Technical Report AC-TR-18-011

December 2018

A*-Based Construction of Decision Diagrams for a Prize-Collecting Scheduling Problem

Matthias Horn, Johannes Maschler, Günther R. Raidl and Elina Rönnberg

A*-Based Construction of Decision Diagrams for a Prize-Collecting Scheduling Problem[*]

Matthias Horn[1], Johannes Maschler[1], Günther Raidl[1], Elina Rönnberg[2]

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

[2]Department of Mathematics, Linkping University, Sweden

{horn|maschler|raidl}@ac.tuwien.ac.at, elina.ronnberg@liu.se

Decision diagrams (DDs) have proven to be useful tools in combinatorial optimization. Relaxed DDs represent discrete relaxations of problems, can encode essential structural information in a compact form, and may yield strong dual bounds. We propose a novel construction scheme for relaxed multi-valued DDs for a scheduling problem in which a subset of elements has to be selected from a ground set and the selected elements need to be sequenced. The proposed construction scheme builds upon A* search guided by a fast-to-calculate problem-specific dual bound heuristic. In contrast to traditional DD compilation methods, the new approach does not rely on a correspondence of DD layers to decision variables. For the considered kind of problem, this implies that multiple nodes representing the same state at different layers can be avoided, and consequently also many redundant isomorphic substructures. For keeping the relaxed DD compact, a new mechanism for merging nodes in a layer-independent way is suggested. For our prize-collecting job sequencing problem, experimental results show that the DDs from our A*-based approach provide substantially better bounds while frequently being an order-of-magnitude smaller than DDs obtained from traditional compilation methods, given about the same time. To obtain a heuristic solution and a corresponding lower bound, we further propose to construct a restricted DD based on the relaxed one, thereby substantially exploiting already gained information. This approach outperforms a standalone restricted DD construction, basic constraint programming and mixed integer linear programming approaches, and a variable neighborhood search in terms of solution quality on most of our benchmark instances.

Keywords. decision diagrams, A* search, scheduling, sequencing

[*]This project is partially funded by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Foundation (FWF) Project No. W1260-N35. The work of Elina Rönnberg is supported by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05.

1. Introduction

In the last decade *decision diagrams* (DDs) have shown to be a powerful tool in combinatorial optimization (Andersen et al. 2007, Bergman et al. 2014b, Cire and van Hoeve 2013). For a variety of problems that classical mixed integer programming (MIP) and constraint programming (CP) techniques cannot address effectively (due, e.g., weak dual bounds), new state-of-the-art methodologies could be obtained with DDs at the core. These problems comprise prominent ones such as the minimum independent set, set covering, and maximum cut problems (Bergman et al. 2016b,a) as well as diverse sequencing and scheduling problems (Cire and van Hoeve 2013), including variants of the traveling salesman problem (TSP). Note that, common for the first group of mentioned problems is that an optimal subset of elements needs to be *selected* from some ground set, while for the sequencing problems, an optimal *order* (permutation) of all elements shall be determined.

Decision diagrams are in essence data structures that provide graphical representations of the solution space of an optimization problem. More specifically, a relaxed DD represents a superset of all feasible solutions in a compact way and can therefore be seen as a discrete relaxation of the problem. A relaxed DD can be used to obtain a dual bound, but it also provides a fruitful basis for alternative branching schemes (Bergman et al. 2016a) and constraint propagation (Andersen et al. 2007), for example.

For the kind of problems where a subset of elements is to be selected, so-called *binary decision diagrams* (BDDs) are typically used. Here, solutions are usually represented by binary vectors and each layer of the BDD is associated with a boolean decision variable indicating whether an element is selected or not. In contrast, for problems in which an optimal sequence of elements shall be found, it is more natural to apply so-called *multi-valued DDs* (MDDs). Then, a solution is represented by a permutation of the given elements, and consequently a layer in the MDD is associated with the decision which element appears at the respective position in the permutation.

We contribute in considering DDs for a problem that combines the selection aspect with the sequencing aspect, i.e., a problem in which a subset of initially unknown size needs to be selected from some ground set of elements and the selected elements need to be ordered to form a complete solution. Problem-specific constraints restrict the solution space so that not all subsets have a feasible order. More specifically, the problem we consider is the *prize-collecting job sequencing problem with one common and multiple secondary resources* (PC-JSOCMSR) from Horn et al. (2018). In this problem, each job is associated with a prize and the objective is to select a subset of jobs and find a feasible schedule such that the total prize is maximized. Note that besides the PC-JSOCMSR, the type of problems that combines the element selection aspect with the sequencing aspect is not uncommon. For example, the prominent class of orienteering problems (Gunawan et al. 2016), also called selective TSP, falls into this category as well as order acceptance and single machine scheduling (Ouz et al. 2010, Silva et al. 2018), prize-collecting single machine scheduling (Cordone et al. 2018), and other scheduling problems in which the number of tardy jobs shall be minimized (which corresponds to selecting and scheduling a subset of the jobs), see., e.g., Moore (1968), Lee and Kim (2012). More generally, similar problems also appear as pricing problems in column generation approaches, for example for vehicle routing and parallel machine scheduling problems. Last but not least, assortment problems also exhibit the selection aspect—although sometimes with decisions beyond binary ones—and occasionally the sequencing aspect, for example when optimizing over time (**?**).

For the type of problem with both selection and sequencing decisions we consider, it is natural to build upon MDDs similar to those from Cire and van Hoeve (2013), as solutions can be represented by permutations of the chosen elements. In contrast, it does not seem possible to effectively cover the sequencing aspect in some BDD variant. A particularity of our case is that feasible solutions may have arbitrary size in terms of the number of selected elements. This leads us to a novel technique for constructing relaxed MDDs. The method is inspired by A* search, a commonly used algorithm in path planning and problem solving (Hart et al. 1968). A priority queue is maintained for open nodes that still need further processing. A fast-to-calculate exogenous dual bound function is used as heuristic function to iteratively select the next node to be processed. To keep the constructed relaxed MDD compact, nodes are merged in a carefully selected way when the open list reaches a certain size. We show for the PC-JSOCMSR that the relaxed MDDs obtained by the A*-based method yield substantially stronger bounds than relaxed MDDs of comparable size constructed by two standard techniques. The main reasons for this advantage are (a) the guidance by the dual bound heuristic, (b) that our construction is able to effectively avoid multiple nodes for identical states at different layers of the MDD, and (c) that similar nodes can also be merged across different layers. Substantial redundancies that cannot be avoided in the standard construction techniques are therefore less problematic in our approach.

In order to not just obtain dual bounds, we further describe the construction of a restricted DD that yields promising heuristic solutions for the PC-JSOCMSR. Restricted DDs in general represent subsets of all feasible solutions. Hereby we contribute with a novel way of utilizing a previously constructed relaxed DD in order to substantially speed up the construction of a restricted DD.

