Augmenting Constraint Programming Variable Selection with Domain-Specific Heuristics for a Prize-Collecting Scheduling Problem

Nikola Petrov
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: Nikola Petrov
Final project course: CSE3000 Research Project
Thesis committee: Dr.Emir Demirović, Maarten Flippo, Imko Marijnissen, Dr.Julia Olkhovskaia

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Abstract

This paper investigates the inclusion of domain-specific variable selection heuristics in Constraint Programming (CP) solvers for the Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR) problem. We propose two variable selection heuristics: a greedy variable selection method based on densities, Highest Density First (HDF), and a modified Variable State Independent Decaying Sum (VSIDS) initialized with job densities, referred to as VSIDS + Density. Experimental results on benchmark instance sets reveal that the proposed heuristics do not outperform the baseline VSIDS heuristic. Overall, they lead to higher conflict counts and slower convergence. These findings highlight the robustness of general-purpose heuristics like VSIDS in diverse problem instances. Future research should explore other domain-specific heuristics, as the current experiment demonstrates that the proposed heuristics do not improve performance.

1 Introduction

Scheduling is vital across industries, including manufacturing, healthcare, transportation, and education [1, 2, 3, 4]. Effective scheduling boosts throughput, ensures timely task completion, and prevents resource overuse [5]. Various scheduling problems arise due to unique operational characteristics [6].

This paper focuses on Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR), an NP-Hard problem introduced by Horn et al. [7]. Each job is associated with a prize, and the objective is to select a subset of jobs and find a feasible schedule that maximizes the total prize. Each job requires a common resource for part of its duration and a secondary resource for its entire duration. The common resource is shared by all jobs, while the secondary resource can differ per job. Time windows specify when each job can be processed. Figure 1 illustrates a simple schedule. Applications include particle therapy patient scheduling [8] and pre-runtime scheduling of avionic systems [9, 10].

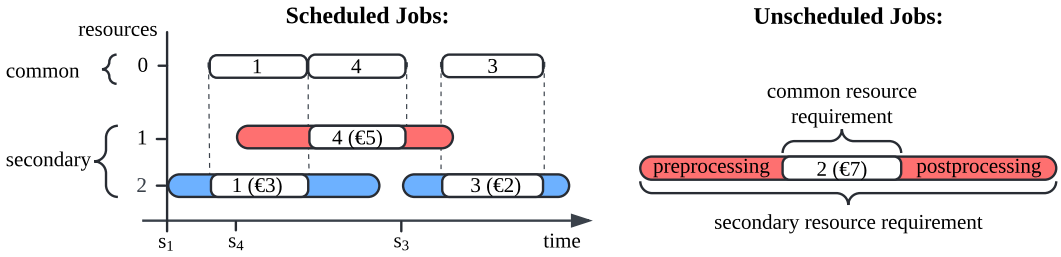

Figure 1: A solution to a Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources problem instance with 4 jobs and 2 secondary resources. Jobs 1, 3, and 4 are scheduled to start at times s_1, s_3, and s_4, respectively. Job 2 is not scheduled due to resource unavailability during its time windows.

Horn et al. explored Mixed Integer Programming (MIP), Constraint Programming (CP), and A* search for PC-JSOCMSR [7]. Improved results were achieved using Multivalued Decision Diagrams (MDDs) [11, 12]. Froger et al. presented a Branch-Cut-and-Price (BCP)

algorithm supported by an Iterated Local Search (ILS) heuristic, solving larger instances [13].

A trend which can be identified is that general-purpose methods such as CP and MIP fall behind algorithms tailored specifically for PC-JSOCMSR [7, 13]. This could stem from the limited problem-specific knowledge available to CP and MIP solvers. For example, Horn et al.s A* search approach [7] decides which job to schedule next based on a problem-specific heuristic, a tactic not employed by CP and MIP solutions. Ruiz et al. suggest that simple heuristics can sometimes outperform more involved solutions [14]. Therefore, enhancing traditional CP solvers to incorporate domain-specific knowledge to help guide their search procedure could potentially boost their performance and make them competitive with stateof- the-art algorithms.

This paper investigates the inclusion of PC-JSOCMSR-specific knowledge in CP solvers. We propose two variable selection heuristics based on job densities (prize to duration ratio):

1. **Highest Density First:** a heuristic prioritizing high-density jobs
2. **VSIDS + Density:** Variable State Independent Decaying Sum (VSIDS) [15] initialized with job densities.

Job densities are precomputed and passed to the CP solver. Experimental results show that neither Highest Density First (HDF) nor VSIDS + Density outperform the baseline VSIDS heuristic, generally leading to higher conflict counts and slower convergence times. These findings higjlight the robustness of general-purpose heuristics like VSIDS across various scenarios.

The paper is organized as follows: Section 2 provides a formal definition of PC-JSOCMSR. Previous work is discussed in Section 3. Section 4 covers Constraint Programming, VSIDS, and the CSP formulation of PC-JSOCMSR. Our heuristics are presented in Section 5. Experimental results are analyzed in Section 6. Section 8 addresses ethical aspects. Finally, Section 9 concludes and discusses future work.

2 Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources

This section provides a formal description of the Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR) problem in Subsection 2.1. Additionally, an important application of PC-JSOCMSR, particle therapy patient scheduling, is briefly introduced in Section 2.2 as it is worth understanding why this problem is worth solving and why it is worth improving current solutions.

