A Variable Neighborhood Search for the Job Sequencing with One Common and Multiple Secondary Resources Problem

Thomas Kaufmann, Matthias Horn(✉), and Günther R. Raidl

Institute of Logic and Computation, TU Wien, Vienna, Austria
thomas@tkaufmann.at, {horn,raidl}@ac.tuwien.ac.at

Abstract. In this work we consider a scheduling problem where a set of non-preemptive jobs needs to be scheduled such that the makespan is minimized. Each job requires two resources: (1) a common resource, shared by all jobs and (2) a secondary resource, shared with only a subset of the other jobs. The secondary resource is required during the job's entire processing time whereas the common resource is only required during a part of a job's execution. The problem models, for instance, the scheduling of patients during one day in a particle therapy facility for cancer treatment. We heuristically tackle the problem by a general variable neighborhood search (GVNS) based on move and exchange neighborhoods and an efficient evaluation scheme to scan the neighborhoods of the current incumbent solution. An experimental evaluation on two benchmark instance sets, including instances with up to 2000 jobs, shows the effectiveness of the GVNS. In particular for larger instances our GVNS outperforms an anytime A* algorithm that was the so far leading method in heuristic terms as well as a constrained programming model solved by ILOG CP optimizer.

Keywords: Sequencing · Scheduling · Variable neighborhood search · Particle therapy patient scheduling

1 Introduction

In this work we apply a *general variable neighborhood search* (GVNS) approach to the *job sequencing with one common and multiple secondary resources* (JSOCMSR) problem. The JSOCMSR has been introduced in [8] and considers a scenario where a finite set of jobs must be scheduled without preemption. Each job requires two resources: (1) a common resource, which is shared by all jobs and (2) a secondary resource which is shared by only a subset of the jobs. The secondary resource is required for the entire processing time of a job whereas

We gratefully acknowledge the financial support of the Doctoral Program "Vienna Graduate School on Computational Optimization" funded by Austrian Science Foundation under Project No W1260-N35.

T. Bäck et al. (Eds.): PPSN 2020, LNCS 12270, pp. 385–398, 2020.
https://doi.org/10.1007/978-3-030-58115-2_27

the common resource is needed only after some pre-processing time for a part of the job's whole processing time. The objective is to minimize the makespan.

The JSOCMSR problem has applications, for example, in the context of the production of certain goods where on a single machine (the common resource, for example an oven used for heat treatment) some fixtures or molds (the secondary resource) filled with some raw material are sequentially processed. Before the fixtures/molds can be processed on the machine there is a setup time during which the secondary resource is already needed (e.g., preparations within the mold) as well as a post-processing time also still requiring the secondary resource (e.g., cooling before the product can be removed from the mold).

Another more specific application is the scheduling of treatments for cancer patients who are to receive a particle therapy [1,9,13]. In this rather novel treatment technique, carbon or proton particles are accelerated in an particle accelerator to almost the speed of light, and this particle beam is directed into one of a few treatment rooms where a patient gets radiated. There are typically two to four treatment rooms that are differently equipped for specific kinds of radiations. In this scenario the JSOCMSR appears as a simplified daily subproblem where the treatment rooms correspond to the secondary resources and the single particle beam, which can only be directed into one of these rooms at a time, corresponds to the common resource. The treatment room for each patient is known in advance and depends on the patients specific needs. Each patient treatment requires a specific preparation time (positioning, fixation, sedation, etc.) in the room before the radiation can be performed and occupies the room after the treatment for some further medical examinations until the patient can eventually leave the room. The JSOCMSR we consider here only represents the "hard core" of the real practical scheduling problem, in which several different objectives, further resources, time windows, and other soft- and hard-constraints need to be taken care of. Maschler et al. [13] tackled this real-world problem with a greedy construction method, which is extended to an iterative greedy metaheuristic and a greedy randomized adaptive search (GRASP).

For the JSOCMSR, Horn et al. [5] proposed an exact anytime A* search. For instances, where the workload over all secondary resources is rather balanced this A* search works extremely well by solving even large instances with up to 2000 jobs to proven optimality. However, on instances where the workload over the secondary resources is skewed, i.e., one resource is more frequently required than the others, the A* algorithm's performance degrades and it is in many cases only able to provide heuristic solutions.

Contribution of This Work. For such hard-to-solve JSOCMSR instances we propose a GVNS heuristic with range-limited neighborhood structures. First, we discuss the related work in Sect. 2 and give a formal problem definition in Sect. 3. The GVNS is described in Sect. 4, where we also introduce the so-called *synchronization mechanism* that allows us to quickly determine the changed makespan of the incumbent solution when a neighborhood move is applied. In this way our GVNS algorithm is able to quickly scan through the used neighborhoods. In Sect. 5 experimental results are provided, which indicate that this mechanism is rather independent of the number of jobs and therefore also applicable for larger problem instances. Ultimately, the proposed GVNS is able to provide new state-of-the-art results for many hard-to-solve instance classes of the JSOCMSR.

