a minor improvement in using the LPT to initialize VSIDS values for the MRCPSP. Using the LPT also to initialize the default increment parameter, did not seem to yield any positive results.

1 Introduction

The Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) is a complex optimization problem that arises in various industries, such as construction engineering, transportation, and software development [27]. It involves scheduling tasks that require resources in multiple possible modes of operation, subject to various constraints, making it a well-established NP-Hard problem [1]. The optimization of the MRCPSP is crucial for improving resource allocation in these industries. However, finding an optimal solution for MRCPSP instances can be computationally intensive [13], requiring efficient solvers.

Extensive research has been conducted on solving the MRCPSP, exploring techniques ranging from exact algorithms to meta-heuristics. In the context of the MRCPSP, numerous domain-specific heuristics have been proposed to enhance the efficiency of scheduling algorithms. Commonly used heuristics include priority rules, which prioritize tasks based on criteria such as the earliest start time, slack time, or criticality. For instance, [15] presented several priority rule-based heuristics, showing their effectiveness in generating feasible schedules quickly. Other heuristic approaches, such as genetic algorithms and simulated annealing, have also been explored for their ability to provide near-optimal solutions within reasonable computational times [11].
Another approach is to use a Constraint Programming (CP) solver [5]. A CP solver is a tool that efficiently finds solutions to combinatorial optimization problems by exploring the search space while enforcing constraints to prune infeasible solutions. It utilizes constraint propagation techniques and search algorithms to iteratively refine solutions until an optimal solution is found. Laborie [18] demonstrates the potential of CP solvers in handling complex scheduling constraints and resource allocations.
Both of these ideas have been researched in the literature, however, there remains a knowledge gap in combining these two optimization techniques. This research aims to address this gap by investigating the potential of integrating domain-specific heuristics into CP solvers to improve the runtime for the MRCPSP. By leveraging insights from previous studies on CP solvers and domain-specific heuristics, this research seeks to identify a promising heuristic and integrate it into a state-of-the-art CP solver. The primary research question driving this study is:

Can domain-specific heuristics be integrated into a CP solver to improve solution times for the MRCPSP?

A novel idea was introduced in this study, to utilize an adaptation to the Longest Processing Time (LPT) heuristic to initialize VSIDS. The LPT is adapted to the MRCPSP by taking the minimum of all the mode durations, this ensures a task will at least that long to schedule. Then the LPT score is used to initialize VSIDS values to prioritize longer-duration tasks. Experimental testing shows a 10% faster average computation time for all instances when initializing VSIDS values with the LPT score. The average deviation from the optimal solution has also been improved from 0.080% to 0.050%. However, using the LPT to initialize the VSIDS default increment, resulted in equal or worse results. In conclusion, the integration of an LPT adaptation to initialize VSIDS values for the MRCPSP has shown to be effective. Yet whether the LPT can also be used in other parameters is not conclusive and still has to be researched further.
The remainder of this paper is structured as follows: First Section 2 provides a formal definition of the MRCPSP. After that Section 3 contains the related work to this subject and Section 4 provides a basic explanation of the methods used in this study. In Section 5 the contribution of this research is introduced. Then Section 6, the experimental setup will be described and the results will be shown and analysed. Section 7 reflects on the ethical aspects of this research. Finally, in Section 8 a conclusion will be drawn and possible future work will be discussed.

2 MRCPSP definition

Given is a set of activities N, numbered from 0 to $n+1$ where both the first and last nodes are dummies. Schedule these activities on R_r renewable resources and R_n non-renewable resources. Each renewable resource $k \in R_r$ has a constant amount of resource available of a_k^r per time unit while each non-renewable resource $l \in R_n$ is restricted to an amount of a_l^n resource over the entire planning. Each activity $i \in N$ can be executed in one of M_i modes ($d_{i,m}$, $r_{i,m,k}^r$, $r_{i,m,l}^n$) with $m \in \{1, 2, ..., M_i\}$. The start and end dummies have one mode and require no resources. Selecting a mode involves a duration $d_{i,m}$ for each activity i which requires a certain amount of resources. The goal is to schedule all tasks to ensure the shortest possible makespan.