2.1 Formal Definition

Let $J = \{1, \ldots, n\}$ denote the set of n jobs. Let $R_0 = \{0\} \cup R$ denote the set of resources, where resource 0 is the common resource, and $R = \{1, \ldots, m\}$ is the set of m secondary resources. Resources are renewable, meaning they replenish after being used, and can be used by one job at a time. Each job $j \in J$ is assigned to a secondary resource $q_j \in R$, which is required for the entire duration $p_j > 0$ of the job. The common resource is used

for p_j^0 units of time, starting after a preprocessing step lasting $p_j^{\text{pre}} \geq 0$ units of time. A postprocessing step lasting $p_j^{\text{post}} = p_j - p_j^{\text{pre}} - p_j^0 \geq 0$ units of time makes use only of the secondary resource. Jobs are processed without interruption once scheduled, meaning preemption is not allowed.

Each job $j \in J$ must be processed within one of ω_j disjoint time windows $W_j = \{W_{jk} \mid k = 0, \ldots, \omega_j\}$, where $W_{jk} = [w_{jk}^{\text{start}}, w_{jk}^{\text{end}}]$ and $w_{jk}^{\text{end}} - w_{jk}^{\text{start}} \geq p_j$. Additionally, jobs have a release time and a deadline:

$$T_j^{\text{rel}} = \min_{k=0,\ldots,\omega_j} w_{jk}^{\text{start}}$$

$$T_j^{\text{dead}} = \max_{k=0,\ldots,\omega_j} w_{jk}^{\text{end}}$$

Finally, each job $j \in J$ is associated with a prize $z_j > 0$. The objective of PC-JSOCMSR is to select a subset of jobs $S \subseteq J$ and find a feasible schedule maximizing the total prize (the sum of prizes of the scheduled jobs):

$$Z^* = \max_{S \subseteq J} Z(S) = \max_{S \subseteq J} \sum_{j \in S} z_j$$

2.2 Particle Therapy Patient Scheduling

An important application of PC-JSOCMSR is daily scheduling of cancer patients who are to receive particle therapy [8]. There, the common resource corresponds to a particle accelerator, whereas the secondary resources correspond to a small set of treatment rooms. The accelerator creates a particle beam which can be directed to only one room at a time. Before treatment begins, time is allocated to setting up the room and possibly sedating the patient and this whole procedure corresponds to the preprocessing step in PC-JSOCMSR. After the treatment is complete, the patient stays in the room a while longer such that medical checks can be carried out (this corresponds to the postprocessing step in PC-JSOCMSR).

3 Related Work

This section discusses problems similar to Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources in Section 3.1 and some solving approaches used in the literature in Section 3.2.

3.1 Related Problems

A problem motivated by particle therapy for cancer treatment [8] is Job Sequencing with One Common and Multiple Secondary Resources (JSOCMSR) [16]. JSOCMSR is a simplified version of PC-JSOCMSR that minimizes makespan without considering job prizes or processing time windows. Real-world applications necessitate prioritization due to time windows. This is addressed by PC-JSOCMSR [7].

PC-JSOCMSR can also be modelled as a Resource-Constrained Project Scheduling Problem (RCPSP) by splitting each job into three sub-jobs processed sequentially [17, 6]. However, reducing PC-JSOCMSR to RCPSP is unlikely to provide advantages due to the complexity of RCPSP.

3.2 Solving Approaches for PC-JSOCMSR

When PC-JSOCMSR was introduced, Horn et al. presented three approaches to solving it: A* search, Mixed Integer Programming (MIP) using order-based variables, 4 and Constraint Programming (CP) using option type variables [7]. They derived upper bounds on the total achievable prize for unscheduled jobs based on relaxations of a 0-1 multidimensional knapsack problem. These upper bounds were used to create a fast-tocalculate problem-specific heuristic that guided the A* search. For the MIP and CP solutions, they employed traditional modelling and solving techniques without incorporating PC-JSOCMSR-specific knowledge for variable/value selection. Horn et al. concluded that their A* search outperforms the MIP and CP solutions in terms of computation time, consistently solving small to medium-sized instances of up to 40 jobs to optimality.

Other methods used to solve PC-JSOCMSR include Multivalued Decision Diagrams (MDDs) and the Branch-Cut-and-Price (BCP) algorithm. Horn et al. introduced a novel construction scheme for relaxed MDDs, providing better bounds and more compact representations, successfully solving instances with up to 50 jobs to optimality [12, 11]. The state-of-the-art algorithm for solving PC-JSOCMSR - BCP [13] combined with an Iterated Local Search (ILS) heuristic iscapable of solving instances with up to 250 jobs to optimality.

Most progress in solving PC-JSOCMSR has been achieved through exact and heuristic solutions that exploit domain-specific knowledge. Using upper bounds of achievable prizes has provided better results than classical CP and MIP approaches, which lack such information. While CP and MIP are general-purpose, incorporating more domain-specific knowledge could improve their performance for PC-JSOCMSR.

4 Preliminaries

The solver used in this paper employs Lazy Clause Generation (LCG) [18]. Section 4.1 introduces Constraint Programming (CP) and how Constraint Satisfaction Problems (CSPs) are solved, including Satisfiability (SAT) solvers with Conflict-Driven Clause Learning (CDCL) and Variable State Independent Decaying Sum (VSIDS) [15]. Section 4.3 details the combination of CP and SAT solvers to achieve LCG solvers. Finally, Section 4.4 presents the CSP formulation of Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR).

4.1 Constraint Programming

Constraint Programming (CP) solves combinatorial optimization problems like PC-JSOCMSR by taking a Constraint Satisfaction Problem (CSP) as input, consisting of variables with predefined domains and constraints. CP solvers explore variable assignments systematically to satisfy all constraints.