2 Related Work

As mentioned the JSOCMSR was already approached by Horn et al. [5,8], who also proved the NP-hardness of the problem. The authors suggested methods for calculating lower bounds for the makespan, given a partial solution with still open jobs. Those lower bounds are then utilized in both a heuristic construction algorithm as well as a novel exact anytime A* search. The latter performs after a certain number of classical A* node expansions a beam search, starting from the currently selected node. In this way the A* search is able to provide besides a proven optimal solution at the end also promising intermediate heuristic solutions. The latter are especially valuable for hard instances where runtimes would be too excessive and the search must be terminated prematurely. This A* search was compared, among others, to a compact position based *mixed integer linear programming* (MIP) model solved with CPLEX as well as a *constraint programming* (CP) model solved with ILOG CP Optimizer. The experimental evaluation shows that the A* search clearly dominates the considered competitors.

A problem strongly related to the JSOCMSR is considered by Veen et al. [16] with the important difference that post-processing times are negligible compared to the total processing times of the jobs. This property allows to treat the problem as a traveling salesman problem with a special cost structure, which can be solved efficiently in time $O(n \log n)$, where n is the number of jobs. For other related problems we refer to [5].

A prize-collecting variant of the JSOCMSR (PC-JSOCMSR) is considered by Horn et al. [7] as well as by Maschler and Raidl [12]. In both works, each job is further equipped with a prize and a set of time windows such that the job can only be scheduled within one of it's time windows. The objective is to find a subset of jobs together with a feasible schedule such that the overall prize of the scheduled jobs is maximized. In Horn et al. [7] an exact A* algorithm is proposed for the PC-JSOCMSR, where corresponding upper bound calculations are based on Lagrangian and linear programming relaxations. The A* algorithm is able to solve small instances with up to 30 jobs to optimality; see [6] for an extended version of the original conference paper. Experiments showed that A* search outperforms a compact MIP model solved by CPLEX as well as a MiniZinc CP model solved by different back-end solvers. Maschler and Raidl [12] investigated different heuristic methods to solve larger instances with up to 300 jobs. These methods are based on *multivalued decision diagrams* (MDDs) and general variable neighborhood search. Both works, [7,12], where then extended by Horn et al. [4] by utilizing a novel construction algorithm for relaxed MDDs. On the basis of these, new state-of-the-art results could be obtained for PC-JSOCMSR instances with up to 500 jobs.

3 Problem Formalization

The JSOCMSR consists of a finite set $J = \{1, \ldots, n\}$ of n jobs, the common resource 0, and a set $R = \{1, \ldots, m\}$ of m secondary resources. Let $R_0 = \{0\} \cup R$ be the set of all resources. Each job $j \in J$ requires one specific secondary resource $q_j \in R$ for its whole processing time $p_j > 0$. Let $J_r = \{j \in J \mid q_j = r\}$ be the subset of jobs requiring resource $r \in R$ as secondary resource. Moreover, each job j needs after some pre-processing time $p_j^{\mathrm{pre}} \geq 0$, counted from the job's start time, also the common resource 0 for a time $0 < p_j^0 \leq p_j - p_j^{\mathrm{pre}}$. For convenience we define the post-processing time, where the secondary resource is still needed but not the common resource anymore, by $p_j^{\mathrm{post}} = p_j - p_j^{\mathrm{pre}} - p_j^0$. A solution to the problem is described by the jobs' starting times $s = (s_j)_{j \in J}$ with $s_j \geq 0$. A solution s is feasible if no two jobs require the same resource at the same time. The objective is to find a feasible solution s that minimizes the makespan $\mathrm{MS}(s) = \max\{s_j + p_j \mid j \in J\}$, i.e., the time the last job finishes its execution.

Since jobs acquire the common resource 0 excursively, a solution implies a total ordering of the jobs. Vice versa, any permutation $\pi = (\pi_i)_{i=1,\ldots,n}$ of jobs in J can be decoded into a feasible solution in a greedy way by scheduling each job in the given order at the earliest feasible time. We refer to a schedule obtained in this way as a *normalized schedule*. By the notation $\mathrm{MS}(\pi)$ we refer to the makespan of a normalized schedule induced by the job permutation π. Since any optimal solution is either a normalized schedule or there exists a corresponding normalized schedule with the same objective value, we can restrict our search to job permutations and their corresponding normalized schedules. Job permutations are therefore the primary solution representation in the suggested GVNS.

