Domain-specific heuristic augmentation of SAT solvers on prize-collecting job scheduling problem

Filip Dashtevski
Supervisor(s): Emir Demirović
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

Abstract

We study the prize-collecting job scheduling problem with one common and multiple secondary resources, where the target is to collect the biggest score possible by constructing a feasible schedule with given constraints of jobs and resources. Many scheduling configurations, where a single asset can be used by all jobs separately and a set of assets can be shared among the same jobs, may be transformed to the format of this problem. Two main tasks of the problem are job selection and job sequencing, which we solve by domain-specific heuristic augmentation of a maximum satisfiability (Max-SAT) solver. The heuristic is based on dual bound values generated by relaxed (lower-bound) and restricted (upper-bound) multi-valued decision diagrams. To convert the data in a format compatible for the solver, we encode it to conjunctive normal form (CNF), and then, we augment the solver by initially modifying the lower bound to be calculated based on a solution obtained by the heuristic approach. For testing we compare the approaches to each other and review different key performance indicators. Based on the experiments results, we observed an improvement in the performance of the solver when it was augmented with the lower bound solution from the heuristic.

Keywords: job-sequencing, job-scheduling, prize-collecting, SAT, domain-specific heuristic.

1 Introduction

Scheduling problems are among the most widely studied problems in operations research and are of big importance in many different industries. A combination of different constraints makes it greatly complex NP-Hard problem, for which many scientists are testing plenty of different approaches.

There are many different sub-problems that branch out of the original scheduling problem. In this paper we will be focused on the prize-collecting job scheduling problem with one common and multiple secondary resources(PC-JSOCMSR). The goal of this problem variation is to collect as big prize as possible, by constructing a feasible schedule where each job has a specific prize value. Additionally, for each job, time interval windows are given which correspond to the time intervals when a job can be processed. The processing of each job demands exactly two resources: one is the common resource, which is required by every job in the set, and the other is one of the secondary resources, which is shared among a subset of the jobs. Each resource can be used by only one job at a time.

The motivation to work on this problem comes from its application in our daily life. For example, two real-life applications in avionics and particle therapy scheduling are described in 2.2. Also, many different scheduling problem configurations can be converted to be PC-JSOCMSR, and then be solved with the same methods that we describe in this article.

PC-JSOCMSR is a recently introduced problem which has not been studied much as of today. A job sequencing variation of the problem was introduced by Horn et al. [9] and a follow up paper from the same authors [8] is used as a main reference paper for the following work in this article. Additionally, a related problem to PC-JSOCMSR scheduling problem with a single machine was introduced by Van der Veen et al. [13] in which specific constraints on a job transition must be satisfied. Different existing approaches to PC-JSOCMSR are presented in the paper from Froger et al. [5]. Since PC-JSOCMSR is closely related to resource constrained project scheduling problem (RCPSP) [1], we often refer to articles solving that problem. For instance, in the second part of our approach we need to encode every instance to a satisfiability (SAT) [6] compatible format, where a boolean configuration has to be satisfied for a schedule to be feasible. For that, articles[14, 7] on how to encode a RCPSP instance may be helpful in understanding the concepts.

The main research question that we are working on in this article is to check whether it is possible to improve the performance of a SAT solver approach to PC-JSOCMSR, by heuristically augmenting it with a previously calculated lower bound solution. The quality of the solution is calculated by testing the performance and precision of the obtained results.

The paper is structured as follows. Firstly, in section 2 the problem will be described formally. Then, section 3 explains the detailed solution to the problem. Section 4 presents the setup and the results of the experiments. Next, in section 5 the ethical aspects of the research are discussed. Finally, in section 6 the conclusion and potential future work are presented.

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

In PC-JSOCMSR, a collection of jobs is provided and each job is associated with a prize. A subset of the maximum total prize is chosen and scheduled among these jobs. Each job has its own set of time windows, and it can only be processed in the range of one of them. Each job requires two resources for processing: a common resource that all jobs require for a portion of their processing, and a secondary resource that is shared by just a subset of the other tasks but is required for the entire processing period. At any time instance, each resource can only process one job.