Rigorous experiments including comparisons with MIP and CP approaches as well as with a variable neighborhood search heuristic on large benchmark instances with up to 500 jobs show the advantages of the proposed relaxed and restricted MDD construction techniques, respectively.

The article is structured as follows. Section 2 reviews DDs with a focus on MDDs for sequencing problems, introduces notations, and discusses related work. The PC-JSOCMSR is formally introduced in Section 3. Section 4 presents the A*-based construction of relaxed DDs in a rather problem-independent way, while Section 5 adds the problem-specific aspects, such as the definition of states, the transitions between them, and the way how nodes are merged. In Section 6, we explain how to boost the construction of a restricted DD by exploiting an existing relaxed DD. Experimental results are presented in Section 7. Finally, Section 8 concludes this work and outlines further research directions.

2. Decision Diagrams for Combinatorial Optimization

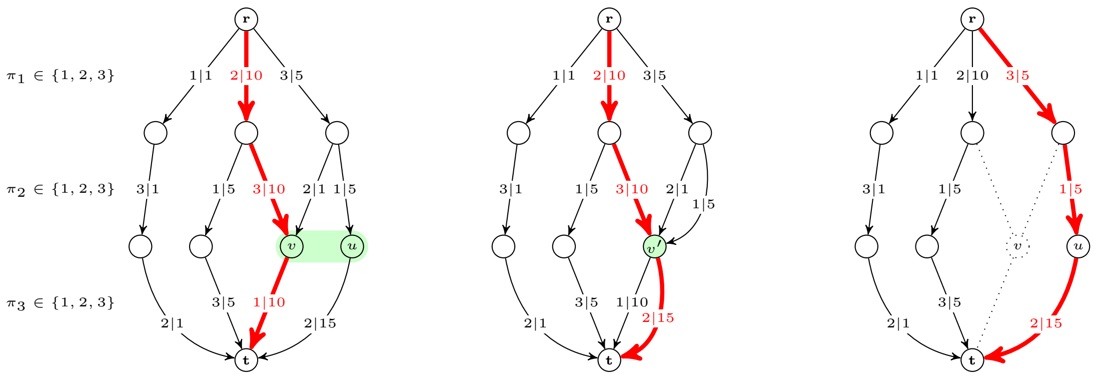
Decision diagrams were originally introduced in the field of electrical circuits and formal verification, see e.g. Lee (1959). For a comprehensive reading on DDs in optimization, their variants, applications, and successes, we refer to the book by Bergman et al. (2016a).

In the context of this work, a DD is a directed weighted acyclic multi-graph $M = (V, A)$ with node set V and arc set A. In the literature, the node set V is usually partitioned into layers $V = V_1 \cup \ldots \cup V_{n+1}$, where n corresponds to the number of decision variables representing a solution. The first and the last layer are singletons and contain the root node $\mathbf{r} \in V$ and the destination node $\mathbf{t} \in V$, respectively. Each arc $a = (u, v) \in A$ is associated with a value $\mathrm{val}(a)$ and directed from a source node u in some layer V_i to a destination node v in the subsequent layer V_{i+1}, $i \in \{1, \ldots, n\}$. Such an arc refers to the assignment of value $\mathrm{val}(a)$ to the i-th decision variable. While the domain of values in BDDs is restricted to $\{0, 1\}$, MDDs have arbitrary finite domains corresponding to those of the respective decision variables.

Each path from the root node $\mathbf{r}$ to the target node $\mathbf{t}$ corresponds to a solution encoded in the DD. An *exact DD* has a one-to-one correspondence between feasible solutions and the existing **r**-**t** paths. Let us consider a sequencing problem where a subset of the permutations $\pi = (\pi_1, \ldots, \pi_n)$ of the ground set $\{1, \ldots, n\}$ forms the set of feasible solutions. Figure 1a shows an example for an exact MDD with $n = 3$ encoding the permutations (1,3,2), (2,1,3), (2,3,1), (3,2,1), and (3,1,2).

Each arc $a \in A$ has a length $z(a)$ (or prize, cost, etc.) which gives the corresponding variable assignment's contribution to the objective value if it is chosen. The total length of an **r**-**t** path thus corresponds to the solution's objective value. We assume throughout this work that the considered optimization problem is a maximization problem. Consequently, we are looking for a longest **r**-**t** path.

As long as the DD is not too large, a longest path can be found efficiently as the DD is acyclic. Unfortunately, exact DDs for NP-hard optimization problems will in general have exponential size. This is where relaxed DDs come into play: They are more compact and they approximate an exact DD by encoding a superset of all feasible solutions. The longest path of a relaxed DD therefore provides



(a) Exact MDD (b) Relaxed MDD (c) Restricted MDD

Figure 1: Examples of an exact, a relaxed, and a restricted MDD for a sequencing problem with ground set $\{1, 2, 3\}$. Each arc label shows both the element to be assigned to the corresponding variable and the arc length as $\text{val}(a)|z(a)$. The longest path is highlighted. For the exact MDD, the longest path encodes the optimal permutation $\pi^* = (2, 3, 1)$ with a total prize of 30. The relaxed MDD approximates the exact MDD by merging nodes u and v into node v'. The corresponding longest path encodes the infeasible solution $\pi = (2, 3, 2)$ and yields the upper bound 35. In the restricted MDD, node v and its incident arcs are removed and therefore only a subset of all feasible solutions is encoded. Its longest path is the permutation $\pi = (3, 1, 2)$ of length 25, which provides a lower bound.

an upper bound to the original problem's optimum solution value. A restricted DD, in contrast, encodes only a subset of the feasible solutions, and its longest path therefore yields a lower bound and a possibly promising heuristic solution. Figure 1b and 1c show a relaxed and a restricted DD for the exact DD in Fig. 1a.

Each node $u \in V$ carries problem-specific information through its *state* $\sigma(u)$ that is reached by all the partial solutions corresponding to the paths from the root node to node u. In our case, when the MDDs represent subsets of permutations, each state includes the subset of elements by which the partial solutions may still be extended, thereby defining which outgoing arcs exist; we denote this set as $P(u)$.

Decision diagrams are usually derived from a *dynamic programming* (DP) formulation of the considered problem, and therefore a strong relationship exists between the DP's state transition graph and the nodes of the DD (Hooker 2013). We will see this relationship in more detail when considering the PC-JSOCMSR specifically in Section 5.

There are two fundamental methods for compiling relaxed DDs of limited size. These are, to the best of our knowledge, used in almost all so far published works where relaxed DDs are applied to address combinatorial optimization problems. The *top-down construction* (TDC) starts with just the root node and creates the DD iteratively layer by layer, essentially performing a breadth-first search. The size of the DD is controlled by imposing an upper bound β, called *width*, on the number of nodes at each layer. If the size of a current layer exceeds β, then some nodes of this layer are selected and *merged* so that the layer's size is reduced to at most β. This merging is done in such a way that no paths corresponding to feasible solutions are lost; new paths corresponding to infeasible solutions may emerge, however.

