# An A* Algorithm for Solving a Prize-Collecting Sequencing Problem with One Common and Multiple Secondary Resources and Time Windows

**Matthias Horn · Günther R. Raidl · Elina Rönnberg**

**Abstract** In the considered sequencing problem, a subset of a given set of jobs is to the be scheduled. A scheduled job has to execute without preemption and during this time, the job needs both a common resource for a part of the execution as well as a secondary resource for the whole execution time. The common resource is shared by all jobs while a secondary resource is shared only by a subset of the jobs. Each job has one or more time windows and due to these, it may not be possible to schedule all jobs. Instead, each job is associated with a prize and the task is to select a subset of jobs which yields a feasible schedule with a maximum total sum of prizes. First, we argue that the problem is NP-hard. Then, we present an exact A* algorithm and derive different upper bounds for the total prize; these bounds are based on constraint and Lagrangian relaxations of a linear programming relaxation of a multidimensional knapsack problem. For comparison, a compact mixed integer programming (MIP) model and a constraint programming model are also presented. An extensive experimental evaluation on two types of problem instances shows that the A* algorithm outperforms the other approaches and is able to solve small to medium size instances with up to about 50 jobs to proven optimality. In cases where A* does not prove that an optimal solution is found, the obtained upper bounds are stronger than those of the MIP model.

**Keywords** Sequencing · A* algorithm · particle therapy patient scheduling

This project is partially funded by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Foundation (FWF) Project No. W1260-N35. The work of Elina Rönnberg is supported by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05. We further thank Lukas Felician Krasel for his help in the implementation and testing.

M. Horn and G. R. Raidl
Institute of Logic and Computation, TU Wien, Austria E-mail: {horn|raidl}@ac.tuwien.ac.at

E. Rönnberg
Department of Mathematics, Linköping University, Sweden
E-mail: elina.ronnberg@liu.se

## 1 Introduction

The *Job Sequencing with One Common and Multiple Secondary Resources* (JSOCMSR) problem has been introduced in [6]. It considers the scenario of scheduling a set of jobs where each job, for part of its execution, requires a common resource and, for the whole processing time, requires a secondary resource that is only shared with a subset of the other jobs. The goal of the JSOCMSR is to find a feasible schedule minimizing the makespan.

Applications of this problem can be found, for example, in manufacturing when the common resource corresponds to a machine on which the jobs are to be processed and the processing of each job also requires a certain accompanying resource like a casting mold, template etc. The important aspect is that there is a setup time (e.g. a preparation of the casting mold) at which the secondary resource is needed before the job can be executed at the machine and also a postprocessing time during which the secondary resource is needed afterwards (e.g. because the produced goods must be cooled in the casting mold). With job-dependent setup, processing and postprocessing times and a limited number of shared secondary resources, the JSOCMSR has been shown to be NP-hard [6].

A more specific application of this problem can be found in the daily scheduling of cancer patients that are to receive particle therapy, [12]. There, the common resource corresponds to a sophisticated particle accelerator, which accelerates proton or carbon particles to almost the speed of light. This particle beam is directed into one of a small set of treatment rooms where one patient can be radiated at a time. The treatment rooms are here the secondary resources. During the setup time, a patient is positioned and possibly sedated and after the actual radiation with the particle beam, typically some medical examinations are to be done before the patient can leave the room and it becomes available for a next patient. In such particle therapy treatment centers, there is usually only a single particle accelerator because of its huge cost and about two to three treatment rooms. Since these treatment rooms are typically individually equipped for handling different kinds of treatments, the assignment of patients to rooms is pre-specified. Ideally, patients are scheduled in such a way that the particle beam is directly switched from one room to another so that patients are radiated without any significant breaks in-between.

However, the JSOCMSR is in most cases only a strongly simplified model of real-world scenarios like the above patient scheduling. Most notably, the jobs start times are in practice frequently constrained to certain time windows arising from the availability of the underlying resources. Furthermore, in practice, it happens frequently that not all jobs can be scheduled due to these time windows and instead, a best subset of jobs that can be feasibly scheduled must be selected.

To also include such aspects is the focus of the current work: We extend the JSOCMSR by considering job-specific time windows, and instead of minimizing the makespan we aim at finding a feasible solution that maximizes the total prize of all scheduled jobs. To this end, each job has an assigned prize,

which can simply take the value one if we want to maximize the number of scheduled jobs or it can take a value representing the priority of the job. One possibility is that these prizes are correlated to the processing times of the jobs to avoid scheduling only short jobs.

These new aspects have a substantial impact on the algorithmic side, and existing methods for the JSOCMSR cannot easily be extended in efficient ways. After a more formal problem definition in the next section and a survey of related work in Section 3, the contribution of this paper is to suggest a new solution approach designed for solving small to medium-sized instances of this extended JSOCMSR variant to proven optimality. To this end, we propose an effective A* algorithm in Section 4, which relies on a state strengthening procedure and an upper bound calculation for partial solutions. We suggest and investigate different ways, based on constraint and Lagrangian relaxations of a linear programming relaxation of a multidimensional knapsack problem, to perform these upper bound calculation. For comparison purposes, we further consider, in Section 5, an order-based Mixed Integer Programming (MIP) model solved by Gurobi Optimizer Version 7.5.1 and, in Section 6, a constraint programming model solved by MiniZinc, using three different backend solvers. Experimental results for instances with up to 90 jobs are presented in Section 7. They show that the proposed A* algorithm can solve substantially larger instances to optimality in shorter times than the compared approached. Section 8 concludes this work with an outline of promising future research directions.

## 2 Problem Formulation

The *Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources with Time Windows* (PC-JSOCMSR) considers the sequencing of a set of jobs where each job needs to respect resource constraints and time windows. The resource constraints refer both to a common resource that is used by all jobs and a set of secondary resources of which each job uses exactly one. It is assumed that it is usually not possible to find a feasible schedule that includes all jobs; instead each job is associated with a prize and the objective is to choose a subset of the jobs such that the sum of prizes of the sequenced jobs is maximized.

Let the set of all jobs be denoted by $J$, with $|J| = n$, and let the prize of job $j$ be $z_j > 0$, $j \in J$. The problem is to find a subset of jobs $S \subseteq J$ that can be feasibly scheduled so that the total prize of these jobs is maximized:

$$Z^* = \max_{S \subseteq J} Z(S) = \max_{S \subseteq J} \sum_{j \in S} z_j. \tag{1}$$

The set of (renewable) resources is denoted by $R_0 = \{0\} \cup R$, with $R = \{1, \ldots, m\}$. During its execution, job $j$ uses resource 0, referred to as the *common resource*, and one of the secondary resources $q_j \in R$, $j \in J$. Let $p_j > 0$ be the processing time of job $j$, during which it fully requires the

secondary resource $q_j$, $j \in J$. Further, let $J_r = \{j \in J \mid q_j = r\}$ be the set of jobs that require resource $r$, $r \in R$. For job $j$, $j \in J$, the use of the common resource begins $p_j^{\text{pre}} \geq 0$ time units after the start of the job, has a duration of $p_j^0$, and ends $p_j^{\text{post}} = p_j - p_j^{\text{pre}} - p_j^0 \geq 0$ time units before the end of the job.

If a job $j$ is scheduled, it must be performed without preemption and within one of its $\omega_j$ time windows $W_j = \bigcup_{w=0,\ldots,\omega_j} \left[W_{jw}^{\text{start}}, W_{jw}^{\text{end}}\right]$, where $W_{jw}^{\text{end}} - W_{jw}^{\text{start}} \geq p_j$, $w = 0, \ldots, \omega_j$, $j \in J$. We assume that each job has at least one time window. For job $j$, let the release time be $T_j^{\text{rel}} = \min_{w=0,\ldots,\omega_j} W_{jw}^{\text{start}}$ and the deadline be $T_j^{\text{dead}} = \max_{w=0,\ldots,\omega_j} W_{jw}^{\text{end}}$, $j \in J$. The overall time interval to consider is then $\left[T^{\min}, T^{\max}\right]$ with $T^{\min} = \min_{j \in J} T_j^{\text{rel}}$ and $T^{\max} = \max_{j \in J} T_j^{\text{dead}}$. Note that the existence of unavailability periods of resources is also covered by the above formulation since these can be translated into time windows of the jobs.

To simplify the consideration of the time windows of a job, we define the function *earliest feasible time*

$$\text{eft}(j,t) = \min\{T^{\max},\ t' \geq t \mid [t', t'+p_j] \subseteq W_j\} \tag{2}$$

that yields the earliest time not smaller than the provided time $t \leq T^{\max}$ at which job $j$ can be scheduled according to the time windows $W_j$, $j \in J$. Hereby, $\text{eft}(j,t) = T^{\max}$ indicates that job $j$ cannot be feasibly included in the schedule anymore.

Since each job requires resource 0 and only one job can use this resource at a time, a solution to PC-JSOCMSR implies a total ordering of the scheduled jobs $S$. Vice versa, a permutation $\pi = (\pi_i)_{i=1,\ldots,|S|}$ of a subset of jobs $S \subseteq J$ that can be feasibly scheduled can be decoded into a feasible schedule in a straight-forward greedy way by, in the order given by $\pi$, placing each job from $S$ at its earliest feasible time with respect to when the resources are available after being used by all its preceding jobs. A schedule derived from a job permutation $\pi$ in this way is referred to as a *normalized schedule*. Note that if this greedy approach is applied to a permutation of jobs and some job cannot be feasibly scheduled in this way, this permutation does not correspond to a feasible solution. Also, an optimal solution is either a normalized schedule or the order of the jobs in this optimal solution can be used to derive a normalized schedule with the same objective value. For this reason the notation $Z(\pi)$ is used for the total prize of the normalized solution given by the job permutation $\pi$.

It is not difficult to see that the PC-JSOCMSR is NP-hard: The decision variant of the JSOCMSR, which looks for a feasible schedule with a makespan not exceeding a given $M$, has already been shown to be NP-hard in [6]. One can reduce this decision problem to the PC-JSOCMSR in polynomial time by setting all time windows to $W_j = [0, M]$ and all prizes to $z_j = 1$. A solution to the JSOCMSR decision problem exists if and only if a solution to the PC-JSOCMSR can be found that has all jobs scheduled.

## 3 Related Work

The JSOCMSR was originally proposed by Horn, Raidl, and Blum in [6]. There, a greedy heuristic, an A* algorithm, and a position-based MIP model are described. A particular contribution of that work is a method for calculating a relatively tight lower bound for the makespan, given a partial solution and the still open jobs. Experimental results show that thanks to this strong bound, already the greedy heuristic yields relatively good results by quickly deriving solutions with optimality gaps of only a few percent. Further, the A* search is capable of solving instances with up to 1000 jobs to proven optimality. In comparison, the presented MIP approach was not competitive and can only solve instances with up to 20 jobs reasonably well and it then requires substantially longer computation times. Unfortunately, the lower bound calculation from this previous work is of no use for our PC-JSOCMSR due to the difference in objective function, the fact that usually not all jobs can be scheduled and the time windows. Note further that it is also not efficiently possible or straightforward to adapt the position based MIP model from [6] to the PC-JSOCMSR due to the time windows.