For instance, consider an application where the common resource is a software which can be used by at most one user at a time, and multiple devices, which can be found in different municipalities, as secondary resources. Every user has been assigned to a municipality and can only use the secondary resource there, while the software can be used on any device, but only one device can use it at a single time instance. Every user needs to first register on the device, then use the software and finally fill in a survey. So, the device(secondary resource) is needed during the whole process and the software(common resource) is needed only during the second phase. After the end of the process, all devices can be used again by other users. This is an informal example of an application of the problem, but in 2.2 more detailed applications are described, to which we refer in the article.

2.1 Formal Problem Description

We are given a set of N non-preemptive jobs $J = [1, N]$, a set of M secondary resources $R = [1, M]$ and a common resource referred to as 0. Hereafter, we denote $R_0 = \{0\} \cup R$ as a set of renewable resources. Each of the jobs is assigned to exactly one secondary resource.

Next, for each job three time periods are given p_j, p_j^{pre}, p_j^0, where $p_j > 0$ is the total time needed for the job to finish, $p_j^{pre} \geq 0$ is the time needed for job pre-processing, and p_j^0 is the time when the common resource is required. Additionally, $p_j^{post} = p_j - p_j^{pre} - p_j^0 \geq 0$, is the time needed for job post-processing. Namely, a previously assigned secondary resource is needed during the whole job process, while the common resource is required only during p_j^0.

The jobs can be processed within one of the time windows of set $W_j = \{T_{jk} | k \in [1, w_j]\}$ with $T_{jk} = min\{t_{jk}^1, t_{jk}^2\}$, without any delay. All of the time windows are indexed in increasing order of the t^1_{jk} values. Denoting $\forall j \in J, t^1 = min\{t^1_{j1}\}$ and $t^2 = max\{t^2_{jw_j}\}$, the planning horizon is defined as the interval $[t^1, t^2]$.

Finally, for every scheduled job $j \in J$, a prize $z_j > 0$ is given. The main problem to be solved is finding a feasible schedule with the biggest prize in return. A feasible solution to the problem is a schedule that satisfies the following constraints: 1) each job is scheduled at most once, 2) if a job is scheduled, it is scheduled uninterruptedly and completely (from its start to its end) within one of its time windows, and 3) at any time, every resource (main or secondary) is used by at most one job. From now on, we refer to constraints 1), 2) and 3) respectively as elementarity, time window and resource usage constraints.

2.2 Applications

Different problem descriptions in which a single action can be applied on every element of a set, along with additional set of actions that can be applied to a single assigned element from the set without any delay, can be transformed to PC-JSOCMSR.

As mentioned in the work of Horn et al.[8], two of the most popular applications for this specific problem are in the field of avionics and particle therapy scheduling.

The first application is about particle therapy scheduling[11] for cancer treatments. In these instances a single machine, known as synchrotron, is the common resource and 2-4 rooms with different equipment are considered as secondary resources. Each patient has to first pass the initial checks and to some of them anesthesia needs to be provided. This processes are considered as pre-processing time. Then, the patient is sent to radiation without any delay. Finally, after radiation is finished, specific medical inspections are done to the patient, to which we refer as post-processing time. The prize assigned to each job is the time needed for the common resource per job.

The second application is pre-runtime planning of avionic systems(electronic frameworks in airplane). The mechanically relevant instances considered in these articles are as well complex and large-scale to be tended to specifically and instead they ought to be solved by some decomposition. Briefly portrayed, the considered system consists of a set of hubs and each of these contains a set of modules (processors) with employments to be planned. In each hub, there's a single module called the communication module, which corresponds to the common resource in PC-JSOCMSR. Each hub also has a set of application modules, which compare to the secondary resources. Here, to every job a prize is assigned based on priority.