The second frequently applied approach for constructing relaxed DDs is *incremental refinement* (IR). It starts with a trivial relaxed DD, e.g., a DD of width one, which has just one node in each layer. Then two major steps are repeatedly applied until some termination condition is fulfilled, e.g., a maximum number of nodes is reached. In the *filtering* step, the relaxation represented by the DD is strengthened by removing arcs that cannot be part of any path corresponding to a feasible solution. In the *refinement* step, nodes are split into pairs of new replacement nodes in order to remove some of the paths that correspond to infeasible solutions.

Besides TDC and IR, Bergman and Cire (2017) proposed to consider the compilation of a relaxed DD as an optimization problem and investigated a MIP formulation. While this approach is useful for benchmarking different compilation methods on small problem instances, it is computationally too expensive for any practical application. In another work, Römer et al. (2018) suggested a local search framework that serves as a more general scheme to obtain relaxed DDs. It is based on a set of local operations for manipulating and iteratively improving a DD, including the node splitting and merging from IR and TDC, respectively, and arc redirection as a new operator. Again, both approaches are strongly layer-oriented.

Especially in the context of binary DDs, a commonly used extension that frequently yields more compact DDs are so-called *long arcs* (Bryant 1986, Minato 1993). They skip one or more layers and represent multiple variable assignments with one arc. In *zero-suppressed DDs*, variables corresponding to skipped layers take the value zero, while in *one-suppressed* DDs they get value one. Alternatively, a long arc may indicate that the skipped variables can take either value. For example, Bergman et al. (2014b) suggested to use zero-suppressed DDs for the independent set problem, while Kowalczyk and Leus (2018) applied them to solve the pricing problem in a branch-and-price algorithm for parallel machine scheduling.

In conjunction with scheduling and sequencing problems, MDDs were already successfully applied e.g. to single machine scheduling problems (Cire and van Hoeve 2013), the time-dependent traveling salesman problem with and without time windows, the time-dependent sequential ordering problem (Kinable et al. 2017), and job sequencing with time windows and state-dependent processing times (Hooker 2017).

All these approaches utilize MDDs for permutations similar to our example in Fig. 1. An alternative way of representing sets of permutations as DDs has been described by Minato (2011). It builds upon zero-suppressed decision diagrams and encodes permutations by binary decision variables that indicate the transposition of pairs of elements. While this approach offers interesting advantages concerning certain algebraic operations, it appears nontrivial to efficiently express typical objective functions from routing and scheduling in terms of arc lengths on such DDs.

2.1. MDDs for Problems with Both Selection and Sequencing Decisions

To address problems like the PC-JSOCMSR, the above described MDDs for permutations can be extended in natural ways.

A commonly used approach for modeling problems with multiple different goal states is to use a single target node **t** and connect each other node that corresponds to a feasible end state to this target node with a special *termination arc* of length zero. Such a termination arc a has a special value $\mathrm{val}(a) = \mathrm{T}$ and does not correspond to any classical variable assignment. See Fig. 2a for an example of such an approach in our case.

A simpler method can be used for optimization problems where appending an element to a solution, if feasible, always leads to a solution that is not worse. This is, in particular, the case when all arc lengths are non-negative. Here, we can avoid additional artificial arcs and simply redirect all arcs that lead to a non-extendable state directly to the target node **t**; see Fig. 2b. These redirected arcs may now skip layers. In contrast to the previously discussed long arcs, however, our arcs here still represent single variable assignments. In the remainder of this work, we consider just this simpler redirection approach without explicit termination arcs. However, the algorithmic concepts we present can also be adapted in a straightforward way to the more general DD structure with termination arcs.

A further advantage of our MDDs for problems with both selection and sequencing decisions is illustrated in Fig. 2b–c. In Fig. 2b, consider the substructures rooted at nodes v and v' and note thereby that due to the variable solution length, isomorphic substructures appear. In Fig. 2c, the MDD is condensed by storing this substructure just once, with all arcs leading to the two substructures in Fig. 2b redirected to the single substructure. In this way, many redundancies might be avoided and substantially more compact MDDs representing the same set of solutions may be obtained. Note,

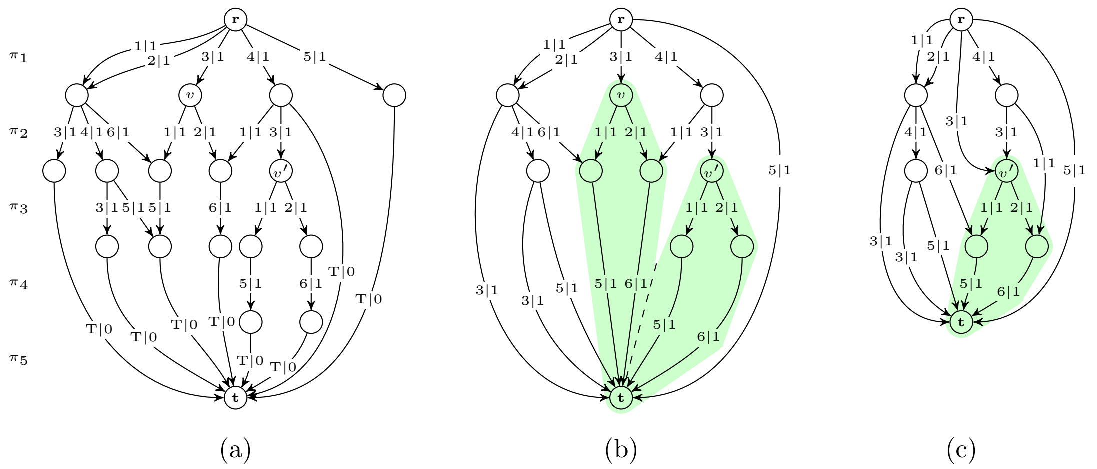


Figure 2: MDD variants for a problem with both selection and sequencing decisions encoding the same set of solutions: (a) using artificial termination arcs with val(a) = T; (b) redirecting all arcs leading to non-extendable states directly to the target node (if appending an element may never yield a worse feasible solution); (c) additionally avoiding multiple instances of isomorphic substructures (shaded parts).

however, that classical DD construction techniques such as TDC and IR are not able to create such an MDD as they rely on the notion that an arc originating at layer i, $i = 1, \dots, n$, (or a long arc passing a layer i) assigns a value to the i-th decision variable. The A*-based construction method we will propose in Section 4 does not rely on the layer-to-variable relationship but more generally just assumes that on any **r**-**t** path, the i-th arc represents an assignment to the i-th variable.