Concerning the PC-JSOCMSR, we point out that in parallel to the work on exact solution approaches presented here, our research group also investigated methods based on multivalued decision diagrams and general variable neighborhood search for addressing larger problem instances that cannot be solved to proven optimality; a corresponding article is concurrently submitted to PATAT 2018 by Maschler and Raidl [10].

To the best of our knowledge, there are only a few further publications dealing with other scenarios similar to the (PC-)JSOCMSR. Veen et al. [17] considers a similar setup as JSOCMSR with jobs requiring one common resource and individual secondary resources. However, in their case, the postprocessing times are negligible compared to the total processing times of the jobs. This implies that the start time of each job only depends on its immediate predecessor. This simplifies the situation substantially, since a job $j$ requiring a different resource than its predecessor $j'$ can always be started after a setup time only depending on job $j$, while a job requiring the same resource can always be started after a postprocessing time only depending on job $j'$. Due to these properties, this problem can be interpreted as a *Traveling Salesman Problem* (TSP) with a special cost structure. Veen et al. even show that their problem can be solved efficiently in time $O(n \log n)$.

Somehow related to the JSOCMSR are no-wait flowshop problem variants, see [1] for a survey. Each job needs to be processed on each of $m$ machines in the same order and the processing of the job on a successive machine always has to take place immediately after its processing has finished on the preceding machine. This problem can be solved in time $O(n \log n)$ for two machines via a transformation to a specially structured TSP [2]. In contrast, for three and more machines the problem is NP-hard, although it can still be transformed into a specially structured TSP. Röck [15] proved that the problem is strongly NP-hard for three machines by a reduction from the 3D-matching problem.

Furthermore, the JSOCMSR problem can be modeled as a more general Resource-Constrained Project Scheduling Problem with maximal time lags by splitting each job according to the resource usage into three sub-jobs that must be executed sequentially without any time lags; see [5] for a survey. Such an indirect solution approach, however, is unlikely to yield promising results in practice since problem-specific aspects are not exploited.

Concerning particle therapy patient scheduling, our PC-JSOCMSR is a practically relevant improvement over the simpler JSOCMSR model, but it still is a strongly simplified formulation addressing only certain properties of the real-life problem. In the full practical scenario, many more aspects must be considered such as large time horizons of several weeks, sequences of therapies for patients to be treated, additionally needed resources including medical staff and their availability time windows, and a combination of more advanced objectives and diverse soft constraints. Maschler et al. [12] proposed a greedy construction heuristic which is extended towards an *Iterated Greedy* metaheuristic and a *Greedy Randomized Adaptive Search Procedure* (GRASP), which consider more of these advanced aspects. These approaches treat the whole problem as a bi-level optimization problem in which the upper level is concerned with the assignment of treatments to days and the lower level corresponds to the detailed scheduling of the treatments assigned at each day. In [11,9], the Iterated Greedy metaheuristic was further refined by including an improved makespan estimation for the daily scheduling problems and by considering additional soft constraints for unwanted deviations of start times of the individual treatments for each therapy.

Concerning A*, we point out that it is a well-known and prominent method for finding shortest paths in possibly huge graphs and more generally an informed search method for problem-solving, see [4,14].

## 4 An A* Algorithm for the PC-JSOCMSR

The A* algorithm is a classic search algorithm from the field of path planning on possibly huge graphs [4,14]. In this section, we describe our A* search approach to solve the PC-JSOCMSR problem. The method either yields a proven optimal solution or, in case of an early termination, a heuristic solution together with an upper bound to the optimal solution value. We start by describing the state graph on which the search is performed, continue with the framework of our A* algorithm, and focus in Sections 4.3 and 4.4 on the strengthening of obtained states and propose different possibilities to determine upper bounds for the achievable total prizes of partial solutions, respectively.

### 4.1 State Graph

We consider a weighted directed acyclic state graph $G = (V, A)$ where each node in $V$ represents a unique state $(P, t)$ consisting of

- the set $P \subseteq J$ of jobs that are still available to be scheduled in further steps, and
- the vector $t = (t_r)_{r \in R_0}$ of the earliest times from which each of the resources are available for performing a next job.

The initial (root) state is $\mathbf{r} = (J, (T^{\min}, \ldots, T^{\min}))$ and represents the original PC-JSOCMSR problem instance with no jobs scheduled or excluded yet.

An arc $(u, v) \in A$ represents a transition from a state $u = (P, t)$ to a state $v = (P', t')$ that is achieved by scheduling a job $j$, $j \in P$, at its earliest possible time w.r.t. vector $t$. More precisely, the *start time* of job $j$, $j \in P$, w.r.t. state $(P, t)$ is

$$\mathrm{s}((P,t), j) = \mathrm{eft}(j, \max(t_0 - p_j^{\mathrm{pre}}, t_{q_j})). \tag{3}$$

The transition function to obtain the successor state $(P', t')$ of state $(P, t)$ when scheduling job $j$, $j \in P$, next is

$$\tau((P,t), j) = \begin{cases} (P \setminus \{j\}, t'), & \text{if } \mathrm{s}((P,t), j) < T^{\max}, \\ \hat{0}, & \text{else}, \end{cases} \tag{4}$$

with

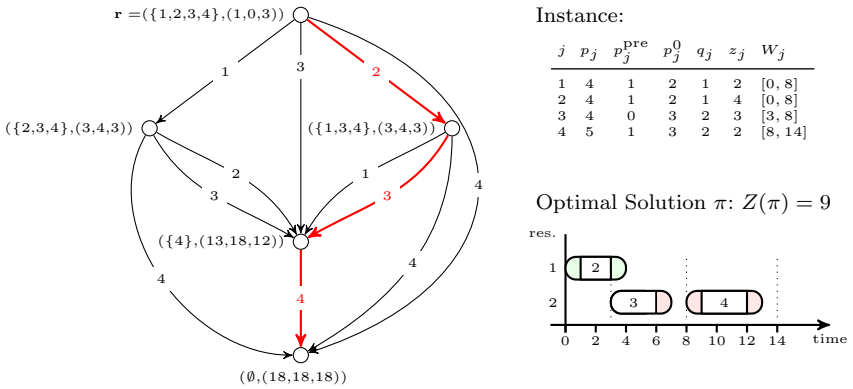
$$t'_0 = \mathrm{s}((P,t), j) + p_j^{\mathrm{pre}} + p_j^0, \tag{5}$$

$$t'_r = \mathrm{s}((P,t), j) + p_j, \qquad \text{for } r = q_j, \tag{6}$$

$$t'_r = t_r, \qquad \text{for } r \in R \setminus \{q_j\}, \tag{7}$$

where $\hat{0}$ represents the infeasible state. The price associated with a state transition is the prize $z_j$ of the scheduled job $j$. Thus, each path in this state graph originating in the initial state $\mathbf{r}$ and ending in some other state than $\hat{0}$ corresponds to a feasible solution in which the jobs associated with the arcs are greedily scheduled in the order in which they appear in the path. Note that a feasible state $(P, t) \in V$ may, in general, be reached via multiple different paths, i.e., by including different sets of jobs $S$ and/or by different orderings of these jobs. Therefore, a feasible state does, in general, not represent a unique solution. As we want to find a solution with maximum total prize, we are primarily interested in a path from $\mathbf{r}$ to $(P, t)$ with maximum total prize. Let $Z^{\mathrm{lp}}(P, t)$ be this maximum total prize to reach a feasible state $(P, t)$. In order to solve the PC-JSOCMSR we are looking for a feasible state with the maximum $Z^{\mathrm{lp}}(P, t)$. To find such a state we perform the A* search detailed in the following subsection.

Figure 1 shows an example of a state graph for an instance with 4 jobs and 2 secondary resources. Each job has exactly one time window. On the right side, the individual characteristics of each job are shown as well as the optimal solution $\pi = (2, 3, 4)$ with a total prize of $Z(\pi) = 9$. On the left side, the corresponding state graph is shown. The path that represents the optimal solution is highlighted. Note that for the shown state graph the strengthening of states was already applied, which will be described in Section 4.3.




| $j$ | $p_j$ | $p_j^{\text{pre}}$ | $p_j^0$ | $q_j$ | $z_j$ | $W_j$ |
|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 2 | 1 | 2 | [0, 8] |
| 2 | 4 | 1 | 2 | 1 | 4 | [0, 8] |
| 3 | 4 | 0 | 3 | 2 | 3 | [3, 8] |
| 4 | 5 | 1 | 3 | 2 | 2 | [8, 14] |

**Fig. 1** Example of a state graph for an instance with $n = 4$ jobs and $m = 2$ secondary resources.

### 4.2 A* Algorithm Framework

In order to solve the PC-JSOCMSR we have to find a feasible state $(P, t)$ with maximum $Z^{\text{lp}}(P, t)$. Such a state cannot have any feasible successor, hence, either $P = \emptyset$ or $\tau((P, t), j) = \hat{0}$, $j \in P$, and otherwise $Z^{\text{lp}}(P, t)$ is not the maximum achievable prize.

A* search belongs to the class of *informed* search strategies that make use of a heuristic estimate for guidance in order to return a proven optimal solution possibly faster than a more naive uninformed search like breadth- or depth-first-search. Within our A*, each encountered feasible state $(P, t)$ of the state graph is evaluated by a priority function $f(P, t) = Z^{\text{lp}}(P, t) + Z^{\text{ub}}(P, t)$ in which $Z^{\text{lp}}(P, t)$ corresponds to the prize of the so far best (i.e., longest) known path from $\mathbf{r}$ to state $(P, t)$ and the value of the function $Z^{\text{ub}}(P, t)$ represents an upper bound on the still achievable prize by extending the path from state $(P, t)$ onward. The value of the priority function $f(P, t)$ is an upper bound on the total prize that a solution may achieve by considering state $(P, t)$. Different options for how to compute this upper bound will be presented in Section 4.4.

Our A* search is shown in pseudo-code in Algorithm 1. It maintains the set $W$ of all so far encountered states, implemented by a hash table; for each contained state $(P, t)$, this data structure also stores the values $Z^{\text{lp}}(P, t)$ and $Z^{\text{ub}}(P, t)$ as well as a reference $\text{pred}(P, t)$ to the predecessor state of a currently longest path from $\mathbf{r}$ to $(P, t)$, and the last scheduled job $\text{j}(P, t)$. Furthermore, the algorithm maintains the *open list* $Q$, which contains all states queued for further expansion. It is realized by a priority queue data structure that considers the states' priority values $f(P, t)$. Last but not least, our A* search maintains a reference $x_{\text{maxlp}}$ to the encountered state $(P, t)$ with the so far largest $Z^{\text{lp}}(P, t)$ value. Both $W$ and $Q$, as well as $x_{\text{maxlp}}$, are initialized with the initial state $\mathbf{r}$.