4 Variable Neighborhood Search

The well known *variable neighborhood search* (VNS) metaheuristic, introduced by Mladenović and Hansen [14], has been successfully applied on many combinatorial optimization problems; for a comprehensive review see [2]. To heuristically solve the JSOCMSR we use a GVNS, where two different sets of neighborhood structures $N^{\mathrm{I}}_{i=1\ldots k_{\max}}$ and $N^{\mathrm{S}}_{i=1\ldots l_{\max}}$ are alternatingly applied in intensification and diversification phases. In the intensification phase, a deterministic *variable neighborhood descent* (VND) uses a set of $k_{\max} = 4$ intensification neighborhood structures, which are searched, depending on their computational cost, in either a first-improvement or best-improvement manner. In the diversification phase a set of $l_{\max} = 23$ increasingly perturbative shaking neighborhood structures are used to perform random moves in order to reach parts of the search space that are farther away from the incumbent solution. Algorithm 1 illustrates this procedure. The initial solution—represented by permutation π—is created uniformly at random. The GVNS terminates if a certain time-limit is exceeded or the incumbent solution's objective value corresponds to the strongest lower bound $\mathrm{MS}^{\mathrm{LB}}$ obtained from [5]. In the latter case a proven optimal solution has been found.

Algorithm 1. General Variable Neighborhood Search

```
1: **Input:** initial solution $\pi$, $N^I_{i=1,\dots,k_{\max}}$, $N^S_{j=1,\dots,l_{\max}}$
2: $\pi^{\text{best}} \leftarrow \pi$; $l \leftarrow 1$
3: **repeat**
4:     $\pi' \leftarrow \text{Shake}(N^S_l, \pi^{\text{best}})$                ▷ diversification
5:     $\pi'' \leftarrow \text{VND}(N^I, \pi')$                ▷ intensification
6:     $l \leftarrow l + 1$
7:     **if** $\text{MS}(\pi') < \text{MS}(\pi^{\text{best}})$ **then**        ▷ new incumbent solution found
8:         $\pi^{\text{best}} \leftarrow \pi'$; $l \leftarrow 1$
9:     **else if** $l > l_{\max}$ **then**    ▷ continue with next shaking neighborhood structure
10:        $l \leftarrow 1$
11:    **end if**
12: **until** $\text{MS}^{\text{LB}} = \text{MS}(\pi^{\text{best}}) \vee$ time-limit reached
13: **return** $\pi^{\text{best}}$
```

4.1 Solution Representation and Evaluation

As mentioned in Sect. 3, our VNS interprets solutions to the JSOCMSR as linear permutations that state the order in which the jobs acquire the common resource 0. To obtain the makespan $\text{MS}(\pi)$ of such a permutation π, the exact starting time s_j for each job $j \in J$ must be determined. This is done by a linear time decoder that greedily schedules each job as soon as its resources become available. As it becomes quite inefficient to naively apply this decoder during neighborhood evaluation, we propose an incremental evaluation scheme in which it is not always necessary to (re-)determine the starting time for each job to obtain its makespan.

However, due to the incremental nature of the decoding mechanism and a solution's consequential characteristic, that even small structural changes—like the removal of a job from its current position–potentially propagate to distant sections in the solution, a strictly constant-time incremental evaluation schema is not possible. Instead, we concentrated on an alternative approach, where a certain subsection of a neighboring solution is evaluated until a point of *synchronization* with respect to the incumbent solution is identified. After this point, no structural differences besides a fixed time offset occur. This point of synchronization in the permutation resides at the end of a so-called *synchronization border*, consisting of a minimal set of jobs on different secondary resources which are aligned w.r.t. their starting times in the incumbent solution and the respective neighboring solution in the same way. In the following we define this formally.

Definition 1 (Synchronization Border). *Given two solutions π, π' and the respective normalized starting times s and s', where π' is a neighbor of π w.r.t. some neighborhood structure $\mathcal{N}$. Assume further that the underlying permutation of jobs has only changed up to position i, $0 \le i < n$. The synchronization point is then the smallest position i' with $i < i' \le n$, where a set of jobs $\mathcal{B} \subseteq \{\pi_k \mid k = i+1, \dots, i'\}$, denoted as the synchronization border, satisfies the following conditions:*

1. *The set contains exactly one job for each secondary resource that is still claimed by a job in the permutation at or after the synchronization point i'.*
2. *The jobs are aligned with respect to their starting times in the same way in s and s', i.e., $\exists c \in \mathbb{Z}\ \forall j \in \mathcal{B} : s_j - s'_j = c$.*

In order to evaluate the makespan of a neighbor π' of the incumbent solution π, our approach starts at the first position in the permutation subject to the structural change induced by the move in the neighborhood and scans through the permutation to identify the synchronization border. As soon as the synchronization border is established we are able to determine the *alignment offset* c, i.e., the time difference between the solutions concerning the border, and, consequently, can immediately derive the makespan $\mathrm{MS}(\pi')$ of the neighbor solution π'. Figure 1 illustrates this approach, where a neighboring solution π' on the bottom is derived from π by removing job 4 from position 9 and reinserting it at its new position 4. In this example, the synchronization border $\mathcal{B} = \{5, 7, 6\}$ can be determined already after three steps, allowing to derive the makespan of π' already at position 8.