3 Heuristic augmentation of SAT solvers

As it can be seen from the research question, we first solve the problem with two different approaches, and then we try to combine them and observe if there is any improvement performance-wise.

We start with implementing a domain-specific heuristic for the problem, introduced by Horn et al.[9]. Then, we encode the problem to CNF and compile the obtained formula with a SAT solver. The final and most important step of the research is to optimize the SAT solver by applying the dual bounds obtained from the heuristic approach, with an objective to decrease the number of variable selections done by the solver.

3.1 Heuristic

Since our research question is to boost a SAT solver by adding some domain-specific information to it, the heuristic step is of big importance. Our primary goal regarding the heuristic is to obtain dual bounds in reasonable computational time, so that the SAT solver can be augmented by integrating the obtained bounds to it.

Due to time limitations that we are given only 9 weeks to finish the research and elaborate everything in this paper, we decided to use the heuristic implemented by Horn et al.[8]. They have done an incredible work in implementing the multi-valued decision diagram(MDD) approach with A* [3] to solve PC-JSOCMSR.

In the following subsections, main steps of the heuristic calculation will be described. We start by explaining what a MDD is and how to represent one with no optimisations. Then, the process of obtaining the lower bound is represented by generating relaxed MDDs. Finally, we discuss how to use the relaxed MDDs to generate a restricted MDD which calculates the upper bound of the instance.

3.1.1 Exact MDD

Decision diagrams are compact graphical representations of boolean functions, originally introduced for applications in circuit design by Lee[10], and widely studied and applied in computer science. MDD is a decision diagram where a different number of actions can be done from any node, representing a state. In this problem, a state of the MDD is represented as a permutation of a subset of jobs $j \in J$.

The simplest form of a MDD is an Exact MDD, which is generated by adding all feasible permutations of jobs. Namely, a new job is added to a permutation of jobs if and only if the new permutation remains feasible. After all the additions to the diagrams, an A* algorithm finds an optimal path.

3.1.2 Relaxed MDD

Since exact MDD is built without any optimisations, the complexity to construct such a diagram quickly increases and therefore, we need to decrease the number of nodes in the MDD. In that regard, we construct a relaxed MDD by merging some of the nodes.

To implement A*, we have a priority queue Q that keeps all the unvisited nodes and based on an evaluation formula, these nodes are ranked in the queue. In contrast, when implementing a relaxed MDD we limit the openlist to a constant number of nodes. That is achieved by merging nodes that are assumed to be similar and nodes that are most probable to be excluded in the final path iteration.

Due to the merging and the optimality condition of A* search, with the relaxed MDDs we have obtained a path whose length is a valid upper bound to the optimal solution value.

3.1.3 Restricted MDD

To generate lower bound of the objective value, we need to construct a restricted MDD. While the relaxed MDD provided an upper bound with constructing a potentially unfeasible schedule, restricted MDD represents only a superset of all feasible schedules. It is primarily used to obtain feasible solutions and corresponding lower bounds. Similar to the limit of the openlist size in relaxed MDD, restricted MDDs are limited by imposing a maximum width β for each of the layers. Relaxed MDDs are used to construct restricted MDD faster.

To read in more detail about the heuristic you can refer to the paper from Horn et al. [8].

3.2 SAT Encoding

Given a boolean formula B composed of a set of propositional variables V and the $\wedge$ (and), $\vee$ (or), and $\neg$ (negation) boolean connectives, the satisfiability (SAT) problem asks â is there an assignment M of true or false values to the variables V such that B evaluates to true under M? ". If such an assignment M exists, then it is said to satisfy B and it is referred to as a model of B. This approach is known as a constraint programming (CP) [2] technique.

For most SAT problems to be solved efficiently, the boolean formula must be of Conjunctive Normal Form (CNF). A formula in CNF only contains $\wedge$ (AND), $\vee$ (OR) and $\neg$ (negation) operators. Additionally, the formula must be a conjuction of disjuntions, or it must be in the form of ANDs of ORs.