At each major iteration, a state $(P, t)$ with maximum priority value $f(P, t)$ is taken from $Q$. This state $(P, t)$ is then *expanded*, which means that each job

**Algorithm 1:** A* Algorithm for PC-JSOCMSR

```
1  Input: Initial state r;
2  set of encountered states W ← {r},  Z^lp(r) ← 0;
3  open list Q ← {(r, f(r) = Z^ub(r))};
4  state with maximum Z^lp so far x_maxlp ← r;
5  do
6      if Q = ∅ then
7          return opt. solution given by x_maxlp and its predecessor chain
8      (P, t) ← pop state with maximum f(P, t) from Q;
9      if f(P, t) ≤ Z^lp(x_maxlp) then
10         return opt. solution given by x_maxlp and its predecessor chain
11     // expand state (P, t):
12     foreach j ∈ P do
13         (P', t') ← τ((P, t), j); strengthen state (P', t');
14         if (P', t') = 0̂ ∨ (P', t') ∈ W ∧ Z^lp(P, t) + z_j ≤ Z^lp(P', t') then
15             // infeasible or existing state reached in no better way, skip
16             continue
17         if (P', t') ∉ W then
18             // new state reached
19             W ← W ∪ {(P', t')};
20         Z^lp(P', t') ← Z^lp(P, t) + z_j,  pred(P', t') ← (P, t),  j(P', t') = j ;
21         if Z^ub(P', t') ≠ 0 then
22             Q ← Q ∪ ((P', t'), f(P', t') = Z' + Z^ub(P', t'))
23         if Z^lp(x_maxlp) < Z^lp(P', t') then
24             x_maxlp ← (P', t');
25 while time or memory limit not reached;
26 // terminate early:
27 (P, t) ← state with maximum f(P, t) from Q;
28 derive solution π from x_maxlp following predecessors;
29 π ← greedily augment π with jobs from P;
30 return heuristic solution π and upper bound f(P, t);
```

in $P$ is considered as next job to be scheduled by calculating the respective successor state obtained by the transition $(P', t') = \tau((P, t), j)$. If a job yields the infeasible state $\hat{0}$, it is skipped. Similarly, if the obtained state has been encountered before and the new path via state $(P, t)$ is not longer than the previously identified path to $(P', t')$, we skip job $j$. Otherwise, if a new feasible state is reached, it is added to set $W$. Since a new longest path to state $(P', t')$ via $(P, t)$ has been found, line 20 sets $Z^{\text{lp}}(P', t') = Z^{\text{lp}}(P, t) + z_j$ and stores $(P, t)$ as predecessor of $(P', t')$ and job $j$ as the last scheduled job. The upper bound $Z^{\text{ub}}(P', t')$ is then also calculated, and if there is potential for further improvement, i.e., $Z^{\text{ub}}(P', t') > 0$, state $(P', t')$ is added to the open list $Q$ for a possible future expansion. Finally, reference $x_{\text{maxlp}}$ is updated if a new overall longest path is obtained.

A special aspect of our A* search therefore is that we do not have a specific target state that is known in advance, and we only add states that may yield further successor states to the open list. Lines 6 to 10 makes sure that we

nevertheless recognize when a proven optimal solution has been reached: This is the case when either the open list $Q$ becomes empty or the priority value $f(P,t)$ of $Q$'s top element $(P,t)$ (i.e., the maximum priority value) is not larger than the length $Z^{\mathrm{lp}}(x_{\mathrm{maxlp}})$ of the so far longest identified path. Note that the priority value of $Q$'s top element always is a valid overall upper bound for the total achievable prize. This follows from the fact that $Z^{\mathrm{ub}}(P,t)$ is an *admissible heuristic* according to [4], i.e., it never underestimates the real prize that can still be achieved from $(P,t)$ onward. Further, an optimal solution is derived from state $x_{\mathrm{maxlp}}$ by following its chain of predecessor states and respectively scheduled jobs, and the corresponding solution is returned.

A particular feature of our A* search is that it can also be terminated early by providing a time or memory limit for the execution and it still yields a heuristic solution together with an upper bound on the optimal solution value in this case. This heuristic solution is derived from the so far best state $x_{\mathrm{maxlp}}$ by following its chain of predecessors and additionally considering all remaining jobs in $P$ in their natural order (i.e., as given in the instance specification) for further addition in a greedy way. The returned upper bound is the priority value of $Q$'s top element.

### 4.3 Strengthening of States

Frequently, we can safely replace a state $(P,t)$ by a *strengthened* state $(P',t')$, with $P' \subseteq P$ and $t'_r \geq t_r$, $r \in R_0$, where either $P' \subset P$ or $t'_r > t_r$ for one or more $r$, $r \in R_0$, without losing possible solutions. This state strengthening is applied in Algorithm 1 at line 13 to any state that is obtained from the transition function $\tau$.

By considering the earliest start time $\mathrm{s}((P,t),j)$ for job $j$, $j \in P$, we can first remove all jobs from $P$ that actually cannot be scheduled anymore, i.e., $P' = \{j \in P \mid \mathrm{s}((P,t),j) \neq T^{\max}\}$. Then, time $t'_r$, $r \in R_0$, is set to the earliest possible time when resource $r$ can be used considering all remaining jobs $P'$, i.e.,

$$t'_0 = \min_{j \in P'} \left(\mathrm{s}((P,t),j) + p_j^{\mathrm{pre}}\right), \tag{8}$$

$$t'_r = \begin{cases} \min_{j \in J_r \cap P'} \mathrm{s}((P,t),j), & \text{if } J_r \cap P' \neq \emptyset, \\ T^{\max}, & \text{else,} \end{cases} \qquad r \in R. \tag{9}$$

Here, $t'_r$ is set to $T^{\max}$ if no job that requires resource $r$ remains in $P'$.

### 4.4 Upper Bounds for the Total Prize of Remaining Jobs

For a given state $(P,t)$, an upper bound for the still achievable total prize for the remaining jobs in $P$ can be calculated by solving a linear programming

(LP) relaxation of a multi-constrained 0–1 knapsack problem

$$Z_{\mathrm{KP}}^{\mathrm{ub}}(P,t) = \max \quad \sum_{j\in P} z_j x_j \tag{10}$$

$$\text{s.t} \quad \sum_{j\in P} p_j^0 x_j \le W_0(P,t), \tag{11}$$

$$\sum_{j\in P\cap J_r} p_j x_j \le W_r(P,t), \qquad r\in R, \tag{12}$$

$$x_j \in [0,1], \qquad j\in P, \tag{13}$$

where $x_j$ is a continuous relaxation of a binary variable that indicates if job $j$ is included (=1) or not (=0), $j \in P$. The right-hand-sides of the knapsack constraints

$$W_0(P,t) = \left| \bigcup_{j\in P} \bigcup_{\substack{w=1,\ldots,\omega_j| \\ W_{jw}^{\mathrm{end}} - p_j^{\mathrm{post}} \ge t_0 + p_j^0}} \left[ \max\left(t_0, W_{jw}^{\mathrm{start}} + p_j^{\mathrm{pre}}\right), W_{jw}^{\mathrm{end}} - p_j^{\mathrm{post}} \right] \right| \tag{14}$$

and

$$W_r(P,t) = \left| \bigcup_{j\in P\cap J_r} \bigcup_{w=1,\ldots,\omega_j | W_{jw}^{\mathrm{end}} \ge t_r + p_j} \left[ \max\left(t_r, W_{jw}^{\mathrm{start}}\right), W_{jw}^{\mathrm{end}} \right] \right| \tag{15}$$

represent the total amount of still available time for resource 0 and resource $r$, $r \in R$, respectively, considering the current state and the time windows.

To solve this upper bound calculation problem for each state with an LP solver is computationally too expensive. Instead we compute upper bounds by solving two types of relaxations. The first one

$$Z_0^{\mathrm{ub}}(P,t) = \max \quad \sum_{j\in P} z_j x_j \tag{16}$$

$$\text{s.t} \quad \sum_{j\in P} p_j^0 x_j \le W_0(P,t), \tag{17}$$

$$x_j \in [0,1], \qquad j\in P, \tag{18}$$

is obtained by relaxing inequalities (12); the second one

$$h^{\mathrm{ub}}(P,t,u) = \max \quad \sum_{j\in P} z_j x_j + u\left(W_0(P,t) - \sum_{j\in P} p_j^0 x_j\right) \tag{19}$$

$$\text{s.t} \quad \sum_{j\in P\cap J_r} p_j x_j \le W_r(P,t), \qquad r\in R, \tag{20}$$

$$x_j \in [0,1], \qquad j\in P, \tag{21}$$

is obtained by performing a Lagrangian relaxation of inequality (11), where $u \geq 0$ is the Lagrangian dual multiplier associated with inequality (11).

Both $Z_0^{\text{ub}}(P,t)$ and $h^{\text{ub}}(P,t,u)$ are computed by solving LP-relaxations of knapsack problems. In the latter case, this is possible since the problem separates over the resources and for each resource, the resulting problem is an LP-relaxation of a knapsack problem. An LP-relaxation of a knapsack problem can be efficiently solved by a greedy algorithm that packs items in decreasing prize/time-ratio order; the first item that does not completely fit is packed partially so that the capacity is exploited as far as possible, see [7].

It follows from weak duality (see e.g. [13], Prop. 6.1) that $h^{\text{ub}}(P,t,u)$ yields an upper bound on $Z_{\text{KP}}^{\text{ub}}(P,t)$ for all $u \geq 0$, but the quality of this upper bound depends on the choice of $u$. We have chosen to consider $h^{\text{ub}}(P,t,u)$ for the values $u = 0$ and $u = z_{\bar{j}}/p_{\bar{j}}^0$, where $\bar{j}$ is the last, and typically partially, packed item in an optimal solution to the problem solved to obtain $Z_0^{\text{ub}}(P,t)$. The value $u = z_{\bar{j}}/p_{\bar{j}}^0$ is chosen since it is an optimal LP dual solution associated with inequality (17) and therefore has a chance to be a good estimate of a value for $u$ that gives a strong upper bound.

By solving the relaxations introduced above, the strongest bound on $Z_{\text{KP}}^{\text{ub}}(P,t)$ we can obtain is

$$Z_*^{\text{ub}}(P,t) = \min\left(Z_0^{\text{ub}}(P,t), h^{\text{ub}}(P,t,0), h^{\text{ub}}(P,t,z_{\bar{j}}/p_{\bar{j}}^0)\right). \tag{22}$$

In our experimental comparisons in Section 7, we will illustrate the practical strengths of the bounds

$$Z_0^{\text{ub}}(P,t), \tag{23}$$

$$Z_{00}^{\text{ub}}(P,t) = \min\left(Z_0^{\text{ub}}(P,t), h^{\text{ub}}(P,t,0)\right), \text{ and} \tag{24}$$

$$Z_{0\bar{j}}^{\text{ub}}(P,t) = \min\left(Z_0^{\text{ub}}(P,t), h^{\text{ub}}(P,t,z_{\bar{j}}/p_{\bar{j}}^0)\right), \tag{25}$$

and study if the additional computational effort required to consider also $h^{\text{ub}}(P,t,0)$ and/or $h^{\text{ub}}(P,t,z_{\bar{j}}/p_{\bar{j}}^0)$ pays off in the context of our A* search.