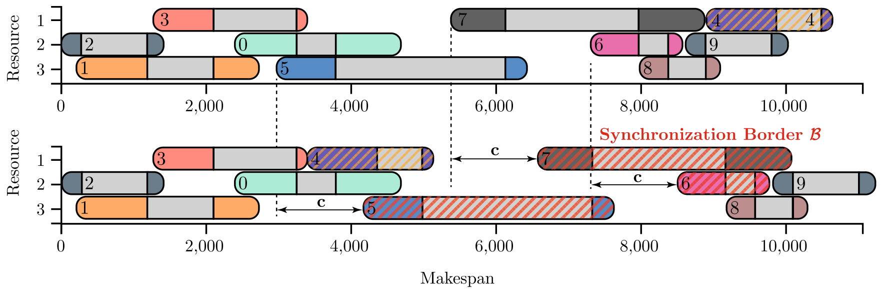

Fig. 1. Illustration of an incumbent solution (top) and a neighboring solution obtained after moving job 4 (bottom) and their synchronization border $\{5, 7, 6\}$.

As identifying the synchronization border in a naive iterative way requires time $\mathcal{O}(nm)$ in the worst case, we use additional auxiliary data structures for each incumbent solution that frequently allow to skip certain parts of the scan through the permutation. In this way the synchronization border can typically be identified much quicker and as a consequence the exploration of the neighborhoods is more efficient. Besides simple lookup tables to detect, for instance, the last job on a particular resource, most importantly, our approach relies on a data structure $\alpha(\pi) = (\alpha_{i,r}(\pi))_{i=1,\ldots,n,\, r\in R_0}$ indicating for each position i in permutation π the time from which on each resource r is available for scheduling a job at this position i. Thus, $\alpha_{i,r}(\pi)$ can be used to quickly determine the starting time of a job which should be inserted in π at position i. As all our neighborhood structures are essentially defined by removing and re-inserting jobs in the permutation representation in certain ways, this data structure allows to immediately determine the starting time of an inserted job at any position, subsequently requiring only the identification of the synchronization border to determine the implied change in the makespan. Although the preparation of these data structures comes with an additional computational cost of $\mathcal{O}(nm)$ per incumbent solution for which the VND is started, our experiments in Sect. 5 indicate that in practice the whole approach requires only constant amortized runtime with respect to the number of jobs for identifying the synchronization border and thus the makespan of a neighboring solution.

4.2 Intensification

The VND, which is responsible for intensification within the GVNS, makes use of a set of neighborhood structures for linear permutations, as formally defined by Schiavinotto and Stützle [15].

The *insertion neighborhood* $\mathcal{N}_I(\pi)$ of an incumbent solution π consists of any solution π' obtained by removing any job j from its current position in π and reinserting it at any other position. We efficiently evaluate the whole neighborhood by considering the removal of each job $j \in J$ in an outer loop, yielding a partial solution $\pi \ominus j$ for which the corresponding auxiliary data structure $\alpha(\pi \ominus j)$ is derived and the partial neighborhood $\mathcal{N}'_I(\pi \ominus j, j)$ corresponding to the re-insertion of j at any position except the original one is evaluated in an inner loop. Algorithm 2 shows in more detail how the neighbor solution in which job j is re-inserted at a position i in the partial solution $\pi \ominus j$ is evaluated by determining the synchronization border and the respective alignment offset.

Based on this evaluation scheme, it turned out to be advantageous in the implementation to further divide the insertion neighborhood $\mathcal{N}_I(\pi)$ into forward and backward insertion neighborhoods such that jobs are only allowed to move forward or backward in the permutation, respectively. This allows to reuse some part of the auxiliary data structures for the entire neighborhood evaluation.

Algorithm 2. Evaluation of the neighbor in which job j is reinserted at position i

```
 1: Input: partial solution $\pi \ominus j$, insertion position $i$, resource availability times $\alpha(\pi \ominus j)$
 2: $t_r \leftarrow \alpha_{i,r}(\pi \ominus j)$, $\forall r \in R_0$
 3: synchronization border $B = \emptyset$, aligned offset $c \leftarrow 0$
 4: for $k = i, \ldots, |\pi \ominus j|$ do        ▷ evaluate $\pi \ominus j$ from insert position onwards
 5:     $j' \leftarrow (\pi \ominus j)_k$
 6:     $s_{j'} \leftarrow \max\{t_0 - p^{\mathrm{pre}}_{j'}, t_{q_{j'}}\}$        ▷ evaluate new starting time for $j'$
 7:     $t_0 \leftarrow s_{j'} + p^{\mathrm{pre}}_{j'} + p^0_{j'}$; $t_{q_{j'}} \leftarrow s_{j'} + p^{\mathrm{pre}}_{j'}$
 8:     update $B$ with job $j'$
 9:     if $B$ satisfies conditions from Definition 1 then
10:         $c \leftarrow$ derive alignment offset from $B$ and $\pi$
11:         break
12:     end if
13: end for
14: return $\mathrm{MS}(\pi \ominus j) + c$
```

The exchange neighborhood $\mathcal{N}_X(\pi)$, contains any solution derived from the incumbent π by exchanging any pair of jobs in the permutation. Again, the neighborhood evaluation is based on determining synchronization borders, but instead of using intermediate partial solutions, a dual synchronization approach has been devised, where the neighborhood operation is essentially reduced to two insertion operations, where both the offset between the respective exchanged jobs as well as the offset of the latter job to the makespan are obtained with the synchronization technique.