This subsection will describe how a PC-JSOCMSR instance is encoded to CNF to then be solved by a Max-SAT [15] solver. First, we will explain how the starting variables are assigned. Then, encoding of both common and secondary resources is defined. Next, At-Most-One cardinality constraints are described. Lastly, we explain how specific prize is assigned to each job from the instance and discuss how to decode the obtained model.

3.2.1 Starting Variables

To keep track of the processing time of every job we assign a starting variable for each job. Since for each job we are given time windows of availability, we need to assign a new variable for every available time instance for every job as follows.

$$\forall j \in J, \forall t \in T_j, s_{jt} = \begin{cases} 1 \text{ If the job } j \text{ starts at } t \\ 0 \text{ If the job } j \text{ does not start at } t \end{cases} \tag{1}$$

By having only this there might be cases when there are multiple starting times per job, but in one of the following sections At-Most-One constraints will restrict these variable selections.

3.2.2 Resource Clauses

After assigning the starting variables, we need to think of a way to keep track of the activity of resources while following specific rules about their activity times.

As previously mentioned, for each job $j \in J$ we are given three activity times p_j, p_j^{pre}, p_j^0, and $p_j^{post} = p_j - p_j^{pre} - p_j^0$. Each secondary resource needs to be active through the whole activity p_j, while the common resource needs to be active only during p_j^0.

Based on the activity times, we can define an implication with the starting variables. Namely, if job j started at time t, then the secondary resource assigned to that job must be active in the whole interval from t to $t + p_j$. The equation is as follows:

$$\forall j \in J, \forall t \in T_j, s_{jt} \implies sr_{jrt'} \tag{2}$$

In the equation above $t' \in [t, t + p_j]$ and r is the assigned secondary resource to job j.

Similarly, an implication can be defined for common resources with the starting variables. In this case, if job j started at time t, then the common resource must be active during all time instances from $t + p_j^{pre}$ to $t + p_j^{pre} + p_j^0$. The equation is as follows:

$$\forall j \in J, \forall t \in T_j, s_{jt} \implies cr_{jt'} \tag{3}$$

In the equation above $t' \in [t + p_j^{pre}$, t $+$ p_j^{pre} $+$ $p_j^0]$

Note that in the implication for secondary resources, variable r seems redundant, but it is of big importance when checking for AMO constraints in the next subsection.

3.2.3 At-Most-One Cardinality Constraints

At-Most-One cardinality constraints are constraints that guarantee that there will be at most one variable selected from a set of variables.

These constraints are of big importance for the encoding, because any resource $r \in R_0$ can be working on at most one job at any time and any job $j \in J$ can have at most one starting time s_{jt}.

$$\forall j \in J, \sum_{t=T_{j0}}^{T_j} s_{jt} <= 1 \tag{4}$$

$$\forall j \in J, \forall r \in R, \sum_{t=T_{j0}}^{T_j} sr_{jrt} <= 1 \tag{5}$$

$$\forall j \in J, \sum_{t=T_{j0}}^{T_j} cr_{jt} <= 1 \tag{6}$$

3.2.4 Job Selection Clauses

The objective of the problem is to select and order jobs, such that the sum of their prizes is the maximum possible and the schedule is feasible. Therefore, we have to include the prizes in the clauses in some way. The most convenient way for that is using Max-SAT solver with hard and soft clauses, described detailedly in the next section.

To indicate if a job was selected or not we have to create a disjunction clause of all available starting time instances per job.

$$\forall j \in J, \forall t \in T_j, \quad S_j = \vee s_{jt} \tag{7}$$

To every disjunction clause per job, its prize z_j is assigned as a weight.

3.3 Max-SAT solver

Since we need to find the most profitable variable assignment, non-binary prizes should be encoded in some way also. The most convenient approach for these kind of variables is to encode to Max-SAT, by using soft and hard clauses. A soft clause is a clause that is not mandatory to be satisfied, but there is a weighted penalty for such cases. On the other hand, hard clauses must be satisfied, so their penalty weight is infinite. Our objective is to find an assignment with the least possible penalty(cost).