In Fig. 1 an example can be found of the MRCPSP. On the left is a graph showing the tasks to be scheduled. The nodes represent the tasks, where nodes 1 and 9 are the dummies. The arrows connecting the nodes represent precedence constraints. For instance, the arrow from node 2 to node 4 indicates that task 2 must be completed before task 4 can be scheduled. The number at the top of a node represents the duration, while the number at the bottom is the resource consumption per time unit. Nodes 3,5,6,8 have 2 numbers each, which represent 2 different modes. The image on the right represents the solution to the problem. Each task is scheduled on the resource while adhering to the precedence and resource constraints. This optimal solution results in a final makespan of 10, when the last task is finished.

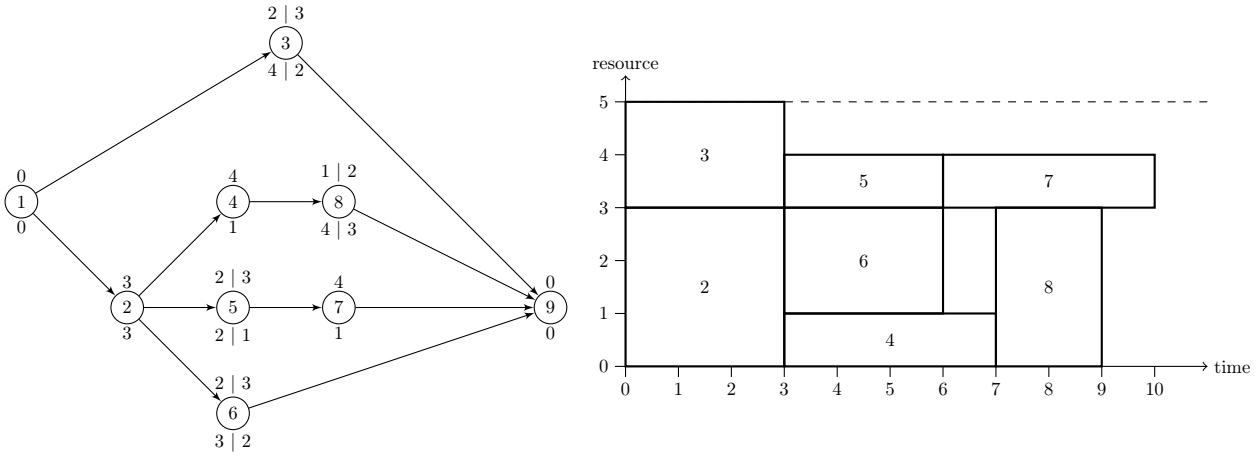

Figure 1: An example of the MRCPSP (J. Rezaeian, 2015)

A MIP model of the problem is defined by Talbot [26] as the following:

$$\text{Minimize} \quad \sum_{t=es_{n+1}}^{l_{n+1}tx_{n+1,1,t}} \tag{1}$$

$$\text{Subject to:} \quad \sum_{m=1}^{M_i} \sum_{t=es_i}^{ls_i} (t + d_{i,m}) x_{i,m,t} \leq \sum_{m=1}^{M_j} \sum_{t=es_j}^{ls_j} t x_{j,m,t} \quad \forall (i,j) \in A, \tag{2}$$

$$\sum_{m=1}^{M_i} \sum_{t=es_i}^{ls_i} x_{i,m,t} = 1 \quad \forall i \in N, \tag{3}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r^r_{i,m,k} \sum_{s=\max(t-d_{i,m},es_i)}^{\min(t-1,ls_i)} x_{i,m,s} \leq a^r_k \tag{4}$$

$$\forall k \in R^r \text{ and } t = 1, \ldots, T, \tag{5}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r^n_{i,m,l} \sum_{t=es_i}^{ls_i} x_{i,m,t} \leq a^n_l \quad \forall l \in R^n, \tag{6}$$

$$x_{i,m,t} \in \{0,1\} \quad \forall i \in N; m = 1, \ldots, M_i; t = 1, \ldots, T. \tag{7}$$

x_i, m, t is equal to 1 if activity i is performed in mode m and started at time t, otherwise 0. The first line is the optimization function which tries to minimize the total time it takes to schedule. The second line is a constraint that ensures that an activity can start without any delay after its predecessor finishes. The third line makes every activity only performed once and in one mode. Lines 4 and 5 satisfy the renewable resource constraint where T is an upper bound on the project duration. Line 6 limits the use of renewable sources. Finally, line 7 ensures binary decision variables. Note that the abbreviations es_i and ls_i are used to denote the earliest and latest start for activity i given the project upper bound T.

