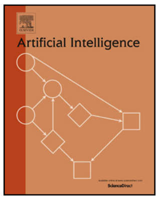
Artificial Intelligence 277 (2019) 103173

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Job sequencing with one common and multiple secondary resources: An A*/Beam Search based anytime algorithm[☆]

Matthias Horn[a,*], Günther Raidl[a], Christian Blum[b]

[a] *Institute of Logic and Computation, TU Wien, Favoritenstraße 9–11/E192-01, 1040 Vienna, Austria*
[b] *Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, Bellaterra, Spain*

A R T I C L E   I N F O

Article history:
Received 23 March 2018
Received in revised form 8 August 2019
Accepted 8 September 2019
Available online 12 September 2019

Keywords:
Sequencing
Scheduling
A* algorithm
Beam Search
Particle therapy patient scheduling

A B S T R A C T

We consider a sequencing problem that arises, for example, in the context of scheduling patients in particle therapy facilities for cancer treatment. A set of non-preemptive jobs needs to be scheduled, where each job requires two resources: (1) a common resource that is shared by all jobs and (2) a secondary resource, which is shared with only a subset of the other jobs. While the common resource is only required for a part of the job's processing time, the secondary resource is required for the whole duration. The objective is to minimize the makespan. First we show that the tackled problem is NP-hard and provide three different lower bounds for the makespan. These lower bounds are then exploited in a greedy construction heuristic and a novel exact anytime A* algorithm, which uses an advanced diving mechanism based on Beam Search and Local Search to find good heuristic solutions early. For comparison we also provide a basic Constraint Programming model solved with the ILOG CP optimizer. An extensive experimental evaluation on two types of problem instances shows that the approach works even for large instances with up to 2000 jobs extremely well. It typically yields either optimal solutions or solutions with an optimality gap of less than 1%.

1. Introduction

This work considers the following combinatorial optimization problem. A finite set of jobs must be executed without preemption. Each job requires two resources during its execution: (1) a *common resource*, which is required during a certain part of the job's processing period, and (2) a *secondary resource*, which is required during the whole processing period. The common resource is shared by all jobs whereas the secondary resource is shared only by a subset of the other jobs.

Problems with such characteristics arise, for example, in the context of the production of certain products. Imagine a single machine (the common resource) sequentially processing some fixtures or molds (the secondary resource) that contain some raw material. Before the processed fixtures/molds are available for further usage again, some postprocessing (e.g., cooling) might be required. However, our motivation for tackling this problem in this work has a different source: the scheduling of patients in modern particle therapy for cancer treatment [20,17,18]. In this rather novel cancer treatment technique, carbon or proton particles are accelerated in a cyclotron or synchrotron (i.e., a specific kind of particle accelerator) to almost

☆ We gratefully acknowledge the financial support of this project by the Doctoral Program "Vienna Graduate School on Computational Optimization" funded by the Austrian Science Foundation (FWF) under contract No. W1260-N35.

* Corresponding author.

E-mail addresses: horn@ac.tuwien.ac.at (M. Horn), raidl@ac.tuwien.ac.at (G. Raidl), christian.blum@iiia.csic.es (C. Blum).

https://doi.org/10.1016/j.artint.2019.103173
0004-3702/© 2019 Elsevier B.V. All rights reserved.

the speed of light. The particle beam is then directed into a treatment room where it is used to radiate a patient. Typically, between two and four differently equipped treatment rooms are available but only one particle accelerator, and the single particle beam can only be directed into one of these rooms at a time. The treatment room for each patient is determined in advance in dependence on the patient's specific needs. Each patient generally requires a specific preparation (related to positioning, fixation, sedation, etc.) in the room before the actual irradiation can start. Moreover, after the treatment some further time is usually needed for medical inspections before the patient can actually leave the room and the treatment of a next patient may start in the same room. Note that the available rooms correspond to the secondary resources mentioned above, while the particle beam is the common resource. We consider the scheduling of a set of patients in a given time period (for example, one day) in such a facility. The optimization goal is to finish the treatment of all patients as early as possible, which is known as *makespan minimization* in the related literature. This problem, whose technical description will be provided in Section 3, is henceforth denoted as *Job Sequencing with One Common and Multiple Secondary Resources* (JSOCMSR).

Note that the JSOCMSR problem is rather easy to solve when (1) the common resource usage is the exclusive bottleneck and enough secondary resources are available or (2) the pre- and postprocessing times during which only the secondary resources are required are negligible in comparison to the jobs' total processing times. In such cases the jobs can, essentially, be performed in almost an arbitrary ordering and the common resource is exploited without any breaks. The problem, however, becomes challenging when pre- and postprocessing times are substantial and many jobs require the same secondary resources. In this work we consider such difficult scenarios.

1.1. Contribution of this work

We first show that the JSOCMSR problem is NP-hard and derive three different functions for lower bounds on the makespan. These lower bounds are then exploited both in the context of a construction heuristic and a novel anytime A* algorithm. The classical A* algorithm is a well known standard algorithm in the field of Artificial Intelligence. Our novel A* algorithm includes an advanced diving mechanism based on *Beam Search* (BS) to provide intermediate heuristic solutions in regular intervals until a proven optimal solution is found. Hence, our A* algorithm is able to return a feasible heuristic solution whenever the algorithm is stopped (except for a short time period until the first solution is found). In addition, a *Local Search* (LS) procedure is applied to any intermediate heuristic solution obtained from BS to further improve these solutions. Our A* algorithm is able to efficiently exploit symmetries by performing the search on a special state graph structure. Last but not least, for comparison purposes we also consider a *Mixed Integer Linear Programming* (MIP) model which we solve with CPLEX as well as a *Constraint Programming* (CP) model for the JSOCMSR, which we solve with ILOG CP Optimizer, and we consider another anytime A* variant from the literature. Extensive experiments on difficult classes of problem instances demonstrate the excellent performance of the hybrid A*. Many even large instances with up to 2000 jobs can be solved to proven optimality within seconds, and for the remaining instances solutions with small optimality gaps of usually less than one percent can be obtained quite fast.

This work is an extension of our preliminary conference article [12], where we already proved that the JSOCMSR is NP-hard and provided a basic lower bound for the makespan. We also presented the construction heuristic and the anytime A* algorithm there, however, with only a simple greedy-based diving. For the sake of completeness we repeat here the NP-hardness proof, the definition of the basic lower bound regarding the makespan, as well as the construction heuristic.

2. Related work

Our already mentioned preliminary work on the JSOCMSR [12] considered, besides the construction heuristic and the simple anytime A* approach, three different MIP models for the JSOCMSR problem: (1) a time-indexed based formulation, (2) a position-based formulation and (3) a disjunctive MIP model. All three variants were clearly outperformed by our anytime A* approach. The best results among the three MIP models were obtained from the position based formulation, and its results were therefore included in [12]. For the sake of completeness we reconsider here also the position-based MIP formulation.

We split the further treatment of related work into three parts. First we consider the specific application background in particle therapy patient scheduling. The second part addresses further related problems, and the third part deals with related solution techniques.

Concerning particle therapy patient scheduling, the JSOCMSR clearly is a strongly simplified formulation picking out only certain aspects. The complete practical scenario comprises many more aspects, such as, for example, 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. A significant amount of previous work covers patient scheduling for radio therapy on a higher level without planning the detailed timing within each day; see, e.g., [13,4].

The single particle accelerator, which is only found in modern particle treatment facilities, is an expensive bottleneck resource that needs to be exploited in conjunction with multiple treatment rooms in the best possible way. As this technology is quite recent, there are only few existing works. Maschler et al. [20] proposed a greedy construction heuristic which is extended towards an *Iterated Greedy* (IG) metaheuristic and a *Greedy Randomized Adaptive Search Procedure* (GRASP). 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. Thus, our JSOCMSR represents the core of these daily sub-problems. In [19], the Iterated Greedy metaheuristic was further refined by including an improved makespan estimation for the daily scheduling problems, which, however, is still a rather crude approximation. Last but not least, this Iterated Greedy metaheuristic was extended in [17] to additionally consider soft constraints for unwanted deviations of starting times of the individual treatments for each therapy.

To the best of our knowledge, there are only few further publications dealing with other scenarios similar to the JSOCMSR. Among those, Veen et al. [34] consider a problem that has the closest relation to the JSOCMSR problem. The common resource corresponds to a machine on which the jobs are processed and secondary resources needed during pre- and postprocessing are called templates. However in this problem it can be assumed that the postprocessing times are negligible compared to the total processing times of the jobs. This implies that the starting time of each job only depends on its immediate predecessor. More specifically, 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 characteristics, this problem can be interpreted as a *Traveling Salesman Problem* (TSP) with a special cost structure. It is shown that this problem can be solved efficiently in time $O(n \log n)$, where n is the number of jobs.

Another related problem is the no-wait flowshop problem. A survey covering this problem in addition to similar problems can be found in [2]. 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 [8]. In contrast, for three and more machines the problem is NP-hard, although it can still be transformed into a specially structured TSP. Röck [25] proved that the problem is strongly NP-hard for three machines by a reduction from the 3D-matching problem.

Finally, note that the JSOCMSR problem can be modeled as a more general *Resource-Constrained Project Scheduling Problem* (RCPSP) with maximal time lags. For a survey on RCPSPs with various extensions and respective solution methods see Hartmann and Briskorn [11]. Among the state-of-the-art-techniques for solving general RCPSPs are in particular CP techniques utilizing lazy clause generation and satisfiability modulo theory [28,3]. A corresponding RCPSP instance can, for example, be obtained from a JSOCMSR instance by splitting each job into three activities which are the preprocessing, the main part also requiring the common resource, and the postprocessing. These activities must be performed for each job in this order with a maximal time lag of zero. Moreover, all resource requirements must be respected. Another way to obtain a corresponding RCPSP instance from a JSOCMSR instance is to use for each job two activities that must be scheduled at respectively related times: one activity for the part where the common resource is required and a second activity for the whole processing time where the secondary resource is required. We essentially make use of this point-of-view in our CP approach presented in Section 7.

As mentioned already above, one of our core contributions on the algorithmic side is an A* algorithm for solving the JSOCMSR problem. A* is one of the standard algorithms in the field of Artificial Intelligence for path planning and, more generally, finding shortest paths in possibly huge graphs [10,24]. In particular, our approach belongs to the class of Anytime A* algorithms, which are complete but also can be stopped at almost any time and are then likely to yield a reasonable approximate solution. Most of the Anytime A* algorithms from the literature are based on so-called heuristic weighted A* algorithms that were first introduced by Pohl [22]. These algorithms guarantee that the objective value of the obtained solution is not worse than the objective value of the optimal solution times an approximation ratio $\varepsilon \geq 1$. Thus, ε determines a trade-off between solution quality and the time it takes until the first heuristic solution is found. A large value for ε makes the algorithm greedier, meaning that the algorithm will faster find a heuristic solution. Hansen and Zhou [9] provide an Anytime Weighted A* algorithm which does not stop after the first solution is found but rather continues with the search. This algorithm produces a sequence of improved solutions as well as a sequence of improved dual bounds until the optimal solution is found. A similar approach is suggested by Likhachev, Gordon and Thrun [14], called Anytime Repairing A* (ARA*), where ε is decreased each time a new solution is found, instead of letting the parameter constant over the whole search. Restarting Weighted A* (RWA*) introduced by Richter, Thayer and Ruml [23] is based on ARA* and restarts the search each time ε is decreased. In this way, the algorithm is able to undo bad decisions, which may have been taken at the beginning of the search earlier. Another A* based anytime algorithm which does not require any parameter is Anytime Nonparametric A* (ANA*) introduced by van den Berg et al. [33]. The same algorithm was also independently introduced under the name Anytime Potential Search (APTS) by Stern et al. [30,31] via a different derivation [29]. Both algorithms maximize the greediness of the search based on the current incumbent solution. This leads to the fact that APTS/ANA* finds an initial solution faster and needs less time between improving solutions when compared to ARA*.

As we will see in the remainder of the paper, our new A* variant is able to rapidly find high quality solutions for the JSOCMSR problem by exploiting strong lower bounds for the makespan as search guidance. In order to obtain solutions of similar quality with one of the above weighted A* algorithms, parameter ε would need to be set very close to one. However, such a parameter choice typically leads in our case to a long running time until a first solution is found, which stays in contrast to our goal of achieving a good anytime behavior. Using a larger ε to find a first solution faster in general reduces the quality of this first solution substantially. In Section 8.8 we experimentally compare ARA* to our anytime A* algorithm and the results verify this behavior. Therefore using a weighted A* based algorithm is not promising for solving the JSOCMSR problem.

An anytime A* algorithm that is not based on weighted A* is Anytime Window A* (AWA*) as introduced by Aine et al. [1]. This algorithm uses a technique based on sliding windows to reduce the set of partial solution extensions to quickly find a heuristic solution. At each iteration the window width is increased such that improved solutions can be found until AWA* can eventually prove optimality.

Anytime Pack Search (APS), introduced by Vadlamudi et al. [32], is based on BS. The algorithm consecutively performs BS iterations, keeping partial solutions which are pruned during the BS iterations in memory. A BS iteration ends if a heuristic solution is found. Then a new BS iteration starts by selecting the best partial solution from the set of pruned partial solutions. The algorithm terminates if the set of pruned partial solutions is empty. In this case the last found solution must be an optimal solution. The APS approach is similar to our anytime A* approach since we also use BS to find intermediate heuristic solutions. The difference is that we embed BS into an A* algorithm such that BS is performed each time after a specific number of classical A* iterations. Our experimental comparison shows that for the JSOCMSR our anytime A* algorithm performs better than APS, cf. Section 8.4.

3. Problem definition, complexity, and bounds

In formal terms, an instance of the JSOCMSR problem consists of a set of n jobs $J = \{1, \dots, n\}$ that are to be executed without preemption, the common resource 0, and a set of m secondary resources $R = \{1, \dots, m\}$. The set of all resources is denoted by $R_0 = \{0\} \cup R$. Each job $j \in J$ has a total processing time $p_j > 0$ during which it fully requires a secondary resource $q_j \in R$. Furthermore, each job j requires the common resource 0 for a duration p_j^0 with $0 < p_j^0 \le p_j - p_j^{\text{pre}}$, where $p_j^{\text{pre}} \ge 0$ is the *preprocessing time*. In particular, the need for resource 0 starts at time p_j^{pre}, counted from the start of the job's processing. A solution to the problem is described by the starting times $s = [s_j]_{j \in J}$ of all jobs, with $s_j \ge 0$. Such a solution s is feasible if no two jobs require a resource at the same time.

The objective is to find a feasible schedule that minimizes the finishing time of the job that finishes last. This optimization criterion is known as the *makespan*, and it can be calculated for a solution s by

$$\text{MS}(s) = \max_{j \in J} (s_j + p_j). \tag{1}$$

As each job requires the common resource 0, and only one job can use this resource at a time, a solution implies with respect to $s_j + p_j^{\text{pre}}$ a total ordering of the jobs. Vice versa, any ordering—i.e., permutation—$\pi = [\pi_k]_{k=1,\dots,n}$, of the jobs in J can be decoded into a feasible solution in the straight-forward greedy way by scheduling each job in the given order at the earliest feasible time. We call a schedule in which, for a certain job permutation π, each job is scheduled at its earliest time, a *normalized schedule*. Obviously, any optimal solution is either a normalized schedule or there exists a corresponding normalized schedule with the same objective value. We therefore also use the notation $\text{MS}(\pi)$ for the makespan of the normalized solution induced by the job permutation π.

For convenience we further define the duration of the postprocessing time by $p_j^{\text{post}} = p_j - p_j^{\text{pre}} - p_j^0$, $\forall j \in J$ and denote by $J_r = \{j \in J \mid q_r = r\}$ the subset of jobs requiring resource $r \in R$ as secondary resource. Note that $J = \bigcup_{r \in R} J_r$. The minimal makespan over all feasible solutions, i.e., the optimal solution value, is denoted by MS^*.

3.1. Computational complexity

The decision variant of the JSOCMSR problem answers the question if there exists a feasible solution with a makespan corresponding to a given constant MS^*.

Theorem 1. *The decision variant of the JSOCMSR problem is NP-complete for $m \ge 2$.*

Proof. Our problem is in class NP since a solution can be checked in polynomial time. We show that the decision variant of the JSOCMSR problem is NP-complete by a polynomial reduction from the well-known NP-complete *Partition Problem* (PP) [7], which is stated as follows: Given a finite set of positive integers $A \subset \mathbb{N}$, partition it into two disjoint subsets A_1 and A_2 such that $\sum_{a \in A_1} a = \sum_{a \in A_2} a$.

An instance of the PP is transformed into an instance of the JSOCMSR problem as follows. Let $m = 2$ and J consist of the following jobs:

- For each $a \in A$ there is a corresponding job $j \in \{1, \dots, |A|\} \subset J$ with processing time $p_j = a$ requiring resource $q_j = 1$ and the common resource 0 the whole time, i.e., $p_j^0 = p_j$ and $p_j^{\text{pre}} = 0$.
- Furthermore, there are two jobs $j \in \{|A| + 1, |A| + 2\} \subset J$ with processing times $p_j = \frac{1}{2} \sum_{a \in A} a + 1$ requiring resource $q_j = 2$ the whole time but the common resource 0 just at the first time slot, i.e., $p^0 = 1$ and $p_j^{\text{pre}} = 0$.

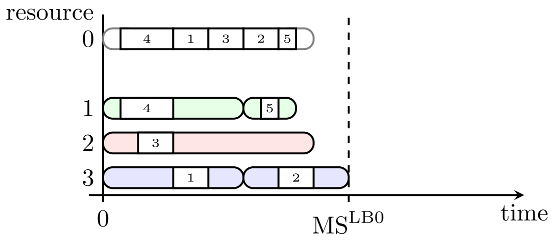


Fig. 1. Resource-specific individual lower bounds and the overall lower bound MS^{LB0} for an example instance with $n = 5$ jobs and $m = 3$ secondary resources.

Let $MS^* = p_{|A|+1} + p_{|A|+2} = \sum_{a \in A} a + 2$. A feasible solution to the JSOCMSR problem with makespan MS^* must have the jobs $|A|+1$ and $|A|+2$ scheduled sequentially without any gap and all other jobs in parallel to those two. A corresponding solution to the PP can immediately be derived by considering the integers associated with the jobs scheduled in parallel to job $|A+1|$ as A_1 and those scheduled in parallel to job $|A+2|$ as A_2. The obtained solution to the PP must be feasible since $\sum_{a \in A_1} a = \sum_{a \in A_2} a = \frac{1}{2}\sum_{a \in A} a$ holds as the jobs corresponding to the integers do not overlap and there is exactly $\frac{1}{2}\sum_{a \in A} a$ time left at the common resource 0 when processing job $|A|+1$ and job $|A|+2$, respectively. It also follows that if there is no JSOCMSR solution with makespan MS^*, then there cannot exist a feasible solution to the PP.