3. Prize-Collecting Job Sequencing with One Common and Multiple Secondary Resources

We consider here specifically the *prize-collecting job sequencing with one common and multiple secondary resources* (PC-JSOCMSR), which was introduced by Horn et al. (2018).

Given is a set of jobs where each job is associated with a prize. Among these jobs, a subset of maximum total prize shall be selected and feasibly scheduled. Each job has individual time windows and can only be performed during one of these. The processing of each job requires two resources: a *common resource*, which all the jobs need for a part of their processing, and a *secondary resource* which is shared by only a subset of the other jobs but needed for the whole processing time. Each resource can only be used for processing one job at a time. As example, consider an application similarly as the one described by Van der Veen et al. (1998): The common resource might be an oven and the secondary resources molds of different types; jobs correspond to different products to be processed in the oven and each product requires a particular mold. There is some preprocessing, where the mold is already required before the prepared product can be put with its mold into the oven; after the heat treatment, the product is taken from the oven but still needs to be cooled within the mold for some time. Only thereafter, the mold can be used for a successive product.

Figure 3 illustrates a solution in which jobs 8, 1, 7, 10, and 4 have been selected and scheduled. The processing of a job is illustrated as a bar with the white part referring to the usage of the common resource. The top row shows when a job uses the common resource (e.g., the oven) and rows one to three show the schedule for each of the three secondary resources (e.g., the three molds).

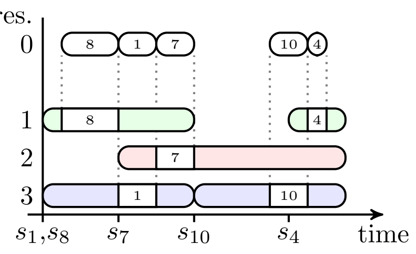


Figure 3: A feasible schedule of jobs $1, 4, 7, 8, 10$ of a PC-JSOCMSR instance with $n = 10$ jobs and $m = 3$ secondary resources.

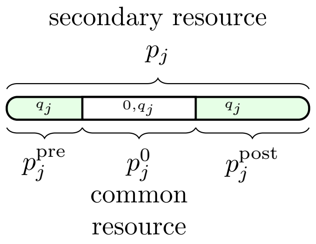


Figure 4: Phases of a job $j \in J$ with respective times and required resources.

3.1. Formal Definition

More formally, we denote the set of (renewable) resources by $R_0 = \{0\} \cup R$, where resource 0 refers to the common resource and $R = \{1, \ldots, m\}$ to m secondary resources. The set of jobs to be considered is $J = \{1, \ldots, n\}$, where n denotes the number of jobs. Each job $j \in J$ has an assigned secondary resource $q_j \in R$ needed for its whole processing time $p_j > 0$, while the common resource is required for the time p_j^0, starting after a pre-processing time $p_j^{\text{pre}} \geq 0$ from the the job's start time. The remaining time for which the secondary resource is still needed, but the common resource is not, is the post-processing time $p_j^{\text{post}} = p_j - p_j^{\text{pre}} - p_j^0 \geq 0$. A selected job must always be performed without preemption. Figure 4 illustrates a job with its required resources and respective times.

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

Last but not least, each job $j \in J$ is associated with a prize $z_j > 0$ and the objective is to select a subset of jobs $S \subseteq J$ and find a feasible schedule for them so that the total prize of these jobs is maximized:

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

A feasible schedule assigns each job in S a feasible starting time. Since each job requires the common resource and since this resource can be used only by one job at a time, any feasible schedule is characterized by the sequence $\pi = (\pi_i)_{i=1,\ldots,|S|}$ in which the selected jobs use the common resource; for example $\pi = (8, 1, 7, 10, 4)$ for the solution in Fig. 4. For such a sequence, valid job starting times can always be computed in a straightforward greedy way. This is done by assigning the jobs in the given order the earliest possible starting times with respect to their time windows and the resource availabilities considering all earlier scheduled jobs. Such a schedule is referred to as a *normalized schedule*. It is easy to see that any feasible schedule always has a corresponding normalized schedule that is also feasible. Consequently, it is sufficient to search the space of all normalized schedules and thus all respective job sequences to solve PC-JSOCMSR.

3.2. Earlier Work on the PC-JSOCMSR

Originally, Van der Veen et al. (1998) has considered a variant of the problem in which all jobs have to be scheduled and the objective is to minimize the makespan. A crucial condition in this work was that the post-processing times were assumed to be negligible compared to the total processing times of the jobs. This simplifies the problem substantially since it implies that the start time of each job only depends on its immediate predecessor. The problem could therefore be modeled as a Traveling

Salesman Problem (TSP) with a special cost structure that allows to solve the problem efficiently in time $O(n \log n)$.

The variant of the problem with general post-processing times, but still with the aim to schedule all jobs and minimize the makespan, has been treated in Horn et al. (2019a) by an A*/beam search anytime algorithm. The authors showed the problem to be NP-hard and could solve instances with up to 1000 jobs to proven optimality. The good computational performance is mainly thanks to a detailed study of the problem structure which yielded a tight dual bound calculation.

The prize-collecting problem variant we consider here, the PC-JSOCMSR, was first described in Horn et al. (2018), where an exact A* algorithm is proposed; see also **?** for an extended version of the original conference paper. To determine tight bounds for this problem variant turned out to be more difficult due to the prize-collecting aspect and the time windows. Different variants of upper bound calculations based on Lagrangian and linear programming relaxation were investigated. Comparisons to a compact MIP model and a MiniZinc CP model solved by different backends indicated the superiority of the A* search. Nevertheless, only instances with up to 30 jobs could be consistently solved to optimality. For instances with 80 jobs, an optimal solution was only found and verified for a few of them.

To obtain dual bounds and heuristic solutions for larger PC-JSOCMSR instances, Maschler and Raidl (2018a) investigated classical TDC and IR approaches to construct relaxed MDDs and a top-down construction for restricted MDDs. In addition, an independent *general variable neighborhood search* (GVNS) metaheuristic was considered. Instances with up to 300 jobs were studied. It turned out that IR frequently yields relaxed DDs of roughly comparable size with stronger upper bounds than TDC. Differences are particularly significant for instances with a larger number of jobs. IR's running times are, however, in general higher than those of TDC for constructing MDDs of comparable size. The heuristic solutions obtained from the restricted DDs were usually better than or on par with the solutions obtained from the GVNS, except for the largest skewed instances, where the GVNS performed better. In our experimental investigations in Section 7 we will also compare to these approaches.