3 Related work

The Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP) has been extensively researched. In this section, we review some of the key contributions and approaches in the literature addressing MRCPSP.

Early work on MRCPSP focused on developing exact algorithms and mathematical formulations to solve instances of the problem optimally. Talbot [26] introduced one of the pioneering formulations of the MRCPSP, laying the groundwork for subsequent research. Brucker et al. [4] proposed a branch-and-bound algorithm for solving MRCPSP instances to optimality, showing the NP-hardness of the problem and the need for efficient exact algorithms.

Researchers have developed a variety of heuristic and metaheuristic approaches to tackle MRCPSP instances. Some of the most notable work on heuristics include: Boctor [2, 3] developed constructive heuristics that prioritize activities based on specific rules, such as the earliest start time or minimum slack, to efficiently allocate resources and schedule activities. Drexl [9] introduced a priority-rule-based heuristic that dynamically adjusts priorities as the project progresses, improving the flexibility and adaptability of the scheduling process. Kolisch [16] proposed a set of randomized priority rules combined with a backward-forward scheduling approach to enhance the search for optimal solutions.

On the other hand, meta-heuristic approaches offer more sophisticated mechanisms to explore the solution space and escape local optima. There are many different meta-heuristics, some notable ones include tabu search introduced by Slowsinksi [25], genetic algorithms by Mori [21] and Hartmann [12] implemented a hybrid genetic algorithm that integrates problem-specific heuristics into the genetic framework, improving its efficiency and effectiveness.

Other notable contributions include the work of Ozdamar [22], who developed a simulated annealing algorithm that probabilistically accepts worse solutions to escape local optima and explore the global solution space more thoroughly. Jozefowska [14] employed an ant colony optimization approach, where artificial ants construct solutions based on pheromone trails, reflecting collective learning and exploration.

Recent advancements in MRCPSP research include the integration of machine learning techniques and constraint programming to address complex scheduling scenarios. Chu [6] proposed a reinforcement learning approach for the MRCPSP, where the best algorithm for a specific instance is selected by a machine learning model.

Generally, the idea to solve the MRCPSP is to split the process into two steps. First, a mode assignment step is done which is modeled as a SAT solver. This problem has been researched extensively in literature [20]. The second step is the RCPSP scheduling step this can be done by using any optimization technique to improve this. These two steps result in two lists that can be used to solve the MRCPSP. In [7] it is shown that combining these two lists can also result in an improved runtime. Because recently this idea has shown very good results, less research is being conducted on CP solvers. So this research aims to explore the possibility of domain-specific heuristics in a CP solver. With positive results from this

study, it could open the doors to more research being conducted again for CP solvers.

4 Preliminaries

In this section, an overview will be given of the subjects required to know in order to understand the contribution of this study. In Chapter 4.1 the Longest Processing Time heuristic will be discussed. Then Chapter 4.2 will explain what a Constraint Programming Solver is. After that, in Chapter 4.3 the working of Conflict Driven Clause Learning will be given and how it is used in a CP solver. Finally, in Chapter 4.4 a high-level description of VSIDS will be shown.

4.1 Longest Processing Time

The Longest Processing Time (LPT) heuristic prioritizes tasks that have the longest processing time (duration) first. It is a common heuristic used in various fields of scheduling [23]. It works by sorting all tasks in descending order based on their processing time. Then going down the list schedule the task in the first available time slot, without breaking the resource constraint. The idea of this is that by scheduling longer tasks first, they will not become a bottleneck later. Ending off with a long task can push the makespan a lot further back. It is also harder to optimize resource usage with larger tasks, so by scheduling them first smaller tasks can be scheduled around the larger ones. This is better for resource usage than first scheduling all the small tasks and then having a lot of wasted resources when you are left with the large tasks at the end. Higher resource usage generally correlates with a shorter makespan. This is because when resources are fully utilized, tasks are completed more quickly, reducing the need to delay tasks. If resources are not fully used, it indicates that tasks that could have been scheduled are being postponed, leading to a longer overall project duration.

