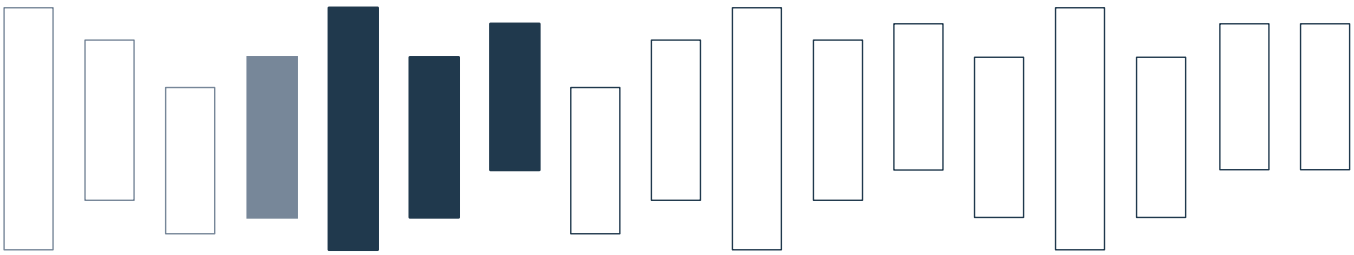
Technical Report AC-TR-19-003

January 2019

Multivalued Decision Diagrams for Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources

Johannes Maschler and Günther R. Raidl



Multivalued Decision Diagrams for Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources*

Johannes Maschler[1], Günther Raidl[1]

[1]Institute of Computer Graphics and Algorithms, TU Wien, Austria

{maschler|raidl}@ac.tuwien.ac.at

Multivalued decision diagrams (MDD) are a powerful tool for approaching combinatorial optimization problems (COPs). Relatively compact relaxed and restricted MDDs are applied to obtain dual bounds and heuristic solutions and provide opportunities for new branching schemes. We consider a prize-collecting sequencing problem in which a subset of given jobs has to be found that is schedulable and yields maximum total prize. The primary aim of this work is to study different methods for creating relaxed MDDs for this problem. To this end, we adopt and extend the two main MDD compilation approaches found in the literature: top down construction and incremental refinement. In a series of computational experiments these methods are compared. The results indicate that for our problem the incremental refinement method produces MDDs with stronger bounds. Moreover, heuristic solutions are derived by compiling restricted MDDs and by applying a general variable neighborhood search (GVNS). Here we observe that the top down construction of restricted MDDs is able to yield better solutions as the GVNS on small to medium-sized instances.

Keywords. Sequencing multivalued decision diagrams incremental refinement particle therapy patient scheduling

*We thank Lukas Felician Krasel for his help in the implementation and testing.

1 Introduction

Horn et al. (2017) originally described the so-called Job Sequencing with One Common and Multiple Secondary Resources (JSOCMSR) problem. We consider an extended and in practice significantly more difficult prize-collecting variant of it which we denote by PC-JSOCMSR. Given a set of jobs, each associated with a prize, the task is to find a subset of jobs with maximal total prize that is feasibly schedulable. Each job requires one of several secondary resources during its whole processing time and a single common resource for a part of its execution. Moreover, each job has to be performed within given time windows. This problem originates from the context of particle therapy for cancer treatment (Maschler et al. 2016, Maschler and Raidl 2018b, Riedler et al. 2017). In this scenario the common resource corresponds to a particle beam that can be directed into one of multiple treatment rooms which are represented by the secondary resources. Jobs describe treatments that consist of several tasks within a treatment room from which only one is the actual irradiation using the beam. While the works concerning particle therapy deal with numerous additional characteristics stemming from the real world application, it is apparent that the most central aspect is the sequencing of the jobs.

In this work we explore the potential of applying the concept of decision diagrams (DDs) to PC-JSOCMSR and in particular investigate different methods for creating them. DDs have been originally developed in the context of circuit design (Lee 1959). In the course of the last decade DDs have shown to be also a powerful tool for tackling combinatorial optimization problems (COPs) (Bergman et al. 2016a). Essentially, DDs are layered directed acyclic multigraphs used to compactly represent a COP's set of feasible solutions. To this end, a DD has a root node and each subsequent layer of the DD is associated with one of the decision variables of the COP. Every arc in the DD describes an assignment of the variable represented by the corresponding layer. Thus, a path starting from the root node represents a variable assignment. The lengths of the arcs are assigned in such a way that the length of a path corresponds to the objective value of the corresponding variable assignment. Depending on whether the COP's objective is to maximize or to minimize a given objective function, we are seeking a longest or a shortest inclusion maximal path to a valid terminal node within the DD. The out-degrees of the DD's nodes directly corresponds with the domain sizes of the respective decision variables. If the COP is modeled with binary variables, then all nodes have at most two outgoing arcs and the DD is called binary decision diagram (BDD). In the more general case with finite variable domains, the number of arcs leaving nodes is not restricted. In this case, DDs are called multivalued decision diagrams (MDDs).

DDs resemble in many aspects a dynamic programming's state graph (Hooker 2013). Likewise, the size of exact DDs grows in general exponentially with the problem size. To overcome the resulting limitations, Andersen et al. (2007) proposed the concept of relaxed DDs. The basic idea is to merge nodes on the same layer and to redirect the affected arcs. This might introduce new paths in the DD that, however, do not represent feasible solutions. Consequently, relaxed DDs encode a superset of the feasible solutions and represent a discrete relaxation of the problem that provides dual bounds. Another way to cope with the in general exponential number of nodes are restricted DDs (Bergman et al. 2014b). A restricted DD is obtained from an exact DD by removing nodes and all incident arcs. Clearly, this also removes all paths from the DD that included at least one of the removed nodes. Therefore, a restricted DD represents only a subset of all feasible solutions, and it is used to obtain a feasible heuristic solution and a respective primal bound.

Beside upper and lower bounds, relaxed DDs in particular also provide promising opportunities for new inference techniques in constraint programming (Cire and Hoeve 2013, Kinable et al. 2017), novel branching schemes (Bergman et al. 2016b) for branch-and-bound, as well as primal heuristics (Bergman et al. 2014b, 2016b).

The concept of DDs has been successfully applied to a variety of problems, ranging from binary optimization problems to sequencing problems. The former include set covering (Bergman et al. 2011, 2014b), maximum independent set (Bergman et al. 2014a, 2016b), maximum cut (Bergman et al. 2016b), and maximum 2-satisfiability (Bergman et al. 2016b) problems and are approached using

BDDs. Sequencing problems on the other hand typically suggest the use of MDDs. In the literature already considered sequencing problems include the time dependent traveling salesman problem with and without time windows and the time-dependent sequential ordering problem (Cire and Hoeve 2013, Kinable et al. 2017). For a comprehensive overview on DDs see Bergman et al. (2016a).

One fundamental difference to the DDs considered in the literature is the prize-collecting aspect. While the so far considered problems define solutions by paths traversing all layers, in PC-JSOCMSR every path starting at the root node corresponds to a valid solution. Two main approaches have been proposed for compiling MDDs. The first starts at the root node and constructs the MDD layer by layer (Bergman et al. 2011, 2014b). If the number of nodes within a layer exceeds a given limit, then either nodes are merged or removed to obtain a relaxed or a restricted MDD, respectively. The second approach, starts with a simplistic relaxed MDD and applies incremental refinements by splitting nodes in order to iteratively strengthen the relaxation (Cire and Hoeve 2013, Kinable et al. 2017). We adapt both approaches for PC-JSOCMSR here and are, to our knowledge, the first who directly compare the two techniques in an experimental fashion. Moreover, we investigate the derivation of heuristic solutions by constructing a restricted MDD and provide an independent general variable neighborhood search (GVNS) (Hansen et al. 2010) to set the DD-based approaches into perspective. Our computational experiments show that the incremental refinement approach provides on most of our benchmark instances better dual bounds than the top down compilation. While the top down compilation for restricted MDDs outperforms the GVNS on small to medium-sized instances, the GVNS is mostly superior on larger instances. This work is a revised and extended version of the preliminary conference paper from Maschler and Raidl (2018a). New is in particular a technique that detects and removes redundancies during the incremental refinement of MDDs. This redundancy detection and removal yields to significantly smaller DDs.

The remainder of this work is organized as follows. In the following we start by giving a formal description of the considered problem. Section 3 provides a recursive dynamic programming model for PC-JSOCMSR which serves as basis for deriving MDDs in Section 4. Section 5 describes the top down compilation of relaxed and restricted MDDs, while the incremental refinement algorithm for PC-JSOCMSR is given in Section 6. Section 7 sketches the standalone GVNS. Results of computational experiments of all approaches are discussed in Section 8. Finally, Section 9 concludes with an outlook on future research directions.