As mentioned in the previous subsection, we model job selection clauses to which a job prize is assigned. In combination with all the AMO constraints and the implied CNF encodings, we construct a model that encodes PC-JSOCMSR.

To solve that encoding we use the Pumpkin solver, to which we gained access from our supervisor Emir Demirović .

3.4 Decoding the Model

When the solver compiles the most profitable model we need to decode the obtained variables and check what is the obtained feasible schedule (if any).

To do that, we iterate through all of the available starting time variables per job. If the value of the variable s_{jt} is true, that means that the job j was assigned to start at time instance t.

After iterating through all starting times, the total prize obtained is the sum of jobs with exactly one true starting time variable.

3.5 Heuristic Augmentation of SAT

As a final step of the process, we describe a method how to augment the SAT solver with a previously computed heuristics.

The aforementioned heuristic is a powerful approach to obtain dual bounds of an instance quickly, while aiming for optimality requires more computational time. When solving WCNF instances, the Pumpkin solver is able to find intermediate solutions, but it starts with finding the most obvious models first. Since we already have some final results from heuristic, we skip that part in the solver by passing the obtained solutions to it.

One of the most influential computational time factors in the heuristic method is the size limit ϕ of the openlist. For that reason, we tested the heuristic with ϕ values that do not increase the computational time of the heuristic method substantially, but generate decent bounds that would skip some redundant models in the solver.

We boost the solver by passing a hints file containing internal indices of variables and their assigned values. The hints file is generated based on a lower bound solution found by the heuristic approach. When the hints file is passed to the solver, it generates the model accordingly and sets up a new upper bound of the cost. In this way, all the redundant work is skipped and only better solutions are considered on output.

4 Experimental Setup and Results

Evaluation of the implemented methods is an important step of the research which determines the answer to the main research question.

In the following subsections we first describe the environment in which we generated all the results and what were the instances we used to obtain the results. Then, results from heuristic, encoding, solver and augmented solver are reported, respectfully.

4.1 Experiment Setup

The implementation of the whole heuristic method to first create a relaxed MDD and then apply beam search to generate a restricted MDD is implemented in C++ using GNU g++ 10.2.0 compiler. Similarly, the Pumpkin solver is implemented in C++ using the same

ϕ	1000		5000		10000		30000	
	time	$\Delta bounds$	*time*	$\Delta bounds$	*time*	$\Delta bounds$	*time*	$\Delta bounds$
j10	0	0	0	0	0	0	0	0
j20	0	150	8	86	479	94	1	181
j30	1	194	18	140	32	99	366	70
j40	1	83	10	137	173	161	437	73
j50	2	45	10	27	36	146	845	111
j60	4	99	11	130	29	44	201	143
j70	7	162	10	77	30	85	305	173
j80	4	156	10	129	29	147	142	44
j90	5	177	12	154	25	6	125	95
j100	5	66	14	212	28	62	191	82
j150	16	158	23	121	42	104	127	100
j200	31	221	49	178	45	250	136	120
j250	42	190	56	171	67	248	161	101
j300	65	132	64	104	89	149	221	89

Table 1: Heuristic results in which computational time and difference between upper and lower bound is reported per instances with same number of jobs for every ϕ

compiler. However, processing of the instances and encoding them to WCNF is implemented in Python.

To obtain results, we used existing instances generated by Algorithms and Complexity Group at Vienna University of Technology.[1] They provide both particle therapy and avionics instances, described in 2.2, compatible for the PC-JSOCMSR problem, but due to time limitations, we report results only for the avionic instances. There are 1080 avionic instances in total. They are divided in 18 groups of 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450 and 500 jobs per instance. In each group there are 60 instances. The number of secondary resources is 3 for half of the instances and 4 in the other half. Time windows are distributed between 1 and 1000 time instances. Maximum prize that can be obtained per job is 70 among all instances.