## 5 A Mixed Integer Programming Model

We use the binary variable $t_j$ to indicate if job $j$, $j \in J$, is included in the schedule (=1) or not (=0) and the binary variable $t_{jw}$ to indicate if job $j$ is assigned to time window $w$ (=1) or not (=0), $w = 1, \ldots, \omega_j$, $j \in J$. Let the continuous variable $s_j$ be the start time of job $j$. Binary variable $y_{jj'}$ is further used to indicate if job $j$ is scheduled before $j'$ w.r.t. the common resource (=1) or not (=0), if both jobs are scheduled, $j, j' \in J,\ j \neq j'$.

Let

$$\delta_{jj'} = \begin{cases} p_j, & \text{if } q_j = q_{j'}, \\ p_j^{\text{pre}} + p_j^0 - p_{j'}^{\text{pre}}, & \text{if } q_j \neq q_{j'}, \end{cases} \tag{26}$$

be the minimum time between the start of job $j$ and the start of job $j'$ if job $j$ is scheduled before job $j'$, which depends on whether both jobs use the same resource or not.

A solution to PC-JSOCMSR can be obtained by solving the MIP model

$$\max \quad \sum_{j \in J} z_j t_j \tag{27}$$

$$\text{s.t} \quad t_j = \sum_{w=1,\ldots,\omega_j} t_{jw}, \qquad j \in J, \tag{28}$$

$$y_{jj'} + y_{j'j} \geq t_j + t_{j'} - 1, \qquad j, j' \in J,\ j \neq j', \tag{29}$$

$$s_{j'} \geq s_j + \delta_{jj'} - (T_j^{\text{dead}} - p_j - T_{j'}^{\text{rel}} + \delta_{jj'})(1 - y_{jj'}), \qquad j, j' \in J,\ j \neq j' \tag{30}$$

$$s_j \geq T_j^{\text{rel}} + \sum_{w=1,\ldots,\omega_j} \left(W_{jw}^{\text{start}} - T_j^{\text{rel}}\right) t_{jw}, \qquad j \in J, \tag{31}$$

$$s_j \leq T_j^{\text{dead}} - p_j + \sum_{w=1,\ldots,\omega_j} (W_{jw}^{\text{end}} - T_j^{\text{dead}}) t_{jw}, \qquad j \in J, \tag{32}$$

$$t_j \in \{0,1\}, \qquad j \in J, \tag{33}$$

$$t_{jw} \in \{0,1\}, \qquad w = 1,\ldots,\omega_j,\ j \in J, \tag{34}$$

$$s_j \in [T_j^{\text{rel}}, T_j^{\text{dead}} - p_j], \qquad j \in J, \tag{35}$$

$$y_{jj'} \in \{0,1\}, \qquad j, j' \in J,\ j \neq j'. \tag{36}$$

Equations (28) state that each scheduled job must be assigned to a time window and inequalities (29) ensure that if two jobs $j$ and $j'$ are scheduled, either $y_{jj'}$ or $y_{j'j}$ must be set to one, i.e., one of them needs to precede the other. If a job is to precede another one, inequalities (30) ensure this w.r.t. the jobs' start times. If a job is assigned to a time window, inequalities (31) and (32) make its start time comply with this time window, and otherwise the job only complies with its release time and deadline.

In the previous work [6], a MIP model with position based variables was introduced for JSOCMSR since this model showed better computational performance than a MIP model with order based variables. Such position based model does, however, not extend well to the current setting with multiple time windows since the time windows require explicit knowledge of the start time of each job, and the position based model only has explicit times for the start time of a certain position.

## 6 A Constraint Programming Model

We further propose the following Constraint Programming (CP) model for the PC-JSOCMSR, which we implemented in the constraint modeling language MiniZinc[1]. The model makes use of so-called *option type* variables. Such a

[1] https://www.minizinc.org

variable may either have a value of a certain domain assigned or set to the special value $\top$ that indicates the absence of a value. For job $j \in J$ we use the option type variable $s_j$ for the job's start time. An absent start time, i.e., $s_j = \top$, indicates that the job is not scheduled. The CP model is given by

$$\max \sum_{j \in J \mid \text{occurs}(s_j)} z_j \tag{37}$$

$$\text{disjunctive\_strict}(\{(s_j + p_j^{\text{pre}}, p_j^0) \mid j \in J\}), \tag{38}$$

$$\text{disjunctive\_strict}(\{(s_j, p_j) \mid j \in J \wedge q_j = r\}), \qquad r \in R, \tag{39}$$

$$\text{occurs}(s_j) \implies \left( s_j \in \bigcup_{\omega=1,\ldots,\omega_j} \left[W_{j\omega}^{\text{start}}, W_{j\omega}^{\text{end}} - p_j\right] \right), \qquad j \in J, \tag{40}$$

$$s_j \in [T_j^{\text{rel}}, \ldots, T_j^{\text{dead}} - p_j] \cup \{\top\}, \qquad j \in J, \tag{41}$$

where for job $j \in J$ the predicate $\text{occurs}(s_j)$ yields true if the option type variable $s_j$ is not absent, i.e., job $j$ is scheduled. The strict disjunctive constraints (38) and (39) ensure that all scheduled jobs do not overlap w.r.t. their usage of the common resource and the secondary resource $r \in R$, respectively. Absent jobs are hereby ignored. Constraints (40) state that if job $j \in J$ is scheduled, it must be performed within one of the job's time windows.

## 7 Experimental Results

The proposed A* algorithm from Section 4 was implemented in C++ using GNU G++ 5.4.1 for compilation. The MIP model from Section 5 was solved with Gurobi[2] Optimizer Version 7.5.1. The CP model from Section 6 was implemented with MiniZinc 2.1.7 and solved with three different backbone solvers: (1) Gecode, (2) G12 LazyFD and (3) Chuffed. All tests were performed on a cluster of machines with Intel Xeon E5-2640 v4 processors with 2.40 GHz in single-threaded mode with a CPU time limit of 900 seconds and a memory limit of 15GB per run.

We created two non-trivial benchmark instance sets in order to test our solution approaches. These instances are, with respect to their basic characteristics, inspired from the particle therapy patient scheduling application, c.f. [6], and available online[3]. Each set consists of 30 instances for each combination of $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90\}$ jobs and $m \in \{2, 3\}$ secondary resources. The difference between these two sets lies in the distributions of the secondary resources and each job's pre-processing, post-processing and $p^0$ times. The first instance set B consists of instances in which all secondary resources are used by the jobs equally likely and the distribution of processing times is *balanced*. This was achieved by sampling for job $j$, $j \in J$: (1) the secondary resource $q_j$ from the discrete uniform distribution $\mathcal{U}\{1, m\}$, (2) the pre-processing times

[2] http://www.gurobi.com

[3] https://www.ac.tuwien.ac.at/research/problem-instances

$p_j^{\text{pre}}$ and the post-processing times $p_j^{\text{post}}$ from $\mathcal{U}\{0, 8\}$ and (3) the times $p_j^0$ from the random variable $\mathrm{p}_\mathrm{B}^0 \sim \mathcal{U}\{1, 8\}$. In contrast, the second instance set S exhibits a *skewed* workload such that the secondary resource $m$ is chosen with a probability of 0.5 and all remaining secondary resources with a probability of $1/(2m-2)$. Furthermore the time to claim the common resource 0 is more dominant by sampling $p_j^{\text{pre}}$ and $p_j^{\text{post}}$ from $\mathcal{U}\{0, 5\}$ and $p_j^0$ from the random variable $\mathrm{p}_\mathrm{S}^0 \sim \mathcal{U}\{1, 13\}$. For both instance sets, the prize $z_j$ is determined in a way to correlate to the usage of the common resource of job $j$ by sampling it from $\mathcal{U}\{p_j^0, 2p_j^0\}$, $j \in J$. The time windows are chosen such that, on average about, 30% of the jobs can be scheduled. For this purpose let $T_i = \lfloor 0.3\, n\, \mathrm{E}(\mathrm{p}_i^0) \rfloor$ be the expected maximum resource usage regarding instance type $i$, $i \in \{\mathrm{B}, \mathrm{S}\}$. First, the number of time windows $\omega_j$ of job $j$ is sampled from $\mathcal{U}\{1, 3\}$, i.e., a job can have up to three time windows. Second, for time window $w$, $w = 1, \ldots, \omega_j$, we sample its start time $W_{jw}^{\text{start}}$ from $\mathcal{U}\{0, T_i - p_j\}$ and its end time $W_{jw}^{\text{end}}$ from $W_{jw}^{\text{start}} + \max(p_j, \mathcal{U}\{\lfloor 0.1\, T_i/\omega_j \rfloor, \lfloor 0.4\, T_i/\omega_j \rfloor\})$ for instance type $i$, $i \in \{\mathrm{B}, \mathrm{S}\}$. Overlapping time windows are merged, and all time windows of a job are sorted according to increasing start times.

Note that since we only use integral time windows and processing times, all start times can also safely be assumed to be integral, and we round down any fractional upper bound to the closest integer value in our implementation.

### 7.1 Comparison of Upper Bound Functions

We start by experimentally evaluating the impact of the individual components of our strongest upper bound function proposed in Section 4.1, which is $Z_*^{\text{ub}}(P, t) = \min(Z_0^{\text{ub}}(P, t), h^{\text{ub}}(P, t, 0), h^{\text{ub}}(P, t, z_{\bar{j}}/p_{\bar{j}}^0))$. To this end, we performed the A* search on all benchmark instances using $Z_*^{\text{ub}}(P, t)$ to evaluate all states and count for the sub-functions $Z_0^{\text{ub}}(P, t)$, $h^{\text{ub}}(P, t, 0)$, and $h^{\text{ub}}(P, t, z_{\bar{j}}/p_{\bar{j}}^0)$ how often each one of them yields the minimum, i.e., determines $Z_*^{\text{ub}}(P, t)$. Figure 2 shows the obtained average success rates grouped according to the instance type, the number of jobs $n$, and the number of secondary resources $m$ for all three upper bounds.