2 The Problem

The Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources (PC-JSOCMSR) problem is formally defined as follows. Let $J = \{1, \dots, n\}$ be a set of n jobs of which a subset shall be scheduled using renewable resources $R_0 = \{0\} \cup R$ with $R = \{1, \dots, m\}$. To be processed, each job $j \in J$ requires a resource $q_j \in R$ for its entire processing time $p_j > 0$ and additionally resource 0 for a duration of p_j^0 after time p_j^{pre} from the job's start; $0 < p_j^0 \leq p_j - p_j^{\text{pre}}$. For convenience, we denote with p_j^{post} the duration after the common resource is used until the job j is completed, i.e., $p_j^{\text{post}} = p_j - p_j^{\text{pre}} - p_j^0$. Moreover, we write J_r for the subset of all jobs in J which require secondary resource $r \in R$.

We associate with each job j a set of time windows $W_j = \bigcup_{w=0,\dots,\omega_j} W_{j,w}$ with $W_{j,w} = [W_{j,w}^{\text{start}}, W_{j,w}^{\text{end}}]$, where $W_{j,w}^{\text{end}} - W_{j,w}^{\text{start}} \geq p_j$. Jobs can only be performed within these time windows and are assumed to be non-preemptive, i.e., may not be interrupted. We denote the whole relevant time horizon, encompassing all time windows of all jobs, with $[T^{\min}, T^{\max}]$.

Finally, each job j has associated a prize (utility value, priority) $z_j > 0$. We assume that there exists, in general, no schedule that assigns feasible starting times to all jobs in J. Instead, we aim for a subset of jobs $S \subseteq J$ that can be feasibly scheduled and maximizes the total prize of these jobs, i.e.,

$$Z(S) = \sum_{j \in S} z_j. \tag{1}$$

A schedule of S implies a total ordering of the scheduled jobs because all jobs require resource 0 and this resource can be used by only one job at a time. Vice versa, such an ordering $\pi = (\pi_i)_{i=1,\dots,|S|}$ of S can be decoded into a schedule by scheduling each job from S in the order given by π at the earliest feasible time after the preceding job. If at least one of the jobs cannot be feasibly scheduled in this way, then ordering π does not represent a feasible solution. We call the schedule obtained from ordering π by the above decoding a normalized schedule. Clearly, for every feasible solution there exists a normalized schedule with the same objective value. Hence, we write $Z(\pi)$ for the total prize of the normalized solution given by the ordering π of jobs.

Above problem variant extends the Job Sequencing with One Common and Multiple Secondary Resources (JSOCMSR) problem originally proposed by Horn et al. (2017) by the considered time windows and the maximization of the scheduled jobs' prizes. In Horn et al.'s JSOCMSR, the objective is to minimize the makespan. Horn et al. showed that the decision variant of JSOCMSR is NP-hard for $m \geq 2$. PC-JSOCMSR is NP-hard as well, which can be shown by a simple reduction. To this end, we construct an instance for PC-JSOCMSR by associating each job with a single time window $[0, M]$, where M is the given constant for the makespan. There exists a solution for the decision variant of JSOCMSR if and only if there exists a solution for the constructed PC-JSOCMSR instance in which all jobs can be scheduled.

In a concurrently submitted work, Horn et al. (2018) focus on solving the PC-JSOCMSR exactly by means of A* search, mixed integer programming, and constraint programming. While excellent results are obtained in particular for the A* search, the applicability of these methods is strongly limited to rather small or medium sized-problem instances. A sequencing problem with similar job characteristics to ours, requiring one common and a secondary resource, has been considered by Van der Veen et al. (1998). However, in their case post-processing times are negligible and as a result the problem reduces to a special variant of the traveling salesman problem that can be efficiently solved in polynomial time. Last but not least, we point out that PC-JSOCMSR is somewhat related to variants of no-wait flowshop problems (Allahverdi 2016) and more general resource-constrained project scheduling problems (Hartmann and Briskorn 2010).

3 Recursive Model for PC-JSOCMSR

We provide a dynamic-programming-like recursive model for PC-JSOCMSR. The induced state graph will then serve as a basis for deriving MDDs. To simplify the handling of time windows let us define the function earliest feasible time $\mathrm{eft}(j, t)$ that computes for a given job j and time point t the earliest time not smaller than t at which job j can be performed according to the time windows, i.e.,

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

Note that $\mathrm{eft}(j, t) = T^{\max}$ if job j cannot be scheduled within its time windows.

The main components of the recursive model are the states, the control variables that conduct transitions between states, and finally the prizes associated with the transitions. In our recursive formulation a state is a tuple (P, t) consisting of the set $P \subseteq J$ of jobs that are still available for scheduling and a vector $t = (t_r)_{r \in R_0}$ of the earliest times from which on each resource r is available. The initial state corresponding to the original PC-JSOCMSR instance without any jobs scheduled yet is $\mathbf{r} = (J, (T^{\min}, \dots, T^{\min}))$.

The control variables are $\pi_1, \dots, \pi_n \in J$. Starting from the root node they select the jobs to be scheduled. Variable π_1 selects the first job j to be scheduled, and we transition from state $\mathbf{r}$ to a successor state (P', t'), where π_2 decides with which next job to continue. This is repeated for all control variables. If a job selected by a control variable cannot be feasibly scheduled as next job, then we obtain the special infeasible state $\hat{0}$. Any further transition from $\hat{0}$ yields $\hat{0}$ again.

To specify the transitions, let the starting time of a next job $j \in J$ w.r.t. a state (P, t) be

$$\mathrm{s}((P, t), j) = \begin{cases} \mathrm{eft}(j, \max(t_0 - p_j^{\mathrm{pre}}, t_{q_j})) & \text{if } j \in P \\ T^{\max} & \text{else.} \end{cases} \tag{3}$$

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

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

with

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

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

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

All states except the infeasible state $\hat{0}$ are possible final states. The prize associated with a state transition is job j's prize z_j. Any sequence of state transitions $\tau(\dots\tau(\mathbf{r}, \pi_1)\dots, \pi_i)$ yielding a feasible state (P, t) from the initial state $\mathbf{r}$ represents a solution. In fact, the respective states map directly to the normalized schedule obtained by decoding the jobs $\pi_1, \dots, \pi_i$ as stated in Section 2. Moreover, the sum of the prizes of all these transitions corresponds to $Z(\pi_1, \dots, \pi_i)$, the total prize of the solution.

Note that a feasible state does not have to describe a single solution, because the same state might be reached by multiple transition sequences. These different transition sequences yielding the same state might also have distinct total prizes. Since we are maximizing the total prize, we are primarily interested in sequences with maximum total prize. To this end, let $Z^{\mathrm{lp}}(P, t)$ be this maximum total prize for any sequence $\tau(\dots\tau(\mathbf{r}, \pi_1)\dots, \pi_i)$ resulting in state (P, t). Ultimately, we are looking for a feasible state with maximum $Z^{\mathrm{lp}}(P, t)$.

Looking at these relationships from a dynamic programming perspective, we can express the maximum total prize for jobs that can still be scheduled from any feasible state (P, t) onward by

$$Z^*(P,t) = \max\{0, z_j + Z^*(\tau((P,t),j)) \mid j \in P \wedge \tau((P,t),j) \neq \hat{0}\}, \tag{8}$$

and $Z^*(\mathbf{r})$ then denotes the overall maximum achievable prize, i.e., the optimal solution value.

Strengthening of States. The individual states obtained by the transitions can be safely strengthened in many cases, typically leading to a smaller state graph. We aim at replacing state (P, t) by state (P', t') with either $P' \subset P$ or $t'_r > t_r$ for one or more $r \in R_0$ without losing possible solutions. This is done by first considering the earliest starting times $\mathrm{s}((P, t), j)$ for all jobs $j \in P$. Jobs that cannot be feasibly scheduled next can be safely removed from P, i.e., $P' = \{j \in P \mid \mathrm{s}((P,t),j) \neq T^{\max}\}$.

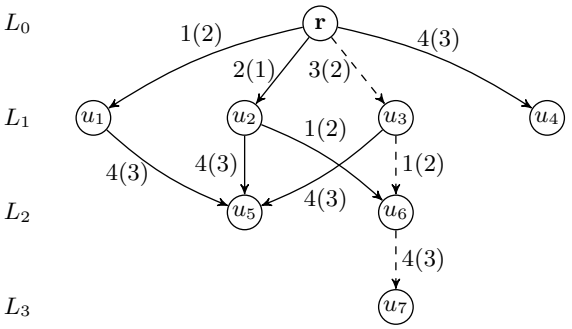
Afterwards, we set the times t'_r, $\forall r \in R_0$, to the earliest time resource r is actually used by the jobs in P'. If a resource is not required by any of the remaining jobs then we set the corresponding time t'_r to $T^{\max}$. More formally,

$$t'_0 = \min_{j \in P'} \left(\mathrm{s}((P,t),j) + p_j^{\mathrm{pre}}\right) \tag{9}$$

$$t'_r = \begin{cases} \min_{j \in J_r \cap P'} \mathrm{s}((P,t),j) & \text{if } J_r \cap P' \neq \emptyset \\ T^{\max} & \text{else} \end{cases} \qquad \forall r \in R. \tag{10}$$