In addition to efficient evaluation schemes for the considered neighborhood structures, we further studied different approaches to reduce neighborhood sizes in order to avoid the evaluation of unpromising neighbors at all. Besides neighborhood reduction based on critical jobs as proposed already by Horn et al. [5], we also considered heuristic approaches like avoiding to schedule two jobs of the same secondary resource consecutively or reducing the size of neighborhoods by limiting the maximum distance of move operations. While these pruning techniques bring the danger of quickly approaching local optima of rather poor quality, concentrating on critical jobs is particularly advantageous in the very beginning of the search. Limiting the maximum distance of move operations particularly showed its effectiveness for exchange neighborhoods, where instead of the dual synchronization evaluation scheme, it becomes more the better option to partially evaluate the entire range between the positions of the two exchanged jobs and perform a single synchronization step at the end of this range. Experimentally, we determined a move distance limitation of $k = 50$ to provide a good trade-off between the size of the neighborhood and its evaluation's efficiency in the context of our benchmark instances. Nevertheless note that these restricted neighborhoods are primarily used in early VND phases, while more comprehensive neighborhoods become important in latter phases to compensate the limitations. More details on the pruning techniques and their impacts can be found in the first author's master thesis [10]. Here, we will only look more closely on the limitation of move distances.

We used findings of a landscape analysis, where the average quality and depth of local optima were studied to prepare a meaningful parameter tuning configuration, and then applied `irace` [11] to select concrete neighborhood structures and parameters like the step function by which the neighborhoods are searched in the VND. For details regarding the parameter tuning setup we refer to [10]. Finally, we investigated the temporal behavior of our algorithm in a set of experiments to decide the neighborhood change function in the VND [3]. Again, more details on this preliminary investigations can be found in [10].

The finally resulting VND configuration uses four neighborhood structures, subject to a piped neighborhood change function [3]. First, an exchange neighborhood structure with a move distance limitation of 50 is used in conjunction with a first-improvement step function to quickly identify local optima of already relatively high quality. This is followed by the backward insertion neighborhood structure searched in a best improvement manner. Next, the unconstrained exchange neighborhood structure is used and finally the unconstrained insertion neighborhood structure, again searched in first and best improvement manners, respectively.

4.3 Diversification

For diversification, the GVNS applies moves from a total of $l_{\max} = 23$ shaking neighborhood structures to the incumbent solution, where each shaking neighborhood N_i^S is parametrized by κ_i describing the number of subsequent applications of the underlying neighborhood move. In order to enable our shaking procedure to introduce fine-grained structural changes into the incumbent solution, we use the exponentially growing function $\kappa_i = \lceil \exp(\frac{i \cdot log(n)}{\kappa_{\max} - 1}) \rceil$, with a maximum number of applied moves per shaking neighborhood of $\kappa_{\max} = 32$, to generate two sets of 10 insertion and exchange shaking neighborhood structures respectively. Starting with insertions, those sets are then interleaved and at positions four, ten and twenty extended by a subsequence inversion shaking neighborhood applying one, two and four inversions of five jobs respectively.

This configuration was mainly hand-crafted based on characteristics of the ruggedness of the respective neighborhood structures and the general structure of the search space. We used the autocorrelation function on random walks of length 10^6 to estimate the ruggedness of neighborhood structures and analyzed a large set of globally optimal solutions obtained from 3.75×10^6 runs on a diverse set of 300 instances with $n = 30$ jobs, to gain insight on the distribution of globally optimal solutions in the search space. We found that the studied instances contain a relatively high number of distinct globally or at least nearly optimal solutions, being widely distributed in the search space. A primary reason for this is likely the dependency structure inherent to the problem and the induced symmetries, caused by resource imbalance or utilization gaps on secondary resources, frequently allowing to exchange of jobs on secondary resources without affecting the makespan. For more details, see [10].

