Job Sequencing with One Common and Multiple Secondary Resources: A Problem Motivated from Particle Therapy for Cancer Treatment

Matthias Horn[1(✉)], Günther Raidl[1], and Christian Blum[2]

[1] Institute of Computer Graphics and Algorithms, TU Wien, Vienna, Austria
{horn,raidl}@ac.tuwien.ac.at
[2] Artificial Intelligence Research Institute (IIIA-CSIC), Campus of the UAB, Bellaterra, Spain
christian.blum@iiia.csic.es

Abstract. We consider in this work the problem of scheduling a set of jobs without preemption, where each job requires two resources: (1) a common resource, shared by all jobs, is required during a part of the job's processing period, while (2) a secondary resource, which is shared with only a subset of the other jobs, is required during the job's whole processing period. This problem models, for example, the scheduling of patients during one day in a particle therapy facility for cancer treatment. First, we show that the tackled problem is NP-hard. We then present a construction heuristic and a novel A* algorithm, both on the basis of an effective lower bound calculation. For comparison, we also model the problem as a mixed-integer linear program (MILP). An extensive experimental evaluation on three types of problem instances shows that A* typically works extremely well, even in the context of large instances with up to 1000 jobs. When our A* does not terminate with proven optimality, which might happen due to excessive memory requirements, it still returns an approximate solution with a usually small optimality gap. In contrast, solving the MILP model with the MILP solver CPLEX is not competitive except for very small problem instances.

1 Introduction

This work considers the following combinatorial optimization problem. A finite set of jobs must be processed without preemption. Each job requires two resources: (1) a common resource, shared by all jobs, is required during a certain part of the job's processing period, while (2) a secondary resource, which is shared with only a subset of the other jobs, is required during the job's whole processing period. This is the case, for example, in the context of the production of certain products where some raw material is put into specific fixtures or

G. Nicosia et al. (Eds.): MOD 2017, LNCS 10710, pp. 506–518, 2018.
https://doi.org/10.1007/978-3-319-72926-8_42

molds (the secondary resources), which are then sequentially processed on a single machine (the common resource). Finally, some further postprocessing (e.g., cooling) might be required before the fixtures/molds are available for further usage again. In order to perform this process as efficiently as possible, the aim is to minimizing the makespan, i.e., the total time required to finish the processing of all jobs. In the following we refer to this problem as *Job Sequencing with One Common and Multiple Secondary Resources* (JSOCMSR).

The technical definition of the problem, which is provided later on, was inspired by a more specific application scenario: the scheduling of patients in radiotherapy for cancer treatment [2,7] and particle therapy for cancer treatment [8]. In modern particle therapy, carbon or proton particles are accelerated in cyclotrons or synchrotrons to almost the speed of light and from there directed into a treatment room where a patient is radiated. A number of differently equipped treatment rooms is available (typically two to four) and the particle beam can only be directed into one of these rooms at a time. For each patient it is known in advance in which room she or he has to be treated in dependence on her/his specific needs. Moreover, each patient requires a certain preparation (such as positioning, fixation, possibly sedation) in the room before the actual irradiation can start. Upon finishing the irradiation of a patient, some further time is usually needed for medical inspections before the patient can actually leave the room and the treatment of a next patient can start. Note that the available rooms correspond to the secondary resources mentioned above, while the particle beam is the common resource. The scheduling of a set of patients at, e.g., one day in such a facility is considered.

For further information on particle therapy patient scheduling, in which JSOCMSR appears as sub-problem, the interested reader is referred to [8]. The whole practical scenario has to consider a time horizon of several weeks, additional resources, their availability time windows, and a combination of more advanced objectives.

The JSOCMSR is rather easy to solve when (1) only the common resource usage is the bottleneck and enough secondary resources are available or (2) the pre- and postprocessing times in 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. 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

In addition to formally proving that the JSOCMSR is NP-hard, we provide a lower bound on the makespan objective, which is then exploited both in the context of a constructive heuristic and a novel A* algorithm. The latter works on a special graph structure that allows to efficiently exploit symmetries and features a diving mechanism in order to obtain also heuristic solutions in regular intervals. In addition, we present a mixed-integer linear programming (MILP)

model for the JSOCMSR. Our experiments show that the A* algorithm performs excellently. Even many large problem instances with up to 1000 jobs can be solved to proven optimality. There are, however, also difficult problem instances for which A* terminates early due to excessive memory requirements. In these cases, heuristic solutions together with lower bounds and typically small optimality gaps are returned. In comparison, solving the MILP model by the general purpose MILP solver CPLEX[1] cannot compete with A*, as only solutions to rather small problem instances can be obtained in reasonable time.