4 Multivalued Decision Diagrams for PC-JSOCMSR

This section explains the relationships between the state graph of a PC-JSOCMSR problem instance and exact, relaxed, and restricted MDDs. An exact MDD is a layered directed acyclic multigraph $G = (V, A)$ with node set V and arc set A. The node set V is partitioned into layers $L_0, \dots, L_n$. The first layer L_0 consists only of a single node associated with the initial state $\mathbf{r}$. Each subsequent layer L_i contains nodes for all states obtained from feasible state transitions from states associated with nodes in layer L_{i-1}. Moreover, the MDD has arcs for all feasible state transitions in the state graph connecting the corresponding nodes. Observe that arcs exist only between directly successive layers



Instance:

j	p_j	p_j^{pre}	p_j^0	q_j	z_j	W_j
1	4	1	2	1	2	[3, 8]
2	3	1	1	1	1	[0, 4]
3	4	0	3	2	2	[0, 4]
4	3	1	2	2	3	[3, 8]

States:

V	P	t
r	$\{1,2,3,4\}$	$(0,0,0)$
u_1	$\{4\}$	$(6,8,5)$
u_2	$\{1,4\}$	$(4,3,3)$
u_3	$\{1,4\}$	$(4,3,4)$
u_4	$\{\}$	$(8,8,8)$
u_5	$\{\}$	$(8,8,8)$
u_6	$\{4\}$	$(6,8,5)$
u_7	$\{\}$	$(8,8,8)$

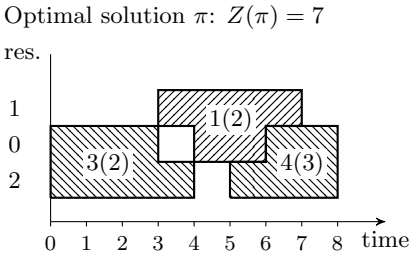


Figure 1: A MDD for an example instance with four jobs and two secondary resources.

and there might be nodes for identical states on different layers. The length of these arcs are the state transition prizes z_j. The infeasible state $\hat{0}$ and all transitions to it are omitted. In the literature, a target node is typically defined and arcs with zero length exist from any feasible end node to this target. Since in our case any node represents a valid end state, we deviate here from the literature and do not make explicit use of this target state.

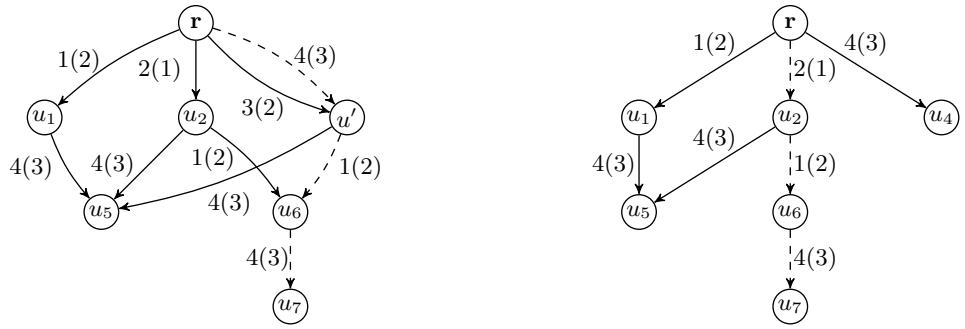
Let us denote by $j(a) \in J$ the job that is considered in the state transition associated with arc $a \in A$. Moreover, let $A^+(u)$ and $A^-(u)$ indicate the set of all incoming and outgoing arcs of a node $u \in V$, respectively. Moreover, for a node u we write $P(u)$ and $t(u)$ as a shorthand for the set P and vector t of the node's state. In particular, we denote with $t_r(u)$ for a node u the time from which on each resource $r \in R_0$ is available for performing a next job.

An optimal solution is obtained from an exact MDD by determining a longest path from **r** to some end node v and scheduling the jobs associated with each arc in the respective order and at the starting times $\text{s}((P,t),j)$. The length of this path, i.e., the sum of the respective arcs' transition prizes, corresponds to the optimal solution value $Z^*(\mathbf{r})$.

Figure 1 shows an exact MDD for an instance with four jobs and two secondary resources. Details of the PC-JSOCMSR instance are given on the top right, while the MDD is depicted on the top left. Each arc's label indicates the job that is scheduled by the respective state transition and in parentheses the arc's length. We indicate with dashed arcs the in our case unique longest path of length seven. The corresponding optimal solution, scheduling the jobs $\pi = (3,1,4)$ with a total prize of $Z(\pi) = 7$, is shown on the bottom left. Moreover, states of all nodes are given on the bottom right.

Exact MDDs grow in general in an exponential way with the problem size as they basically represent the complete state graph. We are more interested in more compact MDDs that represent the state graph only in an approximate way. This is usually done by limiting the number of nodes allowed in each layer to a fixed maximum $\beta \geq 1$. The number of nodes in a layer is called the layer's width, and the maximum width over all layers is the width of an MDD. To receive MDDs of limited width, there have been proposed two concepts with contrary effects: *relaxed MDDs* (Andersen et al. 2007) and *restricted MDDs* (Bergman et al. 2014b).

Relaxed MDDs cover all feasible solutions as a subset plus possibly a set of solutions that are invalid for the original problem. Thus, they represent a discrete relaxation of the original problem, and the length of a longest path of a relaxed MDD is a dual bound to the original problem's optimal solution



(a) Relaxed MDD

(b) Restricted MDD

Figure 2: A relaxed and a restricted MDD for the example instance in Fig. 1.

value $Z^*(\mathbf{r})$. To have limited width, a relaxed MDD in general superimposes states of the original state graph: Sets of states of an exact MDD are combined into so-called *merged nodes*; all affected arcs are redirected to the respective merged node. To ensure that a valid relaxation is obtained, the state of a merged node must be set so that it is in no dimension stricter than each original state. In case of our PC-JSOCMSR, if a set M of original states is merged, the state of the respective merged node is

$$\oplus(M) = \left(\bigcup_{(P,t)\in M} P, \left(\min_{(P,t)\in M} t_r \right)_{r\in R_0} \right). \tag{11}$$

Figure 2a shows for the exact MDD in Figure 1 a relaxed MDD where nodes u_3 and u_4 are merged resulting in node u'. The width of the relaxed MDD decreases from four to three. Recall that the optimal solution of the considered instance has a total prize of seven. The longest path within the relaxed MDD, indicated by the dashed arcs, has a total length of eight. This is achieved by scheduling job 4 twice, which clearly does not correspond to a feasible solution of the original problem. Moreover, notice that the relaxed MDD contains all paths from the exact MDD. The original optimal solution is still represented by a respective path, however, it is not a longest anymore. The state of the merged node is given by $(\{1,4\},(4,3,4))$, while the states of all remaining nodes do not change.

Restricted MDDs are the second option for approximate MDDs with limited width. They are obtained by removing nodes from an exact MDD with all incoming and outgoing arcs. Whenever a node is removed, also all paths containing the node are not anymore encoded in the MDD. Consequently, a restricted MDD represents only a subset of all feasible solutions, and the length of a longest path in a restricted MDD might be shorter than one in an exact MDD. For this reason the length of a longest path in a restricted MDD is a primal bound to the original problem's optimal solution value $Z^*(\mathbf{r})$.

A restricted MDD for the exact MDD from Figure 1 is depicted in Figure 2b. The node u_3 and all its incoming and outgoing arcs are removed. All other nodes, arcs, and states remain unchanged. The longest path in the restricted MDD, again indicated by dashed arcs, has a total length of six. This longest path encodes a feasible solution to the original problem, however, not an optimal one.

5 Top-Down Construction

The top-down construction (Bergman et al. 2016b, 2014b,a) compiles exact MDDs, as well as relaxed and restricted MDDs by traversing the state graph in a breadth-first fashion. The method starts with an empty first layer L_0 and adds a node for the initial state $\mathbf{r}$. Then, one layer after the other is filled with nodes. For a subsequent layer L_i, this is done by adding all feasible states that can be obtained by a transition from any node $u \in L_{i-1}$, i.e.,

$$L_i = \{\tau(u,j) \mid u \in L_{i-1}, j \in P(u)\}. \tag{12}$$

Note that identical states produced by different transitions are represented by a single common node within a layer. In addition to the nodes, we also add corresponding arcs for each of the conducted transitions.

When we are compiling relaxed or restricted MDDs, we have to check at this point the width of the current layer L_i. If it exceeds a given maximum β, nodes have either to be merged or dropped, respectively. The quality of the obtained primal and dual bounds from the produced relaxed and restricted MDDs is predominantly influenced by the strategy to select the nodes for merging or removal. The basic idea is to prefer nodes for merging or removal that are unlikely part of any optimal solution. Bergman et al. (2014a) considered three different merging heuristics: random nodes, nodes with the shortest longest path $Z^{\text{lp}}(u)$, and nodes with the most elements in $P(u)$. In their experiments the second strategy achieved the best results. Moreover, Bergman et al. (2014b) suggest the same node selection heuristic for the compilation of restricted MDDs. We observed that merging or removing nodes with the smallest $Z^{\text{lp}}(u)$ values is disadvantageous for PC-JSOCMSR. This can be explained by the fact that this strategy focuses just on the longest path, but does not respect how well the jobs fit next to each other. Therefore, we set the longest path to a node into perspective with the time the common resource is occupied by the corresponding jobs. The nodes within the currently considered layer L_i, $i > 0$, are sorted according to the ratio $Z^{\text{lp}}(u)/(t_0(u) - T^{\min})$ in increasing order. We then merge respectively remove the first nodes until the width of L_i becomes β. Afterwards, we continue with the next layer. The algorithm terminates when either no further state transitions are possible or we completed layer L_n.