4.2 Constraint Programming Solver

A Constraint Programming (CP) solver is a tool used to solve combinatorial optimization problems. It does so by efficiently exploring the search space of possible solutions. It works by modeling the problem as a set of variables, each with a domain of possible values, and a set of constraints that must be satisfied by the variables. The solver searches through the possible variable assignments, using inference techniques to prune the search space and improve efficiency. These techniques use logical deductions to infer extra information from the current state of variable assignments and constraints. This information can then be used to reduce the search space by removing variable assignments that would eventually lead to an unsatisfied constraint.

4.3 Conflict Driven Clause Learning

Conflict Driven Clause Learning (CDCL) is a technique used in the field of SAT solving. It works the following way:

1. Initialization: The algorithm starts with an empty assignment and the original set of clauses.

2. Decision: A variable is chosen based on heuristics, and is either assigned true or false.

3. Propagation: Propagation is performed to reduce the search space based on the current assignment.

4. Conflict Detection: If a conflict is detected during propagation, the solver performs conflict analysis.

5. Learning: A new clause is learned from the conflict, representing the cause of the conflict.

6. Backtracking: The solver backtracks to an earlier decision level where the conflict can be resolved.

7. Restart: If necessary, the solver may restart, keeping the learned clauses.

CDCL is used in some CP solvers by integrating a lazy clause generator (LCG). LCG works by mapping the integer and other domain variables to boolean literals (e.g. x=0, x=1, x>1). Then when a conflict occurs during the search, conflict analysis is performed similarly to CDCL in SAT solvers. From the analysis, clauses are then generated that must be satisfied to prevent the same conflict from happening. These clauses essentially act as additional constraints that are generated during the search. An in-depth explanation of an LCG CP solver can be found in [10]

4.4 Variable State Independent Decaying Sum

Variable State Independent Decaying Sum, also known as VSIDS, is a heuristic created for SAT solvers. The idea behind VSIDS is to prioritize variables based on their past conflicts. It works by keeping track of the activity score for each variable. The initial score is usually set to 0. Then the variable with the highest activity will be selected. In case of a tie, it will be randomly selected. If a conflict is found during the search, all variables participating in the conflict will get their activity values increased. Over time all values decay, allowing for more exploration. This is done by dividing the activity scores by a decay factor. In order to optimize speed, the decay of activity values is done instead by multiplying future bumps by

$$1/decay\_factor$$

. This gives the same result as dividing the activity values but is computationally faster.

The VSIDS will have the following parameters to be initialized:

- default VSIDS value: the value at which an activity value starts. This can be any number but is usually set to 0.
- default VSIDS increment: the value that is added when activities participate in conflicts. This can also be set to any number, however, generally is 1.
- decay factor: the ratio that the activity values are decayed by. This is a number between 0 and 1.
- max threshold: the maximum value variables can get. Since the decay factor is applied using multiplication, activity values can become very large. Thus in order to prevent overflows, once this value is reached by a variable, all variables will be divided by this max threshold.

5 LPT adaptation in VSIDS

In this section, we go into the main contribution of this research: the use of an adaptation of the LPT for the MRCPSP to enhance VSIDS. Although VSIDS is highly efficient during the later stages of the search process, at the start of the search, all activity values are set equally. This causes a slow start since it will randomly select a variable. Instead of selecting randomly, a heuristic can be used to make a better initial guess on which variable will be most efficient to start with.

Inspired by the research conducted by [24] on augmenting VSIDS with starting values for the Resource-Constrained Project Scheduling Problem (RCPSP). Schutt initialized the values of VSIDS with domain-specific knowledge to guide the start of the search. Subsequently extended by [19] to RCPSP with Time Windows (RCPSP-t), we propose a novel enhancement. This enhancement involves incorporating an adaptation of the LPT to initialize the start values and increment of VSIDS for the MRCPSP.