Most importantly we can see that not a single sub-function is dominating, i.e., it makes sense to calculate all three functions and to combine their results by taking the minimum in order to get a generally tighter bound. More specifically, the success of each sub-function obviously also depends on the specific characteristics of the problem instances. For instances of type B with two secondary resources, $h^{\text{ub}}(P, t, 0)$ is for each instance class on average more than 50% of the times the strongest upper bound. In all other cases $h^{\text{ub}}(P, t, z_{\bar{j}}/p_{\bar{j}}^0)$ is on average most successful, and for the instances of type S it is clear that both the bounds $h^{\text{ub}}(P, t, z_{\bar{j}}/p_{\bar{j}}^0)$ and $Z_0^{\text{ub}}(P, t)$ are of importance.

The strongest upper bound function, however, does not necessarily yield the best performing A* search, since the time for calculating the bound also plays a major role. As already stated in Section 4.1, we consider the upper

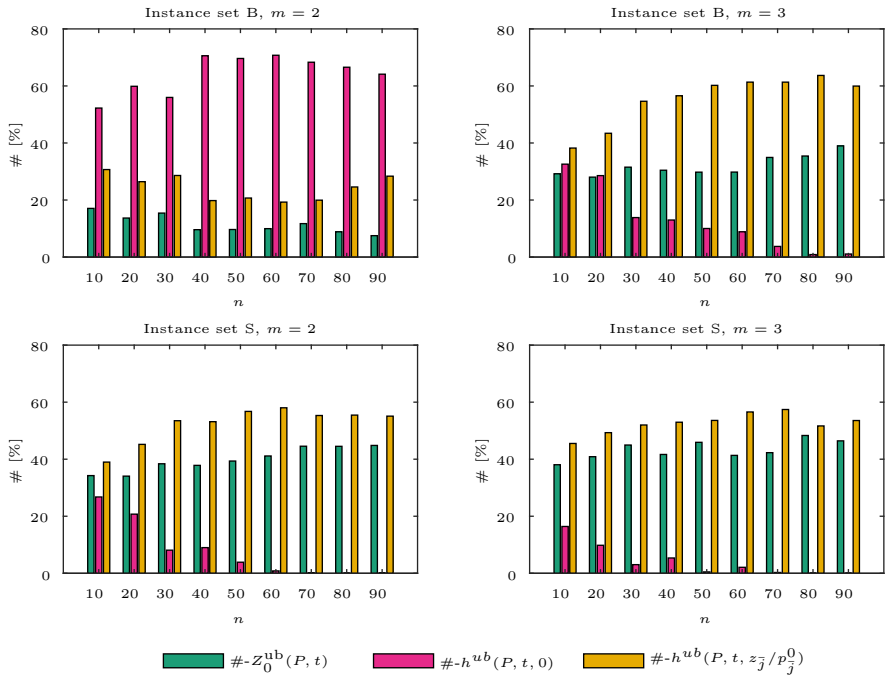



**Fig. 2** Success rates of upper bound subfunctions $Z_0^{\text{ub}}(P,t)$, $h^{\text{ub}}(P,t,0)$ and $h^{\text{ub}}(P,t,z_{\bar{j}}/p_{\bar{j}}^0)$ to yield the smallest value, i.e., to determine $Z_*^{\text{ub}}(P,t)$.

bound functions $Z_0^{\text{ub}}(P,t)$, $Z_{00}^{\text{ub}}(P,t)$, $Z_{0\bar{j}}^{\text{ub}}(P,t)$, and $Z_*^{\text{ub}}(P,t)$ that make use of the above sub-functions in different ways, see Eqs. (22)–(25).

Table 1 presents the aggregated results for each combination of instance type, numbers of jobs, and secondary resources for our A* search using these different upper bound calculations. Columns %-opt show the percentage of instances which could be solved to proven optimality. Columns $\overline{Z^{\text{ub}}}$ state the average final upper bounds and columns $\overline{\%\text{-gap}}$ list the average optimality gaps which are calculated by $100\% \cdot (Z^{\text{ub}} - Z(\pi))/Z^{\text{ub}}$, where $\pi$ is the final solution and $Z^{\text{ub}}$ the final upper bound. Columns t[s] list the median computation times in seconds, whereas columns $\overline{|W|}$ state the average number of encountered states during the A* search.

In almost all cases our strongest bound $Z_*^{\text{ub}}(P,t)$ provides the tightest bounds. There are only four exceptions where $Z_{00}^{\text{ub}}(P,t)$ or $Z_{0\bar{j}}^{\text{ub}}(P,t)$ yield tighter bounds on average, but $Z_*^{\text{ub}}(P,t)$ is not far behind. Using just $Z_0^{\text{ub}}(P,t)$ yields in all cases worse upper bounds than using $Z_{00}^{\text{ub}}(P,t)$, $Z_{0\bar{j}}^{\text{ub}}(P,t)$ or $Z_*^{\text{ub}}(P,t)$. However, in cases where not all instances could be solved to optimality, the A* algorithm with $Z_0^{\text{ub}}(P,t)$ was able to provide the smallest average optimality gaps in almost all cases. We conclude that $Z_0^{\text{ub}}(P,t)$ might be a slightly better guidance for our simple greedy heuristic used to find solutions when terminating early. Only for instances of type B with two secondary