6 Incremental Refinement

The basic idea of an incremental refinement approach is to apply filtering and refinement steps iteratively on an initial simple relaxed MDD in order to improve it and approximate an exact MDD. Filtering steps remove arcs that are only contained in root to sink paths that represent infeasible solutions. The refinement steps consist of splitting nodes to represent so far merged states in more detail and as a consequence to trigger further filtering of arcs. The main goal of incremental refinement is to decrease the length of longest paths in the MDD, i.e., the obtained upper bound on an instance's solution value.

Incremental refinement has been initially proposed by Hadzic et al. (2008) and Hoda et al. (2010) for constraint satisfaction systems. The central aspect of this approach is the division of filtering and refinement into independent operations. As a consequence, the overall algorithm can apply and combine these operations however it is appropriate. A relaxed MDD for the PC-JSOCMSR problem contains in general paths that do not represent feasible solutions, either because jobs occur more than once or not all jobs can be scheduled within their time windows. Therefore, we have to find refinement and filtering operations that allow us to exclude job repetitions and time window violations.

Due to the fact that exact MDDs have in general an exponential number of nodes w.r.t. the problem size, we cannot hope to apply refinement and filtering until all invalid paths are sorted out for problem instances of practically relevant size. Hence, a key aspect of an incremental refinement approach is the order in which the refinement steps are applied on the nodes. The works from Cire and Hoeve (2013) and Kinable et al. (2017) provide an incremental refinement method for sequencing problems in which a permutation of jobs has to be found. Essentially, they order the jobs according to the processing times and with it to a certain extent according to the length of the corresponding arcs within the MDD. Their approach removes repetitions of jobs according to that order until the maximal allowed width of the MDD is reached. The rationale behind this strategy is that repetitions of jobs represented by long arcs are more frequently contained within longest paths. For PC-JSOCMSR this method is, however, not suitable because we have to assume that only a fraction of the jobs can be actually scheduled. Hence, it is not clear in advance which jobs play a key role for deriving a good approximation of an exact MDD.

Our incremental refinement for PC-JSOCMSR uses a current longest path as guidance. We follow the arcs on such a longest path, starting from the root node, and check for each arc whether the associated job can be feasibly scheduled. In case that a job occurs more than once, we refine the MDD s.t. repetitions of this job are not possible anymore. If a job cannot be feasibly scheduled within its time windows, we split nodes to allow excluding this path.

Algorithm 1: Incremental Refinement Guided by Longest Paths (IRLP)

Input: initial relaxed MDD $G = (V, A)$ with $V = L_0 \cup \cdots \cup L_n$

```
1  while termination criterion not met do
2      Let p be a longest path in G;
3      if p admits a feasible schedule then
4          return;                                  /* optimal solution has been found */
5      if p contains a job repetition then
6          for i ← 1 to n do
7              foreach node u ∈ L_i do
8                  update node u and filter incoming and outgoing arcs;
9                  split node u into two if it allows to avoid the node repetition;
10             merge nodes with identical states in L_i;
11     else                                         /* p contains a time window violation */
12         Split nodes on p to avoid the identified time window violation;
13         for i ← 1 to n do
14             foreach node u ∈ L_i reachable from the splitted nodes do
15                 update node u and filter incoming and outgoing arcs;
16             merge nodes with identical states in L_i;
```

Algorithm 1 shows an outline of the proposed Incremental Refinement Guided by Longest Paths (IRLP). It acts on a given relaxed MDD, which is obtained in our case by the top down construction from Section 5 with a small initial width. In each iteration of the main while loop we obtain a longest path. If the sequence of jobs represented by the path can be feasibly scheduled, then we have found an optimal solution and terminate.

Depending on whether we detected a job repetition or a time window violation on the currently considered longest path the following steps differ. In the former case we traverse the MDD starting from the root node **r** layer by layer. For each considered node we try to filter arcs and update the node's state if necessary. We check next if the node has to be refined and perform a node split if it allows to remove the considered job repetition. After all nodes of a layer have been considered, we might encounter that nodes are associated with an identical state. Such nodes are merged to avoid redundancies in the MDD. In the latter case of a time window violation we perform a much more local refinement operation in which only nodes along the considered path are split. In the subsequent filtering we consider all nodes reachable from the previously split nodes. We enforce also here that all nodes within each layer represent a distinct state. Notice that the refinement of job repetitions is preferred over the refinement of time window violations if both are contained in the longest path. This has shown to provide better bounds if the algorithm has to terminate prematurely due to a time limit. The applied filtering techniques and the updating of the nodes' state are described in Section 6.1. The two types of refinement operations are presented in more detail in Section 6.2 and Section 6.3. Finally, in Section 6.4 we explain how we avoid producing nodes representing identical states within layers.

6.1 Node Updates and Filtering

Filtering applied in an incremental refinement method aims at identifying and removing arcs that are only contained in paths corresponding to infeasible solutions. The filtering techniques generally rely on the Markovian property of the MDD's states, which means that a state is defined by its predecessors and the transitions. This allows specifying tests that use information local to a considered node to decide whether incoming or outgoing arcs can be removed.

An intrinsic part of the presented filtering method is to keep the node's states always up to date, which is necessary because the removal of a node's incoming arcs may change its associated state. Moreover, an adjustment of a node's state may imply further changes on the nodes reachable from the currently considered node. Therefore, we traverse the MDD s.t. we reach a node after we have processed all its predecessors. Consequently, we end up in each iteration of the IRLP with an MDD where all states fulfill the Markovian property. For each considered node we first update the node's state and then check whether incoming or outgoing arcs can be removed. In case incoming arcs are removed the node's state has to be reevaluated again. An update of a state consists of reassessing and merging the transitions from all predecessors, which means for a node u to compute

$$\oplus \left(\{ \tau(v, j(a)) \mid a = (v, u) \in A^+(u) \} \right). \tag{13}$$

Such a state update is a computational expensive operation and should only be performed if a node's state may actually change. For this reason, we recompute a node's state only if either a predecessors state has changed or if an incoming arc has been removed.

Let (P, t) and (P', t') be node u's state before and after a reevaluation, respectively. Due to the definition of the relaxation scheme (11) and the fact that we are only removing arcs during filtering, it holds that $t'_r \geq t_r$ for all $r \in R_0$ and $P' \subseteq P$. In case $P' \subset P$, we remove all outgoing arcs $a \in A^-(u)$ with $j(a) \notin P'$ since they cannot be part of any feasible solution represented by a path reaching u from $\mathbf{r}$. If any node except $\mathbf{r}$ ends up without any incoming arc, it is removed together with all its outgoing arcs.

6.2 Refinement of Job Repetitions

We discuss in this section a technique that modifies an MDD in such a way that a considered job j occurs on each path at most once. This method is conceptually an adaptation from the one proposed by Cire and Hoeve (2013), but takes into account that in PC-JSOCMSR usually only a subset of the jobs can be scheduled. The refinement is based on the observation that a job repetition occurs if a job j is contained on a path starting from node $\mathbf{r}$ to a node u and job j is still included in $P(u)$. Consequently, node u has an outgoing arc associated with job j which represents a repetition. Before we can derive a splitting strategy, we first have to verify if the above condition is sufficient to detect all job repetitions. To this end we denote with $\text{Some}_u^{\downarrow} \subseteq J$ the subset of jobs appearing in some path from $\mathbf{r}$ to a node $u \in V$. For a node $u \in V$ the set $\text{Some}_u^{\downarrow}$ can be calculated recursively by

$$\text{Some}_u^{\downarrow} = \bigcup_{a=(v,u) \in A^+(u)} \left(\text{Some}_v^{\downarrow} \cup \{ j(a) \} \right). \tag{14}$$

We show next that we can determine repetitions of a considered job j occuring on some path in a MDD by using $P(u)$ and $\text{Some}_u^{\downarrow}$ of the nodes u in the MDD.

Lemma 1. *A job j is assigned on each path starting from $\mathbf{r}$ at most once if and only if $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ holds for all nodes $u \in V$.*

Proof. Assume first that a job j is associated with at most one arc in every path starting from $\mathbf{r}$ of a given MDD G and consider an arbitrary node $u \in V$. If no path from $\mathbf{r}$ to u has an arc labeled j then it holds by definition that $j \notin \text{Some}_u^{\downarrow}$ and consequently $j \notin \text{Some}_u^{\downarrow} \cap P(u)$. If on the other hand there exists a path from $\mathbf{r}$ to u with an arc associated with j then no path starting from u can contain

an arc labeled j. Moreover, it holds by definition that a node $v \in V$ can only have an outgoing arc a with $j(a) = j$ if $j \in P(u)$. Therefore, $j \notin P(u)$ and $j \notin \text{Some}_u^{\downarrow} \cap P(u)$.