2   Related Work

In the literature there are only few publications dealing with scenarios similar to JSOCMSR. Veen et al. [10] studied a related problem in which the common resource corresponds to a machine on which the jobs are processed and secondary resources needed in a pre- and postprocessing are called templates. An important restriction in their problem is that the postprocessing times are assumed to be 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 $O(n \log n)$.

Somewhat related is the no-wait flowshop problem; see [1] for a survey on this problem and related ones. Here, 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 [4]. In contrast, for three and more machines the problem is NP-hard, although it can still be transformed into a specially structured TSP. Röck [9] proved that the problem is strongly NP-hard for three machines by a reduction from the 3D-matching problem.

A more general problem as which our JSOCMSR can be modeled is the Resource-Constrained Project Scheduling Problem (RCPSP) with maximal time lags. We obtain a corresponding RCPSP instance 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 maximal time lags of zero, and all resource requirements must be respected. For a survey on RCPSPs with various extensions and respective solution methods see Hartmann and Briskorn [6]. For practically solving the JSOCMSR, however, such a mapping does not seem to be effective due to the increased number of required activities and since specificities of the problem are not exploited.

---

[1] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer.

3   Problem Definition and Complexity

An instance of JSOCMSR consists of a set of n jobs $J = \{1, \ldots, n\}$, the common resource 0, and a set of m secondary resources $R = \{1, \ldots, m\}$. By $R_0 = \{0\} \cup R$ we denote the set of all resources. 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 from a time $p_j^{\mathrm{pre}} \geq 0$ on, counted from the job's start, for a duration p_j^0 with $0 < p_j^0 \leq p_j - p_j^{\mathrm{pre}}$. A solution to the problem is described by the jobs' starting times $s = (s_j)_{j \in J}$ with $s_j \geq 0$. 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 latest. This optimization criterion is known as the *makespan*, and it can be calculated for a solution s by

$$\mathrm{MS}(s) = \max \{s_j + p_j \mid j \in J\}. \tag{1}$$

As each job requires the common resource 0, and only one job can use this resource at a time, a solution implies a total ordering of the jobs. Vice versa, any ordering—i.e., permutation—$\pi = (\pi_i)_{i=1,\ldots,n}$, 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 $\mathrm{MS}(\pi)$ 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 $p_j^{\mathrm{post}} = p_j - p_j^{\mathrm{pre}} - p_j^0$, $\forall j \in J$ 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

Let the decision variant of JSOCMSR be the problem in which it has to be determined if there exists a feasible solution with a makespan corresponding to a given constant MS^*.

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

Proof. Our problem is in class NP since a solution can be checked in polynomial time. We show that JSOCMSR is NP-complete by a polynomial reduction from the well-known NP-complete *Partition Problem* (PP) [3], 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 $\sum_{a \in A_1} a = \sum_{a \in A_2} a$.

We transform an instance of the PP into an instance of the JSOCMSRC as follows. Let $m = 2$ and J consist of the following jobs:

– For each $a \in A$ there is a corresponding job $j \in \{1, \ldots, |A|\} \subset J$ 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 $p_j^{\text{pre}} = 0$.

Let $\text{MS}^* = p_{|A|+1} + p_{|A|+2} = \sum_{a \in A} a + 2$. A feasible solution to JSOCMSR 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 jobs $|A| + 1$ and $|A| + 2$, 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., polynomial time.

Consequently, the decision variant of the JSOCMSR is NP-complete. $\qquad\square$

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

3.2 Lower and Upper Bounds

For an instance of JSOCMSR a lower bound for the makespan can be calculated on the basis of each resource $r \in R$ by taking the total time $\sum_{j \in J_r} p_j$. Similarly, one more lower bound can also be obtained from the total time resource 0 is required, i.e., $\sum_{j \in J} p_j^0$. The latter can further be improved by adding the minimal time for preprocessing and postprocessing for the first and last scheduled jobs, respectively. Taking the maximum of these $m + 1$ individual lower bounds yields

$$\text{MS}^{\text{LB}} = \max \left(\min_{j,j' \in J \,|\, j \neq j' \vee |J| = 1} (p_j^{\text{pre}} + p_{j'}^{\text{post}}) + \sum_{j \in J} p_j^0, \; \max_{r \in R} \sum_{j \in J_r} p_j \right). \quad (2)$$

Figure 1 illustrates these relationships.