**Table 1** Average results of A* search for different upper bound functions.

| | | | $Z_0^{\mathrm{ub}}(P,t)$ | | | | $Z_{00}^{\mathrm{ub}}(P,t)$ | | | | $Z_{0\dagger}^{\mathrm{ub}}(P,t)$ | | | | $Z_{*}^{\mathrm{ub}}(P,t)$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| type | $n$ | $m$ | %-opt | $Z^{\mathrm{ub}}$ | %-gap | t[s] | $\|W\|$ | %-opt | $Z^{\mathrm{ub}}$ | %-gap | t[s] | $\|W\|$ | %-opt | $Z^{\mathrm{ub}}$ | %-gap | t[s] | $\|W\|$ | %-opt | $Z^{\mathrm{ub}}$ | %-gap | t[s] | $\|W\|$ |
| B | 10 | 2 | **100** | **30.93** | **0.00** | <0.1 | $2.48{\cdot}10^{1}$ | **100** | **30.93** | **0.00** | <0.1 | $2.15{\cdot}10^{1}$ | **100** | **30.93** | **0.00** | <0.1 | $2.32{\cdot}10^{1}$ | **100** | **30.93** | **0.00** | <0.1 | $2.15{\cdot}10^{1}$ |
| B | 20 | 2 | **100** | **50.37** | **0.00** | <0.1 | $2.62{\cdot}10^{2}$ | **100** | **50.37** | **0.00** | <0.1 | $1.96{\cdot}10^{2}$ | **100** | **50.37** | **0.00** | <0.1 | $2.50{\cdot}10^{2}$ | **100** | **50.37** | **0.00** | <0.1 | $1.96{\cdot}10^{2}$ |
| B | 30 | 2 | **100** | **75.33** | **0.00** | <0.1 | $1.64{\cdot}10^{3}$ | **100** | **75.33** | **0.00** | <0.1 | $1.33{\cdot}10^{3}$ | **100** | **75.33** | **0.00** | <0.1 | $1.59{\cdot}10^{3}$ | **100** | **75.33** | **0.00** | <0.1 | $1.32{\cdot}10^{3}$ |
| B | 40 | 2 | **100** | **98.93** | **0.00** | 0.2 | $1.64{\cdot}10^{4}$ | **100** | **98.93** | **0.00** | 0.1 | $1.15{\cdot}10^{4}$ | **100** | **98.93** | **0.00** | 0.1 | $1.57{\cdot}10^{4}$ | **100** | **98.93** | **0.00** | 0.1 | $1.14{\cdot}10^{4}$ |
| B | 50 | 2 | **100** | **123.27** | **0.00** | 2.7 | $2.18{\cdot}10^{5}$ | **100** | **123.27** | **0.00** | 2.1 | $1.62{\cdot}10^{5}$ | **100** | **123.27** | **0.00** | 2.4 | $2.09{\cdot}10^{5}$ | **100** | **123.27** | **0.00** | 1.8 | $1.61{\cdot}10^{5}$ |
| B | 60 | 2 | **100** | **146.80** | **0.00** | 42.0 | $2.79{\cdot}10^{6}$ | **100** | **146.80** | **0.00** | 26.1 | $1.66{\cdot}10^{6}$ | **100** | **146.80** | **0.00** | 46.0 | $2.63{\cdot}10^{6}$ | **100** | **146.80** | **0.00** | 26.8 | $1.65{\cdot}10^{6}$ |
| B | 70 | 2 | 50 | 179.60 | 6.06 | 724.3 | $1.46{\cdot}10^{7}$ | **77** | **174.17** | **2.36** | 379.4 | $1.14{\cdot}10^{7}$ | 47 | 179.37 | 6.20 | 698.1 | $1.41{\cdot}10^{7}$ | 70 | 174.50 | 2.78 | 380.3 | $1.11{\cdot}10^{7}$ |
| B | 80 | 2 | 3 | 217.27 | 14.55 | 683.1 | $2.05{\cdot}10^{7}$ | **7** | 209.57 | **10.23** | 700.0 | $1.98{\cdot}10^{7}$ | 3 | 216.23 | 14.25 | 698.5 | $2.01{\cdot}10^{7}$ | **7** | **209.53** | 10.25 | 762.4 | $1.95{\cdot}10^{7}$ |
| B | 90 | 2 | 0 | 258.37 | 20.55 | 594.8 | $1.96{\cdot}10^{7}$ | 0 | 250.20 | 18.52 | 638.6 | $1.94{\cdot}10^{7}$ | 0 | 256.90 | 20.44 | 653.4 | $1.95{\cdot}10^{7}$ | **3** | **249.83** | **18.19** | 687.2 | $1.94{\cdot}10^{7}$ |
| B | 10 | 3 | **100** | **36.17** | **0.00** | <0.1 | $3.09{\cdot}10^{1}$ | **100** | **36.17** | **0.00** | <0.1 | $2.87{\cdot}10^{1}$ | **100** | **36.17** | **0.00** | <0.1 | $3.06{\cdot}10^{1}$ | **100** | **36.17** | **0.00** | <0.1 | $2.86{\cdot}10^{1}$ |
| B | 20 | 3 | **100** | **59.27** | **0.00** | <0.1 | $2.92{\cdot}10^{2}$ | **100** | **59.27** | **0.00** | <0.1 | $2.74{\cdot}10^{2}$ | **100** | **59.27** | **0.00** | <0.1 | $2.81{\cdot}10^{2}$ | **100** | **59.27** | **0.00** | <0.1 | $2.68{\cdot}10^{2}$ |
| B | 30 | 3 | **100** | **86.30** | **0.00** | <0.1 | $3.50{\cdot}10^{3}$ | **100** | **86.30** | **0.00** | <0.1 | $3.28{\cdot}10^{3}$ | **100** | **86.30** | **0.00** | <0.1 | $3.24{\cdot}10^{3}$ | **100** | **86.30** | **0.00** | <0.1 | $3.09{\cdot}10^{3}$ |
| B | 40 | 3 | **100** | **112.00** | **0.00** | 0.2 | $3.67{\cdot}10^{4}$ | **100** | **112.00** | **0.00** | 0.2 | $3.46{\cdot}10^{4}$ | **100** | **112.00** | **0.00** | 0.2 | $3.52{\cdot}10^{4}$ | **100** | **112.00** | **0.00** | 0.2 | $3.35{\cdot}10^{4}$ |
| B | 50 | 3 | **100** | **140.33** | **0.00** | 5.6 | $4.93{\cdot}10^{5}$ | **100** | **140.33** | **0.00** | 5.9 | $4.83{\cdot}10^{5}$ | **100** | **140.33** | **0.00** | 6.0 | $4.80{\cdot}10^{5}$ | **100** | **140.33** | **0.00** | 5.1 | $4.64{\cdot}10^{5}$ |
| B | 60 | 3 | **97** | 165.60 | **0.28** | 74.5 | $4.54{\cdot}10^{6}$ | **97** | 165.67 | 0.33 | 74.3 | $4.41{\cdot}10^{6}$ | **97** | **165.57** | **0.28** | 68.4 | $4.26{\cdot}10^{6}$ | **97** | 165.63 | 0.32 | 66.1 | $4.13{\cdot}10^{6}$ |
| B | 70 | 3 | 40 | 202.07 | **6.18** | 541.6 | $1.58{\cdot}10^{7}$ | 40 | 201.80 | 7.32 | 548.3 | $1.58{\cdot}10^{7}$ | **43** | 201.77 | 6.51 | 608.4 | $1.55{\cdot}10^{7}$ | 40 | **201.63** | 7.37 | 588.0 | $1.54{\cdot}10^{7}$ |
| B | 80 | 3 | **17** | 247.13 | **13.63** | 548.8 | $1.97{\cdot}10^{7}$ | **17** | 247.17 | 15.18 | 541.4 | $1.95{\cdot}10^{7}$ | **17** | **246.47** | 13.88 | 567.8 | $1.95{\cdot}10^{7}$ | **17** | 246.50 | 14.66 | 558.7 | $1.94{\cdot}10^{7}$ |
| B | 90 | 3 | **0** | 282.53 | **17.35** | 505.1 | $2.07{\cdot}10^{7}$ | **0** | 282.50 | 18.10 | 504.3 | $2.07{\cdot}10^{7}$ | **0** | **282.13** | 17.88 | 508.7 | $2.07{\cdot}10^{7}$ | **0** | **282.13** | 18.35 | 567.2 | $2.07{\cdot}10^{7}$ |
| S | 10 | 2 | **100** | **50.93** | **0.00** | <0.1 | $2.39{\cdot}10^{1}$ | **100** | **50.93** | **0.00** | <0.1 | $2.33{\cdot}10^{1}$ | **100** | **50.93** | **0.00** | <0.1 | $2.38{\cdot}10^{1}$ | **100** | **50.93** | **0.00** | <0.1 | $2.33{\cdot}10^{1}$ |
| S | 20 | 2 | **100** | **89.93** | **0.00** | <0.1 | $3.33{\cdot}10^{2}$ | **100** | **89.93** | **0.00** | <0.1 | $3.17{\cdot}10^{2}$ | **100** | **89.93** | **0.00** | <0.1 | $3.27{\cdot}10^{2}$ | **100** | **89.93** | **0.00** | <0.1 | $3.15{\cdot}10^{2}$ |
| S | 30 | 2 | **100** | **131.37** | **0.00** | <0.1 | $3.87{\cdot}10^{3}$ | **100** | **131.37** | **0.00** | <0.1 | $3.80{\cdot}10^{3}$ | **100** | **131.37** | **0.00** | <0.1 | $3.68{\cdot}10^{3}$ | **100** | **131.37** | **0.00** | <0.1 | $3.63{\cdot}10^{3}$ |
| S | 40 | 2 | **100** | **180.07** | **0.00** | 0.5 | $8.04{\cdot}10^{4}$ | **100** | **180.07** | **0.00** | 0.5 | $7.85{\cdot}10^{4}$ | **100** | **180.07** | **0.00** | 0.6 | $7.92{\cdot}10^{4}$ | **100** | **180.07** | **0.00** | 0.5 | $7.79{\cdot}10^{4}$ |
| S | 50 | 2 | **100** | **225.67** | **0.00** | 8.3 | $9.83{\cdot}10^{5}$ | **100** | **225.67** | **0.00** | 8.2 | $9.62{\cdot}10^{5}$ | **100** | **225.67** | **0.00** | 8.4 | $9.59{\cdot}10^{5}$ | **100** | **225.67** | **0.00** | 8.2 | $9.47{\cdot}10^{5}$ |
| S | 60 | 2 | **53** | 285.90 | **4.47** | 678.1 | $1.38{\cdot}10^{7}$ | **53** | **285.33** | 4.81 | 589.6 | $1.38{\cdot}10^{7}$ | **53** | 285.87 | 4.77 | 606.7 | $1.37{\cdot}10^{7}$ | 50 | 285.87 | 4.95 | 553.8 | $1.36{\cdot}10^{7}$ |
| S | 70 | 2 | **3** | 346.97 | **11.07** | 507.1 | $2.19{\cdot}10^{7}$ | **3** | 347.03 | 11.58 | 539.9 | $2.18{\cdot}10^{7}$ | **3** | **346.80** | 11.40 | 503.1 | $2.19{\cdot}10^{7}$ | **3** | **346.80** | 11.58 | 562.4 | $2.19{\cdot}10^{7}$ |
| S | 80 | 2 | **7** | 394.93 | **11.68** | 454.0 | $2.10{\cdot}10^{7}$ | **7** | 394.93 | 11.80 | 491.5 | $2.10{\cdot}10^{7}$ | **7** | **394.77** | 12.05 | 483.5 | $2.09{\cdot}10^{7}$ | **7** | **394.77** | 12.11 | 525.1 | $2.09{\cdot}10^{7}$ |
| S | 90 | 2 | **0** | 460.37 | **16.33** | 409.8 | $2.13{\cdot}10^{7}$ | **0** | 460.37 | 16.45 | 390.6 | $2.13{\cdot}10^{7}$ | **0** | **460.20** | 17.23 | 433.4 | $2.12{\cdot}10^{7}$ | **0** | **460.20** | 17.30 | 445.9 | $2.12{\cdot}10^{7}$ |
| S | 10 | 3 | **100** | **51.97** | **0.00** | <0.1 | $2.62{\cdot}10^{1}$ | **100** | **51.97** | **0.00** | <0.1 | $2.47{\cdot}10^{1}$ | **100** | **51.97** | **0.00** | <0.1 | $2.47{\cdot}10^{1}$ | **100** | **51.97** | **0.00** | <0.1 | $2.44{\cdot}10^{1}$ |
| S | 20 | 3 | **100** | **96.47** | **0.00** | <0.1 | $2.96{\cdot}10^{2}$ | **100** | **96.47** | **0.00** | <0.1 | $2.92{\cdot}10^{2}$ | **100** | **96.47** | **0.00** | <0.1 | $2.94{\cdot}10^{2}$ | **100** | **96.47** | **0.00** | <0.1 | $2.90{\cdot}10^{2}$ |
| S | 30 | 3 | **100** | **135.90** | **0.00** | <0.1 | $3.54{\cdot}10^{3}$ | **100** | **135.90** | **0.00** | <0.1 | $3.54{\cdot}10^{3}$ | **100** | **135.90** | **0.00** | <0.1 | $3.52{\cdot}10^{3}$ | **100** | **135.90** | **0.00** | <0.1 | $3.52{\cdot}10^{3}$ |
| S | 40 | 3 | **100** | **185.43** | **0.00** | 0.8 | $1.64{\cdot}10^{5}$ | **100** | **185.43** | **0.00** | 0.8 | $1.56{\cdot}10^{5}$ | **100** | **185.43** | **0.00** | 0.9 | $1.55{\cdot}10^{5}$ | **100** | **185.43** | **0.00** | 0.9 | $1.51{\cdot}10^{5}$ |
| S | 50 | 3 | **97** | **234.50** | **0.11** | 18.7 | $2.46{\cdot}10^{6}$ | **97** | **234.50** | **0.11** | 19.3 | $2.44{\cdot}10^{6}$ | **97** | **234.50** | **0.11** | 19.3 | $2.41{\cdot}10^{6}$ | **97** | **234.50** | **0.11** | 19.5 | $2.41{\cdot}10^{6}$ |
| S | 60 | 3 | **47** | 295.60 | **5.13** | 508.0 | $1.58{\cdot}10^{7}$ | **47** | 295.57 | 5.18 | 475.9 | $1.58{\cdot}10^{7}$ | **47** | **295.43** | 5.50 | 467.6 | $1.57{\cdot}10^{7}$ | **47** | **295.43** | 5.41 | 501.2 | $1.57{\cdot}10^{7}$ |
| S | 70 | 3 | **13** | 352.60 | **11.72** | 445.5 | $2.12{\cdot}10^{7}$ | **13** | 352.60 | 12.10 | 487.2 | $2.12{\cdot}10^{7}$ | **13** | **352.23** | 11.73 | 552.6 | $2.10{\cdot}10^{7}$ | **13** | **352.23** | 11.89 | 499.2 | $2.11{\cdot}10^{7}$ |
| S | 80 | 3 | **0** | 413.10 | **14.92** | 466.3 | $2.18{\cdot}10^{7}$ | **0** | 413.10 | 14.94 | 473.6 | $2.18{\cdot}10^{7}$ | **0** | **413.07** | **14.92** | 494.8 | $2.18{\cdot}10^{7}$ | **0** | **413.07** | 14.98 | 535.3 | $2.18{\cdot}10^{7}$ |
| S | 90 | 3 | **0** | 464.33 | **19.71** | 419.7 | $2.12{\cdot}10^{7}$ | **0** | 464.33 | **19.71** | 432.3 | $2.12{\cdot}10^{7}$ | **0** | **464.20** | 20.00 | 443.1 | $2.12{\cdot}10^{7}$ | **0** | **464.20** | 20.00 | 454.3 | $2.12{\cdot}10^{7}$ |

resources, the bound $Z_0^{\text{ub}}(P,t)$ is not able to provide the smallest average optimality gaps. This observation is in accordance with our previous observation concerning Fig. 2, where $h^{\text{ub}}(P,t,0)$ provides more often the strongest upper bound.

Considering only instance classes where all instances could be solved to optimality, the A* algorithm with upper bound function $Z_*^{\text{ub}}(P,t)$ encounters less states than the A* with one of the other functions $Z_{00}^{\text{ub}}(P,t)$, $Z_{0\bar{j}}^{\text{ub}}(P,t)$, or $Z_*^{\text{ub}}(P,t)$, respectively. This is not surprising since $Z_*^{\text{ub}}(P,t)$ dominates the other bounds.