Conversely, suppose that $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ for all nodes $u \in V$. In case $j \notin \text{Some}_u^{\downarrow}$ we cannot have a repetition of node j on any path from $\mathbf{r}$ to u. If a node u is reached by an arc associated with job j then $j \in \text{Some}_u^{\downarrow}$ and thus, $j \notin P(u)$. Since node u can have only outgoing arcs for the jobs in $P(u)$, node u cannot have an outgoing arc labeled j. Moreover, since $j \in \text{Some}_v^{\downarrow}$ for all nodes v reachable from node u we can conclude by the same argument that also for these nodes $j \notin P(u)$ and hence there are no respective outgoing arcs. Thus, job j is assigned on each path starting from $\mathbf{r}$ at most once. □ □

Whenever we detect a node repetition, i.e., $j \in \text{Some}_u^{\downarrow} \cap P(u)$ for some node u, we perform a node split to obtain a node u_1 with $j \notin P(u)$ and a node u_2 with $j \notin \text{Some}_u^{\downarrow}$ as follows.

Theorem 1. *Given job j and a MDD, we replace all nodes $u \in V$ with $j \in \text{Some}_u^{\downarrow} \cap P(u)$ by two nodes u_1 and u_2, s.t. all incoming arcs $a = (v, u)$ are redirected to u_1 if $j \notin P(\tau(v, j(a)))$ and to u_2 otherwise. All outgoing arcs are replicated for both nodes. The resulting MDD satisfies $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ for all nodes $u \in V$.*

Proof. For the root node $\mathbf{r}$ we have by definition that $\text{Some}_{\mathbf{r}}^{\downarrow} = \emptyset$ and, thus, $j \notin \text{Some}_u^{\downarrow} \cap P(u)$. Assume as induction hypothesis that the desired condition $j \notin \text{Some}_u^{\downarrow} \cap P(u)$ holds for all predecessors of a node u. In addition, consider that we have replaced node u by the nodes u_1 and u_2 as described above. From the relaxation scheme (11) we know that set P of node u_1 cannot contain j. For all of u_2's incoming arcs $a = (v, u_2)$ it holds that $j \notin \text{Some}_v^{\downarrow}$ since otherwise $P(\tau(v, j(a)))$ could not contain j. Consequently, u_1 as well as u_2 satisfy the stated condition. □ □

The actual refinement is done by enforcing Lemma 1 in a single top down pass. To this end, we start with the root node and process all nodes layer by layer. For each considered node u we first update its state if needed and apply the filtering as described in Section 6.1. Afterwards, we determine the set $\text{Some}_u^{\downarrow}$ and split node u as described in Theorem 1 if necessary. Whenever a node is split, new states are calculated for the two new nodes. Furthermore, we perform filtering on the new nodes' incoming and outgoing arcs.

6.3 Refinement of Time Window Violations

Let sequence $(u_1, a_1, u_2, \ldots, u_k, a_k, u_{k+1})$ of alternating nodes and arcs denote a path in our MDD starting at the root node $\mathbf{r}$ (i.e., $u_1 = \mathbf{r}$) where $(u_1, a_1, u_2, \ldots, u_{k-1}, a_{k-1}, u_k)$ corresponds to a feasible solution but the job represented by arc a_k cannot be additionally scheduled within its time windows. For the considered path we denote with $(u_1^{\downarrow}, \ldots, u_k^{\downarrow})$ the not relaxed states along the considered path. That is, $u_1^{\downarrow} = \mathbf{r}$ and $u_i^{\downarrow} = \tau(u_{i-1}^{\downarrow}, j(a_{i-1}))$ for $1 < i \leq k$. Due to the state relaxations of the nodes in the MDD we observe that $j(a_k) \in P(u_k)$ but $j(a_k) \notin P(u_k^{\downarrow})$. The basic idea is to split the nodes on the path in such way that job $j^{(k)}$ can be removed from $P(u_k)$ and with it also the arc a_k.

In general, it is not sufficient to just split node u_k but a subset of the path's nodes $u_l, \ldots, u_k$, with $1 < l \leq k$, has to be refined. Ideally, the number of nodes to be refined should be small and the refinement should exclude other time window violations as well. We compute the subset of nodes to be refined as follows: We first check whether $s(\tau(u_{k-1}, j(a_{k-1})), j(a_k))$ evaluates to $T^{\max}$. If it does, then job $j(a_k)$ cannot be feasibly scheduled on the state resulting from the transition from state u_{k-1}. Consequently, it suffices to refine node u_k. If it does not, then we consider one predecessor more, i.e., we check whether $s(\tau(\tau(u_{k-2}, j(a_{k-2})), j(a_{k-1})), j(a_k))$ results in $T^{\max}$. This step is repeated until we find a node u_{l-1} on the considered path which allows excluding job $j(a_k)$ if we follow exact transitions from it.

The actual refinement works as follows: We replace each node u_i with $i = l, \ldots, k$ by nodes $u_{i,1}$ and $u_{i,2}$. The incoming arcs $a = (v, u_i) \in A^+(u_i)$ are redirected to $u_{i,1}$ if $t_r(\tau(v, j(a))) \geq$

$t_r(\tau(u_{i-1}, j(a_{i-1})))$ for all $r \in R_0$, otherwise, they are redirected to $u_{i,2}$. Outgoing arcs of u_i are replicated for $u_{i,1}$ and $u_{i,2}$. After a node split we determine for the two resulting nodes the corresponding states and perform a filtering of their incoming and outgoing arcs as described in Section 6.1. Last but not least, we have to possibly reevaluate the states and filter all incident arcs of all nodes reachable from each node u_i.

6.4 Duplicate State Elimination

A side effect of the node updates and the refinement methods is that we might end up with multiple nodes within one layer that represent an identical state. For example, assume that a considered layer already contains a node u without any outgoing arcs that represents state $(\emptyset, (T^{\max}, \ldots, T^{\max}))$. Moreover, suppose that due to updating another node u' we encounter that its state cannot be feasibly extended and remove all outgoing arcs. Both, u and u', then represent the same identical state, but are reached by different paths. We can avoid this redundancy in the MDD by redirecting all incoming arcs from u' to u and removing u' from V.

In the more general case, where u and u' have outgoing arcs merging nodes with duplicate states is more involved. First of all, we have to ensure that our MDD still remains a valid relaxation. The Markovian property implies that for all feasible extensions of u there exist an equivalent extension of u' and vice versa (Cire and Hoeve 2013). This allows us to remove node u' including its outgoing arcs after redirecting all incoming arcs to u. Obviously, the state of node u remains valid. However, if not done carefully, this operation may reintroduce paths encoding infeasible solutions which have been already excluded. Furthermore, we have to make sure that the duplicate state elimination does not produce cycles with the refinement operations in order to guarantee that IRLP terminates. After performing the refinements of job repetitions for a job j on a considered layer it holds that if a job $j \in P(u)$ then $j \notin \text{Some}^{\downarrow}_{u'}$ whenever the states of u and u' are identical. Splitting nodes for refining time window violations aims at increasing the t_r values to trigger filtering. Since our duplicate state elimination does not change the nodes' states, we will from a theoretical point of view always converge to an exact MDD.

Our duplicate state elimination works as follows: After we have performed all refinement, updating, and filtering operations within a layer, we consider all nodes with duplicate states pairwise and remove one of them until all nodes' states are distinct. To this end, we redirect all incoming arcs to the node u having the larger $Z^{\text{lp}}(u)$ value and remove the other node including outgoing arcs. The intention for selecting the node with the larger $Z^{\text{lp}}(u)$ value is that we do not increase the longest path to any node. Moreover, the nodes reachable from u are more likely to be already refined, as IRLP focuses on longest paths.

7 General Variable Neighborhood Search

In this section the General Variable Neighborhood Search (GVNS) is presented which serves us as a reference approach for obtaining heuristic solutions. GVNS (Hansen et al. 2010) is a prominent local search based metaheuristic which operates on multiple neighborhoods. The basic idea is to systematically change local search neighborhood structures until a local optimum in respect to all these neighborhood structures is found. This part is called variable neighborhood descent (VND). To further diversify the search, the GVNS performs so-called shaking for local optimal solutions by applying random moves in larger neighborhoods. These perturbed solutions then undergo VND again, and the whole process is repeated until a termination condition is met at which point the best solution encountered is returned.

In the context of this metaheuristic we represent a solution by a permutation $\pi = (\pi_i)_{i=1,\ldots,|J|}$ of the entire set of jobs J. Starting times and the subset of jobs $S \subseteq J$ that actually is scheduled is obtained by considering all jobs in the order of π and determining each job's earliest feasible time; jobs that cannot be feasibly scheduled w.r.t. their time windows anymore are skipped. This solution

representation allows us to use rather simple neighborhood structures.