Clearly, the described transformation of a PP instance into a JSOCMSR instance as well as the derivation of the PP solution from the obtained schedule can both be done in time $O(|A|)$, i.e., in polynomial time.

Consequently, the decision variant of the JSOCMSR problem is NP-complete.

Corollary 1. *The makespan minimization variant of the JSOCMSR problem is NP-hard.*

3.2. Basic lower and upper bounds

First of all, a trivial upper bound for MS^* is obtained when scheduling all jobs strictly sequentially, yielding $MS^{UB} = \sum_{j \in J} p_j$.

A simple lower bound for the makespan can be calculated on the basis of each resource $r \in R$ by taking the total time over all jobs requiring resource r, i.e.,

$$MS_r^{LB0} = \sum_{j \in J_r} p_j. \tag{2}$$

Similarly, another lower bound can be obtained on the basis of resource 0 by

$$MS_0^{LB0} = \min_{j,j' \in J \mid j \neq j' \vee |J|=1} (p_j^{pre} + p_{j'}^{post}) + \sum_{j \in J} p_j^0. \tag{3}$$

Instead of taking only the sum of the time requirements for resource 0, this calculation also considers the minimal times for preprocessing and postprocessing of the first and last scheduled jobs, respectively. Now, we can take the maximum of these $m+1$ resource-specific lower bounds and obtain

$$MS^{LB0} = \max_{r \in R_0} MS_r^{LB0}. \tag{4}$$

Fig. 1 illustrates these relationships. It follows that taking any normalized solution has an approximation factor of no more than m, since $MS^{UB} \leq m \cdot MS^{LB0}$.

Theorem 2. MS^{LB0} *is a lower bound for the makespan of the JSOCMSR problem.*

Proof. Consider an arbitrary instance of the JSOCMSR problem and let MS^{LB0} be the computed lower bound according to (4). Assume that there exists a normalized feasible solution s with $MS(s) < MS^{LB0}$. Let r be a resource that determines the maximum in (4), i.e., $MS_r^{LB0} = MS^{LB0}$. We have to consider two cases:

- $r \neq 0$: The lower bound is determined by the sum of the total processing times of those jobs which require the secondary resource $r \in R$, that is, $\sum_{j \in J_r} p_j$. Since the makespan of solution s is smaller than the sum $\sum_{j \in J_r} p_j$, resource r must be used by more than one job at the same time. Hence, solution s would be infeasible, which contradicts our original assumption.
- $r = 0$: The lower bound is determined by the common resource 0. We have to distinguish between two cases: (1) $MS(s) < \sum_{j \in J} p_j^0$ and (2) $\sum_{j \in J} p_j^0 \leq MS(s) < \min_{j,j' \in J \mid j \neq j' \vee |J|=1}(p_j^{pre} + p_{j'}^{post}) + \sum_{j \in J} p_j^0$. In the first case solution s is not feasible because the common resource 0 must be used by more than one job at the same time to achieve a

makespan $\mathrm{MS}(s) < \mathrm{MS}^{\mathrm{LB0}}$. In the second case, the possibility remains that the common resource 0 is not used by more than one job at the same time. But then either there must exist a job $j \in J$ with a starting time $s_j < 0$ or there must exist a job $j' \in J$ such that $s_{j'} + p_{j'} > \mathrm{MS}(s)$. Again, all these cases contradict our original assumption.

We conclude that $\mathrm{MS}^{\mathrm{LB0}}$ is indeed a lower bound for MS^*.

3.3. Strengthened lower bounds

The assumption of lower bound $\mathrm{MS}^{\mathrm{LB0}}$ (from Eq. (4)) is that regarding a determining resource $r \in R$ all jobs J_r can be scheduled consecutively one after the other without any gaps. $\mathrm{MS}^{\mathrm{LB0}}$ can be further strengthened by checking if the usage of the common resource 0 by all other jobs in $J \setminus J_r$ causes conflict with this assumed schedule of jobs in J_r. This is done by considering the maximal possible continuous duration in which resource 0 is not used when scheduling the jobs in J_r consecutively in any order. To determine this duration we consider the maximum gap in the usage of resource 0 for any consecutively scheduled pair of jobs from J_r if $|J_r| > 1$, or just the maximum of the preprocessing and postprocessing times in case J_r is singleton, i.e.,

$$p^{\mathrm{prepost}}_{\max}(J_r) = \begin{cases} \max\limits_{j,j' \in J_r \mid j \neq j'} (p^{\mathrm{pre}}_j + p^{\mathrm{post}}_{j'}) & \text{if } |J_r| > 1 \\ \max\limits_{j \in J_r}(p^{\mathrm{pre}}_j, p^{\mathrm{post}}_j) & \text{if } |J_r| = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Now, each job $j \in J \setminus J_r$ that requires the common resource 0 longer then $p^{\mathrm{prepost}}_{\max}(J_r)$ causes a delay of at least $p^0_j - p^{\mathrm{prepost}}_{\max}(J_r)$ in the schedule of the jobs in J_r. Hence, the sum of all minimum delays

$$h^1_r = \begin{cases} \sum\limits_{j \in J \setminus J_r} \max\left(p^0_j - p^{\mathrm{prepost}}_{\max}(J_r), 0\right) & \text{if } J_r \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

for each resource $r \in R$ can be added to the corresponding lower bounds $\mathrm{MS}^{\mathrm{LB0}}_r$. We obtain $\mathrm{MS}^{\mathrm{LB1}}_r = \mathrm{MS}^{\mathrm{LB0}}_r + h^1_r$ as new bound for each secondary resource $r \in R$ and the overall strengthened lower bound

$$\mathrm{MS}^{\mathrm{LB1}} = \max\left(\mathrm{MS}^{\mathrm{LB0}}_0, \max_{r \in R} \mathrm{MS}^{\mathrm{LB1}}_r\right). \tag{7}$$

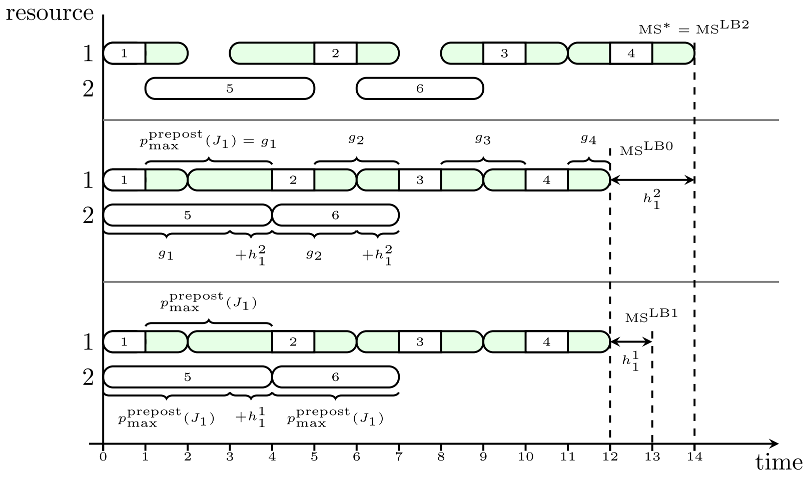
Lower bound $\mathrm{MS}^{\mathrm{LB1}}$ can be even further strengthened. Instead of considering always only the largest possible gap $p^{\mathrm{prepost}}_{\max}(J_r)$ in the usage of resource 0 to compute the sum of delays as done in Eq. (6), we can more precisely consider a tighter set of maximal time gaps $[g_i]^k_{i=1}$ with $k = |J_r|$ that possibly may occur together in a consecutive scheduling of the jobs in J_r. Algorithm 1 shows the calculation of this set, which is obtained as a decreasingly sorted sequence of time gaps $[g_i]^k_{i=1}$, i.e., $g_1 = p^{\mathrm{prepost}}_{\max}(J_r) \geq g_2 \geq \cdots \geq g_k$, for $r \in R$.

Line 3 ensures that the first element of the sequence is always $g_1 = p^{\mathrm{prepost}}_{\max}(J_r)$. Sets J^{pre} and J^{post} contain the jobs whose pre- and postprocessing times are not already consumed for calculating the next time gap, respectively. At each iteration k of the while-loop, jobs $j_1 \in J^{\mathrm{pre}}$ and $j_2 \in J^{\mathrm{post}}$ are determined such that $g_k = p^{\mathrm{pre}}_{j_1} + p^{\mathrm{post}}_{j_2}$ is the next largest time gap. At the end of the loop jobs j_1 and j_2 are removed from sets J^{pre} and J^{post}, respectively.

Algorithm 1 Determine time gaps $[g_i]^k_{i=1}$ for jobs J_r.

```
1: Input: Jobs $J_r$
2: Output: Sequence of time gaps $[g_i]^k_{i=1}$
3: $g_1 \leftarrow p^{\mathrm{prepost}}_{\max}(J_r)$ with corresponding jobs $j_1$ and $j_2$ for $p^{\mathrm{pre}}_{j_1}$ and $p^{\mathrm{post}}_{j_2}$, respectively
4: $J^{\mathrm{pre}} \leftarrow J^{\mathrm{pre}} \setminus \{j_1\}$,   $J^{\mathrm{post}} \leftarrow J^{\mathrm{post}} \setminus \{j_2\}$
5: $k \leftarrow 1$
6: while $J^{\mathrm{pre}} \neq \emptyset \wedge J^{\mathrm{post}} \neq \emptyset$ do
7:     $k \leftarrow k + 1$
8:     $(j_1, j_2) \leftarrow \arg\max_{(j,j') \in J^{\mathrm{pre}} \times J^{\mathrm{post}}} (p^{\mathrm{pre}}_j + p^{\mathrm{post}}_{j'})$
9:     $g_k \leftarrow p^{\mathrm{pre}}_{j_1} + p^{\mathrm{post}}_{j_2}$
10:    $J^{\mathrm{pre}} \leftarrow J^{\mathrm{pre}} \setminus \{j_1\}$,   $J^{\mathrm{post}} \leftarrow J^{\mathrm{post}} \setminus \{j_2\}$
11: end while
12: return $[g_i]^k_{i=1}$
```

Algorithm 2 shows the calculation of the strengthened sum of minimum delays h^2_r exploiting the time gaps $[g_i]^k_{i=1}$ for resource $r \in R$. Set J^0 contains all jobs $J \setminus J_r$ which are not yet consumed. After determining the time gaps $[g_i]^k_{i=1}$ the



resource
$\mathrm{MS}^* = \mathrm{MS}^{\mathrm{LB2}}$
$p^{\mathrm{prepost}}_{\max}(J_1) = g_1$
g_2
g_3
g_4
$\mathrm{MS}^{\mathrm{LB0}}$
h^2_1
g_1
$+h^2_1$
g_2
$+h^2_1$
$p^{\mathrm{prepost}}_{\max}(J_1)$
$\mathrm{MS}^{\mathrm{LB1}}$
h^1_1
$p^{\mathrm{prepost}}_{\max}(J_1)$
$+h^1_1$
$p^{\mathrm{prepost}}_{\max}(J_1)$
time

Fig. 2. Example for the lower bounds $\mathrm{MS}^{\mathrm{LB0}}$, $\mathrm{MS}^{\mathrm{LB1}}$ and $\mathrm{MS}^{\mathrm{LB2}}$.

for-loop iterates through them starting with the largest time gap g_1. At each iteration, the job j that, among those currently being in J^0, has the largest requirement of resource 0 is determined. The loop terminates if set J^0 becomes empty or p^0_j is less than the currently largest time gap g_i. If $p^0_j \geq g_i$ then it is not possible to schedule job j without shifting some of the starting points of the jobs in J_r. Therefore, we add the difference $p^0_j - g_i$ to h^2_r. Afterwards job j is removed from set J^0. In the last step of Algorithm 2 the total processing times from the remaining jobs in J^0 are added to h^2_r.

Algorithm 2 Determine the strengthened sum of the minimum delays h^2_r.

```
 1: **Input:** Resource $r$
 2: **Output:** $h^2_r$
 3: **if** $J_r = \emptyset$ **then return** 0
 4: $h^2_r \leftarrow 0$
 5: $J^0 \leftarrow J \setminus J_r$
 6: determine time gaps $[g_i]^k_{i=1}$ concerning $J_r$ according to Algorithm 1
 7: **for** $i = 1$; $J^0 \neq \emptyset \wedge i \leq k$; $i \leftarrow i + 1$ **do**
 8:     $j \leftarrow \arg\max_{j \in J^0} p^0_j$
 9:     **if** $p^0_j < g_i$ **then return** $h^2_r$
10:     $h^2_r \leftarrow h^2_r + p^0_j - g_i$
11:     $J^0 \leftarrow J^0 \setminus \{j\}$
12: **end for**
13: **return** $h^2_r + \sum_{j \in J^0} p^0_j$
```

Once all terms h^2_r for $r \in R$ are computed by Algorithm 2, they are added to the basic individual lower bounds, that is, $\mathrm{MS}^{\mathrm{LB2}}_r = \mathrm{MS}^{\mathrm{LB0}}_r + h^2_r$. Finally, the joined strengthened lower bound is obtained by

$$\mathrm{MS}^{\mathrm{LB2}} = \max\left(\mathrm{MS}^{\mathrm{LB0}}_0, \max_{r \in R} \mathrm{MS}^{\mathrm{LB2}}_r\right). \tag{8}$$

Fig. 2 shows an example where the bound $\mathrm{MS}^{\mathrm{LB2}}$ corresponds to the optimum makespan MS^* and the bounds $\mathrm{MS}^{\mathrm{LB1}}$ and $\mathrm{MS}^{\mathrm{LB0}}$ are weaker. The example illustrates an instance with $n = 6$ jobs and $m = 2$ secondary resources. Jobs $J_1 = \{1, 2, 3, 4\}$ need secondary resource 1 and jobs $J_2 = \{5, 6\}$ resource 2. Note that all jobs in J_1 require the common resource 0 exactly one unit of time whereas all jobs in J_2 need resource 0 during their whole execution. The optimal solution is shown in the upper third of the figure with an optimal makespan of $\mathrm{MS}^* = 14$. However, according to Eq. (4) $\mathrm{MS}^{\mathrm{LB0}} = 12$, which is illustrated in the lower part of the figure. The common resource 0 is at most $p^{\mathrm{prepost}}_{\max}(J_1) = 3$ units of time free when considering only the jobs in J_1. Hence, $h^1_1 = 1$ according to (6) since job 5 claims resource 0 for more than 3 units of time and job 6 claims resource 0 for exactly 3 units. This gives us a strengthened bound of $\mathrm{MS}^{\mathrm{LB1}} = 13$ according to (7), which is illustrated in the lower part of Fig. 2. However, considering the time gaps $[g_i]^4_{i=1} = [3, 2, 2, 1]$ for resource 1 according to Algorithm 1, we get $h^2_1 = p^0_5 - g_1 + p^0_6 - g_2 = 4 - 3 + 3 - 2 = 2$. This results in an improved lower bound $\mathrm{MS}^{\mathrm{LB2}} = 14$ as shown in the middle part of Fig. 2.

Theorem 3. $\mathrm{MS}^{\mathrm{LB1}}$ *and* $\mathrm{MS}^{\mathrm{LB2}}$ *are lower bounds for the makespan and* $\mathrm{MS}^{\mathrm{LB0}} \leq \mathrm{MS}^{\mathrm{LB1}} \leq \mathrm{MS}^{\mathrm{LB2}}$ *for any instance of the JSOCMSR problem.*

Proof. See the Appendix A.

4. Least lower bound heuristic

We construct a heuristic solution by iteratively selecting a not yet scheduled job, which is executed after all already scheduled jobs at the earliest possible time. The crucial aspect is the greedy selection of the job to be scheduled next, which is based on the lower bound calculation from the previous section. Therefore we call this heuristic *Least Lower Bound Heuristic* (LLBH).

Let π^{p} be the current partial job permutation representing the current normalized schedule and $J' \subseteq J$ be the set of remaining unscheduled jobs. Initially, π^{p} is empty and $J' = J$. Given π^{p}, the *earliest availability time* for each resource—that is, the earliest time when the resource could be used by a next yet unscheduled job—can be calculated from the respective finishing time of the last job using this resource:

$$t_0 = \begin{cases} \max_{j \in J \setminus J'} s_j + p_j^{\mathrm{pre}} + p_j^0 & \text{for } J' \neq J \\ 0 & \text{else} \end{cases} \tag{9}$$

$$t_r = \begin{cases} \max_{j \in J_r \setminus J'} s_j + p_j & \text{for } J_r \setminus J' \neq \emptyset \\ 0 & \text{else} \end{cases} \qquad \forall r \in R \tag{10}$$

These times, however, can possibly be further increased (*trimmed*) as the earliest usage time of resource $r \in R$ by a successive job also depends on the remaining unscheduled jobs and the earliest usage time of the common resource 0. We therefore apply the rule

$$t_r \leftarrow \max\left(t_r,\, t_0 - \max_{j \in J_r \cap J'} p_j^{\mathrm{pre}}\right) \qquad \forall r \in R \mid J_r \cap J' \neq \emptyset. \tag{11}$$

Moreover, also t_0 might be increased as its earliest usage time also depends on the remaining unscheduled jobs and the earliest usage times of their secondary resources. These relations are considered by applying the rule

$$t_0 \leftarrow \max\left(t_0,\, \min_{j \in J'}(t_{q_j} + p_j^{\mathrm{pre}})\right) = \max\left(t_0,\, \min_{r \in R \mid J_r \cap J' \neq \emptyset}(t_r + \min_{j \in J_r \cap J'} p_j^{\mathrm{pre}})\right). \tag{12}$$

Also note that after a successful increase of t_0 by rule (12), it might be possible to further increase the t_r-value of some resource $r \in R$ according to the respective rule (11). We therefore apply the trimming rules repeatedly until no further increase of any of the earliest availability times is achieved.

Following our general lower bound calculation for the makespan in (4), it is possible to derive a more specific lower bound for a given partial permutation π^{p} considering any possible extension to a complete solution on the basis of each resource $r \in R \mid J_r \cap J' \neq \emptyset$ by

$$\mathrm{MS}_r^{\mathrm{LB0}}(\pi^{\mathrm{p}}) = \begin{cases} t_r + \sum_{j \in J_r \cap J'} p_j & \text{for } J_r \cap J' \neq \emptyset \\ 0 & \text{else} \end{cases} \qquad \forall r \in R. \tag{13}$$

We define $\mathrm{MS}_r^{\mathrm{LB0}}(\pi^{\mathrm{p}}) = 0$ for any resource r that is not required by any remaining job in J' since these bounds are not relevant for our further considerations.