The LPT adaptation will work as follows: For each task, all possible modes will be considered. The duration with the shortest possible mode will be used for the LPT score. For task i we can calculate the LPT score the following:

$$LPT_i = min(d_{i,m}) \quad \forall m \in M_i$$

By using the value of the smallest mode, we ensure that the processing time of that task will be at least the same if not longer no matter what mode will be selected. Thus this should essentially work the same as the LPT heuristic for the RCPSP, but with a slight adaptation to make it suitable for the MRCPSP. The LPT score is then used to initialize the variable values for VSIDS. Since the VSIDS can now be initialized with higher scores, the default increment parameter of VSIDS will also be tested in the experimental section. This will be done by trying to set the default increment to a value relative to the size of the LPT scores.

6 Experimental Setup and Results

In this section, the goal is to assess whether the adaptation of the LPT can be used to initialize VSIDS values and increments to improve the efficiency of the solver. In Chapter 6.1 the experimental setup will be described in detail. The results and analysis of the experiment are in Chapter 6.2.

6.1 Experimental Setup

The VSIDS enhanced with an adapted LPT score is implemented in the CP solver Pumpkin, developed in Rust. Pumpkin is an LCG CP solver, so the variables are boolean literals as explained in Chapter 4.3. All literals associated with a task will receive their corresponding LPT score. The literals that aren't associated with any tasks, will receive a score of 0. For the value selection a solution-guided value selector is used, an explanation of how this works can be found in [8]. The experiment will use the dataset generated by ProGen [17], featuring 30 activities with 2 renewable and 2 non-renewable resources. This dataset can be found in the PSPLIB and is also known as J30. This dataset also contains infeasible problems however, these will be excluded from the results since the main focus of this experiment is

to see if the improved solver can find solutions better and faster. The experiment will be performed locally on a computer with the following hardware:

- Processor: AMD Ryzen 7 2700X, 8 cores
- Memory: 16 GB DDR4-3200 Dual-channel (2 x 8 GB)
- Storage: Samsung 980, 3500 MB/s read, 3000 MB/s write
- OS: Windows 10 Home

By running the tests locally there is a chance of the results being skewed by performing other tasks at the same time. Thus the computer had no other applications open or was being used for anything else while the tests were running, to keep the results as accurate as possible.

The experiment will be conducted with four different configurations. The first configuration will be the default CP solver. This run will act as a baseline, to compare the others to. The other three configurations will be the augmented CP solver but with a different default VSIDS increment parameter. For all four the decay factor will be 0.95 and the max threshold will be 1e100. The other two parameters will be different per run as shown in Table 1.

Parameters	value	increment
VSIDS	0	1
LPT	LPT	1
LPT Avg	LPT	AVG_LPT
LPT Max	LPT	MAX_LPT

Table 1: VSIDS parameters per run

Two stop criteria of 30 and 100 seconds will be used, for each instance in the dataset. It will stop if it either reaches this time limit or finds an optimal solution. The stop criterion of 30 seconds is used to see how fast small instances are solved and for larger instances how close they can get to the optimal solution in a short period. The 100-second stop criterion will evaluate the solver's ability to converge to the optimal solution and see if it can solve some of the harder instances. Each time a solution is found, it is logged along with a timestamp. The performance of the baseline and enhanced CP solver will be evaluated using the following metrics:

- The average deviation from the optimal solution
- The computation time
- Amount of optimal solutions

The solver determines if a solution is optimal if no more solutions are available to explore. This implies that the solver may have already found the optimal solution, but it has not yet confirmed whether it is truly optimal.