Last but not least, we point out that the memory limit of 16GB was the termination reason in several runs for the largest instances. Thus, memory consumption plays a significant role in our A* algorithm. One way to save memory would be to adopt the technique applied in [6]: States with the same $P$ are stored in an aggregated fashion by storing $P$ only once and the individual vectors $t$ and further information in an attached list of so-called non-dominated time records.

### 7.2 Evaluation of the CP Model with Different Backbone Solvers

The CP model from Section 6 was tested with the MiniZinc 2.1.7 backbone solvers Gecode, G12 LazyFD, and Chuffed. Aggregated results for each combination of instance type, number of jobs, and number of secondary resources are shown in Table 2. Columns %-opt state the percentage of instances that could be solved to proven optimality, columns $\overline{Z^{\text{lp}}}$ the obtained average total prizes, and columns t[s] median computation times.

The results give a rather clear picture: Gecode is clearly outperformed by the two lazy clause based solvers G12 LazyFD and Chuffed. Obviously, the SAT propagation technologies behind the latter two here give a substantial advantage. Among these two solvers, Chuffed performed significantly better and was able to consistently solve all instances with up to 40 jobs and some instances with up to 60 jobs. Furthermore, Chuffed is typically also faster on those instances for which both solvers could prove optimality of their solutions.

### 7.3 Comparison of A*, MIP, and CP

We finally compare our A* search using the dominating upper bound function $Z_*^{\text{ub}}(P,t)$ to solving the MIP model from Section 5 using Gurobi and the CP model from Section 6 using Chuffed. Table 3 shows the aggregated results. Regarding the number of instances that could be solved to proven optimality, the A* consistently outperforms Gurobi and Chuffed. A* could solve all instances with up to 50 jobs to proven optimality, except one skewed instance with three secondary resources. The largest instance which A* could solve to proven optimality consists of 90 jobs, whereas the largest instances that

**Table 2** Average results of MiniZinc with different backbone solvers.

| | | | Gecode | | | G12 LazyFD | | | Chuffed | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| type | $n$ | $m$ | %-opt | $\overline{Z^{\text{lp}}}$ | t[s] | %-opt | $\overline{Z^{\text{lp}}}$ | t[s] | %-opt | $\overline{Z^{\text{lp}}}$ | t[s] |
| B | 10 | 2 | **100** | **30.93** | 0.8 | **100** | **30.93** | 0.8 | **100** | **30.93** | 0.8 |
| B | 20 | 2 | 33 | 48.27 | 900.0 | **100** | **50.37** | 0.6 | **100** | **50.37** | 0.4 |
| B | 30 | 2 | 0 | 64.23 | 900.0 | **100** | **75.33** | 2.5 | **100** | **75.33** | 0.6 |
| B | 40 | 2 | 0 | 74.67 | 900.0 | **100** | **98.93** | 181.5 | **100** | **98.93** | 3.0 |
| B | 50 | 2 | 0 | 83.70 | 900.0 | 0 | 117.07 | 900.0 | **100** | **123.27** | 56.3 |
| B | 60 | 2 | 0 | 92.27 | 900.0 | 0 | 131.40 | 900.0 | **13** | **142.23** | 900.0 |
| B | 70 | 2 | **0** | 108.77 | 900.0 | **0** | 141.50 | 900.0 | **0** | **154.93** | 900.0 |
| B | 80 | 2 | **0** | 120.73 | 900.0 | **0** | 157.10 | 900.0 | **0** | **171.70** | 900.0 |
| B | 90 | 2 | **0** | 142.27 | 900.0 | **0** | 177.13 | 900.0 | **0** | **188.53** | 900.0 |
| B | 10 | 3 | **100** | **36.17** | 0.4 | **100** | **36.17** | 0.4 | **100** | **36.17** | 0.6 |
| B | 20 | 3 | 43 | 55.50 | 900.0 | **100** | **59.27** | 0.7 | **100** | **59.27** | 0.5 |
| B | 30 | 3 | 0 | 71.73 | 900.0 | **100** | **86.30** | 2.5 | **100** | **86.30** | 0.5 |
| B | 40 | 3 | 0 | 84.83 | 900.0 | 80 | 111.87 | 272.7 | **100** | **112.00** | 4.0 |
| B | 50 | 3 | 0 | 90.40 | 900.0 | 0 | 131.17 | 900.0 | **93** | **140.20** | 116.9 |
| B | 60 | 3 | 0 | 103.37 | 900.0 | 0 | 149.00 | 900.0 | **13** | **160.43** | 900.0 |
| B | 70 | 3 | **0** | 120.17 | 900.0 | **0** | 166.57 | 900.0 | **0** | **179.60** | 900.0 |
| B | 80 | 3 | **0** | 138.10 | 900.0 | **0** | 185.10 | 900.0 | **0** | **202.17** | 900.0 |
| B | 90 | 3 | **0** | 154.57 | 900.0 | **0** | 206.17 | 900.0 | **0** | **220.80** | 900.0 |
| S | 10 | 2 | **100** | **50.93** | 0.4 | **100** | **50.93** | 0.5 | **100** | **50.93** | 0.5 |
| S | 20 | 2 | 23 | 85.23 | 900.0 | **100** | **89.93** | 0.5 | **100** | **89.93** | 0.3 |
| S | 30 | 2 | 0 | 108.97 | 900.0 | **100** | **131.37** | 5.7 | **100** | **131.37** | 0.8 |
| S | 40 | 2 | 0 | 129.53 | 900.0 | 43 | 178.70 | 900.0 | **100** | **180.07** | 10.9 |
| S | 50 | 2 | 0 | 156.20 | 900.0 | 0 | 205.87 | 900.0 | **46** | **222.23** | 900.0 |
| S | 60 | 2 | **0** | 179.66 | 900.0 | **0** | 242.87 | 900.0 | **0** | **254.83** | 900.0 |
| S | 70 | 2 | **0** | 210.63 | 900.0 | **0** | 272.07 | 900.0 | **0** | **289.53** | 900.0 |
| S | 80 | 2 | **0** | 258.70 | 900.0 | **0** | 312.87 | 900.0 | **0** | **318.50** | 900.0 |
| S | 90 | 2 | **0** | 288.13 | 900.0 | **0** | 344.60 | 900.0 | **0** | **349.60** | 900.0 |
| S | 10 | 3 | **100** | **51.97** | 0.3 | **100** | **51.97** | 0.4 | **100** | **51.97** | 0.4 |
| S | 20 | 3 | 10 | 87.60 | 900.0 | **100** | **96.47** | 0.5 | **100** | **96.47** | 0.3 |
| S | 30 | 3 | 0 | 109.87 | 900.0 | **100** | **135.90** | 5.1 | **100** | **135.90** | 0.9 |
| S | 40 | 3 | 0 | 136.17 | 900.0 | 30 | 183.20 | 900.0 | **100** | **185.43** | 20.4 |
| S | 50 | 3 | 0 | 160.57 | 900.0 | 0 | 217.63 | 900.0 | **13** | **228.33** | 900.0 |
| S | 60 | 3 | **0** | 191.67 | 900.0 | **0** | 253.97 | 900.0 | **0** | **262.93** | 900.0 |
| S | 70 | 3 | **0** | 224.27 | 900.0 | **0** | 287.07 | 900.0 | **0** | **292.00** | 900.0 |
| S | 80 | 3 | **0** | 276.03 | 900.0 | **0** | **326.63** | 900.0 | **0** | 324.80 | 900.0 |
| S | 90 | 3 | **0** | 308.43 | 900.0 | **0** | **359.73** | 900.0 | **0** | 355.30 | 900.0 |

Gurobi and Chuffed could solve to proven optimality have 50 and 60 jobs, respectively. Gurobi could solve all balanced instances with up to 40 jobs and all skewed instances with up to 30 jobs to proven optimality. Computation times for those are, however, significantly larger than for A*. In particular for small instances with up to $n = 30$ jobs, A* only required median computation times of no more than 0.1 seconds. The CP solver Chuffed could solve all instances up to $n = 40$ jobs to optimality. The A* algorithm was able to provide equally good or better final solutions than Gurobi and Chuffed in almost all cases. Exceptions occurred only for some of the largest instances with 90 jobs, where Gurobi's heuristic performance proved to be superior. Note, however, that the derivation of heuristic solutions for large instances that cannot be solved to optimality is not in the foreground of our research here. Concerning obtained upper bounds, the A* search again clearly outperforms the MIP approach by a