A lower bound w.r.t. the common resource 0 can be calculated similarly by

$$\mathrm{MS}_0^{\mathrm{LB0}}(\pi^{\mathrm{p}}) = \max\left(t_0 + \min_{j \in J'} p_j^{\mathrm{post}},\, \min_{j,j' \in J' \mid j \neq j' \vee |J'|=1}(t_{q_j} + p_j^{\mathrm{pre}} + p_{j'}^{\mathrm{post}})\right) + \sum_{j \in J'} p_j^0. \tag{14}$$

Clearly, an overall lower bound for the partial solution π^{p} is obtained by taking the maximum of the individual bounds

$$\mathrm{MS}_{\max}^{\mathrm{LB0}}(\pi^{\mathrm{p}}) = \max_{r \in R_0} \mathrm{MS}_r^{\mathrm{LB0}}(\pi^{\mathrm{p}}). \tag{15}$$

For selecting the next job in LLBH to be appended to π^{p}, we always consider the impact of each job $j \in J'$ on each individual bound $\mathrm{MS}_r^{\mathrm{LB0}}$, $r \in R_0$, as this gives a more fine-grained discrimination than just considering the impact on the overall bound $\mathrm{MS}_{\max}^{\mathrm{LB0}}(\pi^{\mathrm{p}})$, which would often lead to ties.

More specifically, let $\vec{f}(\pi^{\mathrm{p}}) = [f_0(\pi^{\mathrm{p}}), \ldots, f_m(\pi^{\mathrm{p}})]$ be the vector of the bounds $\mathrm{MS}_r^{\mathrm{LB0}}(\pi^{\mathrm{p}})$ for $r \in R_0$ *sorted in non-increasing value order*, i.e., $f_0(\pi^{\mathrm{p}}) = \mathrm{MS}_{\max}^{\mathrm{LB0}}(\pi^{\mathrm{p}}) \geq f_1(\pi^{\mathrm{p}}) \geq \ldots \geq f_m(\pi^{\mathrm{p}})$ holds.

Let $\pi^{\mathrm{p}} \oplus j$ denote the partial solution obtained by appending job $j \in J'$ to π^{p}. We consider $\pi^{\mathrm{p}} \oplus j$ *better than* $\pi^{\mathrm{p}} \oplus j'$ for $j, j' \in J'$ iff there exists an $i \in \{0, \ldots, m\}$ such that

$$f_i(\pi^{\mathrm{p}} \oplus j) < f_i(\pi^{\mathrm{p}} \oplus j') \ \wedge \forall i' < i \mid f_{i'}(\pi^{\mathrm{p}} \oplus j) = f_{i'}(\pi^{\mathrm{p}} \oplus j'). \tag{16}$$

In other words, the sorted vectors $\vec{f}(\pi^{\mathrm{p}} \oplus j)$ and $\vec{f}(\pi^{\mathrm{p}} \oplus j')$ are compared in a lexicographic way.

LLBH selects at each iteration a job $j \in J'$ yielding a (locally) best extension. In the case when multiple extensions have equal $\vec{f}$-vectors, one of them is chosen at random.

4.1. Strengthened lower bounds for partial solutions

As done in Section 3.3, we can also strengthen the lower bound $\mathrm{MS}^{\mathrm{LB0}}(\pi^{\mathrm{p}})$ for a partial solution π^{p} by checking for each secondary resource $r \in R$ if a consecutive schedule of all not yet scheduled jobs $J \cap J_r$ is possible. This is done by adding the strengthening terms $h_r^i(\pi^{\mathrm{p}})$ to $\mathrm{MS}_r^{\mathrm{LB0}}(\pi^{\mathrm{p}})$ for each secondary resource $r \in R$—that is, $\mathrm{MS}_r^{\mathrm{LB}i}(\pi^{\mathrm{p}}) = \mathrm{MS}_r^{\mathrm{LB0}}(\pi^{\mathrm{p}}) + h_r^i(\pi^{\mathrm{p}})$—and taking the maximum of these lower bounds to get the final strengthened lower bound for partial solution π^{p} and strengthening type $i \in \{1, 2\}$, i.e.,

$$\mathrm{MS}^{\mathrm{LB}i}(\pi^{\mathrm{p}}) = \max\left(\mathrm{MS}_0^{\mathrm{LB0}}(\pi^{\mathrm{p}}), \max_{r \in R} \mathrm{MS}_r^{\mathrm{LB}i}(\pi^{\mathrm{p}})\right). \tag{17}$$

The terms h_r^1 and h_r^2, for all $r \in R$, can be computed in a way similar to Section 3.3, by either considering the maximum time gap $p_{\max}^{\mathrm{prepost}}(J' \cap J_r)$ (see Eq. (5)) or by considering a sequence of time gaps $[g_i]_{i=1}^k$ as computed by Algorithm 1, respectively. This time, however, we only consider jobs $J' \cap J_r$ that are not already scheduled and require the secondary resource r. Furthermore we also have to take into account that some resources are already used. If the earliest availability time of resource r is greater than the earliest availability time of resource 0, i.e., $t_r > t_0$ then we also have to consider the additional time lag $t_r - t_0$ for computing h_r^1 or h_r^2. This can be achieved by temporarily adding a *pseudo job* $n+1$ with $q_{n+1} = r$, $p_{n+1}^{\mathrm{pre}} = 0$, $p_{n+1}^0 = t_0$ and $p_{n+1} = t_r$ to set J'. Note that the postprocessing time is $p_{n+1}^{\mathrm{post}} = t_r - t_0$. Hence, for the calculation of the strengthening terms we assume that the already consumed parts of resource r are represented by job $n+1$. Concerning the LLBH, the vector of lower bounds $\vec{f}(\pi^{\mathrm{p}})$ can be computed in exactly the same way as described before by using the strengthened lower bounds $\mathrm{MS}_r^{\mathrm{LB}i}(\pi^{\mathrm{p}})$ instead of the basic lower bounds $\mathrm{MS}_r^{\mathrm{LB0}}(\pi^{\mathrm{p}})$ for the secondary resources $r \in R$.

4.2. Combined lower bounds

Although $\mathrm{MS}^{\mathrm{LB2}}$ is—according to Theorem 3—the strongest lower bound among the ones described in Section 3, in preliminary experiments it turned out that, in practice, the basic lower bound $\mathrm{MS}^{\mathrm{LB0}}$ tends to guide the LLBH better than $\mathrm{MS}^{\mathrm{LB2}}$. This is because, for a given partial solution π^{p}, the evaluation vectors $\vec{f}^2(\pi^{\mathrm{p}} \oplus j) = [f_0^2(\pi^{\mathrm{p}} \oplus j), f_1^2(\pi^{\mathrm{p}} \oplus j), \ldots, f_m^2(\pi^{\mathrm{p}} \oplus j)]$ with respect to $\mathrm{MS}^{\mathrm{LB2}}$, for all $j \in J'$, tend to be more similar to each other than the evaluation vectors $\vec{f}^0(\pi^{\mathrm{p}} \oplus j) = [f_0^0(\pi^{\mathrm{p}} \oplus j), f_1^0(\pi^{\mathrm{p}} \oplus j), \ldots, f_m^0(\pi^{\mathrm{p}} \oplus j)]$ concerning the basic lower bound $\mathrm{MS}^{\mathrm{LB0}}$. Furthermore, the elements of the vectors $\vec{f}^2(\pi^{\mathrm{p}} \oplus j)$ tend to be more similar so that they are not such a good discriminator to select more promising extensions. In order to obtain an evaluation vector with both properties—that is, (1) a good search guidance for the LLBH and (2) a strong lower bound for partial solutions π^{p}—we combine vectors $\vec{f}^2(\pi^{\mathrm{p}})$ and $\vec{f}^0(\pi^{\mathrm{p}})$ in an interleaved way. More precisely, in order to make use of a combination of lower bounds $\mathrm{MS}^{\mathrm{LB0}}$ and $\mathrm{MS}^{\mathrm{LB2}}$, the vector of lower bounds $\vec{f}$ used in LLBH is defined as

$$\vec{f}(\pi^{\mathrm{p}}) := \left[f_0^2(\pi^{\mathrm{p}}), f_0^0(\pi^{\mathrm{p}}), \ldots, f_m^2(\pi^{\mathrm{p}}), f_m^0(\pi^{\mathrm{p}})\right]. \tag{18}$$

Hereby, the first element is always equal to the stronger lower bound $\mathrm{MS}^{\mathrm{LB2}}$, whereas the second element is always equal to the basic lower bound $\mathrm{MS}^{\mathrm{LB0}}$.

5. A* algorithm and extensions

The proposed A* algorithm follows more or less the classical principle as described in [10]. It performs a minimum cost path search on a weighted directed acyclic state graph from a start node x_{start} to a goal node. The algorithm manages on *open list* Q and the set of all already considered nodes. Each node x can be evaluated by a function $f(x) = g(x) + h(x)$, where the term $g(x)$ represents the so far smallest known cost from the start node x_{start} to node x, and the heuristic term $h(x)$ represents estimated cost from node x to a goal node. Initially, the start node x_{start} is created and inserted into the open list. At each iteration of A* a node x minimizing function $f(x)$ is taken from the open list. This node is *expanded* by deriving all possible successor states. For each successor state x' it is checked if a corresponding node already exists. If this is the case, $g(x')$ is updated if a *dominating* (i.e., shorter) path to x' has been found by reaching x' from x; otherwise, the transition from x to x' can be skipped. If no node exists yet for state x', a new one is created and inserted into the open list Q. The algorithm terminates when A* selects the goal node x_{goal} for expansion.

If function h is *admissible*, meaning that $h(x)$ never overestimates the minimum cost from node x to the goal node x_{goal} for all nodes x and $h(x_{\mathrm{goal}}) = 0$, then A* is known to be complete, i.e., it yields a proven minimum cost path. Under some restrictions and conditions, it can further be shown that—using the same heuristic information of function h—there exists no algorithm which expands fewer nodes than A* to find a proven minimum cost path [10,5].

Our idea is to extend the solution construction principle of LLBH to perform a systematic search for a proven optimal solution. Let us reconsider the evaluation vector $\vec{f}$ from the LLBH. Since the first entry of this evaluation vector represents a lower bound for the JSOCMSR according to Theorem 3, it can be used as admissible evaluation function f. The vector

$\vec{t} = [t_r]_{r \in R_0}$ of the trimmed earliest availability times t_r, $r \in R_0$ as defined by Eqs. (9)–(12) can be seen as the already occurred costs, and the remaining sums of processing times are the heuristic information used to estimate the makespan of the optimal solution.

It remains to define the nodes in our weighted directed acyclic state graph in more detail. In principle, each node represents any (partial-)schedule that already schedules exactly a specific set of jobs. More precisely, each node contains the following information: (1) an unordered set $\hat{J} \subset J$ of already scheduled jobs, implemented by a bit-vector, and (2) a set of so-called *Non-Dominated Times* (NDT) records. Each NDT record represents a specific partial solution by storing: (1) the vector $\vec{t}$ of earliest availability times, (2) the last scheduled job $j^{\text{last}} \in \hat{J}$ after which $\vec{t}$ was obtained, and (3) an evaluation vector $\vec{f}'$ similar to $\vec{f}$ with one modification that will be described below. Thus, each node aggregates all partial solutions π^{p} having the same jobs $\hat{J}$ scheduled, and each NDT record provides more specific information for each non-dominated partial solution. In our state graph there is exactly on goal node x_{goal} which represents any complete schedule and one start node x_{start} which represents the empty schedule. The ordering of the scheduled jobs $\hat{J}$ of a specific partial solution is indirectly given. For a pair of a node and one of its NDT records—henceforth called a (node, NDT record) pair—the corresponding ordering can be derived in a reverse iterative manner by considering the fitting preceding (node, NDT record) pairs, always continuing with a pair where the node is characterized by $\hat{J} \setminus \{j^{\text{last}}\}$ and the NDT record has times t_r allowing to schedule job j^{last} without exceeding the t_r values of the previous (node, NDT record) pair.

Initially a starting (node, NDT record) pair corresponding to the empty schedule is generated with $\hat{J} = \emptyset$, $\vec{t} = \vec{0}$, $j^{\text{last}} =$ none, and $\vec{f}' = (\text{MS}^{\text{LB}i}, \dots, \text{MS}^{\text{LB}i})$ with a chosen lower bound type $i \in \{0, 1, 2\}$. The goal node is the node with $\hat{J} = J$, corresponding to all complete solutions.

The set of all so far considered nodes is implemented by a hash-table with $\hat{J}$ as key in order to efficiently find already existing nodes for reached states. The open list is more specifically realized by a priority queue Q containing references to all *open* (node, NDT record) pairs, that is, the non-dominated partial solutions that have not yet been expanded. As order criterion the *is-better* relation from the LLBH (see Eq. 16) is used.

Note that the first entry of our evaluation vector $\vec{f}$ from LLBH is not necessarily monotonically non-decreasing when considering any path from x_{start} to x_{goal}. Thus, it can happen that the lower bound $f_0^i(\pi^{\text{p}} \oplus j)$ of an extension of a partial solution π^{p} with job $j \in J \setminus \hat{J}(\pi^{\text{p}})$ is less than the lower bound $f_0^i(\pi^{\text{p}})$. To establish monotonicity at least for the first element in vector $\vec{f}$—which corresponds to the lower bound of π^{p}—we use a modified evaluation vector $\vec{f}'$ during the A* search, where the first element $f_0'^i(\pi^{\text{p}} \oplus j)$ is always set to $\max(f_0'^i(\pi^{\text{p}}), f_0'^i(\pi^{\text{p}} \oplus j))$. In this way, the largest lower bound along a path from x_{start} to x_{goal} is always preserved.

Algorithm 3 provides a pseudo-code of our A* algorithm, already including details about the embedded Beam Search and Local Search, which will be described in the following subsections. In each major iteration, a best (node, NDT record) pair is taken from the open list Q in line 12 and expanded in line 19 by considering the addition of each job $j \in J \setminus \hat{J}$ as shown separately in Algorithm 4. Hereby, the corresponding node x is looked up—or created, in case it does not yet exist—and a respective NDT record is determined by calculating the earliest usage times $\vec{t}$ and the evaluation vector $\vec{f}'$. If the lower bound of the first element of vector $\vec{f}'$ is larger than the makespan of our current best solution $\text{MS}(\pi^{\text{best}})$, this new NDT record cannot lead to a better solution. Therefore, in this case the new NDT record is disregarded and we move on to the next unscheduled job $j \in J \setminus \hat{J}$. Otherwise, the possibly multiple NDT records in the node are checked for dominance: Only non-identical and non-dominated entries are kept. An NDT record with time vector $\vec{t}$ *dominates* (denoted by $\lhd$) another NDT record with time vector $\vec{t}'$ iff $\forall r \in R_0\ (t_r \le t'_r) \wedge \exists r \in R_0\ (t_r < t'_r)$. Algorithm 4 returns all newly created (node, NDT record) pairs which are then inserted into the open list. Note that we implicitly assume that (node, NDT record) entries in the open list Q corresponding to dominated and therefore removed NDT records are also deleted or skipped when selected for expansion. The A* algorithm stops when the goal node representing a complete solution is selected for expansion. This complete solution must be optimal since f'_0 is an admissible lower bound for the makespan.

5.1. Advanced diving with Beam Search

The A* algorithm described above aims at finding a proven optimal solution as quickly as possible. Feasible solutions are, however, usually only found very late. To turn A* into an anytime algorithm which is able to find also intermediate complete heuristic solutions significantly earlier than when terminating with a proven optimum we proposed in our preliminary work [12] a LLBH-based diving extension: In regular intervals, the search switches from the A* strategy temporarily to depth-first search node expansion following the successor selection from LLBH until a complete solution is obtained. In this work we extend this simple diving to a more powerful BS.

Beam Search is a heuristic search method to solve combinatorial optimization problems. The search is performed on a graph and can be seen as an extension of depth first search. However, instead of expanding always the most promising node at each level of the generated search tree, BS expands the k_{bw}-most promising nodes, where parameter k_{bw} is also called *beam width*. Note that if $k_{\text{bw}} = 1$ BS behaves exactly like greedy depth first search until a complete solution has been reached. Beam Search was first used by Lowerre [15] in a speech recognition system and by Rubin [26] for image recognition. Furthermore BS was also successfully applied to scheduling problems such as job shop scheduling [6,27] and single machine scheduling [35].