6.2 Results

In Table 2 the results with a stop criterion of 30 seconds can be found. For the LPT adaptation, both the LPT and LPT Avg perform very close to each other, with the LPT Avg having a slightly better performance. The LPT Max is by far the worst out of the three, having a much higher deviation from the optimal and less optimal solutions found. Now comparing the LPT enhanced runs to the baseline of Default VSIDS. All three of them have an average computation time of about 0.1-0.2 seconds faster than the baseline. However, VSIDS has the lowest average deviation of 0.126% and the most optimal solutions found with 518. Since the LPT adaptation consistently solved fewer instances optimally but maintained a faster average time across all runs, it must have solved the optimally completed instances more quickly. From this, we can conclude that using the LPT score to initialize VSIDS can help solve smaller instances faster than default VSIDS. In this case, by using the LPT Avg, it is improved from 3.031 to 2.802 seconds, which is about a 7% increase in speed. However, for the larger instances that could not be solved optimally in 30 seconds, the default VSIDS has the lowest deviation of 0.126% compared to the 0.127% of the LPT Avg. This shows close to no change in the resulting makespans.

Parameters	Avg deviation	Avg time	Optimal	Satisfiable
VSIDS	0.126%	3.031s	518	32
LPT	0.130%	2.837s	517	33
LPT Avg	0.127%	2.802s	517	33
LPT Max	0.138%	2.911s	514	36

Table 2: Results with stop criterion of 30 seconds

Table 3 contains the results with a stop criterion of 100 seconds. We can see that the LPT configuration demonstrates the best overall performance, achieving the lowest average deviation, the fastest computation time, and the highest number of optimal solutions. The LPT Avg also performs well, only performing slightly worse in each metric. The LPT Max lags behind the other LPT adaptations regarding all metrics. This can be because the increments are already much larger than the initialized values, so after the first bumps, the initialized value will have close to no impact. Finally, the Default VSIDS performs the worst in both the average deviation and the average time under a 100-second stop criterion. The average deviation is 0.080% while the LPT Default Increment is 0.050%. This is a solid improvement, but the amount of optimal solutions still differs by merely 1. Most likely this is because the solver has already found a lot more optimal solutions, but is not done exploring the entire solution space. So as a result the LPT Default Increment has only found 1 more optimal solution, while in reality, it has found a lot more. The average computation time of the LPT Default Increment is also about 10% faster.

Parameters	Avg deviation	Avg time	Optimal	Satisfiable
VSIDS	0.080%	6.687s	526	24
LPT	0.050%	6.028s	527	23
LPT Avg	0.051%	6.101s	526	24
LPT Max	0.076%	6.558s	525	25

Table 3: Results with stop criterion of 100 seconds

To summarize the results presented in Table 2 and Table 3, we can draw several key conclusions about the performance of the various VSIDS adaptations and parameters. Using the LPT score to initialize values of VSIDS can offer significant improvements in computation speed, particularly for smaller instances within shorter time limits. For longer computational times, the LPT also seems to perform better than the default VSIDS. The LPT Avg and Max always performed either the same as the default LPT or worse, thus using the LPT to also initialize the default increment has not shown positive results.

7 Responsible Research

In this section, the ethical aspects of this study are discussed. The data sets that were used are all public and available in PSPLIB. All methods and parameters used in the study are thoroughly described. This improves the reproducibility of this study however, the CP solver Pumpkin used to run tests with, is as of the release of this paper still private. In the future, a paper will be released about it and the model will become public. Therefore once the model is public, this will improve the reproducibility. The tests were run locally, which can make it harder to reproduce. But all the hardware requirements and parameters are mentioned and can be used to reproduce the experimental setup.

The results of the experiment are also made public, they can be found on Github[1]. Public results allow for the possibility of critique and further analysis by others. The postprocessing scripts used to gather these results can also be found to verify correctness.

8 Conclusions and Future Work

In this study we looked at if domain-specific heuristics can improve the runtime of a CP solver for the MRCPSP. The augmentation of VSIDS led to a notable improvement of approximately 10% in average computation time across various parameterized runs. There was also a significant reduction in the average deviation from the optimal solution, decreasing from 0.080% to 0.050%. However, for the lower timeout of 30 seconds, there was no change in average deviation from the optimal. These findings indicate that the adapted LPT heuristic effectively enhances the performance of the CP solver with VSIDS for the MRCPSP, particularly for instances that are solved to optimality.