CP solvers operate by:

- **Variable Assignment**: Assigning values to variables.
- **Constraint Propagation**: Pruning values from unassigned variables' domains.
- **Backtracking**: Revisiting previous assignments when conflicts are detected.

For example, in a scheduling problem with tasks A, B, and C under constraints $A \neq B$ and $A \leq C$, a CP solver might proceed as follows:

1. Assign $A = 1$ and prune 1 from B and C's domains.
2. Assign $B = 2$ and check constraints.
3. Assign $C = 3$; $A = 1, B = 2, C = 3$ is a valid schedule.

Informed variable selection techniques can greatly reduce the search space, leading to faster solutions [15].

For optimization problems like PC-JSOCMSR, the search restarts with an additional constraint for a better objective value each time a new solution is found. For more details on CP, refer to [19].

4.2 Satisfiability Solvers

SAT solvers [20] determine the satisfiability of propositional logic formulas in Conjunctive Normal Form (CNF). They efficiently handle the Boolean satisfiability problem using techniques like Conflict-Driven Clause Learning (CDCL) and Variable State Independent Decaying Sum (VSIDS) [15]. Section 4.2.1 and Section 4.2.2 describe CDCL and VSIDS in more detail.

4.2.1 Conflict-Driven Clause Learning

Conflict-Driven Clause Learning (CDCL) enhances solver efficiency by learning from conflicts to prune the search space [15]. For example:

$$\text{Given: } (X \vee Y) \wedge (\neg X \vee Z) \wedge (\neg Y \vee \neg Z)$$

A SAT solver proceeds as follows:

1. Assign $X = \text{true}$ and propagate $Z = \text{true}$.
2. Assign $Y = \text{true}$; conflict triggers CDCL:
   - Analyze conflict: $Y = \text{true}, Z = \text{true}$.
   - Learn clause: $\neg X \vee \neg Y$.
   - Backtrack: Add the new clause and avoid the conflict.

4.2.2 Variable State Independent Decaying Sum

Variable State Independent Decaying Sum (VSIDS) is a heuristic that prioritizes variables likely to cause conflicts [15]. It works by assigning initial weights to variables, increasing weights on conflict, and selecting variables with the highest weights. VSIDS parameters include:

- **Increment**: Amount weight increases on conflict. (a value of 1 is used in this paper)
- **Decay Factor**: Adjusts how quickly the solver "forgets" less important variables. (a value of 0.95 is used in this paper)
- **Max Threshold**: Prevents solver from getting stuck in local extremes by scaling down weights. (a value of 10^{100} is used in this paper)

4.3 Lazy Clause Generation

Lazy Clause Generation (LCG) combines CP and SAT solvers, leveraging CP's concise representations and SAT's learning techniques (CDCL and VSIDS) [18]. CP variables and constraints are converted into SAT clauses.

For example, translating the simple scheduling problem from Section 4.1 into clauses:

Domain Clauses $A \in \{1, 2, 3\}$ becomes $(A = 1 \vee A = 2 \vee A = 3)$. (similarly for B and C)

Mutual Exclusivity $(\neg(A = 1) \vee \neg(A = 2))$ ensures A is assigned to a single value. (similarly for B and C)

***A* ≠ *B* constraint** becomes $(A \neq 1 \vee B \neq 1)$.

***A* ≤ *C* constraint** becomes $(A = 1 \rightarrow C \neq 1)$.

4.4 CSP Formulation of PC-JSOCMSR

The CSP formulation of PC-JSOCMSR, developed by Maarten Filippo and Imko Marijnissen, is defined as follows:

$$\max \sum_{j \in J} S_j \cdot z_j \tag{1}$$

$$\text{s.t. } s_j \in \{T_j^{\text{rel}}, \ldots, T_j^{\text{dead}} - p_j\}, \qquad \forall j \in J, \tag{2}$$

$$S_j \in \{0, 1\}, \qquad \forall j \in J, \tag{3}$$

$$B_{ij} \in \{0, 1\}, \qquad \forall i, j \in J,\ i \neq j, \tag{4}$$

$$B_{ij} \implies S_i \wedge S_j \wedge (P_{ij} \vee P_{ji}), \qquad \forall i, j \in J,\ i \neq j, \tag{5}$$

$$P_{ij} \in \{0, 1\}, \qquad \forall i, j \in J,\ i \neq j, \tag{6}$$

$$P_{ij} \implies (s_i + p_i^{\text{pre}} + p_i^0 \leq s_j + p_j^{\text{pre}}), \qquad \forall i, j \in J,\ i \neq j, \tag{7}$$

$$P_{ij} \implies (s_i + p_i \leq s_j), \qquad \forall i, j \in J,\ i \neq j,\ q_i = q_j \tag{8}$$

$$\Omega_{jk} \in \{0, 1\}, \qquad \forall j \in J,\ k \in \{0, \ldots, \omega_j\}, \tag{9}$$

$$\Omega_{jk} \iff (w_{jk}^{\text{start}} \leq s_j \wedge s_j + p_j \leq w_{jk}^{\text{end}}), \qquad \forall j \in J,\ k \in \omega_j, \tag{10}$$

$$S_j \implies (\Omega_{j0} \vee \ldots \vee \Omega_{j\omega_j}), \qquad \forall j \in J. \tag{11}$$

For each job $j \in J$, s_j is the start time variable (2), and S_j is a binary variable indicating if the job is scheduled (3). B_{ij} indicates if both jobs i and j are scheduled (4), with consistency ensured by (5). P_{ij} indicates if job i precedes job j (6), with (7) and (8) ensuring the end time of i is before the start time of j. Ω_{jk} indicates if job j is scheduled within its time window k (9), with (10) ensuring scheduling within time windows. (11) ensures jobs respect their time windows if scheduled. The objective (1) is to maximize the total prize of scheduled jobs.