Our Algorithm 3 switches at the very beginning and after each δ regular iterations from its classical best-first strategy temporarily to a BS-based completion strategy in order to find promising complete solutions at regular intervals. After such

Algorithm 3 A*+BS+LS Algorithm for JSOCMSR.

```
1: initialize open list $Q$ and $B$ with $(\emptyset, (\vec{0}, \text{none}, (\text{MS}^{\text{LB0}}, \ldots, \text{MS}^{\text{LB0}}))$
2: $iter \leftarrow 0$, diving $\leftarrow true$, $B_{\text{ext}} \leftarrow \emptyset$
3: repeat
4:     if diving then
5:         if $B = \emptyset$ then
6:             $B \leftarrow$ the best $k_{\text{bw}}$ entries from $B_{\text{ext}}$ according to $\vec{f}'$
7:             $B_{\text{ext}} \leftarrow \emptyset$
8:         end if
9:         $(\hat{J}, (\vec{t}, j^{\text{last}}, \vec{f}')) \leftarrow$ select randomly one entry from $B$
10:        remove $(\hat{J}, (\vec{t}, j^{\text{last}}, \vec{f}'))$ from Q and $B$
11:    else
12:        $(\hat{J}, (\vec{t}, j^{\text{last}}, \vec{f}')) \leftarrow$ Q.pop()
13:    end if
14:    if $|\hat{J}| = n$ and not diving then
15:        $\pi^{\text{best}} \leftarrow$ derive complete solution from $(\hat{J}, (\vec{t}, j^{\text{last}}, \vec{f}'))$
16:        return proven optimal solution $\pi^{\text{best}}$
17:    end if
18:    if $iter$ mod $\delta = 0$ then diving $\leftarrow true$ end if
19:    $E \leftarrow$ expand$((\hat{J},(\vec{t}, j^{\text{last}}, \vec{f}')), \pi^{\text{best}}$, diving)      ▷ (see Algorithm 4)
20:    Q.insert($E$)
21:    apply LS to all complete solutions in $E$      ▷ (see Algorithm 5)
22:    if new $\pi^{\text{best}}$ obtained from LS then
23:        start injection of $\pi^{\text{best}}$      ▷ (see Algorithm 6)
24:    end if
25:    if diving then $B_{\text{ext}} \leftarrow B_{\text{ext}} \cup E$ end if
26:    if not diving then $iter \leftarrow iter + 1$ end if
27:    if $B = \emptyset \wedge B_{\text{ext}} = \emptyset$ then diving $\leftarrow false$ end if
28: until time- or memory-limit reached
29: return heuristic solution $\pi^{\text{best}}$ and lower bound $f_0$
```

Algorithm 4 A* Node Expansion Algorithm.

```
1: Input: (node, NDT record) pair $(\hat{J},(\vec{t}, j^{\text{last}}, \vec{f}'))$, current best solution $\pi^{\text{best}}$, parameter diving indicating if in diving mode or not
2: Output: set $E$ of extensions, i.e., new non-dominated (node, NDT record) pairs
3: $E \leftarrow \emptyset$
4: for all $j \in J \setminus \hat{J}$ do
5:     find or create node $x$ with $\hat{J}(x) = \hat{J} \cup \{j\}$
6:     calculate new NDT record $(\vec{t}_{\text{new}}, j, \vec{f}'_{\text{new}})$ from $\vec{t}$
7:     if diving or $\text{MS}(\pi^{\text{best}}) \geq f'_{0,\text{new}}$ then
8:         if $\nexists(\vec{t}_{\text{d}}, j^{\text{last}}_{\text{d}}, \vec{f}'_{\text{d}}) \in \text{NDTs}(x) \mid \vec{t}_{\text{d}} = \vec{t}_{\text{new}} \vee \vec{t}_{\text{d}} \lhd \vec{t}_{\text{new}}$ then
9:             Remove every $(\vec{t}_{\text{d}}, j'_{\text{d}}, \vec{f}'_{\text{d}}) \in \text{NDTs}(x) \mid \vec{t}_{\text{new}} \lhd \vec{t}_{\text{d}}$
10:            Add $(\vec{t}_{\text{new}}, j, \vec{f}'_{\text{new}})$ to NDTs$(x)$
11:            $E \leftarrow E \cup \{(\hat{J}, (\vec{t}_{\text{new}}, j, \vec{f}'_{\text{new}}))\}$
12:        else if $|\hat{J}(x)| = n$ then
13:            $E \leftarrow E \cup \{(\hat{J}, (\vec{t}_{\text{new}}, j, \vec{f}'_{\text{new}}))\}$
14:        end if
15:    end if
16: end for
```

a switch to diving mode (indicated by setting the parameter diving to *true*), the currently selected node is expanded. From all obtained extensions, only those that are new and non-dominated are kept and added to the open list Q and to the set of current beam extensions B_{ext}. Afterwards only the best k_{bw} extensions from set B_{ext} are filtered out as beam set B. The BS continues by expanding all extensions from set B with Algorithm 4. Again, the hereby created extensions are added to open list Q and set B_{ext} if they are new and non-dominated. If an extension represents a complete solution then we also keep it even if the corresponding earliest availability times are dominated. The reason for this exception is that we apply a LS procedure to all complete solutions later in line 21 of Algorithm 3. If all extensions from set B are expanded, set B is replaced by the best k_{bw} extensions from set B_{ext}. The process continues until set B as well as set B_{ext} become empty, in which case the algorithm switches back to A*'s normal best-first search strategy. Note that for beam width $k_{\text{bw}} = 1$, our A* embedded BS corresponds to the simple diving from [12]. Furthermore each (node,NDT) pair which is expanded during the BS phase is kept in the A* search tree such that A* will avoid expanding them a second time.

5.2. Local search and solution injection

We attempt to further improve any complete intermediate solution by the following LS procedure based on the *insertion neighborhood*. Given a solution π, the insertion neighborhood contains all solutions that can be obtained by removing a job $j \in J$ from π, which we denote by $\pi \ominus j$, and reinserting j at another position. Following a best-improvement

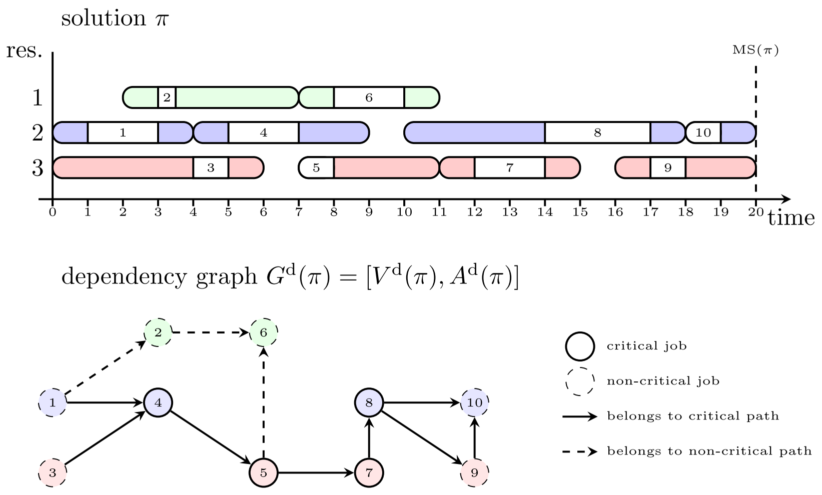

solution π
res.
MS(π)
time
dependency graph $G^{d}(\pi) = [V^{d}(\pi), A^{d}(\pi)]$
critical job
non-critical job
belongs to critical path
belongs to non-critical path

Fig. 3. Example for dependency graph of solution π with critical jobs $J^{c} = \{4, 5, 7, 8\}$.

strategy, a best neighbor of the current solution always is selected as incumbent solution for the next step until no further improvement is possible and thus a local optimum has been reached. Algorithm 5 sketches this procedure.

Algorithm 5 Local Search Procedure.

```
 1: Input: solution $\pi$
 2: $\pi^{best} \leftarrow \pi$
 3: repeat
 4:     $\pi \leftarrow \pi^{best}$
 5:     determine critical jobs $J^{c}(\pi)$ by breadth-first search of dependency graph of $\pi$
 6:     for all $j \in J^{c}(\pi) \mid MS(\pi \ominus j) < MS(\pi^{best})$ do
 7:         $\pi' \leftarrow$ greedily insert $j$ into $\pi \ominus j$ by trying all alternative positions
 8:         if $MS(\pi') < MS(\pi^{best})$ then
 9:             $\pi^{best} \leftarrow \pi'$
10:         end if
11:     end for
12: until $MS(\pi^{best}) = MS(\pi)$
13: return $\pi^{best}$
```

Each insertion neighborhood is efficiently searched by only considering job removals that actually may improve the makespan. To this end, let us consider the dependency graph $G^{d}(\pi) = [V^{d}(\pi), A^{d}(\pi)]$ of solution π whose nodes $V^{d}(\pi)$ correspond to the jobs J and whose arcs $A^{d}(\pi)$ represent the dependencies in such a way that there is an arc (j, j') between any two jobs $j, j' \in J$, $j \neq j'$ iff job j' starts to use some resource immediately after job j has released this resource. All the paths in this dependency graph from any scheduled job starting at time point zero to any job finishing last are so-called *critical paths*, and together, they define the makespan. Now, observe that only the removal of some node (job) lying on *all* these critical paths will yield an immediate reduction of the makespan, and only these jobs are therefore of interest to find a better solution within the insertion neighborhood. We call these jobs *critical jobs* $J^{c}(\pi)$ and determine them by breadth-first search of the dependency graph in time $O(nm)$. Fig. 3 shows an example of such a dependence graph, for more information about critical paths we refer to [21]. Moreover, only those critical jobs whose removal results in a partial solution with a makespan that is lower than the makespan of the current best solution are considered for greedy insertion at a best alternative position in order to determine a best neighbor π^{best}. In preliminary experiments, we also tested an exchange neighborhood in which pairs of jobs are swapped as an alternative or in addition to the insertion neighborhood. However, it turned out that the evaluation of the exchange neighborhood is too expensive regarding running time such that its application usually does not pay off.

If the LS procedure could improve the solution obtained from diving/BS then the solution is injected into the A* search graph to possibly benefit from the respective nodes and NDT records. Furthermore we thereby want to guide the A* search better into more promising areas of the search space so that A* is able to find faster an optimal solution or to return with a smaller optimality gap in case of a termination due to the time or memory limit, and we want to avoid expanding nodes a second time. Algorithm 6 shows the details of the injection operation. Let π^{imp} be the improved solution. The nodes are expanded along the improved solution π^{imp}, starting at the root node with the set of already scheduled jobs $\hat{J} = \emptyset$ and the corresponding NDT record with the vector of earliest availability times $\vec{t}_0 = \vec{0}$. Note, that we assume that this root node/NDT pair is already created by the A* algorithm before Algorithm 6 is executed. If the root node/NDT pair is not yet expanded then we expand it. Starting with $k = 1$, job π^{imp}_k is dealt with by setting $\hat{J} \leftarrow \hat{J} \cup \{\pi^{imp}_k\}$ and computing $\vec{t}_k$ from $\vec{t}_{k-1}$

Algorithm 6 Solution Injection.

```
1: Input: open list $Q$, improved solution $\pi^{\text{imp}}$
2: initialize $\hat{J} \leftarrow \emptyset$, $\vec{t}_0 \leftarrow \vec{0}$
3: retrieve starting node $x$ with $\hat{J}(x) = \emptyset$
4: get NDT record $\eta = (\vec{t}, j^{\text{last}}, \vec{f}) \in \text{NDTs}(x)$ s.t. $\vec{t} = \vec{0}$
5: for $k = 1 \ldots n$ do
6:     if $(\hat{J}, \eta)$ is not expanded then
7:         $E \leftarrow$ expand($(\hat{J}, \eta)$, $\pi^{\text{imp}}$, diving=false)        ▷ (see Algorithm 4)
8:         Q.insert($E$)
9:     end if
10:    $\hat{J} \leftarrow \hat{J} \cup \{\pi_k^{\text{imp}}\}$ and calculate $\vec{t}_k$ from $\vec{t}_{k-1}$ and job $\pi_k^{\text{imp}}$
11:    retrieve node $x$ with $\hat{J}(x) = \hat{J}$
12:    get NDT record $\eta = (\vec{t}, j^{\text{last}}, \vec{f}') \in \text{NDTs}(x)$ s.t. $\vec{t} = \vec{t}_k \vee \vec{t} \lhd \vec{t}_k$
13: end for
```

and π_k^{imp}. Moreover, node x with $\hat{J}(x) = \hat{J}$ is retrieved and, among the NDT records of x, we search for an NDT record η whose earliest availability times are equal to $\vec{t}_k$. If such an NDT record does not exist, there must be at least one dominant NDT record whose earliest availability times dominate $\vec{t}_k$. Therefore we select one such dominant NDT record η with earliest availability time $\vec{t}$ and set $\vec{t}_k \leftarrow \vec{t}$. If the (node, NDT record) pair (x, η) is not yet expanded then we expand it by considering all possible extensions and keep only new and non-dominated (node, NDT record) pairs, which are also inserted into the open list Q. Afterwards, k is incremented and the next job in solution π^{imp} is considered. This procedure is repeated until the last job of solution π^{imp}—that is, π_n^{imp}—has been dealt with. Preliminary tests show that injecting improved solutions from LS back into the A* search graph, leads to an overall performance boost of the A* search. In particular for larger instances, A* benefits from the fact that the search is led faster to promising areas and avoids the expansion of nodes a second time so that on average smaller optimality gaps can be obtained faster.

In the following, we denote the A* algorithm variant with the embedded BS and LS as A*+BS+LS.

6. Mixed integer programming formulation

For comparison purposes, we consider the following position-based MIP formulation from [12], which models solutions to the JSOCMSR problem in terms of permutations of all jobs. Index $i \in \{1, \ldots, n\}$ refers to position i in a permutation. Decision variables $x_{j,i} \in \{0, 1\}$, for all $j \in J$ and $i \in \{1, \ldots, n\}$, are set to one iff job j is assigned to position i in the permutation. Variables $s_i \geq 0$ represent the starting time of the jobs scheduled at each position $i = 1, \ldots, n$ in the permutation. Finally, MS ≥ 0 is the makespan variable to be minimized.

$$\min \text{MS} \tag{19}$$

$$\sum_{j \in J} x_{j,i} = 1 \qquad i = 1, \ldots, n \tag{20}$$

$$\sum_{i=1}^{n} x_{j,i} = 1 \qquad j \in J \tag{21}$$

$$s_i + \sum_{j \in J} x_{j,i} \cdot p_j \leq \text{MS} \qquad i = 1, \ldots, n \tag{22}$$

$$s_1 = 0 \tag{23}$$

$$s_i + \sum_{j \in J} x_{j,i} \cdot p_j^{\text{pre}} \geq s_{i-1} + \sum_{j \in J} x_{j,i-1} \cdot (p_j^{\text{pre}} + p_j^0) \qquad i = 2, \ldots, n \tag{24}$$

$$s_{i'} - s_i + \sum_{j \in J_r} x_{j,i'}(M + p_j) + \sum_{j \in J_r} x_{j,i} M \leq 2M \qquad i = 2, \ldots, n,\ i' = 1, \ldots, i-1,\ r \in R \tag{25}$$

$$x_{j,i} \in \{0, 1\} \qquad j \in J,\ i = 1, \ldots, n \tag{26}$$

$$s_i \geq 0 \qquad i = 1, \ldots, n \tag{27}$$

$$\text{MS} \geq 0 \tag{28}$$

Equations (20) ensure that exactly one job is assigned to the i-th position of the permutation and (21) ensure that each job is assigned to exactly one position. The makespan is determined by inequalities (22). Equation (23) sets the starting time of the first job in the permutation to zero, and the remaining two sets of inequalities make sure that no resource is used by more than one job at a time. Hereby, inequalities (24) take care of the common resource 0, while (25) consider the secondary resources. The Big-M constant in these latter inequalities is set to the makespan obtained by LLBH.

7. Constraint programming formulation

The CP model proposed in the following for the JSOCMSR problem is also used for comparison purposes. It makes use of so-called *interval variables* which represent intervals of time and are a specific feature of ILOG CP Optimizer.[1] For each job $j \in J$ we use two such interval variables: (1) x_j, indicating the time interval during which resource q_j is consumed (p_j units of time), and (2) x_j^0, indicating the time interval during which the common resource 0 is consumed (p_j^0 units of time) by job j. One feature of ILOG CP interval variables is that they can be absent or present in the CP model to consider also scenarios where for instance not all jobs have to be scheduled. Since in our case all jobs needs to be scheduled to get a feasible solution all used interval variables are present in the CP model. The CP model is given by

$$\min \max_{j \in J} \mathrm{end}(x_j) \tag{29a}$$

$$\mathrm{startAtStart}(x_j, x_j^0, p_j^{\mathrm{pre}}) \qquad \forall j \in J \tag{29b}$$

$$\mathrm{noOverlap}(\{x_j \mid j \in J_r\}) \qquad \forall r \in R \tag{29c}$$

$$\mathrm{noOverlap}(\{x_j^0 \mid j \in J\}) \tag{29d}$$

$$x_j : \text{interval variable of size } p_j \quad \forall j \in J \tag{29e}$$

$$x_j^0 : \text{interval variable of size } p_j^0 \quad \forall j \in J \tag{29f}$$

where start(x_j) and end(x_j) represents the start time and the end time of interval x_j, respectively. Constraints (29b) ensure that each job $j \in J$ consumes the common resources 0 exactly p_j^{pre} units of time after the starting time of j using the ILOG CP constraint startAtStart. Constraints (29c) guarantee that each secondary resource $r \in R$ is not used more than once at the same time, whereas Constraint (29d) ensures that the common resource 0 is not used more than once at the same time. This is done using the ILOG CP constraint noOverlap which ensures that all intervals in a given set of interval variables are pairwise non-overlapping.

8. Experimental evaluation

The proposed approaches were implemented in C++ using G++ 5.4.1 for compilation. All tests were performed on a cluster of machines with Intel Xeon E5-2640 v4 processors with 2.40 GHz in single-threaded mode and 15 GB RAM. The CP model from Section 7 was solved with ILOG CP Optimizer 12.7 whereas the MIP model from Section 6 was solved with CPLEX 12.7.

This section is organized as follows. First, the benchmark instances that are used are described and the tuning experiments are summarized. Then, Section 8.3 presents the impact of different algorithmic components used within our A* framework on the solution quality. The anytime behavior of our A* in conjunction with the different options for the lower bound calculation are analyzed in Section 8.5, the improvements of the strengthened lower bound MS$^{\mathrm{LB2}}$ over MS$^{\mathrm{LB0}}$ are studied in Section 8.6, further, the numbers of considered NDT records in our runs (i.e., essentially the sizes of our state graphs) are discussed in Section 8.7. Finally, Section 8.8 presents detailed main results for pure A*, LLBH, A*+BS+LS, the CP approaches with and without using LLBH as search guidance—henceforth denoted by CP+LLBH/ILOG and CP/ILOG, respectively—, the MIP approach, henceforth denoted by MIP/CPLEX, and the weighted A*-based anytime algorithm ARA*.

8.1. Benchmark instances

In our preliminary work [12] we created two non-trivial sets of random instances to test the presented algorithms. Basic characteristics of these instances are roughly inspired from the particle therapy patient scheduling scenario. Each of these sets consists of 50 instances for each combination of $n \in \{10, 20, 50, 100, 200, 500, 1000\}$ jobs and $m \in \{2, 3, 5\}$ secondary resources. As pointed out in Section 1 there are typically only two to four treatment rooms available in the particle therapy scenario, and we therefore do not consider here larger numbers of secondary resources than five. The difference between the two benchmark sets lies in the workload concerning the secondary resources. The first set B is characterized by a *balanced* (B) workload over all resources from R. This was achieved by sampling the secondary resource q_j for each job $j \in J$ uniformly at random from the discrete distribution $\mathcal{U}\{1, m\}$. The second set S has a *skewed* (S) workload. For the S-instances from [12] this was achieved by assigning to resource m a probability twice as high as the probabilities for the remaining secondary resources 1 to $m-1$. This scheme, however, has the disadvantage that for $m > 2$ the common resource tends to become the sole bottleneck, making the instances rather easy to solve. In this work we therefore replace the set of S-instances from [12] by a new set generated with a probability of 0.5 for resource m and a probability of $1/(2m-2)$ for each of the remaining secondary resources. In this way the expected workload on the common resource roughly corresponds to the expected workload of the dominant secondary resource m also for $m > 2$. In addition to these changes, we also

[1] https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-cp-optimizer.

extend both sets B and S with instances with $n = 2000$ jobs. The preprocessing times p_j^{pre} and postprocessing times p_j^{post} were sampled from $\mathcal{U}\{0, 1000\}$ for both instance sets, while times p_j^0 were sampled from $\mathcal{U}\{1, 1000\}$ in case of set B and from $\mathcal{U}\{1, 2500\}$ in case of set S.