All of the tests were performed on a cluster of compute nodes with 8GB RAM and Intel® Xeon Gold 6248R "Cascade Lake" processors in single-threaded mode with a limit of 60 minutes per job. The cluster we used is DelftBlue, operated by Delft High Performance Computing Centre (DHPC)[12].

4.2 Heuristic Results

Calculation of the dual bound values obtained from the heuristic approach, strongly depends on the size limit of the openlist. In that regards, we test every instance with ϕ values of 1000, 5000, 10000, 30000. Lower values are chosen on purpose, because we prioritize lower precision and lower computational time over high computational time to calculate optimal or near-optimal solutions. We require low processing time because the main research question is to check if an augmented SAT solver with models from the heuristic is performing well. Consequently, we need fast heuristic computations which obtain dual bounds quickly.

Key performance indicators that we consider for the heuristic approach are average CPU

[1]Instances can be found at ac.tuwien.ac.at/research/problem-instances

	time	# literals	# clauses
j10	1	7112	283831
j20	5	14600	634271
j30	13	21997	917351
j40	24	28864	1209163
j50	42	36824	1636602
j60	62	44213	1965434
j70	81	51756	2416501
j80	197	59280	2805131
j90	149	65780	3082635
j100	191	73526	3581345
j150	512	110698	5817519
j200	1292	147511	8297194
j250	2006	184673	11109407
j300	2420	221633	14293330

Table 2: Measurement of the encoding time to WCNF, number of literals and clauses per instance group with same number of jobs

time to obtain dual bounds with specific ϕ value and average absolute difference between lower and upper bounds. To model the results in more convenient way, we group them by number of jobs per instance and report results for every ϕ value per group.

4.3 SAT Results

To obtain results for SAT we first need to encode the given instances to WCNF and then pass the encoding to a solver to obtain objective models. Therefore, in this section we report measurements for the encoding process and for the solver.

For the performance results of the encodings we measure time to encode an instance to WCNF, number of clauses and number of literals per encoding. All results are grouped by number of jobs per instance and an average is reported. Note that in the reported time needed to encode an instance, printing to file is not included.

To evaluate the performance of the solver with previously encoded data we use different important indicators. In instances with low number of jobs, it happens often that the solver finds and confirms an optimal solution, so we report the number of instances where an optimum was found. Since the Pumpkin solver is good in finding intermediate solutions, time to obtain the model with least cost is reported. Finally, an important indicator that shows performance difference between heuristic and CP methods is the average difference between lower bound prize values calculated per instance.

4.4 Augmented Solver Results

Measuring the performance of the augmented solver with heuristics is similar to what we have done in the previous subsections for solver and heuristic measurements separately. The difference is that now we have to pass the obtained results from the heuristic method to the solver in a hints format. Therefore, as important performance indicators we consider the same indicators as in the evaluation process of the solver. To calculate the final time needed to obtain upper bound on the cost, which is the lower bound of the prize, we add the

	SAT			*Aug. SAT*		
	time	*# optimum*	*# ALO model*	*time*	*# optimum*	*# ALO model*
j10	0.01	60	60	0.03	60	60
j20	0.06	60	60	0.06	60	60
j30	6.14	38	60	3.25	42	60
j40	18.52	28	60	51.48	32	60
j50	4.27	3	60	16.20	4	60
j60	6.38	0	59	19.15	0	60
j70	5.59	0	59	198.66	0	60
j80	6.39	0	60	172.30	0	60
j90	136.80	0	60	129.53	0	60
j100	195.91	0	60	296.29	0	60
j150	184.04	0	60	214.95	0	60
j200	190.90	0	60	164.68	0	60
j250	158.10	0	60	341.975	0	60
j300	14.29	0	60	236.73	0	60

Table 3: Measurements of the time needed to solve an encoding per job, the number of optimum models found and models where at least one feasible model was found (upper bound of cost)

time needed to obtain dual bounds in the heuristic method and the time needed to obtain the final value of the solver in 60 minutes time limit. To generate hints, we used a ϕ value of 30000, because it appeared to be the most convenient limit that does not increase the computational heuristic time substantially, but obtains good bounds.