5 Domain-Specific Variable Selection Heuristics

In addition to general-purpose variable selection heuristics like VSIDS, problem-specific heuristics can be highly effective in certain contexts [21]. This section introduces two

domain-specific variable selection heuristics tailored for the Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR) problem.

In PC-JSOCMSR, each job $j \in J$ is associated with a prize z_j, and the objective is to maximize the total prize of the produced schedule. This is analogous to the Weighted Knapsack Problem (WKP), where a job's density d_j is defined as the ratio of its prize z_j to its total processing time p_j:

$$d_j = \frac{z_j}{p_j}$$

For combinatorial optimization problems like PC-JSOCMSR, many feasible solutions often exist. Guiding Constraint Programming (CP) solvers to find near-optimal solutions quickly can facilitate early pruning of the solution space, resulting in faster convergence.

Inspired by Dantzig et al.'s greedy algorithm for the 0-1 knapsack problem, which selects items based on the largest density and yields near-optimal solutions [22], we propose two domain-specific heuristic variable selection methods based on job densities:

1. Highest Density First (HDF) in Section 5.1
2. A combination of HDF and VSIDS in Section 5.2

5.1 Highest Density First

The Highest Density First (HDF) heuristic selects variables in descending order of their associated job densities. This approach is inspired by Dantzig et al.'s greedy algorithm for the 0-1 knapsack problem [22]. Our hypothesis is that applying a similar method to PC-JSOCMSR will help find near-optimal solutions early in the CP search procedure, aiding in pruning inferior solutions and speeding up convergence.

To implement HDF in a Lazy Clause Generation (LCG) solver, we must first translate CP variables into clauses suitable for a SAT solver. The objective function is represented by an extra variable $Z \in \{0, \sum_{j \in J} z_j\}$ used for branching. Each PC-JSOCMSR job $j \in J$ is represented by CP variables indicating the job's start time s_j and whether it is scheduled S_j. Since other variables' values can be inferred from s_j and S_j, they are not used for branching. After converting all variables into clauses, we obtain boolean literals for branching on s_j, S_j, and Z.

The boolean literals used for branching are:

- For s_j: corresponding equality literals $(s_j = T_j^{\text{rel}}), \ldots, (s_j = T_j^{\text{dead}})$
- For S_j: $(S_j = 1), (S_j = 0)$
- For Z: corresponding equality literals $(Z = 0), \ldots, (Z = \sum_{j \in J} z_j)$ and lower bound literals $(Z \geq 0), \ldots, (Z \geq \sum_{j \in J} z_j)$

After creating a list of literals corresponding to s_j, S_j, and Z, they are stably sorted based on the respective jobs' densities. Since Z is not associated with any job, its literals are assigned a weight of 0 and remain at the end of the list.

Finally, this ordered list of variables is passed to the solver. The variable selection method picks variables in the order they appear in the list, ensuring the desired behaviour of HDF.

5.2 VSIDS + Density Initialization

VSIDS [15] is a highly effective general-purpose heuristic used with Conflict-Driven Clause Learning (CDCL) solvers [20, 23]. However, the default variable weights in VSIDS may take time to become effective. By initializing VSIDS' variable weights with the corresponding job densities, we aim to make more informed variable selections early in the solving process.

6 Experimental Setup and Results

This section provides an overview of our experimental setup and results. Subsection 6.1 elaborates on the characteristics of the instance sets used in the experiment. Subsequently, the setup and metrics are detailed in 6.2. Finally, subsection 7 provides our results and interpretation.

6.1 Instance Sets

For the purpose of this experiment, we used two instance sets created by Horn et al. [7]. These instance sets, B and S, are inspired by the particle therapy patient scheduling application and are available online[1]. Each set consists of several batches: 30 instances for each combination of $n \in \{10, 20, \ldots, 90\}$ jobs and $m \in \{2, 3\}$ secondary resources.

- **B (Balanced):** Instances generated to ensure that the usage of each secondary resource is roughly balanced.
- **S (Skewed):** Instances generated such that the usage of each secondary resource is skewed, with 50% of the jobs requiring the same secondary resource, while the other 50% of the jobs are spread evenly across the remaining secondary resources.

Details on the creation of each instance set are provided in Appendix A.1.

6.2 Experimental Setup

The purpose of this experiment is to compare our proposed variable selection heuristics: Highest Density First (HDF) and VSIDS initialized using job densities (VSIDS + Densities), against the general-purpose VSIDS heuristics used as a baseline.

Pumpkin, the LCG solver used for this experiment, was developed by Dr. Emir Demirović, Maarten Flippo, and Imko Marijnissen in Rust[2]. The code base was compiled using rustc 1.78.0[3]. All instances were run on an HP ZBook Power G7 Mobile Workstation with an Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz and 16GB of RAM. The operating system

[1] `https://www.ac.tuwien.ac.at/research/problem-instances/`
[2] `https://www.rust-lang.org/`
[3] `https://doc.rust-lang.org/rustc/what-is-rustc.html`

used by this machine is Microsoft Windows 11 Home 10.0.22631 Build 22631, however, all experiments have been run through Windows Subsystem for Linux. The CPU time limit was set to 15 minutes in single-threaded mode, as this is a reasonable time frame for computing a good schedule in real-world scenarios. Memory usage was not restricted.