Our GVNS for PC-JSOCMSR starts with a random permutation of the jobs J as initial solution. In a preliminary study, we also used initial solutions computed by a FRB4$_k$ (Rad et al. 2009) construction heuristic. Although this constructive heuristic provided much better starting solutions, we could not observe significant differences in the quality of final solutions returned by the GVNS.

We employ in our GVNS two local search neighborhood structures. The first one considers all exchanges of pairs of jobs within the permutation, while the second considers the removal of any single job and its re-insertion at another position. To avoid the consideration of moves that do not change the actual schedule, we require that each move changes either S or the order of the jobs within S.

In the VND, we apply any possible improving exchange move before considering the moves that remove and reinsert jobs. Each neighborhood is searched in a first improvement fashion. As shaking operation we perform a sequence of k random remove-and-insert moves. Whenever a new incumbent local optimal solution is found, the following shaking starts with $k = 1$. Parameter k is increased by one up to a maximum value $k_{\max}$ after every unsuccessful shaking followed by the VND. After reaching $k_{\max}$, k is reset to one again.

8 Computational Study

We performed an experimental evaluation of the proposed approaches, i.e., the top down construction (TDC) for relaxed and reduced MDDs, the incremental refinement guided by longest paths (ILRP), and the general variable neighborhood search (GVNS). The algorithms are implemented in C++ and have been compiled by GNU G++ 7.3.1. All experiments are performed on a single core of an Intel Xeon E5-2640 v4 CPU with 2.40GHz and 16 GB of memory.

We use the same two types of test instances as in Horn et al. (2018) but extend these to also include particularly larger instances with up to 300 jobs; all instances are available at http://www.ac.tuwien.ac.at/research/problem-instances. Each set contains in total 840 instances with 30 instances for each combination of $n \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 120, 150, 200, 300\}$ jobs and $m \in \{2, 3\}$ secondary resources. In the first set B of *balanced* benchmark instances the secondary resources are equally distributed among the jobs and each job requires in expectation the common resource for the second third of its processing time. To this end a job's secondary resource is uniformly sampled from R. The processing time of a job j is determined by sampling values for p_j^{pre}, p_j^{post} from $\mathcal{U}\{0, 8\}$ and for p_j^0 from $\mathcal{U}\{1, 8\}$. In the second set S of benchmark instances, which we regard as *skewed*, one of the secondary resources is required predominantly and in expectation the common resource is required more than the half the job's processing time. In detail, a job's secondary resource is set to m with a probability of 0.5 while the other resources in R are selected with a probability of $1/(2m - 2)$. The duration p_j^0 of the jobs $j \in J$ are chosen uniformly from $\{1, \ldots, 13\}$ and the pre-processing and post-processing times p_j^{pre} and p_j^{post} are both uniformly selected from $\{0, \ldots, 5\}$. The remaining characteristics of the two benchmark sets are obtained in the same way: The prize z_j associated with each job is sampled uniformly from $\{p_j^0, \ldots, 2p_j^0\}$ in order to correlate with the time the common resource is used. For the jobs we generate between one and three time windows in such a way that approximately 30% of the jobs fit into a schedule. To this end, we sample for each job the number of time windows ω_j from $\{1, 2, 3\}$. Moreover, let $\mathrm{E}(\mathrm{p}^0)$ be the expected duration a job requires the common resource and let $T = \lfloor 0.3\, n\, \mathrm{E}(\mathrm{p}^0) \rfloor$ be the total expected time required from the common resource to schedule 30% of all jobs. The ω_j time windows W_j for job j are generated as follows: We choose a start time W_{jw}^{start} uniformly from $\{0, \ldots, T - p_j\}$ and an end time W_{jw}^{end} from $\{W_{jw}^{\text{start}} + \max(p_j, \lfloor 0.1\, T/\omega_j \rfloor), \ldots, W_{jw}^{\text{start}} + \max(p_j, \lfloor 0.4\, T/\omega_j \rfloor)\}$. If we obtain overlapping time windows, they are merged and ω_j is adjusted accordingly.

The initial relaxed MDD used by incremental refinement methods in the literature (Cire and Hoeve 2013, Kinable et al. 2017) are typically trivial ones of width one and can be obtained by calling TDC with $\beta = 1$. For PC-JSOCMSR there is a more meaningful initial relaxed MDD of maximum width

m, where on each layer all states are merged that are obtained by jobs requiring the same secondary resource. This initial relaxed MDD has in general already significantly stronger states than the relaxed MDD of width one, because in the latter the advances on the times t_r for the secondary resources $r \in R$ cancel each other out. Preliminary experiments showed that small instances can be optimally solved with fewer iterations and on larger instances stronger bounds can be obtained when starting with the width m initial relaxed MDD. Hence, we do this in all our further IRLP runs.

In other preliminary experiments we investigated different configurations of the GVNS. We tried changing the order of the neighborhood structures within the VND and also shaking operators based on exchanging the positions of randomly selected jobs. The configuration described in Section 7 was found to work best. Moreover, we tuned the maximum shaking size parameter $k_{\max}$. Rather small values for $k_{\max}$ turned out to typically yield better results, and we decided to use $k_{\max} = 4$ for all further GVNS runs in this work.

In the first series of experiments we compare the quality of relaxed MDDs compiled by TDC and IRLP, respectively. IRLP was performed with a CPU-time limit of 900 seconds per run, while for TDC we used different values for the maximum width β in dependence of the number of jobs so that the required CPU-time was in a similar order of magnitude. In Table 1 each row shows average results of 30 instances. The first three columns describe the instance properties. For both approaches mean dual bounds $\overline{Z^{\text{lp}}}$ are listed together with the corresponding standard deviations $\sigma(Z^{\text{lp}})$, the median numbers of nodes of the relaxed MDD $|V|$ and median completion times t in seconds. In addition, the employed maximum width β in the TDC are given. Moreover, for the IRLP we state in column $\Delta[\%]$ the percentage of nodes saved due to the elimination of duplicate states. In other words, we compute $100 - 100 \cdot (|V|/|V'|)$, where $|V'|$ is the size of the MDDs of separate runs of the IRLP without elimination of duplicate nodes, as it was presented in our preliminary work Maschler and Raidl (2018a).

On the smallest instances both algorithms produce relaxed MDDs with the same dual bounds. In these cases the obtained bounds correspond to the optimal objective values, which we verified by checking that the longest paths indeed correspond to feasible schedules. In fact, TDC could solve several instances with up to 60 jobs, while IRLP found optimal solution for some instances with up to 50 jobs. While on the medium to large instances with balanced jobs we cannot observe a clear tendency which method provides tighter bounds, IRLP outperforms TDC by a rather large margin on almost all skewed instances. Notice that the size of the relaxed MDDs produced by both algorithms peaks by instances with 60 or 70 jobs and declines for larger benchmark instances. This can be explained for the TDC by the increasing number of state transitions that have to be performed for each layer and by the increasing number of nodes that have to be merged as a result. For IRLP the reason is similar. IRLP has to reevaluate for larger instances more frequently nodes with many incoming arcs.

The IRLP has been extended in comparison to our preliminary work (Maschler and Raidl 2018a) by the elimination of duplicate states within layers. While the effects on the obtained dual bounds have been negligible, the size of the produced MDDs has been reduced between by 10% to 30% on average. Smaller MDDs of similar quality become especially important if the MDDs are further exploited within a larger algorithmic framework such as in the DD-based branch-and-bound (Bergman et al. 2016b) or for constraint propagation in CP (Cire and Hoeve 2013, Kinable et al. 2017).

In a second series of experiments the heuristic solutions obtained by the TDC for restricted MDDs are compared with the ones computed by the GVNS. We employ for the GVNS a time limit of 900 CPU-seconds as termination criterion. For TDC, different maximum widths β were used again so that the running times are in a similar order of magnitude. Table 2 shows the obtained results. The first three columns describe the instance properties and each row shows average results of 30 corresponding instances. The means and the corresponding standard deviations of the final objective values for TDC and GVNS are shown in the columns $\overline{Z^{\text{lp}}}$, $\sigma(Z^{\text{lp}})$, $\overline{obj}$ and $\sigma(obj)$, respectively. In addition, for TDC the maximum width β, median number of nodes in the restricted MDD $|V|$, and median computation times t in seconds are listed. Moreover, column t^{best} shows for the GVNS median times in seconds when the best solution has been found.