3.3. Applications

As already briefly sketched, the PC-JSOCMSR may arise in the context of the production or processing of certain products on a single machine (the common resource), for example to perform a heat treatment. Some raw material is put into specific molds or fixtures (the secondary resources), which are then sequentially processed on the single machine. In order to use the fixtures/molds again, some post-processing may be required (e.g., cooling). The raw material may only be available on certain times within a limited time horizon and the task is to select the in some sense most valuable subset of products to produce or process on the single machine within the limited time horizon.

Our interest in the PC-JSOCMSR has been primarily motivated from work on two real-world applications in which the PC-JSOCMSR appears as a simplified subproblem at the core.

The first application concerns the scheduling of particle therapies for cancer treatments (Maschler and Raidl 2018b). Here, the common resource corresponds to a synchrotron (i.e., a particle accelerator) in which proton or carbon particles get accelerated to almost light speed and are directed to one of a few treatment rooms in which one patient gets radiated at a time. The typically two to four differently equipped treatment rooms are the secondary resources. Preparations like positioning and a possible anesthesia yield a pre-processing time, while specific medical inspections that are performed after the radiation yield a post-processing time. In the long-term planning, the planning horizon consists of several weeks and due to the overall problem's complexity, a decomposition approach is used in which the PC-JSOCMSR appears as subproblem for each day: To utilize the available working time at a considered day as best as possible, a most valuable subset of all remaining treatments to be performed is selected and scheduled. Not scheduled treatments have to be dealt with at other days. The availability of further required resources, such as medical stuff, is taken into account by setting the jobs' time windows accordingly. Note that the sketched approach is closely related to solving a pricing problem in the context of column generation.

The second application is pre-runtime scheduling of avionic systems (electronic systems in aircraft) as introduced in Blikstad et al. (2018) and **?**. The industrially relevant instances considered in these articles are too complex and large-scale to be addressed directly and instead they need to be solved by some decomposition. The PC-JSOCMSR appears as an important sub-structure both in the exact decomposition approach in Blikstad et al. (2018) and in the matheuristic approach in **?**. Briefly described, the considered system consists of a set of nodes and each of these contains a set of modules (processors) with jobs to be scheduled. In each node, there is a single module called the communication module, which corresponds to the common resource in PC-JSOCMSR. Each node also has a set of application modules, which correspond to the secondary resources. By solving a PC-JSOCMSR, partial and simplified schedules for the nodes can be constructed. In Blikstad et al. (2018) and **?**, this is of relevance in a subproblem where a restriction of the original problem is to be solved and the objective is to schedule as many jobs as possible. Also, to create partial schedules for the nodes would be of interest in rolling-horizon methods for this type of problem.

For more details on both applications we refer to **?**. Benchmark instances reflecting the structure of these two applications have been used in the mentioned earlier works on the PC-JSOCMSR, and we will adopt the larger instances from those and extend the benchmark set in our experimental evaluation.

3.4. Further Related Work

The PC-JSOCMSR might also be modeled as a more general *Resource-Constrained Project Scheduling* (RCPS) problem with maximal time lags by splitting each job according to the resource usage into three sub-jobs; for a survey on RCPS see Hartmann and Briskorn (2010). These three sub-jobs must be executed sequentially without any time lags. However, since each job of the PC-JSOCMSR problem requires three jobs in a RCPS problem and the RCPS also is known to be difficult to solve in practice, this approach does not seem likely to yield better results than a well chosen problem-specific approach.

4. A*-Based Construction of a Relaxed MDD

We propose to construct relaxed MDDs for the PC-JSOCMSR problem and possibly other problems with both selection and sequencing decisions in a novel way that essentially adapts the classical TDC towards the spirit of A* search. Instead of following a breadth-first search we turn towards a best-first search where layers do not play a role anymore. The key characteristics of this scheme are:

1. It naturally avoids multiple nodes for identical states at different layers and consequently multiple copies of isomorphic substructures (cf. Section 2.1 and Fig. 2).

2. Node expansions and the selection of nodes to be merged are guided by an auxiliary upper bound function.

3. Partner nodes for merging are selected by considering state similarity and merging of nodes across different layers is enabled in a natural way.

These features allow to obtain more compact relaxed MDDs that provide tighter upper bounds than the so far used classical construction techniques.

4.1. A* Search

A* search (Hart et al. 1968) is a commonly applied technique in path planning and problem solving. It is well known for its ability to efficiently find best paths in very large (state) graphs. The following brief overview on A* search builds upon the notation already introduced in our DD setting. A* search follows a best-first-search strategy and uses as key ingredient a heuristic function $Z^{\mathrm{ub}}(u)$ that estimates, for each node u reached, the cost to get to the target node $\mathbf{t}$ in a best way, the so-called *cost-to-go*[1]. All not yet expanded nodes, called *open nodes*, are maintained in a priority queue, the *open list* Q. This list is partially sorted according to a priority function

$$f(u) = Z^{\mathrm{lp}}(u) + Z^{\mathrm{ub}}(u) \tag{2}$$

where $Z^{\mathrm{lp}}(u)$ denotes the length of the so far best best path from the root node $\mathbf{r}$ to node u. Initially, Q contains just the root node. The A* search then always takes a node with highest priority from Q and *expands* it by considering all outgoing arcs. Destination nodes that are reached in better ways via the expanded node are updated and newly reached nodes are added to Q. Considering maximization, a heuristic function Z^{ub} that never underestimates the real cost-to-go (i.e., is an upper bound function) is called *admissible*. A* search terminates when $\mathbf{t}$ is selected from Q for expansion. If an admissible heuristic is used, then $Z^{\mathrm{lp}}(\mathbf{t})$ is optimal. From now on let us assume that Z^{ub} is indeed admissible. The efficiency of A* search mostly relies on how well the heuristic function estimates the real cost-to-go.

4.2. Constructing Exact MDDs by A* Search

When performing the A* search, all encountered nodes and arcs that correspond to feasible transitions are stored. If the construction process is carried out until the open list becomes empty and thereby all nodes have been expanded, then a complete MDD is obtained. Alternatively, the A* search's criterion can be applied, and then the search is terminated already when the target node is selected for expansion. In this case, typically substantially less nodes will have been expanded, and the obtained MDD is in general incomplete. Nevertheless, we know due to the optimality condition of A* search that at least one optimal path is contained and thus an optimal solution is indeed represented.

4.3. Constructing Relaxed MDDs

To obtain a compact relaxed MDD we now extend the above A*-based construction by *limiting the open list*. This is achieved by merging similar and less promising nodes when the open list exceeds a certain size ϕ. Details on how we choose the nodes to be merged will be presented in Section 4.4 . Selected nodes are merged in the same problem-specific ways as in traditional relaxed DD construction techniques. In particular, it has to be guaranteed that no paths corresponding to feasible solutions get lost. Section 5.2 will show how this is done in for the PC-JSOCMSR.

When performing this MDD construction until the open list becomes empty, we now obtain a complete relaxed MDD that indeed represents a superset of all feasible solutions and yields an upper bound on the optimal solution value.