5 Computational Results

In our computational study we analyzed the practical applicability and impact of the proposed incremental evaluation technique and compared the GVNS to the baselines provided by Horn et al. [5]. These are the anytime A* algorithm and a CP model. The experiments were conducted on two sets of instances with different characteristics with respect to the workload distribution among the available resources [5]. Balanced instances in set B have the workload uniformly distributed among the secondary resources and obtained durations p_j^{pre} and p_j^{post} for the pre-processing and post-processing of jobs by sampling the discrete uniform distribution $\mathcal{U}\{0, 1000\}$ and durations p_j^0 of the main processing phases by sampling $\mathcal{U}\{1, 1000\}$. Instances in set S, on the other hand, show a skewed workload distribution, both with respect to the assignment to secondary resources and the utilization of the common resource 0. In skewed instances, a job is assigned to the secondary resource 1 with probability 0.5, while the probability for the remaining secondary resources $m > 1$ is $1/(2m-2)$. Both sets consist of instances with $n \in \{50, 100, 200, 500, 1000, 2000\}$ jobs and $m \in \{2, 3, 5\}$ secondary resources with 50 randomly sampled instances for each (n, m) pair. The instance sets are available at https://www.ac.tuwien.ac.at/research/problem-instances/. The proposed GVNS was implemented in C++ using G++ 7.4.0 with *-Ofast* optimization level. The experiments were conducted on a computing cluster of 16 machines, each with two *Intel Xeon E5-2640 v4* CPUs with 2.40 GHz in single threaded mode and 15 GB RAM. All considered approaches where executed with a maximum CPU time limit of 900s. The baseline CP model was solved with ILOG CP Optimizer 12.7.1.

In order to study the practical efficiency of our incremental evaluation approach, an experiment was conducted where 10^4 randomly selected neighborhood moves in exchange and insertion neighborhoods where applied and the distance from the structural change to the last job in the synchronization border—that is the number of steps until the synchronization border could be determined—was traced. Figure 2 shows the synchronization distance of balanced and skewed instances of different sizes. For the considered instances, it can be observed that our approach exhibits an average amortized runtime behavior that is constant in the number of jobs, but increases with the number of secondary resources due to the nature of the synchronization border. Moreover, Fig. 2 illustrates the sensitivity of the approach to significant resource imbalance, indicated by a higher number of outliers observed in the skewed instance set, likely due to large sections in the schedules where secondary resources are not utilized.

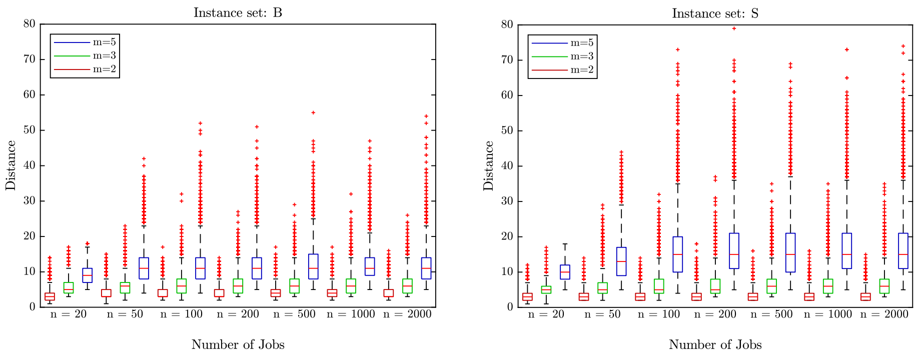

Fig. 2. Synchronization distance: number of steps required to identify the synchronization border in balanced and skewed instances, starting from the position of structural change due to a neighborhood move.

Finally, Table 1 compares average results of our GVNS on different instance classes to the baselines of Horn et al. [5]. Columns %-gap state the final optimality gaps in percent, which is calculated by $100\% \cdot (\mathrm{MS}(\pi) - \mathrm{MS}^{\mathrm{LB}})/\mathrm{MS}^{\mathrm{LB}}$, whereas columns %-opt lists the percentage of proven optimal solutions. Both columns use the best lower bound $\mathrm{MS}^{\mathrm{LB}}$ obtained from Horn et al. [5].

Columns $\sigma_{\%\text{-gap}}$ show the standard deviations of the corresponding average optimality gaps. Column t provides the median time the GVNS required to obtain its best solution in a run. To obtain statistically more stable results, we executed the GVNS ten times for each of the 50 instances per instance class. For the anytime A* algorithm and for the CP solver, column t shows the median time when the algorithms terminated either because the optimal solution has been found or the time- or memory limit was exceeded.

Generally, Table 1 shows that the GVNS manages to obtain heuristic solutions comparable to those of the A* search, while both approaches show their specific advantages on particular subsets of instances. For balanced instances, on the one hand, A* search already showed its effectiveness, where even large instances up to 2000 jobs could be solved to proven optimality. For instances with $m = 2$ and $m = 5$, the GVNS obtains similar results with respect to solution quality, although the temporal performance decreases with increasing instance size in comparison. For instances with $m = 3$ the GVNS's solutions are clearly worse than those of the A* search, although the average optimality-gap of $\leq$0.288% is still small and much better than the one of the CP approach. In Kaufmann [10] we show that providing an initial solution obtained with the least lower-bound construction heuristic of Horn et al. [5] can further improve the solution quality for this particular instance set, however, A* is still superior both with respect to quality as well as temporal behavior.