A trivial upper bound is obtained when scheduling all jobs strictly sequentially, yielding $\text{MS}^{\text{UB}} = \sum_{j \in J} p_j$. It follows that taking any normalized solution has an approximation factor of no more than m, since $\text{MS}^{\text{UB}} \leq m \cdot \text{MS}^{\text{LB}}$.

4 Least Lower Bound Heuristic

We construct a heuristic solution by iteratively selecting a not yet scheduled job and always appending it at the end of the current partial schedule at the
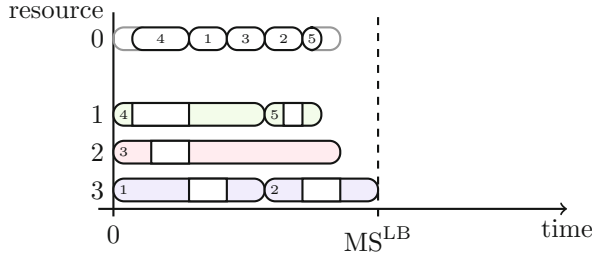
Fig. 1. Resource-specific individual lower bounds and the overall lower bound $\mathrm{MS}^{\mathrm{LB}}$ for an example instance with $n = 5$ jobs and $m = 3$ secondary resources.

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 Sect. 3.2. 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. Given π^{P}, the *earliest availability time* for each resource—that is, the time from which on the resource might be used by a next yet unscheduled job—can be calculated from the respective finishing time of the last job using this resource:

$$t_0 = \begin{cases} \max_{j \in J \setminus J'} s_j + p_j^{\mathrm{pre}} + p_j^0 & \text{for } J' \neq J \\ 0 & \text{else} \end{cases} \tag{3}$$

$$t_r = \begin{cases} \max_{j \in J_r \setminus J'} s_j + p_j & \text{for } J_r \setminus J' \neq \emptyset \\ 0 & \text{else} \end{cases} \quad \forall r \in R \tag{4}$$

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

$$t_r \leftarrow \max(t_r, t_0 - \max_{j \in J_r \cap J'} p_j^{\mathrm{pre}}) \quad \forall r \in R \mid J_r \cap J' \neq \emptyset. \tag{5}$$

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

$$t_0 \leftarrow \max \left(t_0, \min_{j \in J'} (t_{q_j} + p_j^{\mathrm{pre}}) \right) = \max \left(t_0, \min_{r \in R \mid J_r \cap J' \neq \emptyset} (t_r + \min_{j \in J_r \cap J'} p_j^{\mathrm{pre}}) \right). \tag{6}$$

Further note that after a successful increase of t_0 by rule (6), some resource $r \in R$ might become available for a further increase of its t_r by the respective rule (5). We therefore apply all these trimming rules repeatedly until no further increase can be achieved.

Following our general lower bound calculation for the makespan in (2), it is now 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

$$\mathrm{MS}_r^{\mathrm{LB}}(\pi^{\mathrm{P}}) = \begin{cases} t_r + \sum_{j \in J_r \cap J'} p_j & \text{for } J_r \cap J' \neq \emptyset \\ 0 & \text{else} \end{cases} \qquad \forall r \in R. \qquad (7)$$

Note that we define $\mathrm{MS}_r^{\mathrm{LB}}(\pi^{\mathrm{P}}) = 0$ for any resource r that is not required by any remaining job in J' since these bounds should not be relevant for our further considerations.