Alternatively, we may also here already terminate early once the target node is selected for expansion. Due to the merging and the optimality condition of A* search, we have then obtained a path whose length is a valid upper bound to the optimal solution value, and this bound cannot be further improved by continuing the MDD construction. Only longer paths corresponding to weaker bounds may later arise due to further node merges. Let us denote this best obtained bound by $Z^{\mathrm{ub}}_{\min}$.

Thus, the termination criterion to be used depends on the goal for which the MDD is constructed. If we are only interested in the upper bound or, for example, a DD-based branch-and-bound (Bergman et al. 2016a) shall be performed, the early termination may be very meaningful and can save much time. However, should we indeed need a representation of a complete superset of all feasible solutions, the construction has to be continued.

Algorithm 1 shows the proposed MDD construction technique in pseudo-code. After the initialization phase, the main loop is entered. At each major iteration, a node u with maximum priority $f(u)$ is taken from the open list Q. As long as the target node $\mathbf{t}$ was not chosen for expansion, the node's f value provides a valid upper bound and $Z^{\mathrm{ub}}_{\min}$ is updated accordingly in Line 7. If $\mathbf{t}$ was chosen, the optional early termination takes places.

[1]Note that the term cost-to-go is more fitting in the context of minimization. We aim at maximizing the total length, benefit, or prize, and one might therefore consider "length-to-go" more suitable. Nevertheless, we stay here with the term cost-to-go as it is commonly used in the literature.

Algorithm 1: A*-based construction of a relaxed MDD

```
   Input: open list size limit $\phi$
   Output: relaxed MDD $M = (V, A)$ and upper bound to optimal solution value
1  create root node $\mathbf{r}$ corresponding to initial state;
2  open list $Q \leftarrow \{(\mathbf{r}, f(\mathbf{r}) = Z^{\mathrm{ub}}(\mathbf{r}))\}$;
3  $Z^{\mathrm{ub}}_{\min} \leftarrow Z^{\mathrm{ub}}(\mathbf{r})$; $\mathbf{t}$-expanded $\leftarrow$ false;
4  while $Q \neq \emptyset$ do
5      $u \leftarrow$ pop node with largest $f(u)$ from $Q$;
6      if not $\mathbf{t}$-expanded then
7          $Z^{\mathrm{ub}}_{\min} \leftarrow \min(Z^{\mathrm{ub}}_{\min}, f(u))$ ;
8      if $u = \mathbf{t}$ then
9          $\mathbf{t}$-expanded $\leftarrow$ true;
10         // optional, if just the upper bound is of interest:
11         return incomplete MDD $M$ and upper bound $Z^{\mathrm{ub}}_{\min}$;
12     if $u$ not yet expanded then
13         // expand node $u$:
14         foreach feasible successor state $\Sigma$ of $\sigma(u)$ do
15             if $\nexists v \in V \mid \sigma(v) = \Sigma$ then
16                 add new node $v$ to $V$ with $\sigma(v) = \Sigma$, $Z^{\mathrm{lp}}(v) = 0$;
17             add new arc $a = (u, v)$ to $A$;
18             if $Z^{\mathrm{lp}}(u) + \mathrm{z}(a) > Z^{\mathrm{lp}}(v)$ then
19                 $Z^{\mathrm{lp}}(v) \leftarrow Z^{\mathrm{lp}}(u) + \mathrm{z}(a)$;
20                 $Q \leftarrow Q \cup \{(v, f(v) = Z^{\mathrm{lp}}(v) + Z^{\mathrm{ub}}(v))\}$;
21     else
22         // re-expand node $u$:
23         foreach arc $a = (u, v) \in A$ do
24             if $Z^{\mathrm{lp}}(u) + \mathrm{z}(a) > Z^{\mathrm{lp}}(v)$ then
25                 $Z^{\mathrm{lp}}(v) \leftarrow Z^{\mathrm{lp}}(u) + \mathrm{z}(a)$;
26                 $Q \leftarrow Q \cup \{(v, f(v) = Z^{\mathrm{lp}}(v) + Z^{\mathrm{ub}}(v))\}$;
27     // reduce size of $Q$:
28     if $|Q| > \phi$ then
29         try to merge nodes in $Q$ until $|Q| \leq \phi$ according to Alg. 2;
30 return relaxed MDD $M$ and upper bound $Z^{\mathrm{ub}}_{\min}$;
```

Next, the case when node u has not yet been expanded is handled by considering all feasible transitions from state $\sigma(u)$ and creating new nodes and arcs accordingly. If thereby a new path to a node v increases $Z^{\mathrm{lp}}(v)$, then node v is (re-)inserted into Q. If node u was already expanded, a re-expansion has to take place because a longer path to u, yielding a larger $Z^{\mathrm{lp}}(u)$, has been found in an iteration after the node's original expansion. This is done by propagating the updated $Z^{\mathrm{lp}}(u)$ to all its successor nodes and evaluating if they also need re-expansions. Note that, in general, we cannot avoid such re-expansions even when the upper bound function is consistent since node merges may lead to new longer paths.

After each node expansion, the algorithm checks if the size of the open list $|Q|$ exceeds the limit ϕ. If this is the case, then the algorithm tries to reduce Q by merging nodes as explained in the next section. Algorithm 1 terminates regularly when the open list becomes empty and returns the relaxed MDD together with the best obtained upper bound $Z^{\mathrm{ub}}_{\min}$.

Algorithm 2: Reduce Q by merging nodes

```
Input: open list $Q$, global set of collector nodes $V^{\mathrm{c}}$ (initially empty)
Output: possibly reduced open list $Q$
1  for $u \in Q$ in increasing order of values $Z^{\mathrm{lp}}(\cdot)$ do
2      if $|Q| \le \phi$ then
3          break;
4      while $u$ not expanded $\wedge \; \exists v \in V^{\mathrm{c}} \mid L(v) = L(u) \wedge u \neq v \wedge$ $v$ not expanded do
5          create new node $v'$ with merged state $\sigma(v') = \sigma(u) \oplus \sigma(v)$;
6          remove $u$ from $Q$ and $v$ from $Q$ and $V^{\mathrm{c}}$;
7          if $\exists v'' \in V \mid \sigma(v'') = \sigma(v')$ then
8              $f''_{\mathrm{old}} \leftarrow f(v'')$;
9              redirect all incoming arcs from $v'$ to $v''$;
10             if $f(v'') > f''_{\mathrm{old}}$ then
11                 $Q \leftarrow Q \cup \{(v'', f(v'') = Z^{\mathrm{lp}}(v'') + Z^{\mathrm{ub}}(v''))\}$;
12             $v' \leftarrow v''$;
13         else
14             add node $v'$ to $V$;
15             $Q \leftarrow Q \cup \{(v', f(v') = Z^{\mathrm{lp}}(v') + Z^{\mathrm{ub}}(v'))\}$;
16         $u \leftarrow v'$;
17     if $u$ not expanded then
18         insert $u$ into $V^{\mathrm{c}}$;
19 return $Q$;
```

4.4. Reducing the Open List by Merging

Merging different nodes usually introduces new paths corresponding to infeasible solutions, and this typically weakens the upper bound obtained. Therefore we aim at quickly identifying nodes for merging that (a) are less likely to be part of some finally longest path; (b) are associated with similar states, since this should imply that the merged state still is a strong representative for both; (c) do not introduce cycles in the MDD as they would lead to infinite solutions; and (d) ensure that the open list gets empty after a finite number of expansions. The last two aspects are crucial conditions to ensure a proper termination of the approach, and they are not trivially fulfilled due to the possibility to merge across different layers.