For each PC-JSOCMSR instance, we recorded the following:

- The total prize **Z** of each schedule found and the time **t** taken to compute it.
- The number of conflicts **K**which occurred in order to compute each schedule.
- Pumpkin's final state:
  - **O** - An optimal schedule has been found and proven to be optimal.
  - **S** - A solution has been found, but it is not necessarily optimal.
  - **U** - No solution has been found.

For each batch of 30 instances from the same instance set, number of jobs n, and secondary resources m, we collected the following statistics:

- $\bar{Z}_{\text{best}}$ - The average best total prize Z.
- $\overline{AOC}$ - The average area under the curve, used to measure the convergence rate. This metric is central to our experiment, as it evaluates how quickly the proposed heuristics converge compared to VSIDS. To compute this value, we aggregated the results from the three solvers (VSIDS, HDF, and VSIDS + Densities). First, AOC was computed for each run. Then, the maximum AOC across the three algorithms for each instance was normalized. Finally, the average of all normalized AOCs within each n, m batch was taken.
- $\bar{K}$ - The average number of conflicts across all instances in a batch. $\bar{K}$ was computed by first averaging the conflicts K encountered for each schedule, then averaging these values across all instances in the batch.
- Batch status - The number of instances for which Pumpkin finishes in **optimal**, **satisfiable**, and **unknown** states.

7 Results

7.1 Analysis of B (Balanced) Instance Set

Figure 2 illustrates the performance of different heuristics on the B (Balanced) instance set.

- **Solution Quality (Top Row)**: The histograms in the top row of Figure 2 show the percentage of the highest Z_{best} achieved. The VSIDS and VSIDS + Density heuristics generally maintain high solution quality across various job sizes (n), with VSIDS + Density occasionally outperforming the baseline VSIDS. In contrast, HDF shows lower performance, particularly as the number of jobs increases, indicating its inefficiency in balanced scenarios.

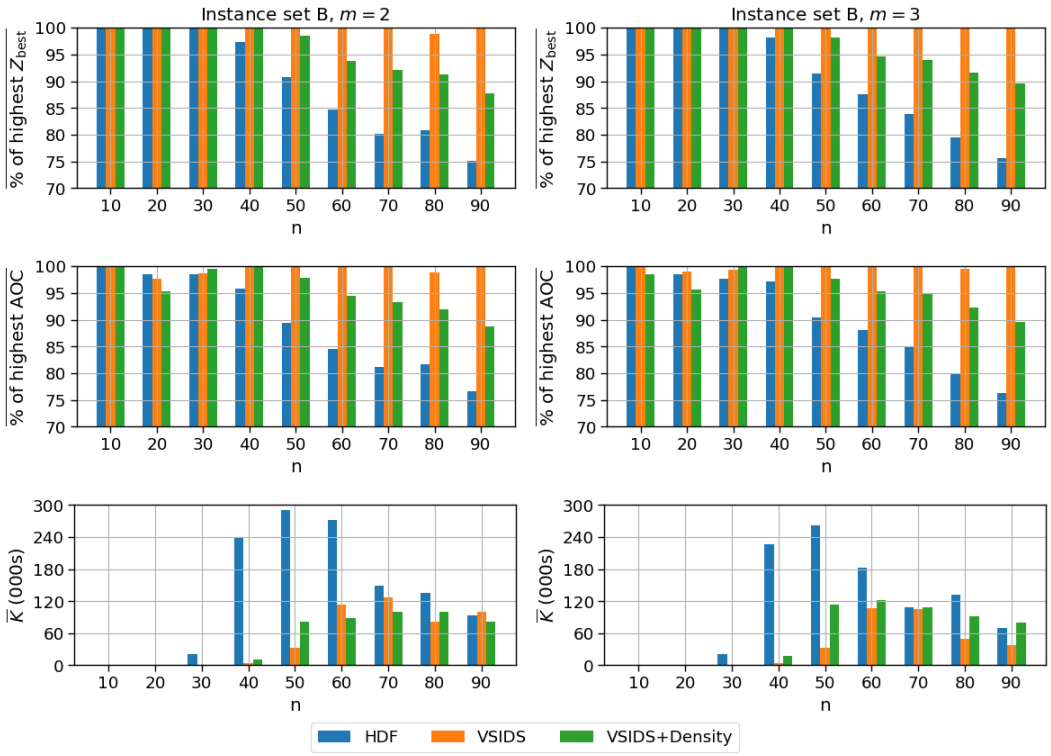

Figure 2: Histograms showing the distribution of total prizes collected across the B (Balanced) instance set for each heuristic.

B	m = 2									m = 3								
	HDF			**VSIDS**			**VSIDS + Density**			HDF			**VSIDS**			**VSIDS + Density**		
n	O	S	U	O	S	U	O	S	U	O	S	U	O	S	U	O	S	U
10-30	optimal									optimal								
40	13	17	0	**30**	**0**	**0**	**30**	**0**	**0**	5	25	0	**30**	**0**	**0**	**30**	**0**	**0**
50-90	satisfiable									satisfiable								

Table 1: Batch status outcomes for the B (Balanced) instance set with 2 and 3 secondary resources using HDF, VSIDS, and VSIDS + Density variable selection heuristics.