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

$$\mathrm{MS}_0^{\mathrm{LB}}(\pi^{\mathrm{P}}) = \max\left(t_0 + \min_{j \in J'} p_j^{\text{post}}, \min_{j,j' \in J' \mid j \neq j' \vee |J'|=1} (t_{q_j} + p_j^{\text{pre}} + p_{j'}^{\text{post}})\right) + \sum_{j \in J'} p_j^0. \qquad (8)$$

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

$$\mathrm{MS}_{\max}^{\mathrm{LB}}(\pi^{\mathrm{P}}) = \max_{r \in R_0} \mathrm{MS}_r^{\mathrm{LB}}(\pi^{\mathrm{P}}). \qquad (9)$$

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 $\mathrm{MS}_r^{\mathrm{LB}}$, $r \in R_0$, as this gives a more fine-grained discrimination than just considering the impact on the overall bound $\mathrm{MS}_{\max}^{\mathrm{LB}}(\pi^{\mathrm{P}})$, which would often lead to ties.

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

$$\exists i \in \{0, \ldots, m\} \mid f_i(\pi^{\mathrm{P}} \oplus j) < f_i(\pi^{\mathrm{P}} \oplus j') \wedge \forall i' < i : f_{i'}(\pi^{\mathrm{P}} \oplus j) = f_{i'}(\pi^{\mathrm{P}} \oplus j'). \qquad (10)$$

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

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

5   Mixed Integer Linear Programming Formulation

The position-based mixed integer linear program (MILP) described in the following models solutions to the JSOCMSR in terms of permutations of all jobs. Index $i \in \{1, \ldots, n\}$ refers hereby to position i in a permutation. 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, $\mathrm{MS} \geq 0$ is the makespan variable to be minimized.

$$\min \mathrm{MS} \tag{11}$$

$$\sum_{j \in J} x_{j,i} = 1 \qquad\qquad i = 1, \ldots, n \tag{12}$$

$$\sum_{i=1}^{n} x_{j,i} = 1 \qquad\qquad j \in J \tag{13}$$

$$s_i + \sum_{j \in J} x_{j,i} \cdot p_j \leq \mathrm{MS} \qquad\qquad i = 1, \ldots, n \tag{14}$$

$$s_1 = 0 \tag{15}$$

$$s_i + \sum_{j \in J} x_{j,i} \cdot p_j^{\mathrm{pre}} \geq s_{i-1} + \sum_{j \in J} x_{j,i-1} \cdot (p_j^{\mathrm{pre}} + p_j^0) \qquad i = 2, \ldots, n \tag{16}$$

$$s_{i'} - s_i + \sum_{j \in J_r} x_{j,i'}(M + p_j) + \sum_{j \in J_r} x_{j,i} M \leq 2M$$
$$i = 2, \ldots, n, \ i' = 1, \ldots, i-1, \ r \in R \tag{17}$$

$$x_{j,i} \in \{0, 1\} \qquad\qquad j \in J, \ i = 1, \ldots, n \tag{18}$$

$$s_i \geq 0 \qquad\qquad i = 1, \ldots, n \tag{19}$$

$$\mathrm{MS} \geq 0 \tag{20}$$

Hereby, Eq. (12) ensure that exactly one job is assigned to the i-th position of the permutation and (13) ensure that each job is assigned to exactly one position. The makespan is determined by inequalities (14). Equation (15) 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 (16) take care of the common resource 0, while (17) consider the secondary resources. The Big-M constant in these latter inequalities is set to the makespan obtained by LLBH.

6   A* Algorithm

Based on the solution construction principle of LLBH it is also possible to perform a more systematic search for a proven optimal solution following the concept of A* search [5]. Our A* algorithm searches in a graph whose nodes correspond to partial solutions and whose arcs represent the extensions of partial solutions by appending not yet scheduled nodes. More precisely, each node in this graph maintains the following information:

1. The *unordered* set $\hat{J} \subset J$ of already scheduled jobs, implemented by a bit-vector.
2. A set of *Non-Dominated Times* (NDT) records, where each NDT record corresponds to an individual, more specific partial solution with an indirectly given ordering for the scheduled jobs by storing:

- the vector $\boldsymbol{t} = (t_r)_{r \in R_0}$ of the trimmed earliest usage times t_r for all resources as defined by (3)–(6);
- the last scheduled job $j^{\text{last}} \in \hat{J}$ after which $\boldsymbol{t}$ was obtained;
- an evaluation vector $\boldsymbol{f}'$ similar to $\boldsymbol{f}$ that will be defined 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. For a given node/NDT record, the corresponding ordering of the scheduled jobs $\hat{J}$ can be derived in a reverse iterative manner by considering the fitting preceding node/NDT records, always continuing with the node $\hat{J} \setminus \{j^{\text{last}}\}$ and an NDT record with times t_r allowing to schedule job j^{last} without exceeding the t_r values of the last node/NDT record.