Aspect (a) is considered by iterating over the nodes in the open list in an increasing Z^{lp}-order and trying to merge each node with a suitably selected *partner node* in a pairwise fashion until the size of the open list does not exceed ϕ anymore. The motivation for the increasing Z^{lp}-order is that A* search has so far postponed the expansion of these nodes while other nodes with comparable Z^{lp} values have already been expanded. Therefore, the nodes with small Z^{lp} values can be argued to be less likely to appear in a longest path.[2]

The selection of the partner node for merging is done considering aspects (b) to (d) by utilizing a global set of so-called *collector nodes* V^{c}. To this end, we define a problem-specific labeling function $L(u)$ that maps the data associated with a node u—in particular its state $\sigma(u)$—to a simpler label of a restricted finite domain $\mathcal{D}_L$, thereby partitioning the nodes into subsets of similar nodes. Our labeling function, for example, may drop, aggregate, or relax parts of the states considered less important and condense the information in this way. Similar principles as in state-space relaxation (Christofides et al.

[2]We remark that we considered in preliminary experiments also an increasing f order, thus processing the priority queue essentially in reverse order. While we obtained mostly MDDs of roughly comparable quality, they were sometimes significantly larger and more computation time was needed.

1981) can be applied. The labeling function, however, may additionally also consider the upper bound $Z^{\mathrm{ub}}(u)$ as criterion for similarity; experimental results in Section 7 will show the particular usefulness of this. The global set of collector nodes V^{c} is initially empty and realized as a dictionary (e.g., hash table) indexed by the labels so that for each label in $\mathcal{D}_L$, there is at most one collector node in V^{c}, and thus $|V^{\mathrm{c}}| \leq |\mathcal{D}_L|$. In this way, we can efficiently determine for any node u if a related collector node with the same label $L(u)$ already exists and, in this case, directly access it.

Algorithm 2 shows the whole procedure to reduce the open list. As long as the open list is too large, nodes are selected in increasing Z^{lp}-order. For a chosen node u, it is checked if it is not yet expanded and if a corresponding collector node v, that is also not yet expanded, exists (Line 4). In this case, u and v are merged, yielding the new node v' with state $\sigma(v') = \sigma(u) \oplus \sigma(v)$, where $\oplus$ denotes the problem-specific state merging operation. All incoming arcs from u and v will be redirected to the new node v'. Consequently, u is removed from Q and v from Q as well as V^{c}. Next, we have to integrate the new node v' into the node set V by avoiding multiple nodes in the set V associated with the same state (Line 7). Furthermore v' becomes a collector node in V^{c}, essentially replacing the former collector node v. Node v' may, however, have a different label than the former v, and some other collector node with the same label as v' may already exists in V^{c}. In this case, we iterate the merging with these nodes by continuing the while-loop in Line 4.

Note that Algorithm 2 shows the main idea pointing out the important steps. In a concrete implementation, a few additional corner cases need to be considered, in particular when collector nodes get changed (e.g., expanded) between two calls of Algorithm 2.

5. A*-Based MDD Construction for PC-JSOCMSR

We now consider the problem-specific details to apply the A*-based construction of a relaxed MDD specifically to PC-JSOCMSR. The root node $\mathbf{r}$ represents the empty schedule. An arc a corresponds to job $\mathrm{val}(a)$ being appended to a partial solution represented by a permutation π and this job being scheduled at its earliest feasible time. The length associated with arc a is the prize of its corresponding job, i.e., $z(a) = z_{\mathrm{val}(a)}$. In this way, the length of a path from $\mathbf{r}$ to some node u corresponds to the total prize of all so far scheduled jobs. The target node $\mathbf{t}$ subsumes all states that cannot be further extended, and thus we want to find a maximum length $\mathbf{r}$-$\mathbf{t}$ path.

In the following we define the states and state transitions as well as the underlying DP formulation for PC-JSOCMSR, as introduced in Horn et al. (2018).

5.1. States and State Transitions

A state in PC-JSOCMSR must describe all relevant aspects in order to determine the earliest starting time of any successive job that can be scheduled. For a node u this is the tuple $\sigma(u) = (P(u), t(u))$ consisting of

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

To simplify the consideration of the time windows, we introduce the function

$$\mathrm{eft}(j,t) = \min(\{t' \geq t \mid \exists k : \left[t', t' + p_j\right] \subseteq W_{jk}\} \cup \{T^{\max}\}), \tag{3}$$

that yields the *earliest feasible time*, not smaller than the provided time $t \leq T^{\max}$, at which job j can be scheduled according to the time windows W_j of the given job $j \in J$. The value $\mathrm{eft}(j,t) = T^{\max}$ indicates that job j cannot be feasibly included in the schedule at time t or later.

The state of the root node is $\sigma(\mathbf{r}) = (P(\mathbf{r}), t(\mathbf{r})) = (J, (T^{\min}, \ldots, T^{\min}))$ and represents the original instance of the problem, with no jobs scheduled or excluded yet, and the target node's state is $\sigma(\mathbf{t}) =$

$(P(\mathbf{t}), t(\mathbf{t})) = (\emptyset, (T^{\max}, \ldots, T^{\max}))$. An arc $a = (u, v)$ represents the transition from state $(P(u), t(u))$ to state $(P(v), t(v))$ that is achieved by scheduling job $j = \mathrm{val}(a)$, $j \in P(u)$, at its earliest possible time w.r.t. vector $t(u)$. When performing this transition, the *start time* of job j w.r.t. state $(P(u), t(u))$ is

$$\mathrm{s}\left((P(u), t(u)), j\right) = \mathrm{eft}\left(j, \max(t_0 - p_j^{\mathrm{pre}}, t_{q_j})\right). \tag{4}$$

The transition function to obtain the successor state $(P(v), t(v))$ of state $(P(u), t(u))$ when scheduling job $j \in P(u)$ is

$$\tau\left((P(u), t(u)), j\right) = \begin{cases} (P(u) \setminus \{j\}, t(v)), & \text{if } \mathrm{s}((P(u), t(u)), j) < T^{\max}, \\ \hat{0}, & \text{else,} \end{cases} \tag{5}$$

with

$$t_0(v) = \mathrm{s}((P(u), t(u)), j) + p_j^{\mathrm{pre}} + p_j^0, \tag{6}$$

$$t_r(v) = \mathrm{s}((P(u), t(u)), j) + p_j, \qquad \text{for } r = q_j, \tag{7}$$

$$t_r(v) = t_r(u), \qquad \text{for } r \in R \setminus \{q_j\}, \tag{8}$$

and where $\hat{0}$ represents the infeasible state. If a transition results in the infeasible state, the corresponding arc and node are omitted in the MDD.