5 Responsible Research

Principles are the basis of integrity in research. According to the " Netherlands Code of Conduct for Research Integrity " [4], there are five main principles to adhere to: honesty, scrupulousness, transparency, independence and responsibility.

Reporting honest results is of big importance. Therefore, when running the tests, we were careful not to influence the results by any means. Also, we explicitly described all important steps taken during the research process to obtain the reported results. Throughout every step, as described in previous sections, we emphasized the importance of using state-of-the-art methods and tended to design up-to-date algorithms.

To make reproducibility of the approach easier for anyone who would want to recreate the implementations, we uploaded the code for encoding to CNF online on a git repository. The encoded data can be then passed to any Max-SAT solver and most of the results can be easily recreated. Additionally, all testing instances used to generate the reported results are available at ac.tuwien.ac.at/research/problem-instances.

All data used during research is publicly available and is not influenced by any user. When recreating results, or when using any part of the code, no personal information is required.

	Heuristic	*SAT*	*Aug. SAT*	*Equal*
Heuristic vs SAT	315	**324**	-	104
Heuristic vs Aug. SAT	221	-	**468**	113
SAT vs Aug. SAT	-	188	**336**	199

Table 4: Pairwise comparison of approaches. A count of prize value closer to optimum is reported per comparison

6 Discussion

In the previous section we reported different results for all the aforementioned approaches, and we discuss all of them in this section.

Firstly, the results in Table 1 show that when we limit the size of the openlist ϕ to be up to 30000, there are not many instances solved to optimality. However, that does not affect our approach, since our primary goal from the heuristics is to obtain the dual bounds and pass them to the solver afterwards. As previously mentioned, as the number of jobs increases, the difference between bounds decreases. For that reason, we decided to do the augmented solver tests with a ϕ value of 30000. We also tested to check if a greater value would be more convenient, but these tests resulted in much higher computational times.

From Table 2 we can see that the average time needed to encode instances to WCNF increases quickly from size to size. That is a result of a big increase in the number of clauses and literals. Big portion of the clauses and literals are due to time instances that can variate from 1 to 1000 and their disjuntion with the common resource and secondary resource literals. A minor relaxation for the solver is that most of the clauses are only consisted of two literals, because of pairwise AMO constraints.

Regarding the SAT solver results in Table 3, not many optimum models were confirmed after the number of jobs per instance increased to 50. Again, that is a result of a rapid increase in the number of clauses and literals in the encoding. But, it is important to note that in almost all instances the solver found an upper bound on the cost in the given time limit of 60 minutes. Similar results are obtained for the augmented solver with slight improvements in the number of optimum solutions found.

The most important results that we obtained can be found in Table 4, where we compare all the approaches to each other, by reporting a count of instances where an according approach obtained a lower bound closer to the optimum value. From the results we can see that when the SAT solver was augmented with the lower bound solution from the heuristic, the number of instances in which the solver obtained a better solution increased from 188 to 336 and a schedule with an equal cost was returned in 199 instances. When compared to the heuristic approach with a ϕ value of 30000, the SAT solver found a feasible schedule with higher prize in 324 instances, which is quite close to the number of instances in which the heuristic found a better model. However, when the heuristic approach is compared to the augmented solver, we see an improvement where the solver found a better solution in more than twice of the instances that the heuristic did better. From these measurements, we can state that there was an obvious advancement when the solver was augmented with the heuristic and it is convenient to apply the heuristic solution with an appropriate ϕ to the CP method using Max-SAT solver.