For the harder skewed instances, on the other hand, our GVNS shows a significant improvement compared to both baseline methods with an average optimality gap below 0.214%. Instances with two secondary resources tend to be among the more difficult ones, where even for small instances with 50 jobs, optimality could be proven with the lower bound only in 42% of the runs. This, however, could as well be an indicator for the lower bounds being off the optimum. Interestingly, the GVNS still shows an improvement with respect to the number of obtained proven optimal solutions, where particularly for small to moderately large instances up to 88% could be solved to proven optimality, despite the inherent incompleteness of the GVNS.

6 Conclusions

In this work, we presented a GVNS to heuristically tackle the JSOCMSR, a combinatorial optimization problem encountered for example in novel cancer treatment facilities. We devised a generally applicable approach to efficiently evaluate solutions in the course of a neighborhood search in incremental ways and applied it to variants of insertion and exchange neighborhood structures. Insertion and exchange moves where utilized in the intensification phase, a piped VND, as well as in the diversification phase as for randomized shaking.

Our experimental analysis first dealt with the practical efficiency of the incremental evaluation scheme, which still has a linear runtime in the number of jobs in the worst-case but exhibits a essentially a constant average runtime on all our benchmark instances. When comparing the GVNS to the state-of-the-art A*

Table 1. Average results of GVNS, A* search, and the CP approach.

Type	n	m	GVNS				Anytime A*				CP/ILOG			
			%-gap	$\sigma_{\%\text{-gap}}$	%-opt	t[s]	%-gap	$\sigma_{\%\text{-gap}}$	%-opt	t[s]	%-gap	$\sigma_{\%\text{-gap}}$	%-opt	t[s]
B	50	2	**0.000**	0.00	100.0	<0.1	**0.000**	0.00	100.0	1.1	**0.000**	0.00	100.0	<0.1
B	100	2	**0.000**	0.00	100.0	<0.1	**0.000**	0.00	100.0	2.0	**0.000**	0.00	100.0	<0.1
B	200	2	**0.000**	0.00	100.0	0.2	**0.000**	0.00	100.0	5.4	**0.000**	0.00	100.0	<0.1
B	500	2	**0.000**	0.00	100.0	2.4	**0.000**	0.00	100.0	35.3	**0.000**	0.00	100.0	1.3
B	1000	2	**0.000**	0.00	100.0	13.0	**0.000**	0.00	100.0	8.9	**0.000**	0.00	100.0	9.2
B	2000	2	**0.000**	0.00	100.0	83.5	**0.000**	0.00	100.0	46.3	<0.001	0.01	98.0	63.5
B	50	3	0.050	0.22	91.2	<0.1	**0.017**	0.08	96.0	1.1	0.068	0.30	92.0	<0.1
B	100	3	0.112	0.29	79.6	0.1	**0.021**	0.09	92.0	2.0	0.226	0.55	78.0	4.2
B	200	3	0.176	0.45	74.0	2.1	**0.016**	0.06	92.0	5.9	0.556	1.12	56.0	319.4
B	500	3	0.260	0.42	47.0	422.3	**<0.001**	<0.01	98.0	35.9	2.212	1.83	20.0	900.0
B	1000	3	0.216	0.33	31.0	385.0	**0.001**	<0.01	98.0	6.1	3.094	1.46	2.0	899.9
B	2000	3	0.288	0.34	15.0	843.2	**0.005**	0.04	98.0	23.8	4.220	1.20	0.0	900.0
B	50	5	<0.001	<0.01	99.4	0.1	**0.000**	0.00	100.0	1.2	**0.000**	0.00	100.0	0.7
B	100	5	**0.000**	0.00	100.0	0.4	**0.000**	0.00	100.0	2.2	**0.000**	0.00	100.0	9.5
B	200	5	**0.000**	0.00	100.0	2.3	<0.001	0.00	98.0	6.5	**0.000**	0.00	100.0	91.3
B	500	5	**0.000**	0.00	100.0	14.3	**0.000**	0.00	100.0	42.3	<0.001	<0.01	86.0	499.7
B	1000	5	<0.001	<0.01	96.0	49.2	**0.000**	0.00	100.0	7.9	0.359	0.12	0.0	900.0
B	2000	5	<0.001	<0.01	86.6	128.8	**0.000**	0.00	100.0	30.4	0.478	0.14	0.0	900.0
S	50	2	**0.163**	0.23	42.0	4.8	0.268	0.38	40.0	11.4	0.210	0.28	42.0	899.9
S	100	2	**0.172**	0.32	33.8	115.5	0.367	0.49	26.0	44.8	0.323	0.47	12.0	900.0
S	200	2	**0.111**	0.18	14.8	606.0	0.440	0.33	2.0	65.2	0.642	0.51	0.0	900.0
S	500	2	**0.095**	0.08	0.0	831.7	0.532	0.18	0.0	88.7	2.736	0.51	0.0	900.0
S	1000	2	**0.105**	0.06	0.0	813.3	0.725	0.20	0.0	176.8	4.636	0.43	0.0	900.0
S	2000	2	**0.214**	0.11	0.0	892.6	0.786	0.18	0.0	252.7	4.784	0.39	0.0	900.0
S	50	3	**0.035**	0.15	82.0	0.2	0.053	0.21	82.0	1.3	**0.035**	0.15	80.0	27.7
S	100	3	**0.030**	0.10	82.8	3.5	0.153	0.37	50.0	16.5	0.060	0.15	52.0	899.7
S	200	3	**0.025**	0.11	78.8	21.5	0.117	0.26	34.0	26.4	0.135	0.21	36.0	899.8
S	500	3	**0.006**	0.02	42.4	370.5	0.177	0.24	14.0	121.6	1.360	0.76	4.0	900.0
S	1000	3	**0.009**	0.02	19.2	584.5	0.621	0.47	2.0	48.0	2.872	0.93	0.0	900.0
S	2000	3	**0.041**	0.05	5.8	863.3	0.701	0.41	0.0	80.2	4.296	0.98	0.0	900.0
S	50	5	0.046	0.14	83.7	<0.1	0.077	0.19	80.0	1.4	**0.045**	0.14	84.0	15.0
S	100	5	**0.006**	0.02	88.4	1.2	0.064	0.18	66.0	6.1	0.019	0.04	70.0	899.6
S	200	5	**0.034**	0.14	77.2	18.3	0.281	0.49	34.0	38.8	0.161	0.25	28.0	900.0
S	500	5	**0.009**	0.02	46.8	351.7	0.347	0.34	16.0	188.6	1.229	0.95	8.0	899.9
S	1000	5	**0.012**	0.02	22.2	625.3	0.702	0.50	0.0	387.3	2.478	1.11	0.0	900.0
S	2000	5	**0.105**	0.10	2.0	893.6	0.915	0.54	0.0	789.3	4.229	1.22	0.0	900.0