Finally note that in [12] we made use of a third set P of instances derived from a real-world particle therapy patient scheduling scenario [20], which consists of 699 instances. However, this instance set consists to a large extent of instances with a similar structure as the balanced instances of type B and of a few instances with a similar structure as the skewed instances of type S. For the sake of completeness we nevertheless show the main results obtained for those set P instances in the main result table in Section 8.8. There, we partitioned the whole set of instances into groups with up to 10, 11 to 20, 21 to 50, and 51 to 100 jobs with 51, 39, 207 and 402 instances, respectively. All these instances use $m = 3$ secondary resources. All instance sets are available from https://www.ac.tuwien.ac.at/research/problem-instances/.

8.2. Parameter tuning

The parameters of our hybrid A*+BS+LS algorithm were tuned for the goal to obtain as good solutions as possible within a CPU time limit of 900 s per run using the automatic parameter configuration tool irace [16] (version 2.1). We distinguish between two different instance sizes: (1) instances with up to $n = 500$ jobs and (2) instances with more jobs. For these we created two independent tuning instance sets, respectively. Both sets contain only instances of type S, because it turned out that these instances are much more difficult to be solved than instances of type B. The first tuning set $T_{\leq 500}$ consists of 10 instances for each combination of $n \in \{10, 20, 50, 100, 200, 500\}$ jobs and $m \in \{2, 3, 5\}$ resources, whereas the second tuning set ($T_{>500}$) consists of 10 instances for each combination of $n \in \{1000, 2000\}$ jobs and $m \in \{2, 3, 5\}$ resources. We tuned four parameters with irace: (1) the lower bound to be used from $\{\text{MS}^{\text{LB0}}, \text{MS}^{\text{LB1}}, \text{MS}^{\text{LB2}}, (\text{MS}^{\text{LB0}}, \text{MS}^{\text{LB2}})\}$ where $(\text{MS}^{\text{LB0}}, \text{MS}^{\text{LB2}})$ denotes the interleaved combination of MS^{LB0} and MS^{LB2} as described in Section 4.2, (2) the usage of LS (on or off), (3) the iteration interval for applying BS $\delta \in \{1, 100, 200, 500, 1\text{k}, 2\text{k}, 5\text{k}, 10\text{k}, 20\text{k}, 50\text{k}, 100\text{k}, 200\text{k}\}$ and (4) the beam width $k_{\text{bw}} \in \{1, 2, 5, 10, 20, 50, 100, 200, 500\}$. For the tuning set $T_{>500}$ we use the different domain $\{1, 2, 3, 4, 5, 6, 7, 8\}$ for the beam width k_{bw} since larger values sometimes do not allow a single BS to be completed within our time limit. Also remember that the case $k_{\text{bw}} = 1$ corresponds to simple diving. The irace tool was applied with a budget of 7000 runs to tuning set $T_{\leq 500}$ and with a budget of 4000 runs to tuning set $T_{>500}$. In this way, the following configurations were obtained by irace: $\mathcal{C}_{\leq 500} = \{(\text{MS}^{\text{LB0}}, \text{MS}^{\text{LB2}}), \text{LS turned on}, \delta = 1000, k_{\text{bw}} = 200\}$ and $\mathcal{C}_{>500} = \{(\text{MS}^{\text{LB0}}, \text{MS}^{\text{LB2}}), \text{LS turned on}, \delta = 100, k_{\text{bw}} = 8\}$, respectively. These results already indicate that the improved lower bound calculations, our advanced diving by BS, as well as the LS procedure are in practice indeed advantageous.

8.3. Analysis of algorithmic components

In this section, the impact of different algorithmic components of the A* framework—in particular BS and LS—is analyzed. For this purpose we compare first LLBH and a standalone variant of BS, both with LS and without. The variants with LS are denoted in the following by LLBH+LS and BS+LS, respectively. Fig. 4 shows results obtained for middle-sized instances of types B and S with $n \in \{100, 200, 500\}$ jobs and $m = 3$ secondary resources. The diagrams on the top present average optimality gaps of obtained solutions π, which are calculated as $100\% \cdot (\text{MS}(\pi) - \text{LB})/\text{LB}$, where LB corresponds to the strengthened lower bound MS^{LB2} according to Eq. (8). The diagrams on the bottom show corresponding average computation times. Note that LLBH described in Section 4 actually is just the special case of BS with beam width $k_{\text{bw}} = 1$. For the standalone BS a beam width of $k_{\text{bw}} = 200$ was used. As expected, LLBH is fastest, however it also yields the highest average optimality gaps in all considered cases compared to the other considered approaches. Conversely, the BS+LS approach leads to the smallest obtained average optimality gaps with the largest obtained average computation times. Remarkable is that BS with or without LS applied always leads to significantly smaller average optimality gaps than LLBH or LLBH+LS. However, these BS variants also require more computation time. Concerning LS, in particular for larger skewed instances of type S, its application has a substantial impact on the obtained average optimality gaps.

Next, we repeat the comparison of the four different heuristics LLBH, LLBH+LS, BS, and BS+LS, however this time embedded in our A* algorithm. Corresponding results are shown in Fig. 5. In the calculation of the optimality gaps, the lower bounds obtained from the corresponding A* runs are now used as LB. In case of A*+LLBH we set the beam width to $k_{\text{bw}} = 1$ otherwise to $k_{\text{bw}} = 200$. In all cases a BS iteration is initiated after every $\delta = 1000$ classical A* iterations. Concerning the computation times, we split each into three parts: the first white block on the bottom of the bars corresponds to the average accumulated time needed by BS. If LS is applied then the black block in the middle corresponds to the average accumulated time used by LS. Finally, the top-most remainder of the bar represents the average time consumed by classical A* iterations. Consequently, the whole bar corresponds to the average total running time of the whole algorithm. At the first glance the results show similar relationships as the results in Fig. 4. Again, in all considered cases the best optimality gaps are obtained by A*+BS+LS whereas the worst ones are obtained by A*+LLBH. However, this time A*+LLBH is not always fastest. In particular for skewed instances with $n \in \{100, 200\}$ jobs A*+LLBH requires more time on average compared to all other considered approaches. This has the following reason: Typically the lower bound obtained from the root node of A* is already rather tight, therefore in order to further reduce the optimality gap it is more important to find good feasible solutions instead of further improving the current best known lower bound by performing classical A* iterations. As we have observed in Fig. 4, the solution quality achieved by LLBH is on average worse than the solution quality achieved by the

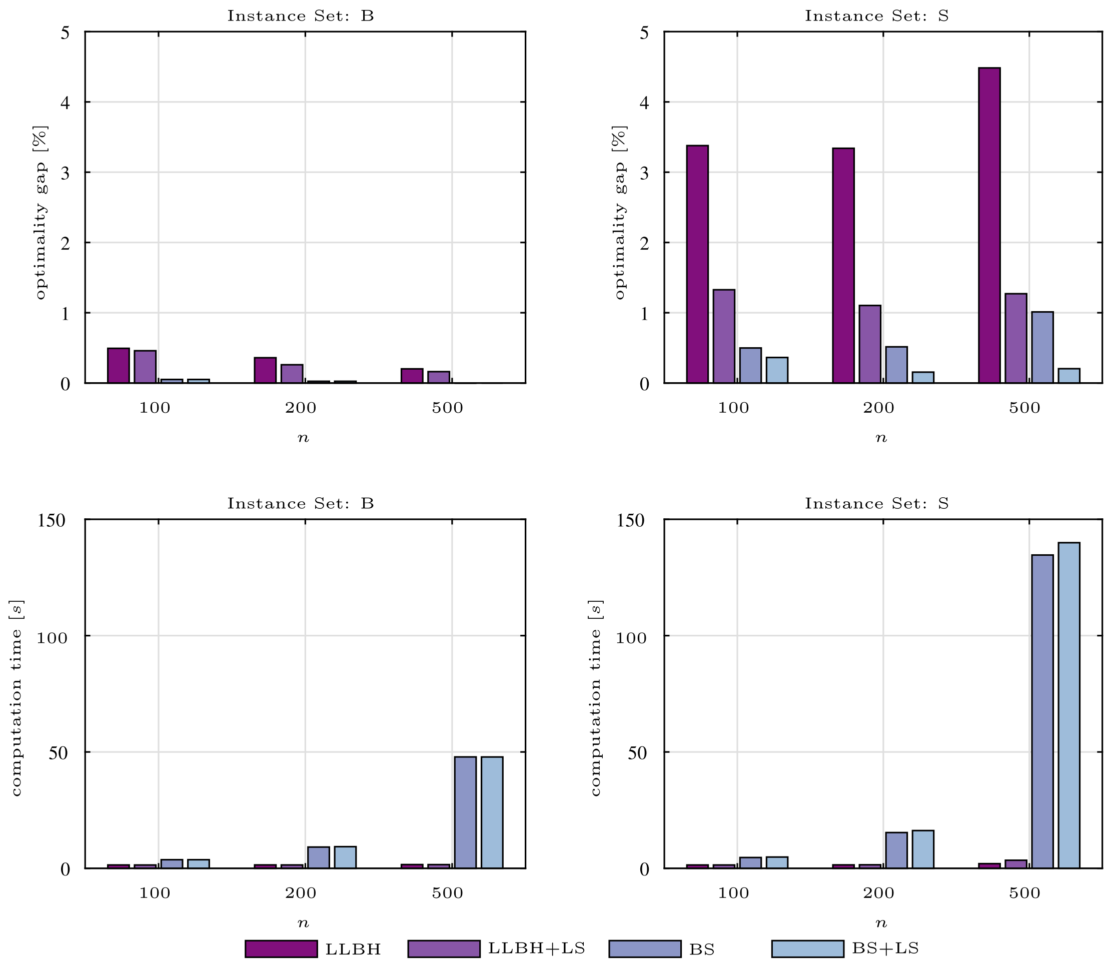


Fig. 4. Comparison of standalone LLBH and BS with and without LS applied afterwards. Results are visualized as barplots which are grouped according to the instance type and number of considered jobs.

other considered approaches. Therefore A*+LLBH has to perform more LLBH iterations until a comparably small optimality gap is achieved. Putting more effort in finding the optimal solution by applying LS afterwards or using BS leads to faster convergence since A* needs less LLBH+LS or BS(+LS) iterations even if a single of such iteration takes more time. For larger instances with $n = 500$ jobs this effect does not pay off anymore. Here, A*+BS and A*+BS+LS are able to return smaller average optimality gaps than A*+LLBH or A*+LLBH+LS, however they also need more time.

8.4. Comparison to anytime pack search

As described in Section 2, *Anytime Pack Search* (APS) from Vadlamudi et al. [32] is an approach similar to our BS-enhanced A* algorithm and based on consecutive BS iterations. In APS terminology the beam width k_{bw} is called *pack size*. There are two major differences: (1) APS does not perform classical A* iterations between consecutive BS iterations—which corresponds in our case to a setting of the diving/BS interval to $\delta = 1$—and (2) if APS starts a new BS iteration then APS selects the best k_{bw} nodes—called *seeds*—from the open list Q instead of just selecting the best node as our A*+BS+LS algorithm does. For comparison purposes we implemented APS based on our combined lower bound. A comparison of A*+BS+LS and APS is shown in Fig. 6. APS was applied to all instances of type B and S, with a pack size corresponding to the beam width k_{bw} of the corresponding A*+BS+LS runs. The results are grouped according to the instance type and number of secondary resources and visualized as boxplots.

These boxplots indicate the following: The obtained results for balanced instances of type B are comparable. The median optimality gaps concerning the subsets of balanced instances of type B are always zero for both algorithms. In particular, for instances with $m = 2$ secondary resources both algorithms could solve all instances to optimality, and for the instances with $m \in \{3, 5\}$ both algorithms could solve almost all instances to optimality, except a few outliers with optimality gaps $< 0.6\%$. In contrast to the results for the instances of type B, the results for the instances of type S show clear differences between A*+BS+LS and APS. Concerning A*+BS+LS, the medians of the obtained optimality gaps for the subsets of instances are 0.330%, 0.004% and 0.001% for 2, 3 and 5 secondary resources, respectively. In contrast, the corresponding medians obtained

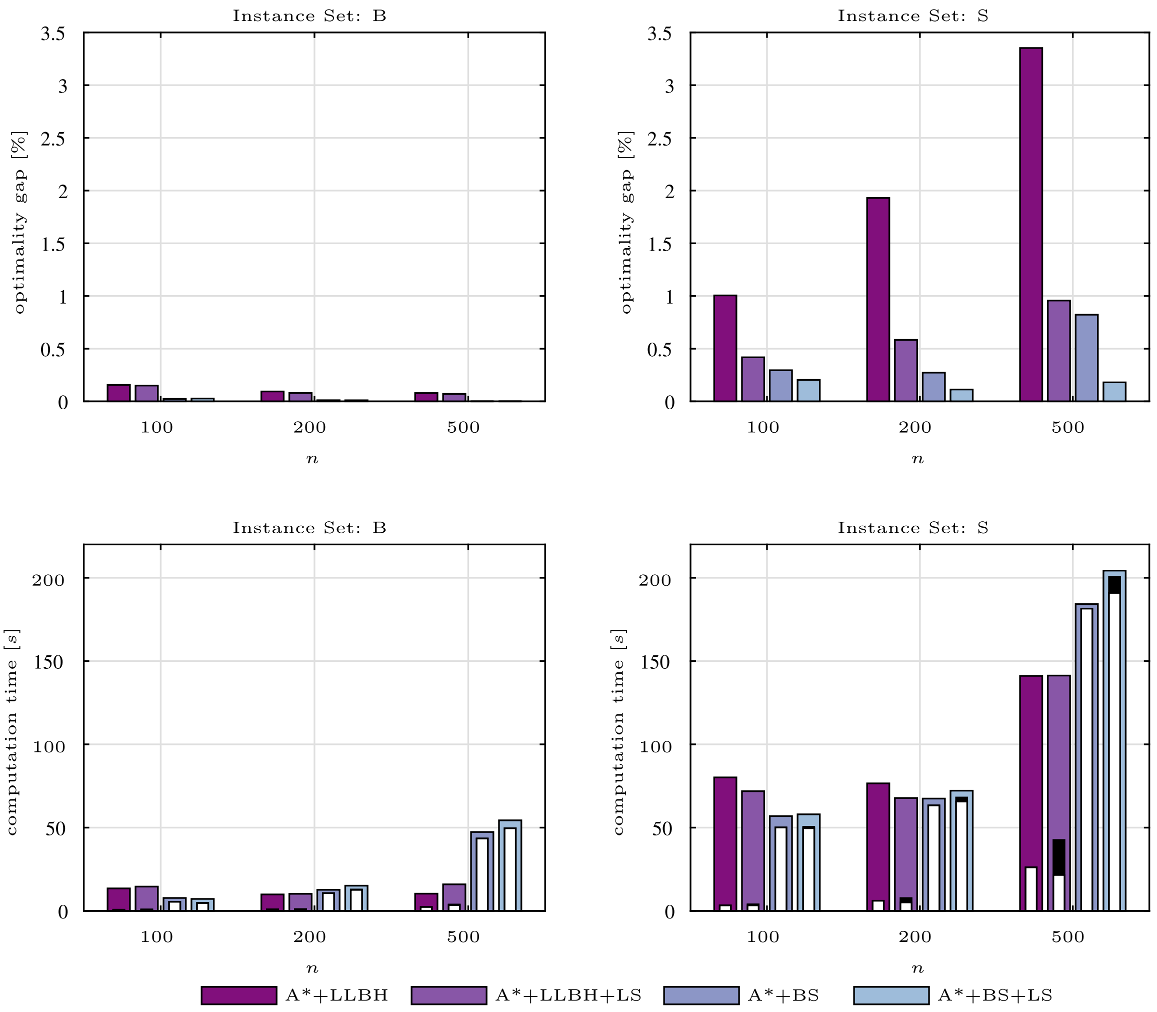


Fig. 5. Comparison of A*+LLBH, A*+LLBH+LS, A*+BS and A*+BS+LS. Results are visualized as barplots which are grouped according to the instance type and number of considered jobs.

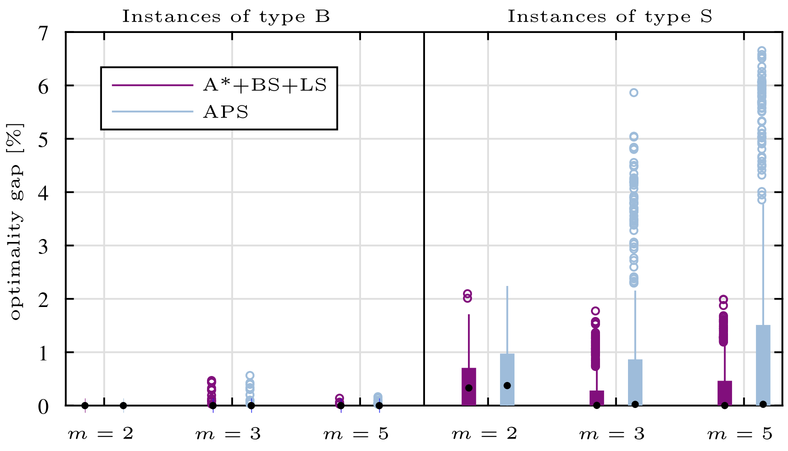


Fig. 6. Comparison of A*+BS+LS and APS. Results are visualized as boxplots which are grouped according to the instance type and number of secondary resources.