Initially a starting node/NDT record corresponding to the empty schedule is generated with $\hat{J} = \emptyset$, $\boldsymbol{t} = \boldsymbol{0}$, $j^{\text{last}} = \text{none}$, and $\boldsymbol{f}' = (\text{MS}^{\text{LB}}, \ldots, \text{MS}^{\text{LB}})$. The goal node is a 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. Furthermore, the A* algorithm maintains a priority queue containing references to all *open* node/NDT record pairs, i.e., the non-dominated partial solutions that have not yet been expanded. The order criterion in this priority queue extends the *is-better* relation (10) from the LLBH heuristic by considering the number of remaining unscheduled jobs $|J \setminus \hat{J}|(\pi^{\text{p}})$ as secondary criterion after $\text{MS}^{\text{LB}}_{\max}(\pi^{\text{p}})$, i.e., vectors

$$\boldsymbol{f}' := (\text{MS}^{\text{LB}}_{\max}(\pi^{\text{p}}) = f_0(\pi^{\text{p}}), |J \setminus \hat{J}|(\pi^{\text{p}}), f_1(\pi^{\text{p}}), \ldots, f_m(\pi^{\text{p}})) \qquad (21)$$

are lexicographically compared. This enhanced relation implies that partial solutions with more scheduled jobs are preferred over partial solutions with the same $\text{MS}^{\text{LB}}_{\max}$ but fewer scheduled jobs, and thus the search adopts depth-first search characteristics when $\text{MS}^{\text{LB}}_{\max}$ does not change. In this way, complete solutions are obtained earlier.

Algorithm 1 sketches our A* algorithm. In each major iteration, a best node/NDT record pair is taken from the priority queue and expanded by considering the addition of each job $j \in J \setminus \hat{J}$. Hereby, the corresponding node is looked up or created when it does not yet exist and a respective NDT record is determined by calculating the earliest usage times $\boldsymbol{t}$ and the evaluation vector $\boldsymbol{f}$. 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 $\boldsymbol{t}$ *dominates* (symbol $\triangleleft$) another NDT record with time vector $\boldsymbol{t}'$ iff $\forall r \in R_0 \ (t_r \leq t'_r) \land \exists r \in R_0 \ (t_r < t'_r)$. The A* algorithm stops with a proven optimal solution when the goal node representing a complete solution is selected for expansion.

Diving: The A* algorithm described above aims at finding a proven optimal solution as quickly as possible. It usually does not yield intermediate complete solutions significantly earlier than when terminating with the proven optimum.

To also obtain intermediate heuristic solutions we extended our A* algorithm by *diving* for a complete solution at regular intervals: At the very beginning

Algorithm 1. A* Algorithm for JSOCMSR

1: Initialize priority queue Q with $(\emptyset, (\mathbf{0}, \text{none}, (\text{MS}^{\text{LB}}, \dots, \text{MS}^{\text{LB}})))$
2: $iter \leftarrow 0$
3: **repeat**
4:    **if** $iter \bmod \delta = 0$ **then**
5:       $\pi \leftarrow$ perform diving to obtain complete solution
6:       $\pi^{\text{best}} \leftarrow \pi$ **if** new best complete solution
7:    **end if**
8:    $(\hat{J}, (\mathbf{t}, j^{\text{last}}, \mathbf{f}')) \leftarrow$ Q.pop()
9:    **if** $|\hat{J}| = n$ **then**
10:       **return** proven optimal solution π^{best}
11:    **end if**
12:    **for all** $j \in J \setminus \hat{J}$ **do**
13:       find or create node N with $\hat{J}(N) = \hat{J} \cup \{j\}$
14:       calculate new NDT record $(\mathbf{t}_{\text{new}}, j, \mathbf{f}'_{\text{new}})$ from $\mathbf{t}$
15:       **if** $\nexists (\mathbf{t}_{\text{dom}}, j^{\text{last}}_{\text{dom}}, \mathbf{f}'_{\text{dom}}) \in \text{NDTs}(N) \mid \mathbf{t}_{\text{dom}} \trianglelefteq \mathbf{t}_{\text{new}}$ **then**
16:          Remove every $(\mathbf{t}_{\text{d}}, j'_{\text{d}}, \mathbf{f}'_{\text{d}}) \in \text{NDTs}(N) \mid \mathbf{t}_{\text{new}} \triangleleft \mathbf{t}_{\text{d}}$
17:          Add $(\mathbf{t}_{\text{new}}, j, \mathbf{f}'_{\text{new}})$ to NDTs(N)
18:          Q.push($\hat{J}(N), (\mathbf{t}_{\text{new}}, j, \mathbf{f}'_{\text{new}})$)
19:          **if** $|\hat{J}(N)| = n$ **then**
20:             $\pi \leftarrow$ derive complete solution from $(\hat{J}(N), (\mathbf{t}_{\text{new}}, j, \mathbf{f}'_{\text{new}}))$
21:             $\pi^{\text{best}} \leftarrow \pi$ **if** new best complete solution
22:          **end if**
23:       **end if**
24:    **end for**
25:    $iter \leftarrow iter + 1$
26: **until** time- or memory-limit reached
27: **return** heuristic solution π^{best} and lower bound f_0