Despite these promising results, the study has certain limitations. The experiments were conducted using a specific dataset (ProGen's J30) and a limited set of parameters. These constraints may affect the generalizability of the findings. Nevertheless, the study makes a novel contribution to the field by integrating domain-specific heuristics into VSIDS, demonstrating significant performance gains for the MRCPSP.

Future research can expand upon this work by investigating the impact of other parameters, such as changing the decay factor or testing a broader range of default increments. The decay factor could scale depending on the size of the initialized values. The higher the initialized values are, the higher the decay factor could be.

[1] https://github.com/JarnoBerger/RP_Results

Testing the solver on different datasets, can also help generalize the results of this study. Different datasets, containing various sizes and combinations of resources could show different results.

Finally, a benchmarking study against other state-of-the-art solvers is necessary to compare the performance gains observed in this study. Using alternative performance metrics such as the minimal critical path length, is also important to verify the strengths and weaknesses of the adaptation. A graph could also be made of how the solution is approached over time to compare the adapted solver to the default.

References

[1] Jacek Blazewicz, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Scheduling under resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[2] F. F. Boctor. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *International Journal of Production Research*, 31(11):2547–2558, 1993.

[3] F. F. Boctor. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research*, 90(2):349–361, 1996.

[4] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1):3–41, 1999.

[5] N. Christofides and R. Alvarez-Valdes. An algorithm for the resource constrained project scheduling problem. In *European Journal of Operational Research*, volume 29, pages 262–273. Elsevier, 1987.

[6] Xianghua Chu, Shuxiang Li, Fei Gao, Can Cui, Forest Pfeiffer, and Jianshuang Cui. A data-driven meta-learning recommendation model for multi-mode resource constrained project scheduling problem. *Computers & Operations Research*, 157:106290, 2023.

[7] JosÃ© Coelho and Mario Vanhoucke. Multi-mode resource-constrained project scheduling using rcpsp and sat solvers. *European Journal of Operational Research*, 213(1):73–82, 2011.

[8] Emir Demirović, 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 *Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24*, pages 99–108. Springer, 2018.

[9] Andreas Drexl and Jörg Grünewald. Nonpreemptive multi-mode resource-constrained project scheduling. *IMA Journal of Management Mathematics*, 5(1):1–18, 1993.

[10] Thibaut Feydy and Peter J Stuckey. Lazy clause generation reengineered. In *International Conference on Principles and Practice of Constraint Programming*, pages 352–366. Springer, 2009.

[11] S. Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL)*, 45(7):733–750, 1998.

[12] Sönke Hartmann. Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research*, 102(1-4):111–135, 2001.

[13] Stefan Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[14] Joanna Józefowska, Jan Weglarz, Marcin J. Sobolewski, and Roman Slowinski. Multi-mode resource-constrained project scheduling with fuzzy activity durations and renewable resources. *European Journal of Operational Research*, 130(2):311–333, 2001.

[15] Rainer Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2):320–333, 1996.

[16] Rainer Kolisch and Andreas Drexl. Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IMA Journal of Management Mathematics*, 8(4):319–334, 1997.

[17] 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, 1995.

[18] Philippe Laborie. Complete mcs-based search: Application to resource constrained project scheduling. In *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 19–33. Springer, 2005.

[19] Tijs Lenssen. Augmenting vsids heuristic for the rcpsp/t by initializing activity values using domain-specific information. 2023.

[20] Joao P Marques-Silva and Karem A Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[21] Michiaki Mori and Choong-Ho Tseng. A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research*, 100(1):134–141, 1997.

[22] Linette Özdamar. A genetic algorithm approach to a general category project scheduling problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 29(1):44–59, 1999.

[23] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 4th edition, 2012.

[24] Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Solving the resource constrained project scheduling problem with generalized precedences by lazy clause generation. *arXiv preprint arXiv:1009.0347*, 2010.

[25] Roman Slowinski, Stefan Son, and Jan Weglarz. Dss for multi-mode resource-constrained project scheduling. *European Journal of Operational Research*, 79(2):299–310, 1994.

[26] F Brian Talbot. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management science*, 28(10):1197–1210, 1982.

[27] Song Zhang. Selection of multimode resource-constrained project scheduling scheme based on dea method. *Scientific Programming*, 2020:1–7, 2020.