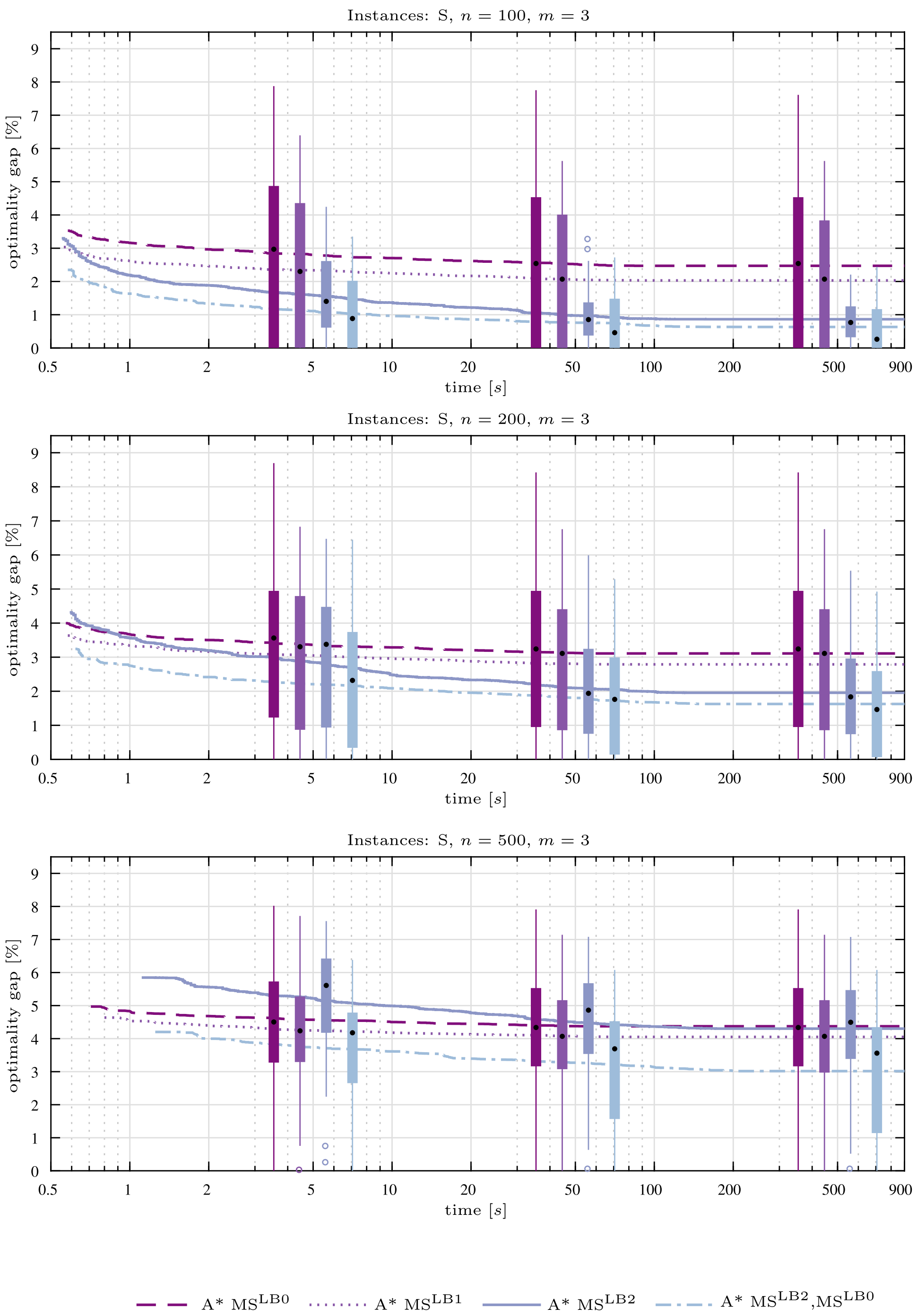


Fig. 7. SQT plots obtained from A* with simple diving ($k_{\mathrm{bw}} = 1$), $\delta = 1000$ and no LS for different choices of the used lower bounds.

from APS are 0.375%, 0.022% and 0.024% for 2, 3 and 5 secondary resources, respectively. The largest optimality gaps obtained from A*+BS+LS are $< 2.1\%$ for all skewed instances of type S, whereas APS yielded a substantial number of times larger gaps of up to 6.7%, in particular for $m = 5$ secondary resources.

8.5. *Anytime behavior*

In order to study the anytime-behavior of the A* variants, Figs. 7 and 8 show *Solution Quality over Time* (SQT) plots for the middle-sized instances of type S (skewed) with $n \in \{100, 200, 500\}$ jobs and $m = 3$ secondary resources. For this purpose, the current optimality gap of A*+BS+LS was taken every 0.01 seconds during each run. At each of the resulting 90,000

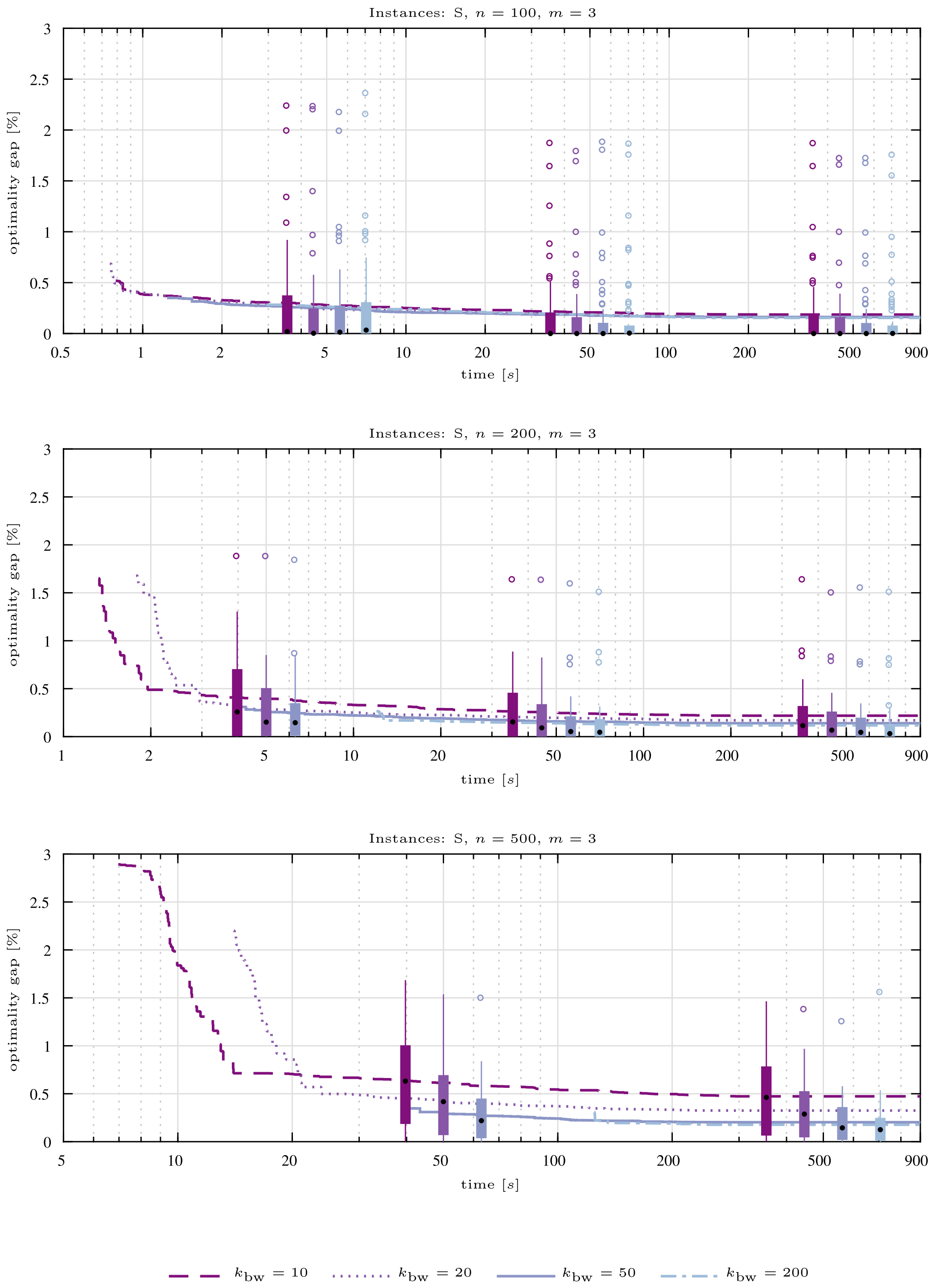


Fig. 8. SQT plots obtained from A*+BS+LS for different choices of the used beam widths. The algorithm was run by using the combined lower bound of $\mathrm{MS}^{\mathrm{LB0}}$ and $\mathrm{MS}^{\mathrm{LB2}}$ and with $\delta = 1000$ and LS turned on.

time points (remember our time limit of 900 s), the average optimality gap for the 50 corresponding problem instances is shown. In addition, in order to visualize the variance, boxplots are shown for time points 5, 50 and 500 seconds. We skip here results showing the anytime behavior of balanced instances if type B, since in most cases the first solution obtained by BS+LS is due to the excellent search guidance of our lower bounds already an optimal solution so that there is no further improvement.

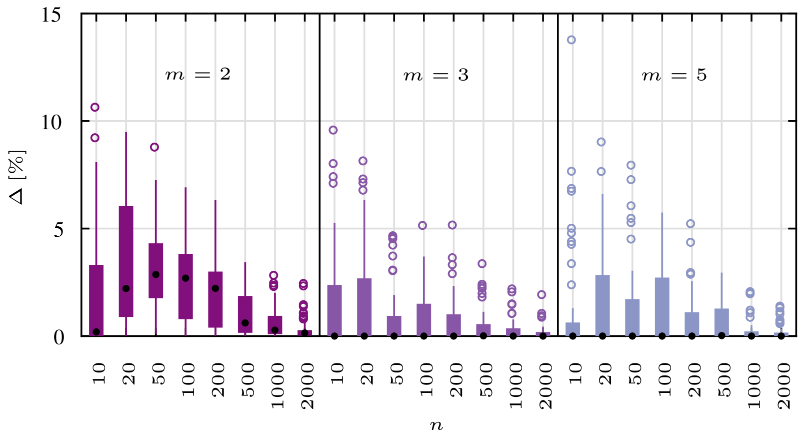


Fig. 9. Relative improvement of bound MS^{LB2} over bound MS^{LB0} in the context of the instances of type S (skewed).

In Fig. 7, the results are shown for four A* variants that differ in the used lower bound: MS^{LB0}, MS^{LB1}, MS^{LB2} and the interleaved combination of MS^{LB0} and MS^{LB2}. The A* algorithm uses just simple diving ($k_{bw} = 1$) every $\delta = 1000$ iterations, and heuristic solutions are here *not* further improved by LS. It can be observed that, for all three values of n, the choice of the combined lower bounds MS^{LB0} and MS^{LB2} almost always provides the smallest average optimality gap at the end of a run.

In Fig. 8 we show SQT plots of A*+BS+LS runs where the combined lower bounds MS^{LB0} and MS^{LB2} are used and BS is started every $\delta = 1000$ iterations with the four different beam widths $k_{bw} \in \{10, 20, 50, 200\}$. Each obtained solution from BS is further improved by LS. The smallest average optimality gaps are obtained from A*+BS+LS when the largest beam width with $k_{bw} = 200$ is used for all three values of n. Furthermore it can be observed that, with increasing beam width, we obtain smaller and smaller average optimality gaps at the end of a run. Overall the SQT plots show a continuous decrease of the optimality gaps over time. Finally, note that in case of the high beam width $k_{bw} = 200$—for instances with $n = 500$ jobs—A*+BS+LS took about 125 seconds until the first solution is obtained. To counteract this initial long waiting time for a first feasible solution, using a smaller beam width for the very first BS application would be an obvious possibility.

8.6. *Comparison of lower bounds* MS^{LB2} *and* MS^{LB0}

We finally aim to quantitatively evaluate the improvement obtained by lower bound MS^{LB2} from Eq. (8) in comparison to the basic lower bound MS^{LB0} from Eq. (4). To this end, we computed both lower bounds for all instances of our benchmark sets and consider the relative improvement $\Delta = (MS^{LB2} - MS^{LB0})/MS^{LB0}$.

In case of the balanced instances of type B, these differences are always zero, thus, no benefits could be observed. The reason for this is that instances of type B are created in such a way that it is likely that the assumption that all jobs J_r which require a secondary resource $r \in R$ can be scheduled consecutively one after the other without any gaps holds. Hence, in such cases the basic lower bound MS^{LB0} will be the same as MS^{LB2}, see Section 3.3 for further details. For the more difficult skewed instances (type S), however, the improvements are sometimes substantial. Fig. 9 shows them as boxplots grouped according to the number of secondary resources m and the number of jobs n. It can be observed that these relative improvements are strongest for the skewed instances with only two secondary resources and in particular just a moderate number of jobs. There they are, however, also most needed: These are generally the instances that are most difficult to solve for our A* variants in terms of the remaining optimality gap, and we have already seen in the SQT plots of Section 8.5 that the usage of the strengthened lower bound boosts the performance significantly.

8.7. *Number of considered NDT records*

Each bar in Fig. 10 shows the average number of NDT records considered by A*+BS+LS, over all 50 instances (per instance type) and using the parameter setting as determined by irace. In essence, these bars show the size of the considered part of the state graph. Each bar is split into a lower part and an upper part. The lower part presents the average number of created NDT records that were never dominated by another NDT record. Thus, this average number of created NDT records essentially is also proportional to the memory usage of A*. The upper part shows the average number of NDT records that were either dominated by another NDT record, or pruned since the lower bound was greater than (or equal to) the makespan of the current incumbent solution. Note that these records are either immediately discarded at the time of their creation or removed during the A*+BS+LS run since they got dominated by another NDT record.

First of all, these plots show that the skewed instances of set S are much more demanding concerning the total number of created NDT records, especially those instances with only two external resources. Clearly, this can be explained with the lower bound calculation being not that tight for these instances as for the balanced set. The number of NDT records for the

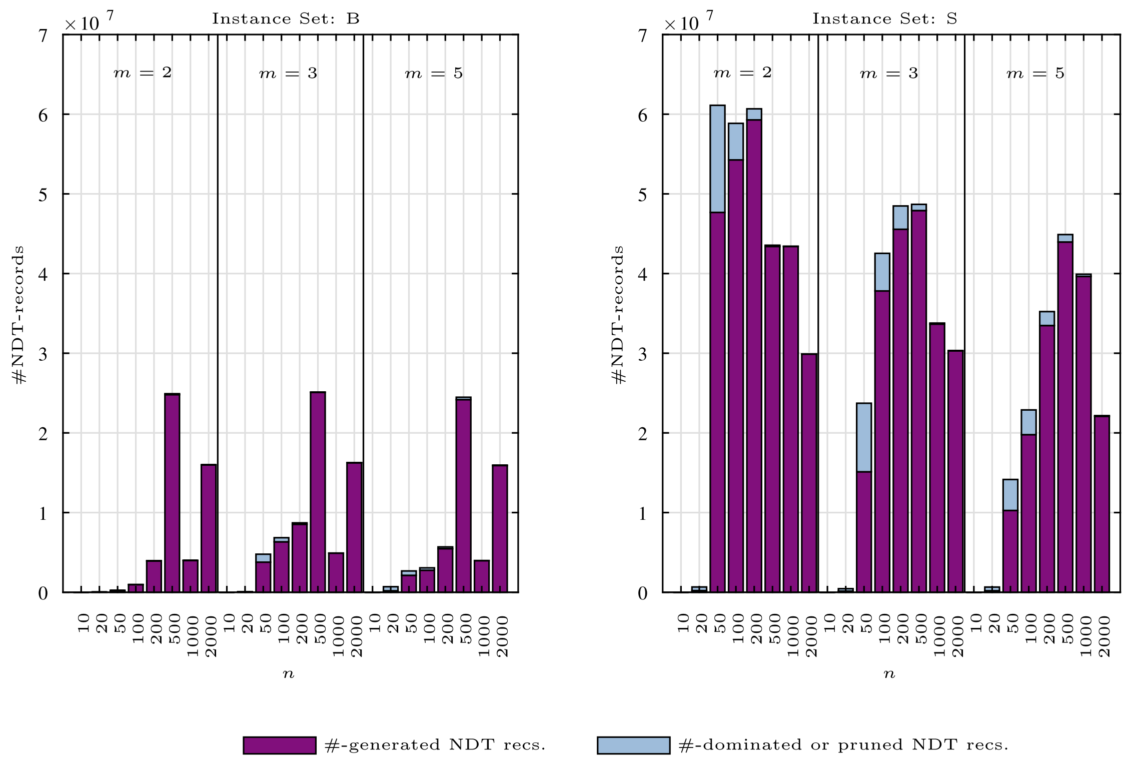


Fig. 10. Number of considered NDT records during the runs of A*+BS+LS, over all instances of type B and S with the parameter setting as determined by irace.

skewed instances has its peak for $n = 50$ jobs and $m = 2$ secondary resources and generally decreases again for instances with more jobs, since fewer node expansions are possible within our CPU time limit of 900 s. Overall, there are instances for which A*+BS+LS explores more than $6 \cdot 10^7$ NDT records.

8.8. Comparison of LLBH, A+BS+LS, CP, MIP, and ARA**

Table 1 presents the aggregated results for each combination of instance type and the different numbers of jobs and secondary resources for LLBH, A*+BS+LS, the CP approach, the MIP approach, pure A* and ARA*. Concerning CP we tested: (1) the CP ILOG solver with default search heuristic—denoted as CP/ILOG—and (2) CP ILOG solver with a custom search heuristic denoted as CP+LLBH/ILOG, which, similar to LLBH, uses the combined lower bound $\mathrm{MS}^{\mathrm{LB2}}$ and $\mathrm{MS}^{\mathrm{LB0}}$ to decide which job should be selected and appended to a current fixed partial schedule first.

In order to compare our A*+BS+LS algorithm also to another state-of-the-art A*-based anytime algorithm we implemented ARA* [14] (see also Section 2, where ARA* was already discussed). ARA* makes use of an inflated evaluation function $f(x) = g(x) + \varepsilon \cdot h(x)$, where ε is the approximation ratio. The algorithm has two parameters: $\varepsilon_{\text{init}} > 1$ and $\varepsilon_{\text{step}} > 0$. Initially, ε is set to $\varepsilon_{\text{init}}$ such that the first heuristic solution that is found is sub-optimal by a factor of at most $\varepsilon_{\text{init}}$. The quality of subsequently encountered heuristic solutions is forced to increase by continuously decreasing ε by step size $\varepsilon_{\text{step}}$. If enough time is given for ε to finally reach value one, ARA* has reached a proven optimal solution. To achieve a reasonable anytime behavior on our benchmark instances we tested different values for $\varepsilon_{\text{init}}$ and $\varepsilon_{\text{step}}$. According to these preliminary experiments we set $\varepsilon_{\text{init}} = 1.5$ and the step size $\varepsilon_{\text{step}} = 0.01$. Moreover, our implementation makes use of the inflated version of evaluation vector $\vec{f}$ with the combined lower bound $\mathrm{MS}^{\mathrm{LB2}}$ and $\mathrm{MS}^{\mathrm{LB0}}$.

Columns %-opt of Table 1 state the percentage of instances that were solved to proven optimality. Columns %-gap list average optimality gaps of final solutions π, which are calculated by $100 \cdot (\mathrm{MS}(\pi) - \mathrm{LB})/\mathrm{LB}$, where LB is the largest lower bound returned from A*+BS+LS, pure A*, CPLEX or from ILOG CP Optimizer. Columns $\sigma_{\text{%-gap}}$ provide the corresponding standard deviations. Columns t[s] show the median computation times in seconds. The last row of each instance type shows the average percentage of instances that were solved to proven optimality, the average optimality gap as well as the corresponding average standard deviations over all instances. Remember that for instances with $n \leq 500$ jobs, A*+BS+LS was run with configuration $\mathcal{C}_{\leq 500}$, and otherwise with configuration $\mathcal{C}_{>500}$. Moreover, LLBH and CP+LLBH/ILOG was applied using the combined lower bounds of $\mathrm{MS}^{\mathrm{LB2}}$ and $\mathrm{MS}^{\mathrm{LB0}}$. Again, the CPU time limit for each run was 900 s. Since the pure A* algorithm does not provide any solution in case of early termination due to the time limit or memory limits, Table 1 just states the percentages of instances that were solved to optimality and the total computation times.