---

and after each δ regular iterations, the algorithm switches from its classical best-first strategy temporarily to a greedy completion strategy which follows in essence LLBH. The currently selected node is expanded by considering all feasible extensions, and each extension is evaluated by calculating the respective evaluation vector $\mathbf{f}'$. From all these extensions, only those that are new and non-dominated—i.e., no corresponding node/NDT entry exists yet—are kept. Should no extension remain in this way, diving terminates unsuccessfully. Otherwise, a best extension is selected from this set according to the lexicographic comparison of the $\mathbf{f}'$ vectors, and the diving continues by expanding this node/NDT record pair next. This methodology guarantees that always not yet expanded nodes are further expanded and the diving, if successful, always yields a different solution.

7   Computational Results

To test our algorithms we created two non-trivial sets of random instances. Set B exhibits a *balanced* (B) workload over all resources R, whereas set S has a *skewed* (S) workload. Each set 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. The required resource q_j for each job $j \in J$ was randomly sampled from the discrete uniform distribution $\mathcal{U}\{1, m\}$ for the balanced set B but in a skewed way for set S: There, resource m is chosen with twice the probability of each of the resources 1 to $m-1$. 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 $\mathcal{U}\{1, 2500\}$ in case of set S.

A third set of instances was derived from the work on patient scheduling for particle therapy in [8]. This set, called P, comprises 699 instances that are expected in practical day-scenarios of this application. We partitioned the whole set 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 three instance sets are available from https://www.ac.tuwien.ac.at/research/problem-instances#JSOCMSR.

The algorithms were implemented using G++ 5.4.1. All tests were done on a single core of an Intel Xeon E5649 with 2.53 GHz with a CPU-time limit of 900 s and a memory limit of 15 GB RAM. The MILP from Sect. 5 was solved with CPLEX 12.7. In A* diving was performed every $\delta = 1000$-th iteration.

Table 1 lists aggregated results for each combination of instance type and the different numbers of jobs and secondary resources. Columns opt state the percentage of instances that could be solved to proven optimality. Columns %-gap list average optimality gaps of final solutions π, which are calculated by $100 \cdot (\text{MS}(\pi) - \text{LB})/\text{LB}$, where LB is the lower bound returned from A* in case of LLBH and A* and the lower bound returned from CPLEX in case of CPLEX. Columns $\sigma_{\text{\%-gap}}$ provide corresponding standard deviations. Columns t show the median running times in seconds. In case of MILP, optimality gaps are list only if solutions for all 50 instances could be obtained.

These results give a rather clear picture: While A* performs very well on essentially all instance sets and sizes—its largest average optimality gaps are $<5\%$—CPLEX applied to our MILP model cannot compete at all. CPLEX is not even able to solve all instances with 10 jobs to optimality, and generally does not yield any solution for instances with 200 and more jobs. With only few exceptions, instances of set B are generally rather easy to solve for A* to either optimality or with a small remaining gap of less than 0.2%. Median running times are here fractions of a second for $n \leq 500$ and under three seconds for $n = 1000$. Here we could observe that the general lower bound MS^{LB} is usually very tight and especially for $m = 2$ often already corresponds to the optimal solution value. Skewed instances of type S but also most instances of type P are more difficult to solve. Especially for set S and $m \in \{2, 3\}$, A* was only able to solve instances up to size 20 consistently to optimality. The reason when A* did not terminate with proven optimality was always that the memory limit had been reached. However, thanks to A*'s diving, heuristic solutions with small remaining optimality gaps could still be found. The LLBH is—as expected—always very fast, nevertheless providing excellent solutions, although without specific performance guarantees.