S	m = 2									m = 3								
	HDF			**VSIDS**			VSIDS + Density			HDF			**VSIDS**			VSIDS + Density		
n	O	S	U	O	S	U	O	S	U	O	S	U	O	S	U	O	S	U
10, 20	optimal									optimal								
30	25	5	0	**30**	**0**	**0**	**30**	**0**	**0**	21	9	0	**30**	**0**	**0**	**30**	**0**	**0**
40	0	30	0	**23**	**7**	**0**	20	10	0	0	30	0	**16**	**14**	**0**	13	17	0
5090	satisfiable									satisfiable								

Table 2: Batch status outcomes for the S (Skewed) instance set with 2 and 3 secondary resources using HDF, VSIDS, and VSIDS + Density heuristics.

- **Convergence Rate (Middle Row)**: The middle row shows the percentage of the highest area under the curve (AOC). Here, both VSIDS and VSIDS + Density display robust convergence rates, frequently hitting high percentages close to the best AOC found in the experiments. HDF, on the other hand, often lags, particularly for larger instances, underscoring its slower convergence in balanced settings.
- **Number of Conflicts (Bottom Row)**: The bottom row shows the average number of conflicts ($\bar{K}$). HDF consistently shows a higher number of conflicts, especially for larger job sizes. This high conflict count can explain its poorer performance in solution quality and convergence rate. In contrast, VSIDS and VSIDS + Density show significantly fewer conflicts, supporting their better performance in achieving high-quality solutions efficiently. By observing the poor results of HDF, we can also conclude that the lower performance of VSIDS + Density compared to VSIDS is due to this strategy of choosing jobs with the highest density. The initialization of VSIDS negatively affects the beginning stages of the search procedure by creating a larger number of conflicts, and therefore the whole search is affected negatively in terms of solution quality and convergence rate when it comes to instances with 40 or more jobs.

The batch status data in Table 1 for the B instance set supports these observations. HDF achieves optimal solutions for smaller job sizes but struggles as the job count increases, resulting in fewer optimal solutions and more satisfiable ones. This aligns with the histogram data, showing that HDF might not be well-suited for balanced scenarios as the complexity increases.

7.2 Analysis of S (Skewed) Instance Set

Figure 3 illustrates the performance across the S (Skewed) instance set.

- **Solution Quality (Top Row)**: The histograms in the top row of Figure 3 show that HDF performs worse than VSIDS and VSIDS + Density in skewed scenarios. VSIDS + Density occasionally matches the baseline VSIDS but generally shows poorer performance compared to VSIDS.
- **Convergence Rate (Middle Row)**: The middle row of histograms shows the percentage of the highest AOC. HDF demonstrates a lower convergence rate in skewed scenarios, often failing to reach high percentages. This indicates that HDF struggles to exploit the resource skewness to find good solutions quickly. VSIDS and VSIDS + Density both show better convergence rates, with VSIDS performing the best.

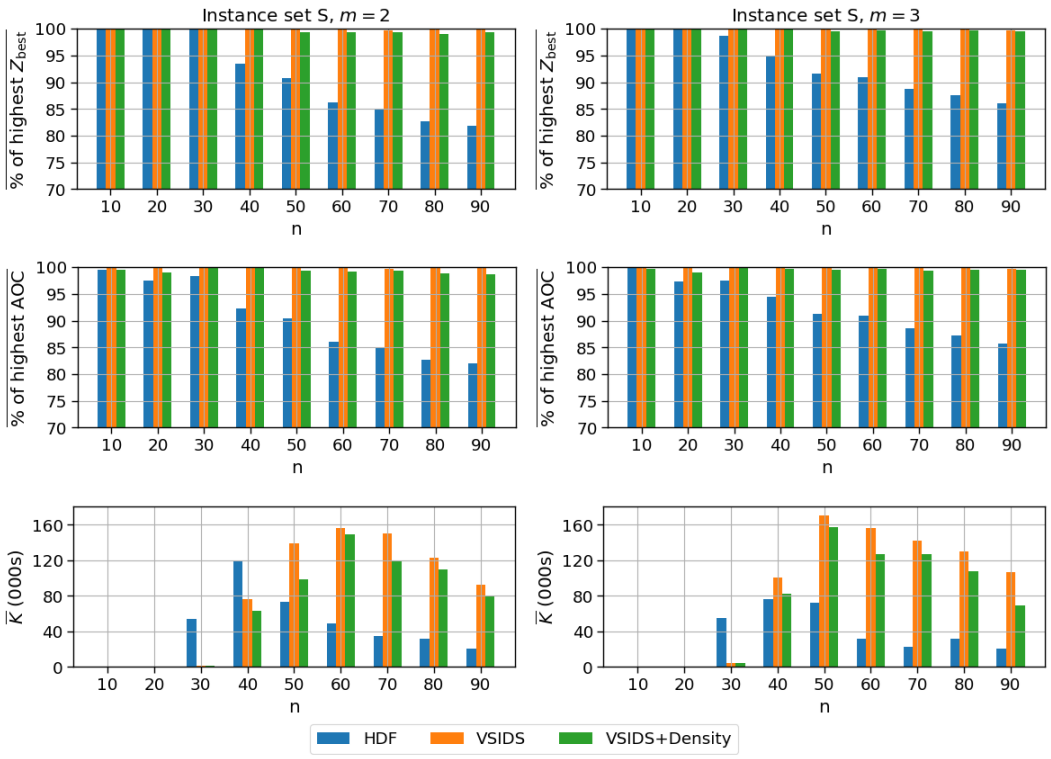

Figure 3: Histograms showing the distribution of total prizes collected across the S (Skewed) instance set for each heuristic.