7 Conclusions and Future Work

The research question that we are answering in this article is to check if any improvement is present in a PC-JSOCMSR solution if a SAT solver is heuristically augmented with data obtaining a lower bound of the prize, compared to a solver where no augmentation is present. In this paper, we first introduced the problem and described all existing approaches. Then, we represented our methods in details and finally, we reported all the obtained results together with a discussion regarding them.
After we reviewed all the results, we observe a significant improvement in calculating the lower bound of the prize. Namely, when a WCNF encoding is passed to the solver together with a list of hints, obtained by the heuristic approach in a low amount of time, the solver manages to find a schedule with a higher prize than the cases in which no hints are provided.

Due to time limitations there are some potential improvements that were not implemented or tested during the research. For instance, a possible improvement could be if instead of doing a single pass of the lower bound model from heuristic to solver, we do that part dynamically for every possible configuration with a small ϕ value. Similarly, both methods may be ran in parallel, with the heuristic periodically increasing the ϕ value and actively passing its best solution to the heuristic. Also, due to huge number of pairwise disjunctions in the encoding, the number of clauses increases substantially. Some improvement could be done regarding this issue, by merging some of the clauses that are implying to same boolean configuration.

Acknowledgments

I am grateful to my professor Emir Demirović for his invaluable help and feedback provided throughout the whole research project.
Further, thanks to Matthias Horn, and the other authors of the main reference paper[8], for providing an advanced heuristic approach.

References

[1] Christian Artigues. *The Resource-Constrained Project Scheduling Problem*, chapter 1, pages 19–35. John Wiley Sons, Ltd, 2008.

[2] Kenneth N. Brown and Ian Miguel. Chapter 21 - uncertainty and change. In Francesca Rossi, Peter van Beek, and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 731–760. Elsevier, 2006.

[3] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. *Procedia Computer Science*, 179:507–514, 2021. 5th International Conference on Computer Science and Computational Intelligence 2020.

[4] Netherlands Organisation for Scientific Research (NWO). Netherlands code of conduct for research integrity. 2021.

[5] Aurelien Froger and Ruslan Sadykov. New exact and heuristic algorithms to solve the prize-collecting job sequencing problem with one common and multiple secondary resources. working paper or preprint, April 2022.

[6] Holger H. Hoos and Thomas Stutzle. 6 - propositional satisfiability and constraint satisfaction. In Holger H. Hoos and Thomas Stutzle, editors, *Stochastic Local Search*, The Morgan Kaufmann Series in Artificial Intelligence, pages 257–312. Morgan Kaufmann, San Francisco, 2005.

[7] Andrei Horbach. A boolean satisfiability approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 181(1):89–107, Dec 2010.

[8] Matthias Horn, Johannes Maschler, Gunther R. Raidl, and Elina Ronnberg. A *-based construction of decision diagrams for a prize-collecting scheduling problem. *Computers Operations Research*, 126:105125, 2021.

[9] Matthias Horn, Gunther Raidl, and Christian Blum. Job sequencing with one common and multiple secondary resources: An a*/beam search based anytime algorithm. *Artificial Intelligence*, 277:103173, 09 2019.

[10] C. Y. Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, 1959.

[11] Johannes Maschler and Gunther R. Raidl. Particle therapy patient scheduling with limited starting time variations of daily treatments. *International Transactions in Operational Research*, 27(1):458–479, 2020.

[12] Delft University of Technology. Delft High Performance Computing Centre. 2021.

[13] Jack A. A. van der Veen, Gerhard J. Woeginger, and Shuzhong Zhang. Sequencing jobs that require common resources on a single machine: A solvable case of the tsp. *Mathematical Programming*, 82(1):235–254, Jun 1998.

[14] Mario Vanhoucke and Jose Coelho. An approach using SAT solvers for the RCPSP with logical constraints. *European Journal of Operational Research*, 249(2):577–591, 2016.

[15] Hantao Zhang, Haiou Shen, and Felip Manya. Exact algorithms for max-sat. *Electronic Notes in Theoretical Computer Science*, 86(1):190–203, 2003. FTP'2003, 4th International Workshop on First-Order Theorem Proving (in connection with RDP'03, Federated Conference on Rewriting, Deduction and Programming).