Table 1. Average results of LLBH, A*, and CPLEX for instances of sets B, S, and P.

type	n	m	LLBH opt[%]	%-gap	$\sigma_{\text{%-gap}}$	t[s]	A* Search opt[%]	%-gap	$\sigma_{\text{%-gap}}$	t[s]	MILP/CPLEX opt[%]	%-gap	$\sigma_{\text{%-gap}}$	t[s]
B	10	2	90	0.197	0.87	<0.1	100	0.000	0.00	<0.1	40	0.007	0.01	22.6
B	20	2	96	0.074	0.37	<0.1	100	0.000	0.00	<0.1	-	-	-	900.1
B	50	2	100	0.000	0.00	<0.1	100	0.000	0.00	<0.1	-	-	-	900.0
B	100	2	100	0.000	0.00	<0.1	100	0.000	0.00	<0.1	-	-	-	900.0
B	200	2	100	0.000	0.00	<0.1	100	0.000	0.00	<0.1	-	-	-	900.0
B	500	2	100	0.000	0.00	0.5	100	0.000	0.00	0.4	-	-	-	900.0
B	1000	2	100	0.000	0.00	3.8	100	0.000	0.00	2.6	-	-	-	900.0
B	10	3	74	1.133	2.48	<0.1	100	0.000	0.00	<0.1	48	0.007	0.01	19.2
B	20	3	76	0.767	1.65	<0.1	100	0.000	0.00	<0.1	2	-	-	900.1
B	50	3	74	0.752	1.40	<0.1	92	0.078	0.30	<0.1	-	-	-	900.0
B	100	3	68	0.632	1.16	<0.1	82	0.168	0.39	<0.1	-	-	-	900.0
B	200	3	68	0.405	0.81	<0.1	82	0.172	0.42	<0.1	-	-	-	900.0
B	500	3	64	0.294	0.46	0.5	68	0.117	0.21	0.4	-	-	-	900.0
B	1000	3	68	0.127	0.25	3.8	76	0.062	0.16	2.7	-	-	-	900.0
B	10	5	50	2.320	3.27	<0.1	100	0.000	0.00	<0.1	74	0.004	0.01	2.2
B	20	5	42	1.634	2.31	<0.1	100	0.000	0.00	<0.1	44	-	-	900.0
B	50	5	52	0.475	0.78	<0.1	94	0.016	0.07	<0.1	34	-	-	900.0
B	100	5	52	0.247	0.45	<0.1	88	0.016	0.06	<0.1	-	-	-	900.0
B	200	5	74	0.076	0.17	<0.1	96	0.002	0.01	<0.1	-	-	-	900.0
B	500	5	80	0.014	0.04	0.5	96	0.001	0.01	0.4	-	-	-	900.0
B	1000	5	76	0.006	0.01	3.8	98	0.000	0.00	2.6	-	-	-	900.0
S	10	2	40	1.387	1.84	<0.1	100	0.000	0.00	<0.1	60	0.004	0.01	2.8
S	20	2	14	1.675	1.41	<0.1	100	0.000	0.00	19.3	2	11.986	10.09	900.1
S	50	2	0	4.739	2.58	<0.1	0	3.374	2.32	154.2	-	-	-	900.1
S	100	2	0	4.122	1.70	<0.1	0	3.271	1.57	153.1	-	-	-	900.0
S	200	2	0	3.678	1.01	<0.1	0	3.163	0.98	166.1	-	-	-	900.0
S	500	2	0	3.662	0.75	0.5	0	3.360	0.70	201.5	-	-	-	900.0
S	1000	2	0	3.626	0.50	3.8	0	3.453	0.48	241.1	-	-	-	900.0
S	10	3	44	1.343	1.73	<0.1	100	0.000	0.00	<0.1	50	0.006	0.01	4.2
S	20	3	20	2.323	1.86	<0.1	100	0.000	0.00	15.2	28	-	-	900.0
S	50	3	18	4.170	2.96	<0.1	20	2.807	2.34	163.3	8	-	-	900.0
S	100	3	18	4.506	3.11	<0.1	20	3.593	2.64	181.4	-	-	-	900.0
S	200	3	10	4.545	2.91	<0.1	10	4.011	2.70	194.1	-	-	-	900.0
S	500	3	0	4.960	1.94	0.5	0	4.672	1.92	236.5	-	-	-	900.0
S	1000	3	0	5.018	1.46	3.8	0	4.852	1.41	246.3	-	-	-	900.0
S	10	5	46	1.496	1.87	<0.1	100	0.000	0.00	<0.1	66	0.004	0.01	0.2
S	20	5	64	0.890	1.80	<0.1	100	0.000	0.00	<0.1	82	0.616	2.42	0.8
S	50	5	74	0.275	0.85	<0.1	88	0.097	0.49	<0.1	84	-	-	16.6
S	100	5	88	0.044	0.17	<0.1	98	0.014	0.10	<0.1	46	-	-	890.5
S	200	5	86	0.010	0.03	<0.1	100	0.000	0.00	<0.1	-	-	-	900.0
S	500	5	96	0.002	0.01	0.5	100	0.000	0.00	0.4	-	-	-	900.0
S	1000	5	96	0.001	0.01	3.8	100	0.000	0.00	2.6	-	-	-	900.0
P	≤10	3	82	0.366	0.93	<0.1	100	0.000	0.00	<0.1	63	0.611	0.84	<0.1
P	≤20	3	64	0.374	0.75	<0.1	100	0.000	0.00	<0.1	59	7.512	17.35	63.5
P	≤50	3	62	0.554	0.96	<0.1	80	0.219	0.55	<0.1	27	-	-	900.0
P	≤100	3	65	0.497	1.01	<0.1	77	0.247	0.58	<0.1	4	-	-	900.0