- **Number of Conflicts (Bottom Row)**: The bottom row illustrates the average number of conflicts ($\bar{K}$). HDF maintains a relatively high number of conflicts even in skewed scenarios, which significantly hinders its solution quality. The negative effect of initializing VSIDS with job densities is more evident in the skewed instance set compared to the balanced one, as the number of conflicts of VSIDS + Density is comparable to the one of HDF. Correlation between the low number of conflicts and the superior solution quality of VSIDS can be observed for the skewed instance set as well.

The batch status data in Table 2 for the S instance set highlights that HDF consistently performs worse for larger job sizes, finding no optimal solutions. This is primarily due to the high number of conflicts generated when prioritizing high-density jobs. The comparison between VSIDS and VSIDS + Density shows that the density initialization negatively affects VSIDS + Density for the same reason. However, VSIDS + Density adjusts the initial density weights in later search stages, which lets it escape the poor performance of HDF. In summary, the inferiority of HDF and VSIDS + Density compared to VSIDS is mainly caused by the large number of conflicts arising from prioritizing high-density jobs.

7.3 General Observations

Across the experiments, it is evident that the effectiveness of each heuristic is primarily influenced by the algorithmic strategy employed rather than the specific characteristics of the instance sets.

- **HDF**: Consistently underperforms due to its strategy of prioritizing high-density jobs, which leads to a large number of conflicts and slower convergence. This makes it unsuitable for both balanced and skewed scenarios.
- **VSIDS + Density**: While incorporating job densities into VSIDS initially appears promising, it ultimately performs poorly compared to the baseline VSIDS. The density-based initialization creates more conflicts, similar to HDF, negatively affecting its overall performance. Although VSIDS + Density adjusts the variable weights over time based such that the ineffective strategy of prioritizing high densities is "forgotten", VSIDS + Density remains inferior to VSIDS.
- **VSIDS**: Outperforms both HDF and VSIDS + Density, maintaining strong performance across both instance sets. This demonstrates the robustness of the VSIDS general purpose variable selection method.

Overall, these results highlight that the choice of a variable selection heuristic plays a crucial role in performance. HDF and VSIDS + Density are less effective due to their high conflict rates stemming from prioritizing high-density jobs. VSIDS, on the other hand, proves to be a more reliable and efficient heuristic across various scenarios.

8 Responsible Research

The Netherlands Code of Conduct [24] outlines five principles: honesty, scrupulousness, transparency, independence, and responsibility. These principles guide our research practices, ensuring adherence to the highest standards of integrity and ethics.

Once the Pumpkin solver paper is published by Emir Demirovi, Maarten Flippo, and Imko Marijnissen, the PC-JSOCMSR model will be made available for public scrutiny. This transparency enables third parties to reproduce the research, increasing its credibility. Detailed descriptions of the methods will facilitate replication.

To further enhance transparency, our raw results, published online[4] with scripts used for processing and visualization, make it possible to validate that our results are not cherry-picked. We have ensured reproducibility by thoroughly documenting machine specifications, including hardware, operating systems. Programming language and compiler versions are specified to minimize inconsistencies. This meticulous documentation enhances the reliability of our findings.

No sensitive data was processed during this project. All datasets are synthetically generated to mimic real-world applications of PC-JSOCMSR (particle therapy patient scheduling), ensuring privacy and data security.

9 Conclusion and Future Work

Constraint Programming (CP) is a versatile approach to solving combinatorial optimization problems but often underperforms compared to tailored algorithms. This paper investigated domain-specific variable selection methods for the Prize-Collecting Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR) problem.

We compared a general-purpose variable selection heuristic, VSIDS, with two proposed domain-specific methods based on job densities. Results showed that the Highest Density First (HDF) heuristic and VSIDS + Density underperformed compared to the baseline VSIDS due to increased conflicts. VSIDS proved to be more reliable and efficient across various scenarios, highlighting the robustness of this general-purpose heuristic.

Future research should explore heuristics based on 0-1 knapsack relaxations of PC-JSOCMSR, as used by Horn et al. [7], and value selection methods that maximize resource usage overlap. These approaches could enhance scheduling efficiency and increase total prize. Additionally, validating the solver's effectiveness with real-world datasets, rather than synthetic ones, is crucial. Collaborating with industry partners to obtain and share real-world data will be beneficial.

References

[1] Michael Pinedo. *Planning and scheduling in manufacturing and services*. Springer, 2005.

[2] Brecht Cardoen, Erik Demeulemeester, and Jeroen Beliën. "Operating room planning and scheduling: A literature review". In: *European journal of operational research* 201.3 (2010), pp. 921–932.

[4] `https://github.com/nppetrov02/cp-density-var-selection-results`

[3] Cynthia Barnhart and Amy Cohn. "Airline schedule planning: Accomplishments and opportunities". In: *Manufacturing & service operations management* 6.1 (2004), pp. 3–22.

[4] Edmund Kieran Burke and Sanja Petrovic. "Recent research directions in automated timetabling". In: *European journal of operational research* 140.2 (2002), pp. 266–280.

[5] Dileep R Sule. *Production planning and industrial scheduling: examples, case studies and applications*. CRC press, 2007.

[6] Sönke Hartmann and Dirk Briskorn. "An updated survey of variants and extensions of the resource-constrained project scheduling problem". In: *European Journal of operational research* 297.1 (2022), pp. 1–14.

[7] Matthias Horn, Günther R Raidl, and Elina Rönnberg. "A* search for prize-collecting job sequencing with one common and multiple secondary resources". In: *Annals of Operations Research* 302.2 (2021), pp. 477–505.