			TDC (relaxed MDDs)					IRLP				
type	m	n	β	$\|V\|$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$	$\|V\|$	$\Delta[\%]$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$
B	2	010	750000	42	<1	**30.93**	6.91	20	11	<1	**30.93**	6.91
B	2	020	750000	526	<1	**50.37**	5.71	182	14	<1	**50.37**	5.71
B	2	030	750000	5365	<1	**75.33**	6.41	1258	17	<1	**75.33**	6.41
B	2	040	750000	78336	<1	**98.93**	7.05	12618	18	2	**98.93**	7.05
B	2	050	750000	850366	18	**123.83**	9.86	210294	34	374	125.80	10.93
B	2	060	750000	6192518	214	181.07	26.79	1502938	36	900	**175.13**	14.89
B	2	070	100000	1821669	136	314.30	30.97	4160276	33	900	**253.23**	31.06
B	2	080	100000	2291714	243	400.57	35.81	4449456	26	900	**342.70**	48.38
B	2	090	100000	3109741	439	497.17	51.51	3722248	27	900	**454.70**	66.55
B	2	100	100000	3885520	683	605.10	47.07	3143184	29	900	**593.33**	100.04
B	2	120	20000	1096548	279	**868.50**	85.35	2581056	19	900	872.83	148.05
B	2	150	20000	1678748	690	**1245.50**	99.96	1867372	14	900	1409.60	148.46
B	2	200	2000	289016	232	**2176.47**	206.23	1300659	8	900	2283.80	117.48
B	2	300	2000	512774	974	**3830.17**	291.98	619908	5	900	3865.90	99.82
B	3	010	750000	52	<1	**36.17**	6.22	30	5	<1	**36.17**	6.22
B	3	020	750000	978	<1	**59.27**	7.85	336	12	<1	**59.27**	7.85
B	3	030	750000	13766	<1	**86.30**	7.08	4046	23	<1	**86.30**	7.08
B	3	040	750000	215763	3	**112.00**	7.79	68336	25	55	112.97	9.82
B	3	050	750000	3893395	84	**154.43**	24.57	718030	32	900	160.80	17.61
B	3	060	750000	10316441	474	241.53	16.68	3910070	21	900	**221.60**	15.19
B	3	070	100000	2441857	193	405.50	51.30	4870403	22	900	**334.90**	47.25
B	3	080	100000	3282533	355	527.47	56.95	3808838	24	900	**477.07**	60.86
B	3	090	100000	4259832	664	**655.80**	68.22	3207024	27	900	672.37	79.23
B	3	100	100000	5214238	981	**783.30**	76.89	3394374	19	900	840.93	67.22
B	3	120	20000	1552652	402	**1176.57**	91.67	2249186	23	900	1186.10	73.63
B	3	150	20000	2290835	1000	1687.27	137.87	1750716	11	900	**1619.23**	96.31
B	3	200	2000	381135	294	2827.77	161.11	1192373	2	900	**2364.27**	131.87
B	3	300	2000	598301	1318	4562.17	122.92	536464	14	900	**3914.33**	127.36
S	2	010	450000	40	<1	**50.93**	8.36	21	11	<1	**50.93**	8.36
S	2	020	450000	1039	<1	**89.93**	8.23	446	9	<1	**89.93**	8.23
S	2	030	450000	21220	<1	**131.37**	10.37	9478	26	1	**131.37**	10.37
S	2	040	450000	430093	7	**180.07**	12.40	167558	25	598	186.27	16.49
S	2	050	450000	4394388	128	300.13	50.61	2079225	25	900	**297.37**	30.72
S	2	060	450000	8549486	530	535.90	77.47	6311374	26	900	**463.60**	44.08
S	2	070	100000	3230809	321	835.43	119.97	5264954	29	900	**717.07**	74.14
S	2	080	100000	4362590	546	1091.63	124.31	4373746	25	900	**931.40**	94.35
S	2	090	100000	5475532	893	1315.33	120.45	4196128	27	900	**1154.03**	100.01
S	2	100	20000	1439179	287	1754.63	163.18	3287628	19	900	**1461.50**	102.69
S	2	120	20000	1840614	537	2276.60	236.54	2713622	15	900	**1891.17**	132.51
S	2	150	20000	2756871	1218	3315.60	209.25	1929511	13	900	**2598.47**	152.95
S	2	200	1000	199201	180	4853.90	171.69	1203598	9	900	**3770.27**	181.75
S	2	300	1000	299301	791	7483.10	187.80	669654	6	900	**6249.60**	213.77
S	3	010	450000	46	<1	**51.97**	9.76	34	7	<1	**51.97**	9.76
S	3	020	450000	1216	<1	**96.47**	9.13	470	20	<1	**96.47**	9.13
S	3	030	450000	23358	<1	**135.90**	9.42	9650	25	1	**135.90**	9.42
S	3	040	450000	1099240	15	**191.20**	17.19	377388	31	900	206.27	20.87
S	3	050	450000	5968862	211	357.60	57.78	5591745	31	900	**341.23**	36.53
S	3	060	450000	11241455	663	610.37	70.78	6410652	29	900	**548.23**	74.03
S	3	070	100000	4134692	401	956.73	114.00	5087212	29	900	**779.00**	73.10
S	3	080	100000	4676286	624	1219.10	166.32	4280442	24	900	**1012.07**	70.41
S	3	090	100000	6803302	1145	1623.87	162.81	3768948	23	900	**1249.00**	100.66
S	3	100	20000	1691990	313	2013.37	239.78	3123381	22	900	**1489.03**	128.71
S	3	120	20000	2298596	648	2696.10	208.48	2441742	15	900	**1926.13**	153.86
S	3	150	20000	2973857	1456	3510.93	225.37	1806430	12	900	**2669.80**	99.54
S	3	200	1000	199201	208	4904.30	165.27	1132778	9	900	**3901.67**	194.42
S	3	300	1000	299301	800	7508.93	188.07	528610	2	900	**6393.83**	250.52

Table 1: Comparison of the relaxed MDDs obtained from TDC and IRLP.

			TDC (restricted MDDs)					GVNS		
type	m	n	β	$\|V\|$	$t[s]$	$\overline{Z^{\text{lp}}}$	$\sigma(Z^{\text{lp}})$	$\overline{obj}$	$\sigma(obj)$	$t^{\text{best}}[s]$
B	2	010	750000	42	<1	**30.93**	6.91	**30.93**	6.91	<1
B	2	020	750000	526	<1	**50.37**	5.71	**50.37**	5.71	<1
B	2	030	750000	5365	<1	**75.33**	6.41	**75.33**	6.41	<1
B	2	040	750000	78336	1	**98.93**	7.05	98.90	7.01	<1
B	2	050	750000	850366	23	**123.27**	10.33	123.20	10.34	<1
B	2	060	750000	6149522	212	**146.80**	10.39	146.53	10.38	<1
B	2	070	750000	10039410	697	**172.23**	11.17	171.93	11.41	8
B	2	080	150000	2678998	259	**199.97**	13.36	199.47	13.13	8
B	2	090	150000	3275627	424	**231.83**	13.36	230.70	13.73	31
B	2	100	150000	3756246	610	**260.40**	11.52	258.83	11.71	67
B	2	120	50000	1592216	371	**315.90**	13.05	312.80	12.44	186
B	2	150	50000	2132142	799	**402.43**	18.65	396.50	17.47	551
B	2	200	6000	353180	226	**528.17**	18.96	526.83	18.73	459
B	2	300	6000	524990	766	**796.17**	16.68	791.67	17.26	751
B	3	010	750000	52	<1	**36.17**	6.22	**36.17**	6.22	<1
B	3	020	750000	978	<1	**59.27**	7.85	**59.27**	7.85	<1
B	3	030	750000	13766	<1	**86.30**	7.08	**86.30**	7.08	<1
B	3	040	750000	215763	8	**112.00**	7.79	111.93	7.85	<1
B	3	050	750000	3891532	105	**140.33**	10.40	140.27	10.40	1
B	3	060	750000	9554024	414	**165.13**	8.83	164.80	8.80	7
B	3	070	750000	13161224	1045	**194.83**	11.13	194.17	11.26	44
B	3	080	150000	3626754	354	**227.87**	13.54	226.70	13.34	72
B	3	090	150000	4210254	585	**257.53**	8.67	255.27	9.14	55
B	3	100	150000	4902283	822	**290.53**	14.30	288.03	14.60	274
B	3	120	50000	1996235	459	**352.27**	15.12	348.63	14.12	255
B	3	150	50000	2489353	936	**439.87**	15.88	436.30	16.68	358
B	3	200	6000	408372	245	579.97	18.74	**583.80**	16.31	507
B	3	300	6000	606960	830	847.67	16.25	**865.20**	19.25	707
S	2	010	750000	40	<1	**50.93**	8.36	**50.93**	8.36	<1
S	2	020	750000	1039	<1	**89.93**	8.23	**89.93**	8.23	<1
S	2	030	750000	21220	<1	**131.37**	10.37	**131.37**	10.37	<1
S	2	040	750000	430093	10	**180.07**	12.40	180.00	12.38	<1
S	2	050	750000	6335677	175	**225.67**	12.77	225.40	12.80	1
S	2	060	750000	10873978	679	**277.53**	11.67	276.87	11.97	10
S	2	070	150000	3233630	293	**326.20**	14.24	325.20	14.51	42
S	2	080	150000	3959832	452	**375.80**	16.50	374.23	16.66	70
S	2	090	150000	4495445	655	**421.50**	17.43	419.27	17.37	159
S	2	100	150000	5105289	962	**479.13**	20.91	476.03	20.84	259
S	2	120	50000	2072133	525	**574.37**	21.79	570.70	22.32	239
S	2	150	50000	2741184	1062	715.93	14.22	**716.37**	16.28	401
S	2	200	6000	432746	270	931.57	21.90	**948.87**	22.25	632
S	2	300	6000	668570	938	1382.70	30.42	**1424.07**	38.56	784
S	3	010	750000	46	<1	**51.97**	9.76	**51.97**	9.76	<1
S	3	020	750000	1216	<1	**96.47**	9.13	**96.47**	9.13	<1
S	3	030	750000	23358	<1	**135.90**	9.42	**135.90**	9.42	<1
S	3	040	750000	1099240	31	**185.43**	10.92	**185.43**	10.92	<1
S	3	050	750000	8572086	298	**234.40**	10.92	234.17	11.10	9
S	3	060	750000	13723665	842	**286.97**	13.00	286.10	13.13	9
S	3	070	150000	3656983	326	**331.30**	17.37	330.20	17.71	51
S	3	080	150000	4253940	475	**384.77**	17.27	383.33	17.38	37
S	3	090	150000	5157731	754	**429.60**	16.92	427.97	16.93	119
S	3	100	150000	5794926	1035	**487.30**	19.41	486.00	18.28	163
S	3	120	50000	2308306	575	565.13	17.12	**568.63**	16.74	448
S	3	150	50000	2904060	1060	708.07	21.61	**716.00**	20.42	442
S	3	200	6000	460541	267	928.80	24.19	**961.70**	24.27	519
S	3	300	6000	715920	926	1378.53	38.42	**1428.57**	39.68	747