search and the CP model, we observed the GVNS's ability to obtain high-quality solutions for a diverse set of large instances with an average optimality-gap of $\leq$0.288%. Although for balanced instances, the anytime A* algorithm of Horn et al. [5] was out of reach for particularly hard instances, our approach showed its effectiveness on harder instances with skewed workloads, where the state of the art could be improved significantly. In future work it would be interesting to investigate the runtime of the incremental evaluation scheme also from a theoretical point-of-view, in the hope that the constant amortized time observed here in practice can even be proven for a larger class of instances. Moreover, it appears promising to apply the underlying ideas of the proposed incremental evaluation scheme also in the context of related scheduling/sequencing problems and local search based metaheuristics.

References

1. Conforti, D., Guerriero, F., Guido, R.: Optimization models for radiotherapy patient scheduling. 4OR **6**(3), 263–278 (2008)
2. Hansen, P., Mladenović, N., Pérez, J.A.M.: Variable neighbourhood search: methods and applications. Ann. Oper. Res. **175**(1), 367–407 (2010)
3. Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S.: Variable neighborhood search: basics and variants. EURO J. Comput. Optim. **5**(3), 423–454 (2016). https://doi.org/10.1007/s13675-016-0075-x
4. Horn, M., Maschler, J., Raidl, G., Rönnberg, E.: A*-based construction of decision diagrams for a prize-collecting scheduling problem. Technical report AC-TR-18-011, Algorithms and Complexity Group, TU Wien (2018)
5. Horn, M., Raidl, G., Blum, C.: Job sequencing with one common and multiple secondary resources: an A*/Beam Search based anytime algorithm. Artif. Intell. **277**(103173) (2019)
6. Horn, M., Raidl, G., Rönnberg, E.: A* search for prize-collecting job sequencing with one common and multiple secondary resources. Ann. Oper. Res. (2020)
7. Horn, M., Raidl, G.R., Rönnberg, E.: An A^* algorithm for solving a prize-collecting sequencing problem with one common and multiple secondary resources and time windows. In: Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling, PATAT 2018, pp. 235–256 (2018)
8. Horn, M., Raidl, G., Blum, C.: Job sequencing with one common and multiple secondary resources: a problem motivated from particle therapy for cancer treatment. In: Nicosia, G., Pardalos, P., Giuffrida, G., Umeton, R. (eds.) MOD 2017. LNCS, vol. 10710, pp. 506–518. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72926-8_42
9. 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 (2006)
10. Kaufmann, T.: A variable neighborhood search for the job sequencing with one common and multiple secondary resources problem. Master's thesis, TU Wien, Vienna, Austria (2019)
11. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
12. Maschler, J., Raidl, G.R.: Multivalued decision diagrams for a prize-collecting sequencing problem. In: Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling, PATAT 2018, pp. 375–397 (2018)
13. Maschler, J., Riedler, M., Stock, M., Raidl, G.R.: Particle therapy patient scheduling: first heuristic approaches. In: Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling, PATAT 2016, pp. 223–244 (2016)
14. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)

15. Schiavinotto, T., Stützle, T.: A review of metrics on permutations for search landscape analysis. Comput. Oper. Res. **34**(10), 3143–3153 (2007). https://doi.org/10.1016/j.cor.2005.11.022
16. 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)