**Table 3** Average results of A*, MIP, and CP.

| | | | A*, $Z_*^{\text{ub}}(P,t)$ | | | | | MIP | | | | | CP, Chuffed | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| type | $n$ | $m$ | %-opt | $\overline{Z^{\text{lp}}}$ | $\overline{Z^{\text{ub}}}$ | $\overline{\%\text{-gap}}$ | t[s] | %-opt | $\overline{Z^{\text{lp}}}$ | $\overline{Z^{\text{ub}}}$ | $\overline{\%\text{-gap}}$ | t[s] | %-opt | $\overline{Z^{\text{lp}}}$ | t[s] |
| B | 10 | 2 | **100** | **30.93** | **30.93** | **0.00** | <0.1 | **100** | **30.93** | **30.93** | **0.00** | <0.1 | **100** | **30.93** | 0.8 |
| B | 20 | 2 | **100** | **50.37** | **50.37** | **0.00** | <0.1 | **100** | **50.37** | **50.37** | **0.00** | 0.1 | **100** | **50.37** | 0.4 |
| B | 30 | 2 | **100** | **75.33** | **75.33** | **0.00** | <0.1 | **100** | **75.33** | **75.33** | **0.00** | 4.4 | **100** | **75.33** | 0.6 |
| B | 40 | 2 | **100** | **98.93** | **98.93** | **0.00** | 0.1 | **100** | **98.93** | **98.93** | **0.00** | 69.0 | **100** | **98.93** | 3.0 |
| B | 50 | 2 | **100** | **123.27** | **123.27** | **0.00** | 1.8 | 30 | 122.73 | 144.60 | 14.13 | 900.3 | **100** | **123.27** | 56.3 |
| B | 60 | 2 | **100** | **146.80** | **146.80** | **0.00** | 26.8 | 0 | 143.73 | 218.37 | 33.91 | 900.2 | 13 | 142.23 | 900.0 |
| B | 70 | 2 | **70** | **169.57** | **174.50** | **2.78** | 380.3 | 0 | 165.33 | 308.00 | 46.12 | 900.1 | 0 | 154.93 | 900.0 |
| B | 80 | 2 | **7** | 187.80 | **209.53** | **10.25** | 762.4 | 0 | **189.27** | 370.97 | 48.95 | 900.4 | 0 | 171.70 | 900.0 |
| B | 90 | 2 | **3** | 203.60 | **249.83** | **18.19** | 687.2 | 0 | **215.27** | 457.10 | 52.80 | 900.2 | 0 | 188.53 | 900.0 |
| B | 10 | 3 | **100** | **36.17** | **36.17** | **0.00** | <0.1 | **100** | **36.17** | **36.17** | **0.00** | <0.1 | **100** | **36.17** | 0.6 |
| B | 20 | 3 | **100** | **59.27** | **59.27** | **0.00** | <0.1 | **100** | **59.27** | **59.27** | **0.00** | 0.1 | **100** | **59.27** | 0.5 |
| B | 30 | 3 | **100** | **86.30** | **86.30** | **0.00** | <0.1 | **100** | **86.30** | **86.30** | **0.00** | 4.6 | **100** | **86.30** | 0.5 |
| B | 40 | 3 | **100** | **112.00** | **112.00** | **0.00** | 0.2 | **100** | **112.00** | **112.00** | **0.00** | 92.2 | **100** | **112.00** | 4.0 |
| B | 50 | 3 | **100** | **140.33** | **140.33** | **0.00** | 5.1 | 10 | 138.70 | 175.73 | 20.02 | 900.5 | 93 | 140.20 | 116.9 |
| B | 60 | 3 | **97** | **165.07** | **165.63** | **0.32** | 66.1 | 0 | 161.97 | 235.63 | 30.87 | 900.6 | 13 | 160.43 | 900.0 |
| B | 70 | 3 | **40** | 185.87 | **201.63** | **7.37** | 588.0 | 0 | **187.20** | 319.23 | 41.22 | 900.1 | 0 | 179.60 | 900.0 |
| B | 80 | 3 | **17** | 209.63 | **246.50** | **14.66** | 558.7 | 0 | **215.97** | 388.50 | 44.21 | 900.0 | 0 | 202.17 | 900.0 |
| B | 90 | 3 | **0** | 229.57 | **282.13** | **18.35** | 567.2 | **0** | **240.67** | 469.97 | 48.70 | 900.0 | **0** | 220.80 | 900.0 |
| S | 10 | 2 | **100** | **50.93** | **50.93** | **0.00** | <0.1 | **100** | **50.93** | **50.93** | **0.00** | <0.1 | **100** | **50.93** | 0.5 |
| S | 20 | 2 | **100** | **89.93** | **89.93** | **0.00** | <0.1 | **100** | **89.93** | **89.93** | **0.00** | 0.3 | **100** | **89.93** | 0.3 |
| S | 30 | 2 | **100** | **131.37** | **131.37** | **0.00** | <0.1 | **100** | **131.37** | **131.37** | **0.00** | 17.8 | **100** | **131.37** | 0.8 |
| S | 40 | 2 | **100** | **180.07** | **180.07** | **0.00** | 0.5 | 60 | 179.87 | 192.80 | 5.92 | 671.4 | **100** | **180.07** | 10.9 |
| S | 50 | 2 | **100** | **225.67** | **225.67** | **0.00** | 8.2 | 0 | 219.97 | 343.97 | 35.61 | 900.7 | 46 | 222.23 | 900.0 |
| S | 60 | 2 | **50** | **271.13** | **285.87** | **4.95** | 553.8 | 0 | 265.73 | 476.83 | 44.06 | 900.4 | 0 | 254.83 | 900.0 |
| S | 70 | 2 | **3** | **305.63** | **346.80** | **11.58** | 562.4 | 0 | 304.40 | 600.70 | 49.19 | 900.3 | 0 | 289.53 | 900.0 |
| S | 80 | 2 | **7** | **346.30** | **394.77** | **12.11** | 525.1 | 0 | 346.23 | 715.73 | 51.51 | 900.3 | 0 | 318.50 | 900.0 |
| S | 90 | 2 | **0** | 379.47 | **460.20** | **17.30** | 445.9 | **0** | **382.87** | 844.50 | 54.55 | 900.1 | **0** | 349.60 | 900.0 |
| S | 10 | 3 | **100** | **51.97** | **51.97** | **0.00** | <0.1 | **100** | **51.97** | **51.97** | **0.00** | <0.1 | **100** | **51.97** | 0.4 |
| S | 20 | 3 | **100** | **96.47** | **96.47** | **0.00** | <0.1 | **100** | **96.47** | **96.47** | **0.00** | 0.4 | **100** | **96.47** | 0.3 |
| S | 30 | 3 | **100** | **135.90** | **135.90** | **0.00** | <0.1 | **100** | **135.90** | **135.90** | **0.00** | 14.7 | **100** | **135.90** | 0.9 |
| S | 40 | 3 | **100** | **185.43** | **185.43** | **0.00** | 0.9 | 33 | 185.03 | 209.70 | 10.85 | 900.3 | **100** | **185.43** | 20.4 |
| S | 50 | 3 | **97** | **234.27** | **234.50** | **0.11** | 19.5 | 0 | 230.07 | 364.60 | 36.47 | 900.1 | 13 | 228.33 | 900.0 |
| S | 60 | 3 | **47** | **278.93** | **295.43** | **5.41** | 501.2 | 0 | 276.50 | 491.83 | 43.60 | 900.1 | 0 | 262.93 | 900.0 |
| S | 70 | 3 | **13** | 308.90 | **352.23** | **11.89** | 499.2 | 0 | **313.13** | 604.00 | 48.04 | 900.1 | 0 | 292.00 | 900.0 |
| S | 80 | 3 | **0** | 350.40 | **413.07** | **14.98** | 535.3 | **0** | **357.93** | 734.10 | 51.14 | 900.3 | **0** | 324.80 | 900.0 |
| S | 90 | 3 | **0** | 370.37 | **464.20** | **20.00** | 454.3 | **0** | **394.27** | 836.17 | 52.66 | 900.0 | **0** | 355.30 | 900.0 |

large margin, especially on the largest instances. Chuffed is not able to return any upper bounds.

## 8 Conclusions and Future Work

We introduced the PC-JSOCMSR as a practically relevant extension of the formerly considered JSOCMSR [6]. The essential differences are the additionally considered time windows, the aspect that usually not all jobs can be feasibly scheduled anymore, and the replacement of the minimization of the makespan by the maximization of the prizes of scheduled jobs. These differences make the problem in practice much more challenging to solve.

Nevertheless, we showed that small to medium sized instances of up to about 50 jobs can relatively consistently be solved to proven optimality by our A* search. For comparison, we considered an order-based MIP model as well as a MiniZinc formulation that was solved with different backend solvers. These approaches, however, are clearly inferior concerning the computation times for proven optimal solutions or obtained upper bounds (actually, the CP approaches are not able to provide any upper bounds in case of early termination).

Within the A* search, the procedure to obtain upper bounds for the still achievable prizes of states plays a crucial role. To this end, we considered a

relaxation of the PC-JSOCMSR that corresponds to an LP relaxation of a multidimensional knapsack problem. Further simplifications based on constraint and Lagrangian relaxation, respectively, yielded the generally fast-to-calculate upper bound functions $Z_0^{\mathrm{ub}}(P,t)$, $Z_{00}^{\mathrm{ub}}(P,t)$, $Z_{0\bar{j}}^{\mathrm{ub}}(P,t)$ and $Z_*^{\mathrm{ub}}(P,t)$. While $Z_*^{\mathrm{ub}}(P,t)$ is strongest, $Z_0^{\mathrm{ub}}(P,t)$ is fastest. In our experiments in the context of the A* search, using the strongest bound $Z_*^{\mathrm{ub}}(P,t)$ turned out to pay off by usually being the best choice. It yields proven optimal solutions a bit more frequently within the allowed time and memory limits and requires less states to be considered. In cases where the limits have been reached, slightly better final upper bounds are frequently achieved.

Our clear focus in this work was on solving the problem to proven optimality. For larger instances, where this goal cannot be achieved within reasonable time, our A* search nevertheless yields a heuristic solution together with the best obtained upper bound. The derivation of this heuristic solution, however, is done in a rather simple way and leaves room for further improvement. For example, local search and other (meta-)heuristic search methods may be easily applied in addition.

Another promising direction for future work would be to turn the A* algorithm into an anytime A* search that deviates from the pure best-first-search strategy by also producing promising complete solutions on a more regular basis from the beginning on; see, for example, the Anytime Weighted A* algorithm [3], the Anytime Repairing A* [8], Anytime Pack Search [16], or the A*/Beam Search hybrid described for the JSOCMSR in [6].

## References

1. Allahverdi, A.: A survey of scheduling problems with no-wait in process. European Journal of Operational Research **255**(3), 665–686 (2016)
2. Gilmore, P.C., Gomory, R.E.: Sequencing a one-state variable machine: A solvable case of the traveling salesman problem. Operations Research **12**(5), 655–679 (1964)
3. Hansen, E.A., Zhou, R.: Anytime heuristic search. Journal of Artificial Intelligence Research **28**, 267–297 (2007)
4. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
5. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research (207), 1–14 (2010)
6. Horn, M., Raidl, G., Blum, C.: Job sequencing with one common and multiple secondary resources: A problem motivated from particle therapy for cancer treatment. In: G. Giuffrida, G. Nicosia, P. Pardalos, R. Umeton (eds.) MOD 2017: Machine Learning, Optimization, and Big Data – Third International Conference, *LNCS*, vol. 10710, pp. 506–518. Springer (2017)
7. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
8. Likhachev, M., Gordon, G.J., Thrun, S.: ARA*: Anytime A* with provable bounds on sub-optimality. In: Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference (NIPS-03), pp. 767–774. MIT Press (2004)
9. Maschler, J., Hackl, T., Riedler, M., Raidl, G.R.: An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In: Proceedings of the 12th Metaheuristics International Conference, pp. 465–474 (2017)

10. Maschler, J., Raidl, G.: Multivalued decision diagrams for a prize-collecting sequencing problem. Tech. rep., TU Wien, Austria (2018). Submitted to PATAT 2018
11. Maschler, J., Riedler, M., Raidl, G.R.: Particle therapy patient scheduling: Time estimation for scheduling sets of treatments. In: R. Moreno-Díaz, F. Pichler, A. Quesada-Arencibia (eds.) Computer Aided Systems Theory – EUROCAST 2017, LNCS. Springer (to appear)
12. Maschler, J., Riedler, M., Stock, M., Raidl, G.R.: Particle therapy patient scheduling: First heuristic approaches. In: PATAT 2016: Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling, pp. 223–244. Udine, Italy (2016)
13. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons (1988)
14. Rios, L.H.O., Chaimowicz, L.: A Survey and Classification of A* Based Best-First Heuristic Search Algorithms, pp. 253–262. Springer (2010). DOI 10.1007/978-3-642-16138-4_26
15. Röck, H.: The three-machine no-wait flow shop is NP-complete. Journal of the ACM **31**(2), 336–345 (1984)
16. Vadlamudi, S.G., Aine, S., Chakrabarti, P.P.: Anytime pack search. Natural Computing **15**(3), 395–414 (2016)
17. Van der Veen, J.A.A., Wöginger, G.J., Zhang, S.: Sequencing jobs that require common resources on a single machine: A solvable case of the TSP. Mathematical Programming **82**(1-2), 235–254 (1998)