Table 1
Average results of LLBH, A*+BS+LS, CP/ILOG, CP+LLBH/ILOG, MIP/CPLEX, pure A*, and ARA* for the instance set B, S, and P.

Type	n	m	LLBH				A*+BS+LS				CP/ILOG				CP+LLBH/ILOG				MIP/CPLEX				A*		ARA*			
			%-opt	%-gap	$\sigma_{\%\text{-gap}}$	t [s]	%-opt	%-gap	$\sigma_{\%\text{-gap}}$	t [s]	%-opt	%-gap	$\sigma_{\%\text{-gap}}$	t [s]	%-opt	%-gap	$\sigma_{\%\text{-gap}}$	t [s]	%-opt	%-gap	$\sigma_{\%\text{-gap}}$	t [s]	%-opt	t [s]	%-opt	%-gap	$\sigma_{\%\text{-gap}}$	t [s]
B	10	2	90	0.197	0.87	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	40	0.007	0.01	22.6	**100**	0.8	**100**	**0.000**	0.00	<0.1
B	20	2	96	0.074	0.37	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	–	–	–	900.0	**100**	15.8	**100**	**0.000**	0.00	<0.1
B	50	2	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	1.1	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	–	–	–	900.0	0	106.7	**100**	**0.000**	0.00	<0.1
B	100	2	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	2.0	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	–	–	–	900.0	0	90.1	78	0.163	0.40	1.3
B	200	2	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	5.4	**100**	**0.000**	0.00	0.1	**100**	**0.000**	0.00	0.1	–	–	–	900.0	0	74.2	30	0.639	0.56	255.0
B	500	2	**100**	**0.000**	0.00	0.3	**100**	**0.000**	0.00	35.3	**100**	**0.000**	0.00	1.3	**100**	**0.000**	0.00	0.3	–	–	–	900.0	0	73.3	2	1.381	0.55	323.3
B	1000	2	**100**	**0.000**	0.00	1.2	**100**	**0.000**	0.00	8.9	**100**	**0.000**	0.00	9.2	**100**	**0.000**	0.00	1.3	–	–	–	900.0	0	66.7	2	2.191	0.68	434.0
B	2000	2	**100**	**0.000**	0.00	6.1	**100**	**0.000**	0.00	46.3	98	<0.001	<0.01	63.5	**100**	**0.000**	0.00	5.1	–	–	–	900.0	0	48.6	0	3.117	0.66	900.0
B	10	3	76	0.819	1.92	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	48	0.007	0.01	19.2	**100**	0.8	**100**	**0.000**	0.00	<0.1
B	20	3	76	0.865	1.87	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	2	–	–	900.1	**100**	14.2	**100**	**0.000**	0.00	0.1
B	50	3	74	0.702	1.32	<0.1	**96**	**0.017**	0.08	1.1	92	0.068	0.30	<0.1	90	0.091	0.32	<0.1	–	–	–	900.0	0	119.8	2	1.028	0.37	284.6
B	100	3	68	0.625	1.13	<0.1	**92**	**0.021**	0.09	2.0	78	0.226	0.55	4.2	78	0.204	0.47	<0.1	–	–	–	900.0	0	101.4	0	1.436	0.34	179.0
B	200	3	68	0.439	0.83	<0.1	**92**	**0.016**	0.06	5.9	56	0.556	1.12	319.4	80	0.198	0.51	0.1	–	–	–	900.0	0	82.5	0	1.595	0.41	153.5
B	500	3	56	0.265	0.42	0.3	**98**	**<0.001**	<0.01	35.9	20	2.212	1.83	900.0	66	0.162	0.29	0.3	–	–	–	900.0	0	75.2	0	2.109	0.80	205.3
B	1000	3	68	0.154	0.32	1.3	**98**	**0.001**	<0.01	6.1	2	3.094	1.46	899.9	74	0.083	0.20	1.3	–	–	–	900.0	0	62.8	0	2.737	0.78	358.9
B	2000	3	66	0.087	0.17	6.3	**98**	**0.005**	0.04	23.8	0	4.220	1.20	900.0	70	0.075	0.16	5.1	–	–	–	900.0	0	50.4	0	3.229	0.29	767.1
B	10	5	50	2.062	3.14	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	74	0.004	0.01	2.2	**100**	0.8	**100**	**0.000**	0.00	<0.1
B	20	5	32	1.971	2.15	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	44	–	–	900.1	**100**	18.6	**100**	**0.000**	0.00	0.1
B	50	5	40	0.623	0.83	<0.1	96	**0.000**	0.00	1.2	**100**	**0.000**	0.00	0.7	94	0.002	0.02	<0.1	34	–	–	900.1	0	142.0	68	0.154	0.30	15.6
B	100	5	28	0.305	0.36	<0.1	96	**0.000**	0.00	2.2	**100**	**0.000**	0.00	9.5	94	0.006	0.04	<0.1	–	–	–	900.0	0	105.7	24	0.453	0.34	290.5
B	200	5	38	0.149	0.20	<0.1	96	<0.001	<0.01	6.5	**100**	**0.000**	0.00	91.3	98	0.002	0.02	0.1	–	–	–	900.0	0	95.8	0	0.729	0.26	256.5
B	500	5	38	0.053	0.06	0.3	**100**	**0.000**	0.00	42.3	86	<0.001	<0.01	499.7	90	0.007	0.02	0.3	–	–	–	900.0	0	87.0	0	0.738	0.25	221.8
B	1000	5	38	0.020	0.03	1.5	**100**	**0.000**	0.00	7.9	0	0.359	0.12	900.0	98	0.002	0.01	1.3	–	–	–	900.0	0	64.3	0	0.670	0.25	329.7
B	2000	5	48	0.011	0.02	7.4	**100**	**0.000**	0.00	30.4	0	0.478	0.14	900.0	94	<0.001	<0.01	5.4	–	–	–	900.0	0	52.1	0	0.767	0.42	715.1
B			68	0.436	0.73		**98**	**0.003**	0.01		76	0.467	0.28		93	0.035	0.09		–	–	–		25		38	0.964	0.32	
S	10	2	56	0.837	1.34	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	0.2	**100**	**0.000**	0.00	0.1	26	0.008	0.01	9.0	**100**	0.8	**100**	**0.000**	0.00	<0.1
S	20	2	28	1.410	1.60	<0.1	**100**	**0.000**	0.00	0.9	24	0.005	0.04	899.8	14	0.486	0.79	899.8	–	–	–	900.0	**100**	4.3	**100**	**0.000**	0.00	0.3
S	50	2	0	2.595	1.89	<0.1	**40**	0.268	0.38	11.4	0	**0.210**	0.28	899.9	0	2.147	1.59	899.8	–	–	–	900.0	0	152.3	0	2.532	1.16	247.2
S	100	2	0	2.589	1.50	<0.1	**26**	0.367	0.49	44.8	0	**0.323**	0.47	900.0	0	2.350	1.33	899.8	–	–	–	900.0	0	135.5	0	3.849	1.17	234.2
S	200	2	0	2.912	1.30	<0.1	**2**	**0.440**	0.33	65.2	0	0.642	0.51	900.0	0	2.865	1.31	899.9	–	–	–	900.0	0	148.0	0	4.606	1.04	244.5
S	500	2	**0**	3.691	1.05	0.5	**0**	**0.532**	0.18	88.7	**0**	2.736	0.51	900.0	**0**	3.679	1.02	899.9	–	–	–	900.0	**0**	222.5	**0**	4.940	0.85	222.7
S	1000	2	**0**	4.381	0.87	3.2	**0**	**0.725**	0.20	176.8	**0**	4.636	0.43	900.0	**0**	4.365	0.87	900.9	–	–	–	900.0	**0**	306.6	**0**	5.026	0.48	276.8
S	2000	2	**0**	4.976	0.86	22.7	**0**	**0.786**	0.18	252.7	**0**	4.784	0.39	900.0	**0**	4.991	0.84	900.0	–	–	–	900.0	**0**	353.6	**0**	5.294	0.40	574.7
S	10	3	30	1.675	1.91	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	0.1	**100**	**0.000**	0.00	<0.1	56	0.005	0.01	2.2	**100**	0.8	**100**	**0.000**	0.00	<0.1
S	20	3	14	1.790	1.42	<0.1	**100**	**0.000**	0.00	0.8	54	**0.000**	0.00	255.0	42	0.446	0.79	899.3	40	–	–	899.3	**100**	6.0	**100**	**0.000**	0.00	<0.1
S	50	3	8	1.937	1.63	<0.1	**80**	0.053	0.21	1.3	54	**0.035**	0.15	27.7	30	1.580	1.63	899.7	10	–	–	900.0	0	194.3	26	0.936	0.97	339.0
S	100	3	12	2.785	1.83	<0.1	**50**	0.153	0.37	16.5	44	**0.060**	0.15	899.7	20	2.720	1.85	899.9	–	–	–	900.0	0	190.0	8	2.483	1.79	268.8
S	200	3	4	3.221	1.89	0.1	34	**0.117**	0.26	26.4	**36**	0.135	0.21	899.8	10	3.202	1.91	899.8	–	–	–	900.0	0	194.5	4	2.939	1.57	212.1
S	500	3	0	4.127	1.70	0.7	**14**	**0.177**	0.24	121.6	4	1.360	0.76	900.0	4	4.124	1.70	899.9	–	–	–	900.0	0	265.9	0	3.781	0.93	158.9
S	1000	3	0	4.123	1.81	4.2	**2**	**0.621**	0.47	248.0	0	2.872	0.93	900.0	0	4.122	1.81	899.9	–	–	–	900.0	0	337.9	0	3.747	0.47	211.9
S	2000	3	**0**	4.142	1.97	31.3	**0**	**0.701**	0.41	480.2	**0**	4.296	0.97	900.0	**0**	4.141	1.97	900.0	–	–	–	900.0	**0**	435.7	**0**	3.794	0.37	735.9
S	10	5	32	1.695	2.20	<0.1	**100**	**0.000**	0.00	0.8	**100**	**0.000**	0.00	0.0	**100**	**0.000**	0.00	<0.1	68	0.004	0.01	0.9	**100**	0.8	**100**	**0.000**	0.00	<0.1
S	20	5	16	1.832	1.97	<0.1	**100**	**0.000**	0.00	0.8	60	**0.000**	0.00	0.6	46	0.708	1.43	899.2	50	7.886	12.25	581.1	**100**	11.0	**100**	**0.000**	0.00	<0.1
S	50	5	16	2.258	1.94	<0.1	**80**	0.077	0.19	1.4	54	**0.045**	0.14	15.0	24	2.023	1.93	899.8	22	–	–	899.7	0	196.9	38	1.096	1.27	362.3
S	100	5	10	2.578	2.06	<0.1	**64**	0.064	0.18	6.1	48	**0.019**	0.04	899.6	18	2.534	2.06	899.8	–	–	–	900.0	0	219.4	14	2.291	1.78	454.2
S	200	5	2	3.472	1.74	0.1	**34**	0.281	0.49	38.8	28	**0.161**	0.25	900.0	6	3.449	1.76	899.8	–	–	–	900.0	0	243.4	8	3.247	1.57	438.5
S	500	5	2	3.653	1.85	0.9	**16**	**0.347**	0.34	188.6	8	1.229	0.95	899.9	4	3.645	1.86	899.9	–	–	–	900.0	0	384.0	4	3.346	1.24	319.7
S	1000	5	**0**	4.371	1.95	5.9	**0**	**0.702**	0.50	387.3	**0**	2.478	1.11	900.0	**0**	4.371	1.95	900.0	–	–	–	900.0	**0**	423.0	**0**	3.613	0.73	396.4
S	2000	5	**0**	4.460	2.07	52.0	**0**	**0.915**	0.54	789.3	**0**	4.229	1.22	900.0	**0**	4.461	2.07	900.0	–	–	–	900.0	**0**	558.4	**0**	3.631	0.50	900.0
S			9	3.040	1.76		**43**	**0.305**	0.25		40	1.272	0.43		22	2.600	1.35		–	–	–		25		29	2.548	0.76	
P	≤10	3	71	0.464	0.93	<0.1	**100**	**0.000**	0.00	0.9	**100**	**0.000**	0.00	<0.1	**100**	**0.000**	0.00	<0.1	63	**0.000**	0.00	<0.1	**100**	1.0	**100**	**0.000**	0.00	<0.1
P	≤20	3	54	0.567	0.86	<0.1	**100**	**0.000**	0.00	0.9	**100**	**0.000**	0.00	<0.1	97	0.011	0.07	<0.1	59	**0.000**	0.00	63.5	**100**	1.7	**100**	**0.000**	0.00	<0.1
P	≤50	3	42	0.634	0.96	<0.1	**87**	**0.110**	0.35	1.0	86	0.120	0.36	0.3	70	0.374	0.78	<0.1	27	–	–	900.0	10	179.3	52	0.580	0.89	257.8
P	≤100	3	48	0.494	0.85	<0.1	83	0.086	0.27	1.2	**85**	**0.070**	0.23	1.6	67	0.396	0.81	<0.1	4	–	–	900.0	0	153.5	21	1.049	1.06	311.1
P			54	0.540	0.90		**93**	0.049	0.16		**93**	**0.048**	0.15		84	0.195	0.42		28	–	–		53		68	0.407	0.49	

The following observations can be made with respect to the balanced instances of set B. First, A*+BS+LS was able to solve all instances with $m = 2$ secondary resources and more than 96% of the instances with $m > 2$ to proven optimality. In contrast, while CP/ILOG also solves nearly all instances with $m = 2$ secondary resources to optimality, its performance degrades significantly with growing n and m. In particular, instances with $m = 3$ secondary resources are hard to solve for CP/ILOG, since those instances are created in a way such that the expected workload of the common resource as well as the expected workloads of the secondary resource are equal, which makes instances hard to solve, whereas for instances with $m \in \{2, 5\}$ secondary resources the common resource becomes the sole bottleneck. Moreover, the average optimality gaps obtained by A*+BS+LS are always smaller than (or equal to) the ones of CP/ILOG. The largest average optimality gap of A*+BS+LS is only 0.021% while that of CP/ILOG is 4.220%. Clearly, the most important ingredient for this success of A*+BS+LS is the excellent guidance of the search by our lower bound calculation so that even for instances with $m = 3$ secondary resources excellent results are obtained. This is documented by the fact that even LLBH yields comparably good results with average optimality gaps of no more than 2.062%. Due to this excellent search guidance it can further be observed that CP+LLBH/ILOG can solve significantly more instances to optimality as CP/ILOG. Over all balanced instances of set B, CP+LLBH/ILOG is able to solve on average 93% of the instances to proven optimality whereas CP/ILOG can only solve on average 76% to proven optimality. However, A*+BS+LS can solve on average 98% of the instances to proven optimality. Moreover, CP+LLBH/ILOG is able to provide better average optimality gaps as CP/ILOG for larger instances. Nevertheless, CP+LLBH/ILOG cannot provide better average optimality gaps than A*+BS+LS, since the combination of BS and LS embedded in A* seems superior to the CP approach in combination with LLBH as search heuristic.

The observations concerning the benchmark set S are as follows. First of all, these instances are clearly more difficult to be solved than the instances from set B. In particular, A*+BS+LS was only able to solve instances up to size $n = 20$ consistently to optimality. However, it still holds that the optimality gaps obtained by A*+BS+LS are, in most cases, smaller than (or equal to) the ones of CP/ILOG. In those cases in which CP/ILOG returns smaller gaps, A*+BS+LS is usually able to solve more instances to optimality. The average optimality gaps obtained by A*+BS+LS never exceed 0.915%, whereas the ones of CP/ILOG range up to 4.784%. Finally, LLBH again provides reasonable results with optimality gaps of no more than 4.976% in the shortest computation times, although as a pure heuristic, LLBH itself does not accompany its results with any lower bound. For CP+LLBH/ILOG, the picture looks different than it does concerning the balanced instances of type B. For instances in data set S, the CP+LLBH/ILOG approach is not able to solve on average more instances to optimality or to provide better optimality gaps than A*+BS+LS or CP/ILOG. Since, CP+LLBH/ILOG uses LLBH as search guidance it is not surprising that in particular for larger instances the average results are similar to those results obtained from LLBH. However, for small and middle sized instances CP+LLBH/ILOG is able to provide smaller average optimality gaps than LLBH.

Regarding the observations of the benchmark set P: the A*+BS+LS approach, CP/ILOG as well as the pure A* approach could solve all instances with up to 20 jobs to optimality. For instances with 21 to 50 jobs A*+BS+LS is able to return the smallest average optimality gaps and can solve most instances to proven optimality. For instances with more than 51 jobs the results of A*+BS+LS and CP/ILOG dominate, and they are essentially on par concerning optimality gaps and instances solved to proven optimality.

The MIP approach is not even able to solve instances with up to $n = 10$ jobs to optimality and in most cases not even able to derive primal solutions for instances with more than 10 jobs. In general, MIP/CPLEX is not able to yield solutions for instances with more than 200 jobs. A reason for this weak performance are clearly the Big-M constraints and the resulting weak linear programming relaxation of the model.

Regarding the pure A* approach, only instances with up to 20 jobs can be solved to proven optimality. For instances with more than 20 jobs A* runs out of memory for each single instance before the proven optimal solution could be found. In particular for balanced instances of type B where A*+BS+LS is able to solve 98% of the instances to proven optimality, the poor performance of the pure A* algorithm seems surprising at the first glance, since in those cases A* without BS and LS might be expected to require less node expansions due to A*'s optimality condition. However, as already mention in Section 5 this optimality condition holds only under certain conditions. Most importantly, the way how ties are handled can substantially impact the number of expanded nodes. In our case the interleaved lower bounds $\mathrm{MS}^{\mathrm{LB2}}$ and $\mathrm{MS}^{\mathrm{LB0}}$ are rather tight and ties occur in general relatively frequently. At least for the smaller instances, it is therefore in many cases sufficient to find an optimal solution quickly in order to also prove optimality. The A*+BS+LS approach frequently achieves this with its strong BS heuristic using $\mathrm{MS}^{\mathrm{LB2}}$ and $\mathrm{MS}^{\mathrm{LB0}}$ as search guidance, whereas the pure A* algorithm tends in case of the ties towards a breadth-first-search behavior, such that it runs rather soon out of memory. This behavior of the pure A* algorithm can be fixed by adding an additional tie breaking criterion which prefers partial schedules where more jobs are already scheduled over partial schedules with less jobs scheduled as we studied it already in [12]. In conjunction with A*+BS+LS, however, this kind of tie breaking would be counter-productive as it is in fact beneficial to initiate the embedded BS from a more diversified set of initial partial solutions, i.e., starting nodes.