[8] Johannes Maschler et al. "Particle therapy patient scheduling: First heuristic approaches". In: *Proceedings of the 11th Int. Conference on the Practice and Theory of Automated Timetabling. Udine, Italy*. 2016, pp. 223–244.

[9] Mathias Blikstad et al. "An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system". In: *Optimization and Engineering* 19 (2018), pp. 977–1004.

[10] Emil Karlsson et al. "A matheuristic approach to large-scale avionic scheduling". In: *Annals of Operations Research* 302.2 (2021), pp. 425–459.

[11] Matthias Horn et al. "A-based construction of decision diagrams for a prize-collecting scheduling problem". In: *Computers & Operations Research* 126 (2021), p. 105125.

[12] Johannes Maschler and Günther R Raidl. "Multivalued decision diagrams for prize-collecting job sequencing with one common and multiple secondary resources". In: *Annals of Operations Research* 302 (2021), pp. 507–531.

[13] Aurélien Froger and Ruslan Sadykov. "New exact and heuristic algorithms to solve the prize-collecting job sequencing problem with one common and multiple secondary resources". In: *European Journal of Operational Research* 306.1 (2023), pp. 65–82.

[14] Luis Fanjul-Peyro and Rubén Ruiz. "Iterated greedy local search methods for unrelated parallel machine scheduling". In: *European Journal of Operational Research* 207.1 (2010), pp. 55–69.

[15] Matthew W Moskewicz et al. "Chaff: Engineering an efficient SAT solver". In: *Proceedings of the 38th annual Design Automation Conference*. 2001, pp. 530–535.

[16] Matthias Horn, Günther Raidl, and Christian Blum. "Job sequencing with one common and multiple secondary resources: A problem motivated from particle therapy for cancer treatment". In: *Machine Learning, Optimization, and Big Data: Third International Conference, MOD 2017, Volterra, Italy, September 14–17, 2017, Revised Selected Papers 3*. Springer. 2018, pp. 506–518.

[17] Soenke Hartman and Dirk Briskorn. "A survey of variants and extensions of the resource-constrained project scheduling problem Cc: 000". In: *Operations Research Management Science* 51.1 (2011), p. 67.

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

[19] Barbara M Smith. "A tutorial on constraint programming". In: (1995).

[20] Karem A Sakallah et al. "Anatomy and empirical evaluation of modern SAT solvers". In: *Bulletin of the EATCS* 103 (2011), pp. 96–121.

[21] Jussi Rintanen. "Planning as satisfiability: Heuristics". In: *Artificial intelligence* 193 (2012), pp. 45–86.

[22] George B Dantzig. "Discrete-variable extremum problems". In: *Operations research* 5.2 (1957), pp. 266–288.

[23] SAT Solvers. "Conflict-Driven Clause Learning". In: *Handbook of Satisfiability* 336 (2021), p. 133.

[24] *Netherlands Code of Conduct for Research Integrity*. Accessed: 2024-06-23. 2018. URL: https://www.universiteitenvannederland.nl/files/publications/Netherlands%20Code%20of%20Conduct%20for%20Research%20Integrity%202018.pdf.

A Instance Set Descriptions

A.1 Particle therapy patient scheduling instance sets

As described by Horn et al. [7], the two particle therapy patient scheduling inspired instance sets B and S differ in the sense that the secondary resources allocated to each job are distributed evenly between the jobs in set B, whereas in set S they are skewed. For each job $j \in J$ in set B, the secondary resource q_j was sampled from a discrete uniform distribution $U(1, m)$ (from now on we use the following notation $U(l, u)$ which refers to a discrete uniform distribution with lower and upper bounds l and u respectively). Meanwhile, in set S, the secondary resource m is chosen with probability 0.5 and all other secondary resources with probability $\frac{1}{2(m-1)}$.

For the creation of set B, processing times, i.e. pre-processing p_j^{pre}, processing on the common resource p_j^0 and post-processing p_j^{post}, were sampled in a balanced manner such that p_j^{pre} and p_j^{post} were sampled from $U(0, 8)$ and p_j^0 was sampled from the random variable $p_B^0 \sim U(1, 8)$. On the other hand, for the creation of set S, p_j^{pre} and p_j^{post} were sampled from $U(0, 5)$ and p_j^0 was sampled from the random variable $p_S^0 \sim U(1, 13)$, which makes the usage of the common resource more dominant with respect to the pre- and post-processing times.

For both instance sets B and S, the prize z_j associated to each job $j \in J$ was generated such that it is correlated to the common resource usage p_j^0. z_j was sampled from $U(p_j^0, 2p_j^0)$.

Finally, the time windows associated to each job $j \in J$ have been sampled such that on average 30% of the jobs can be scheduled. Let $T_i = \lfloor 0.3nE(p_i^0) \rfloor$ be the expected maximum resource usage related to instance set $i \in B, S$. The number of windows ω_j were sampled from $U(1, 3)$. The start $W_{j\omega}^{start}$ and end $W_{j\omega}^{end}$ time of each time window $\omega = 1, \cdots, \omega_j$ was sampled from $U(0, T_i - p_j)$ and from $W_{j\omega}^{start} + max\left(p_j, U\left(\left\lfloor 0.1\frac{T_i}{\omega_j} \right\rfloor, \left\lfloor 0.4\frac{T_i}{\omega_j} \right\rfloor\right)\right)$ respectively for job $j \in J$. Time windows were merged whenever an overlap occurred and sorted by increasing start time.