Table 2: Comparison of heuristic solutions obtained from restriced MDDs compiled by TDC and the GVNS.

The TDC for restricted MDDs is able to outperform the GVNS on most of our benchmark instances. Only for the largest instances with three secondary resources or skewed jobs, GVNS is able to provide better results. The main reason for the superior performance of the TDC on instances with balanced jobs and two secondary resources is that the corresponding exact MDDs are much smaller compared with the other instances. This can be seen on the smallest instances where the imposed maximum width is not yet restrictive. On the instances with 30 jobs, for example, the resulting MDDs for balanced jobs with two secondary resources have on average 5365 nodes, with three secondary resources 13766 nodes, and for the instances with skewed jobs there are 21220 and 23358 nodes. It is safe to assume that this difference in size becomes even larger with more jobs. To stay within the memory and time limits, the maximum allowed width has to be decreased with the increasing number of jobs, which becomes more and more restrictive for the largest instances. Note that this relation can also be observed in Table 1 for relaxed MDDs. The GVNS approach, on the other hand, seems to be less affected by the instance type or by the number of secondary resources. This can be seen by the times the GVNS finds the final solution, which increases with the instance size but does not change substantially with the instance properties.

Concerning the gaps between the upper bounds obtained from the relaxed MDDs and the lower bounds from the heuristic solutions (compare Tables 1 and 2), we can observe that they are only small for the small and medium sized instances but become rather large for our largest instances. For example for the skewed instances with 300 jobs, this gap even exceeds 340%. This also somewhat illustrates the difficulty of the considered problem and the limits of MDDs – or at least the limits of the considered construction methods.

9 Conclusions and Future Work

In this work we studied the application of multivalued decision diagrams (MDDs) for the prize-collecting job sequencing with one common and multiple secondary resources (PC-JSOCMSR) problem. To this end, we first presented a recursive model and showed how to obtain MDDs from the problem's state graph. Whenever, the size of MDDs become to large relaxed and restricted MDDs are employed to obtain dual bounds and heuristic solutions, respectively. We adapted the two main compilation techniques for relaxed MDDs proposed in the literature to PC-JSOCMSR: top down construction (TDC) and incremental refinement (IR). To obtain restricted MDDs we use a variant of the TDC. In our computational study we first compared the relaxed MDDs obtained by TDC and IR. While both methods perform rather similar on balanced instances, IRLP is clearly superior on the benchmark instances sampled from skewed distributions. Afterwards, we assessed the quality of the restricted MDDs compiled with the TDC by comparing the obtained heuristic solutions with the ones from an independent general variable neighborhood search (GVNS). While the TDC performs better than the GVNS on small to medium-sized instances, the GVNS is mostly superior on the largest instances of our benchmark suite.

The focus of this paper is on the compilation of MDDs themself. A next step is to exploit the produced MDDs in various ways in other algorithmic frameworks to ultimately solve also larger instances of the problem in better ways. For example highly promising are novel branching schemes on the basis of relaxed MDDs as described in Bergman et al. (2016b) or the utilization of MDDs in new inference techniques in constraint programming, see e.g. Cire and Hoeve (2013). Last but not least, relaxed MDDs also have a great potential to provide guidance for finding good heuristic solutions in advanced metaheuristics.

A further promising future research direction seems to be to rethink the strict layered structure of the MDDs. For problems like PC-JSOCMSR their exist equivalent states on different layers. If we allow "long arcs" over multiple layers, such equivalent states may be represented by a single node possibly yielding more compact MDDs. Such a generalization provides new opportunities for merging states but also bears new challenges on how to identify nodes to be merged efficiently.

References

Allahverdi A (2016) A survey of scheduling problems with no-wait in process. European Journal of Operational Research 255(3):665–686

Andersen HR, Hadzic T, Hooker JN, Tiedemann P (2007) A constraint store based on multivalued decision diagrams. In: Principles and Practice of Constraint Programming – CP 2007, Springer Berlin Heidelberg, pp 118–132

Bergman D, van Hoeve WJ, Hooker JN (2011) Manipulating MDD relaxations for combinatorial optimization. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer Berlin Heidelberg, pp 20–35

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2014a) Optimization bounds from binary decision diagrams. INFORMS Journal on Computing 26(2):253–268

Bergman D, Cire AA, van Hoeve WJ, Yunes T (2014b) BDD-based heuristics for binary optimization. Journal of Heuristics 20(2):211–234

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2016a) Decision Diagrams for Optimization. Artificial Intelligence: Foundations, Theory, and Algorithms, Springer

Bergman D, Cire AA, van Hoeve WJ, Hooker JN (2016b) Discrete optimization with decision diagrams. INFORMS Journal on Computing 28(1):47–66

Cire AA, Hoeve WV (2013) Multivalued decision diagrams for sequencing problems. Operations Research 61(6):1411–1428

Hadzic T, Hooker JN, O'Sullivan B, Tiedemann P (2008) Approximate compilation of constraints into multivalued decision diagrams. In: Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp 448–462

Hansen P, Mladenović N, Brimberg J, Pérez JAM (2010) Variable neighborhood search. In: Handbook of Metaheuristics, Springer US, pp 61–86

Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research 207(1):1–14

Hoda S, van Hoeve WJ, Hooker JN (2010) A systematic approach to MDD-based constraint programming. In: Principles and Practice of Constraint Programming – CP 2010, Springer Berlin Heidelberg, pp 266–280

Hooker JN (2013) Decision diagrams and dynamic programming. In: CPAIOR 2013: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer, LNCS, vol 7874, pp 94–110

Horn M, Raidl G, Blum C (2017) Job sequencing with one common and multiple secondary resources: A problem motivated from particle therapy for cancer treatment. In: Giuffrida G, Nicosia G, Pardalos P, Umeton R (eds) MOD 2017: Machine Learning, Optimization, and Big Data – Third International Conference, Springer, LNCS, vol 10710, pp 506–518

Horn M, Raidl G, Rönnberg E (2018) An A* algorithm for solving a prize-collecting sequencing problem with one common and multiple secondary resources and time windows. In: PATAT 2018: Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling, Vienna, Austria, pp 235–256

Kinable J, Cire AA, van Hoeve WJ (2017) Hybrid optimization methods for time-dependent sequencing problems. European Journal of Operational Research 259(3):887–897

Lee CY (1959) Representation of switching circuits by binary-decision programs. Bell System Technical Journal 38(4):985–999

Maschler J, Raidl GR (2018a) Multivalued decision diagrams for a prize-collecting sequencing problem. In: PATAT 2018: Proceedings of the 12th International Conference of the Practice and Theory of Automated Timetabling, Vienna, Austria, pp 375–397

Maschler J, Raidl GR (2018b) Particle Therapy Patient Scheduling with Limited Starting Time Variations of Daily Treatments. International Transactions in Operational Research

Maschler J, Riedler M, Stock M, Raidl GR (2016) 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

Rad SF, Ruiz R, Boroojerdian N (2009) New high performing heuristics for minimizing makespan in permutation flowshops. Omega 37(2):331–345

Riedler M, Jatschka T, Maschler J, Raidl GR (2017) An iterative time-bucket refinement algorithm for a high-resolution resource-constrained project scheduling problem. International Transactions in Operational Research

Van der Veen JAA, Wöginger GJ, Zhang S (1998) Sequencing jobs that require common resources on a single machine: A solvable case of the TSP. Mathematical Programming 82(1-2):235–254