The ARA* approach is only competitive to our A*+BS+LS approach in the context of small instances with up to 20 jobs. Concerning larger problem instances A*+BS+LS is generally able to solve more instances to proven optimality and to provide smaller optimality gaps in those cases in which optimality could not be proven. In accordance with observations in the literature [14], with an initial setting of $\varepsilon = 1.5$ ARA* behaves rather greedily and quickly finds a heuristic solution. However, the quality of this solution is generally worse than the initial solution obtained by A*+BS+LS. This seems to be the case because using BS with the non-inflated evaluation vector $\vec{f}$ as search guidance is more powerful. To find better solutions ARA* decreases ε during the search. In particular, the closer ε is to value one, the more ARA* behaves like the pure A* search. However, as already mentioned above, the behavior of pure A* tends to be similar to breadth-first search. Therefore, in many cases, ARA* runs either out of memory or out of time before solutions of similar quality as those obtained by A*+BS+LS are found.

In summary, we have shown that embedding BS and LS in our A* framework has a clearly positive impact on the quality of the obtained solutions and can sometimes even reduce the computation time in contrast to using A*+LLBH. Furthermore, we analyzed the anytime-behavior of A* by using different lower bounds and different beam widths for the BS. Here it turned out that the combined lower bound $\mathrm{MS}^{\mathrm{LB0}}$ and $\mathrm{MS}^{\mathrm{LB2}}$ almost always leads to the smallest average optimality gaps at the end of a run. Regarding different beam widths we could observe that selecting a higher beam width will usually provide solutions with smaller average optimality gaps. This, however, comes at the cost of a higher waiting time until a first complete solution is obtained by the search.

9. Conclusions and future work

We considered the problem of scheduling a set of jobs where each job requires two resources: a common resource shared by all jobs for part of their processing, and a secondary resource for the whole processing time. The objective is to minimize the makespan. We showed that this problem is NP-hard and that we could derive tight lower bounds for the makespan. These lower bounds are exploited in the fast construction heuristic LLBH and a complete A* search. The A* algorithm features in particular a special state graph structure with nodes holding sets of non-dominated time-records in order to exploit symmetries and to keep the memory consumption reasonable. Since the basic A* algorithm may find a complete solution for large, difficult instances only after significant search, we enhanced the search by switching the search strategy in regular intervals to a BS. Nodes where the lower bound is greater than or equal to the makespan of the so far best solution are pruned. Furthermore, complete solutions undergo a LS based on the insertion neighborhood. In this way excellent heuristic solutions are obtained early and the method exhibits a good anytime behavior while remaining complete.

Our experimental evaluation shows that this A*+BS+LS can solve very large non-trivial instances with up to 2000 jobs either to proven optimality or with typically very small remaining gaps of less than one percent. We observed, however, that the difficulty of problem instances is not so much related to the number of jobs, but primarily determined by the distribution of the workload on the resources. More specifically, if the instances exhibit a skewed workload and the workload of the common resource is similar to the workload of the dominant resource among the secondary resources, then these instances are typically hard to solve. In our experiments we therefore considered such difficult skewed instances as well as easier balanced instances and studied the impact of diverse parameters like the different lower bounds, the beam width, and the usage of the LS. Furthermore we compared our approach to the anytime A* variants APS from [32] and ARA* from [14], the pure A* algorithm and a basic CP model solved by the ILOG CP solver. In particular for large instances A*+BS+LS significantly outperforms the ILOG CP approach and in almost all cases where the ILOG CP solver provides smaller average optimality gaps our approach is able to solve more instances to optimality. In addition we compared our anytime A* algorithm to a position-based MIP model solved by CPLEX, which is, however not competitive at all. Only for small instances was CPLEX able to obtain heuristic solutions.

Further research may consider a closer investigation of different starting strategies for the embedded BS as well as an adaptive tuning of the beam width. In fact, the proposed general A*+BS+LS framework is rather problem-independent, and its applications in other problem domains appears promising. Last but not least, the consideration of extended problem variants would be of high practical interest in domains like patient scheduling for particle therapy. Particularly relevant would be the consideration of time windows, alternative objective functions, and a prize-collecting problem version in which only a subset of the jobs can be scheduled.

Declaration of competing interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Appendix A. Proof of Theorem 3

Proof. We first show that $\mathrm{MS}^{\mathrm{LB0}} \leq \mathrm{MS}^{\mathrm{LB1}} \leq \mathrm{MS}^{\mathrm{LB2}}$. The definition of $\mathrm{MS}^{\mathrm{LB1}}$ differs only by the additive terms h_r^1, $r \in R$ from the definition of $\mathrm{MS}^{\mathrm{LB0}}$. Since these terms are all non-negative it follows that $\mathrm{MS}^{\mathrm{LB0}} \leq \mathrm{MS}^{\mathrm{LB1}}$. For $\mathrm{MS}^{\mathrm{LB1}} \leq \mathrm{MS}^{\mathrm{LB2}}$ it is sufficient to show that $h_r^1 \leq h_r^2$, $\forall r \in R$, since the definitions of $\mathrm{MS}^{\mathrm{LB1}}$ (7) and $\mathrm{MS}^{\mathrm{LB2}}$ (8) differ only in the additional strengthened terms h_r^1 and h_r^2. Consider an arbitrary secondary resource $r \in R$. By the definitions of $\mathrm{MS}^{\mathrm{LB1}}$ and $\mathrm{MS}^{\mathrm{LB2}}$ both terms h_r^1 and h_r^2 become zero if $J = \emptyset$ or $J_r = \emptyset$ or $J \setminus J_r = \emptyset$. Therefore, let us assume that all three sets contain at least one element. Let $S = \{j \in J \setminus J_r \mid p_j^0 > p_{\max}^{\mathrm{prepost}}(J_r)\}$ be the set of jobs which require the common resource 0 longer than $p_{\max}^{\mathrm{prepost}}(J_r)$ and which do not require the secondary resource r. In this case it holds that $h_r^1 = \sum_{j \in S}(p_j^0 - p_{\max}^{\mathrm{prepost}}(J_r))$. Next, we show that Algorithm 2 selects in Line 8 or in Line 13 also each job $j \in S$ for adding the corresponding p_j^0 time to variable h_r^2. At the start of the algorithm $S \subseteq J^0$, since J^0 is initially set to $J \setminus J_r$. At each iteration of the for-loop the job which requires resource 0 the most is selected from set J^0 and removed at the end of the loop. Consider the case that job

$j \in S$ is not selected during the loop. There are only two possibilities in which the loop terminates. Either the if condition becomes true and the algorithm terminates, or $J^0 \neq \emptyset \wedge i \leq k$ does not hold anymore. In the former case assume that at iteration i job j_i is selected from set J^0 and $p^0_{j_i} < g_i$. Since $j_i = \arg\max_{j' \in J^0} p^0_{j'}$ it must also hold that $g_i > p^0_{j_i} \geq p^0_j$. But for job j it must further hold that $p^0_j \geq p^{\text{prepost}}_{\max}(J_r) = g_1 \geq g_i$. Hence, we have a contradiction and the if-condition cannot be fulfilled before all jobs from S are selected. The latter case that $J^0 \neq \emptyset \wedge i \leq k$ becomes false remains. Since job j is not selected yet, $J^0 \neq \emptyset$ must hold. This implies that i must be larger than k for terminating the loop. But then job j is still in J^0 and the corresponding p^0_j will be added to h^2_r at the end of the algorithm. Hence, p^0_j, $j \in S$ will be added either during the execution of the for loop or afterwards at the end of the algorithm. Let G be the set of all time gaps $[g_i]^k_{i=1}$ that are subtracted from h^2_r when a job from S is selected in the for-loop. Then, $h^2_r \geq \sum_{j \in S} p^0_j - \sum_{g \in G} g \geq \sum_{j \in J}(p^0_j - p^{\text{prepost}}_{\max}(J_r)) = \sum_{j \in J} p^0_j - |S| p^{\text{prepost}}_{\max}(J_r) = h^1_r$.

It remains to show that MS^{LB2} is a lower bound for the makespan. Assume that there exists a normalized feasible solution s with $\text{MS}(s) < \text{MS}^{\text{LB2}}$. We have to consider two cases:

- $\text{MS}^{\text{LB0}}_0 \geq \max_{r \in R} \text{MS}^{\text{LB2}}_r$: The lower bound is determined by the basic lower bound MS^{LB0}_0. This case has already been shown to yield a contradiction in the proof of Theorem 2.
- $\text{MS}^{\text{LB0}}_0 < \max_{r \in R} \text{MS}^{\text{LB2}}_r$: The lower bound is determined by the strengthened lower bounds. Let $r \in R$ be a secondary resource that determines $\max_{r \in R} \text{MS}^{\text{LB2}}_r$. We have to distinguish between two cases: (1) $\text{MS}(s) < \text{MS}^{\text{LB0}}_r$ and (2) $\text{MS}^{\text{LB0}}_r \leq \text{MS}(s) < \text{MS}^{\text{LB0}}_r + h^2_r$, where $h^2_r > 0$. In the first case solution s cannot be feasible because the makespan of solution s would be smaller than $\text{MS}^{\text{LB0}}_r = \sum_{r \in J_r} p_j$, which would imply that resource r is used by more than one job at the same time. In the second case we assume that resource r is not used by more than one job at the same time. Since $h^2_r > 0$ there must be jobs which are selected by Algorithm 2 to add the corresponding difference at Line 13 and 8 to h^2_r. Let S be the set of these jobs and $[\sigma_i]^{|S|}_{i=1}$ be the sequence of these jobs ordered according to decreasing times p^0, i.e., $p^0_{\sigma_i} \geq p^0_{\sigma_{i+1}}$ for all $i = 1, \ldots, |S| - 1$. Furthermore, let $[g_i]^k_{i=1}$ be the corresponding sequence of time gaps for resource r. Then $h^2_r = \sum^{|S|}_{i=1} p^0_{\sigma_i} - g_i$ where $g_i = 0$ for $i > k$. Now, consider a normalized partial solution s' where exactly the jobs in J_r are scheduled in the exactly same order as in solution s. Then we can determine time gaps between two consecutive jobs in s' regarding the common resource 0. Let $[\bar{g}_i]^{|J_r|+1}_{i=1}$ be the decreasingly sorted sequence of these time gaps. Note, that each element of this sequence is the sum of one preprocessing and one postprocessing time of jobs in J_r. Except for two jobs, which corresponds to the preprocessing time of the first scheduled job from J_r in s' and the postprocessing time of the last scheduled job from J_r in s'. Since Algorithm 1 determines the sequence of time gaps $[g_i]^k_{i=1}$ by taking at each iteration the maximum pre- and postprocessing times which are not already consumed, it holds that $\bar{g}_i \leq g_i$, $\forall i : 1 \leq i \leq k$. This implies that $h^2_r \leq \sum^{|S|}_{i=1} p^0_{\sigma_i} - \bar{g}_i$ where $\bar{g}_i = 0$ for $i > |J_r| + 1$. Now let us insert the jobs from S into solution s' such that job σ_i is placed between the jobs which are associated with time gap $\bar{g}_i$ and causes a delay of $p^0_{\sigma_i} - \bar{g}_i$, $\forall i : 1 \leq |J_r| + 1$. Jobs σ_i where $i > |J_r| + 1$ are appended to solution s'. Then $\text{MS}^{\text{LB0}}_r + h^2_r \leq \text{MS}^{\text{LB0}}_r + \sum^{|S|}_{i=1} p^0_{\sigma_i} - \bar{g}_i \leq \text{MS}(s')$ where $\bar{g}_i = 0$ for $i > |J_r| + 1$. Note that any different schedule order of jobs in S will cause an increase of the delay and therefore of the makespan of solution s'. This holds in particular if we choose the same schedule order for the jobs in S as in solution s. Therefore $\text{MS}(s') \leq \text{MS}(s)$. This contradicts the assumption that $\text{MS}(s) < \text{MS}^{\text{LB0}}_r + h^2_r$.

We conclude that MS^{LB2} is indeed a lower bound for MS^*. Since $\text{MS}^{\text{LB1}} \leq \text{MS}^{\text{LB2}}$ it follows that also MS^{LB1} is indeed a lower bound for MS^*.

References

[1] S. Aine, P.P. Chakrabarti, R. Kumar, AWA* – a window constrained anytime heuristic search algorithm, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers Inc., 2007, pp. 2250–2255.

[2] A. Allahverdi, A survey of scheduling problems with no-wait in process, Eur. J. Oper. Res. 255 (3) (2016) 665–686.

[3] C. Ansótegui, M. Bofill, M. Palahı, J. Suy, M. Villaret, Satisfiability modulo theories: an efficient approach for the resource-constrained project scheduling problem, in: Proceedings of the 9th Symposium on Abstraction, Reformulation and Approximation, AAAI Press, 2011, pp. 2–9.

[4] D. Conforti, F. Guerriero, R. Guido, Optimization models for radiotherapy patient scheduling, Q. J. Oper. Res. 6 (3) (2008) 263–278.

[5] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A*, J. ACM 32 (3) (Jul. 1985) 505–536.

[6] M.S. Fox, Constraint-Directed Search: A Case Study of Job-Shop Scheduling, Ph.D. Thesis, Carnegie-Mellon University, 1983.

[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., 1979.

[8] P.C. Gilmore, R.E. Gomory, Sequencing a one-state variable machine: a solvable case of the traveling salesman problem, Oper. Res. 12 (5) (1964) 655–679.

[9] E.A. Hansen, R. Zhou, Anytime heuristic search, J. Artif. Intell. Res. 28 (2007) 267–297.

[10] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.

[11] S. Hartmann, D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem, Eur. J. Oper. Res. 207 (1) (2010) 1–14.

[12] M. Horn, G. Raidl, C. Blum, 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.), Proceedings of the Third International Conference on Machine Learning, Optimization, and Big Data, in: LNCS, vol. 10710, Springer, 2017, pp. 506–518.

[13] T. Kapamara, D. Petrovic, A heuristics and steepest hill climbing method to scheduling radiotherapy patients, in: Proceedings of the International Conference on Operational Research Applied to Health Services, 2009, pp. 1–7.

[14] M. Likhachev, G.J. Gordon, S. Thrun, ARA*: anytime A* with provable bounds on sub-optimality, in: S. Thrun, L.K. Saul, B. Schölkopf (Eds.), Proceedings of the Conference Advances in Neural Information Processing Systems 16, MIT Press, 2004, pp. 767–774.

[15] B.T. Lowerre, The Harpy Speech Recognition System, Ph.D. Thesis, Carnegie-Mellon University, 1976.

[16] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, T. Stützle, The irace package: iterated racing for automatic algorithm configuration, Oper. Res. Perspect. 3 (Supplement C) (2016) 43–58.

[17] J. Maschler, T. Hackl, M. Riedler, G.R. Raidl, An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem, in: Proceedings of the 12th Metaheuristics International Conference, 2017, pp. 465–474.

[18] J. Maschler, G.R. Raidl, Particle therapy patient scheduling with limited starting time variations of daily treatments, Int. Trans. Oper. Res. 27 (1) (2020) 458–479.

[19] J. Maschler, M. Riedler, G.R. Raidl, 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, in: LNCS, vol. 10671, Springer, 2018, pp. 364–372.

[20] J. Maschler, M. Riedler, M. Stock, G.R. Raidl, Particle therapy patient scheduling: first heuristic approaches, in: Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling, 2016, pp. 223–244.

[21] M.L. Pinedo, Scheduling: Theory, Algorithms, and Systems, 5th edition, Springer, 2015.

[22] I. Pohl, Heuristic search viewed as path finding in a graph, Artif. Intell. 1 (3) (1970) 193–204.

[23] S. Richter, J.T. Thayer, W. Ruml, The joy of forgetting: faster anytime search via restarting, in: Proceedings of the 20th International Conference on Automated Planning and Scheduling, 2010, pp. 137–144.

[24] L.H.O. Rios, L. Chaimowicz, A survey and classification of A* based best-first heuristic search algorithms, in: A.C. da Rocha Costa, R.M. Vicari, F. Tonidandel (Eds.), Proceedings of Advances in Artificial Intelligence: 20th Brazilian Symposium on Artificial Intelligence, in: LNCS, vol. 6404, Springer, 2010, pp. 253–262.

[25] H. Röck, The three-machine no-wait flow shop is NP-complete, J. ACM 31 (2) (1984) 336–345.

[26] S. Rubin, The Argos Image Understanding System, Ph.D. Thesis, Carnegie-Mellon University, 1978.

[27] I. Sabuncuoglu, M. Bayiz, Job shop scheduling with beam search, Eur. J. Oper. Res. 118 (2) (1999) 390–412.

[28] A. Schutt, T. Feydy, P.J. Stuckey, M.G. Wallace, Solving RCPSP/max by lazy clause generation, J. Sched. 16 (3) (Jun 2013) 273–289.

[29] R. Stern, A. Felner, J. van den Berg, R. Puzis, R. Shah, K. Goldberg, Potential-based bounded-cost search and anytime non-parametric A*, Artif. Intell. 214 (2014) 1–25.

[30] R. Stern, R. Puzis, A. Felner, Potential search: a new greedy anytime heuristic search, in: Proceedings of the 3rd Annual Symposium on Combinatorial Search, AAAI Press, 2010, pp. 119–120.

[31] R. Stern, R. Puzis, A. Felner, Potential search: a bounded-cost search algorithm, in: Proceedings of the 21st International Conference on Automated Planning and Scheduling, AAAI Press, 2011, pp. 234–241.

[32] S.G. Vadlamudi, S. Aine, P.P. Chakrabarti, Anytime pack search, Nat. Comput. 15 (3) (2016) 395–414.

[33] J. Van Den Berg, R. Shah, A. Huang, K. Goldberg, ANA*: anytime nonparametric A*, in: Proceedings of Twenty-fifth National Conference on Artificial Intelligence, 2011, pp. 105–111.

[34] J.A.A. Van der Veen, G.J. Wöginger, S. Zhang, Sequencing jobs that require common resources on a single machine: a solvable case of the TSP, Math. Program. 82 (1–2) (1998) 235–254.

[35] M.C. Vélez-Gallego, J. Maya, J.R. Montoya-Torres, A beam search heuristic for scheduling a single machine with release dates and sequence dependent setup times to minimize the makespan, Comput. Oper. Res. 73 (2016) 132–140.