8   Conclusions

In this work we introduced 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. Despite that we could show this problem to be NP-hard, we came up with an excellent lower bound for the makespan, which we exploited in the fast constructive heuristic LLBH and the complete A* search. The A* algorithm features in particular a special graph structure in which each node corresponds to an unordered set of already scheduled jobs in combination with a set of NDT records representing individual non-dominated partial solutions. Hereby it is possible to exploit symmetries and reduce the memory consumption. A diving mechanism is further used to obtain heuristic solutions in regular intervals. It turns out that A* works mostly extremely well. However, some instances especially with skewed resource workloads and competing resources are occasionally hard to solve. The focus of further research will be to better understand difficult instances, to consider extended variants of this problem and to develop advanced heuristic methods.

References

1. Allahverdi, A.: A survey of scheduling problems with no-wait in process. Eur. J. Oper. Res. **255**(3), 665–686 (2016)
2. Conforti, D., Guerriero, F., Guido, R.: Optimization models for radiotherapy patient scheduling. 4OR **6**(3), 263–278 (2008)
3. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., New York (1979)
4. Gilmore, P.C., Gomory, R.E.: Sequencing a one-state variable machine: a solvable case of the traveling salesman problem. Oper. Res. **12**(5), 655–679 (1964)
5. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
6. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. Eur. J. Oper. Res. **207**(1), 1–14 (2010)
7. Kapamara, T., Sheibani, K., Haas, O., Petrovic, D., Reeves, C.: A review of scheduling problems in radiotherapy. In: Proceedings of the International Control Systems Engineering Conference, pp. 207–211. Coventry University Publishing, Coventry (2006)
8. Maschler, J., Riedler, M., Stock, M., Raidl, G.R.: Particle therapy patient scheduling: first heuristic approaches. In: PATAT 2016: Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling, Udine, Italy, pp. 223–244 (2016)
9. Röck, H.: The three-machine no-wait flow shop is NP-complete. J. ACM **31**(2), 336–345 (1984)
10. Van der Veen, J.A.A., Wöginger, G.J., Zhang, S.: Sequencing jobs that require common resources on a single machine: a solvable case of the TSP. Math. Program. **82**(1–2), 235–254 (1998)