Using these definitions of states and transitions, we can express the optimal solution value of the PC-JSOCMSR subproblem represented by a node u by the recursive DP formulation

$$Z^*(u) = \max\left(\{Z^*(\tau(\sigma(u), j)) + z_j \mid j \in P(u), \tau(\sigma(u), j) \neq \hat{0}\} \cup \{0\}\right) \tag{9}$$

and the overall PC-JSOCMSR solution value is $Z^*(\mathbf{r})$.

In our implementation, each determined state further undergoes a strengthening procedure described in Horn et al. (2018) and summarized in A. This state strengthening exploits dominance relationships without omitting any feasible solutions. It typically reduces the number of states that need to be considered substantially and helps to earlier recognize infeasible transitions.

Concerning the auxiliary upper bound function $Z^{\mathrm{ub}}(u)$ for the cost-to-go from a node u, we investigated in **?** different fast-to-compute alternatives for PC-JSOCMSR. We adopt here the strongest one which is based on solving a set of linear programming relaxations of knapsack problems; B repeats details on how this bound is calculated.

5.2. Merging of States

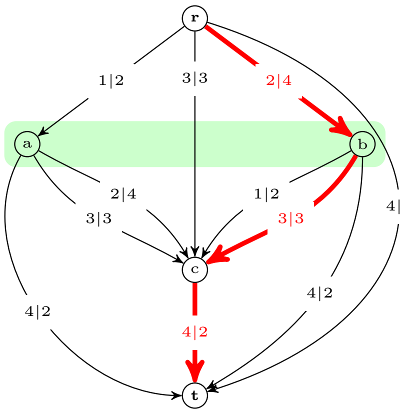
In order to compile a relaxed MDD we further have to define the merging operation. Here, when two nodes $u, v \in V$ are merged into a new single node, the merged state is

$$(P(u), t(u)) \oplus (P(v), t(v)) = \left(P(u) \cup P(v), (\min(t_r(u), t_r(v)))_{r \in R_0}\right). \tag{10}$$

By this construction, the merged state allows all feasible extensions that both original states did. Additional extensions and originally infeasible solutions may, however, become feasible due to the merge, as is usually the case in relaxed DDs. The validity of the merging operation $\oplus$ is discussed in C. If possible, the obtained state is further strengthened as described above.

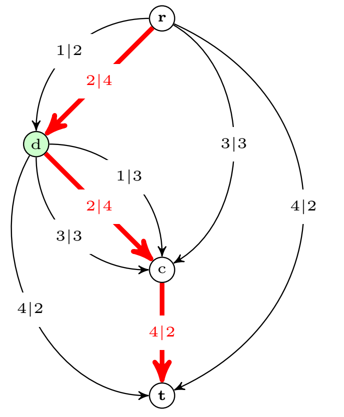
Figure 5 shows an example of an exact MDD and a corresponding relaxed MDD for a small PC-JSOCMSR instance. The states associated with the nodes are detailed in the tables below each MDD. Arc labels indicate the scheduled job and its prize. In the exact MDD, the longest path is highlighted and it has a total length of nine. The corresponding optimal solution is given by the sequence $\pi^* = (2, 3, 4)$ and the respective schedule is depicted on the right side of the figure. A relaxed MDD is shown in the middle; it has been obtained by merging nodes a and b from the exact MDD, yielding node d. The longest path of this relaxed MDD has length ten and it represents the sequence $\pi^* = (2, 2, 4)$, where job 2 is scheduled twice. It can here be easily verified that all $\mathbf{r}$–$\mathbf{t}$ paths in the exact MDD, which correspond to all feasible solutions of this PC-JSOCMSR instance, have corresponding paths in the relaxed MDD, but there exist additional paths representing infeasible solutions such as $(2, 2, 4)$.

Exact MDD



u	$P(u)$	$t(u)$	$Z^{\mathrm{lp}}(u)$
r	$\{1,2,3,4\}$	$(1,0,3)$	0
a	$\{2,3,4\}$	$(3,4,3)$	2
b	$\{1,3,4\}$	$(3,4,3)$	4
c	$\{4\}$	$(9,14,8)$	7
t	$\{\}$	$(14,14,14)$	9

Relaxed MDD



u	$P(u)$	$t(u)$	$Z^{\mathrm{lp}}(u)$
r	$\{1,2,3,4\}$	$(1,0,3)$	0
d	$\{1,2,3,4\}$	$(3,4,3)$	4
c	$\{4\}$	$(9,14,8)$	7
t	$\{\}$	$(14,14,14)$	10

Instance:

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

Opt. solution π^*: $Z(\pi^*) = 9$

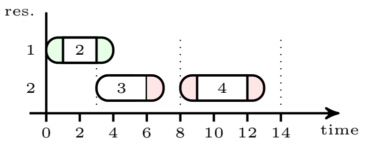


Figure 5: Examples of an exact MDD and a relaxed MDD for a PC-JSOCMSR instance with $n = 4$ jobs and $m = 2$ secondary resources. In the relaxed MDD, the original nodes a and b have been merged.

5.3. Labeling Function for Collector Nodes

As a final major component, closely related to merging, we have to define the labeling function $L(u)$ used for indexing the collector nodes V^c. Remember that this function should partition the set of nodes into subsets such that nodes within a subset are similar enough to be promising to merge; thus, similar nodes should tend to get the same label.

In case of PC-JSOCMSR, we use for a node u the triple $L(u) = (t_0(u), r(u), Z^{\mathrm{ub}}(u))$ as label, where $t_0(u)$ is again the time from which on the common resource is available, $r(u)$ refers to the secondary resource of the job scheduled last in the so far longest path to node u (ties are resolved by using the resource identified first), and $Z^{\mathrm{ub}}(u)$ is the upper bound for the cost-to-go.

Note that by this definition, we do not explicitly consider $P(u)$, the set of jobs that might still be scheduled, nor t_r, $r \in R$, the individual availability times of the secondary resources. Instead of the latter, $r(u)$ is used as a rough substitute. The upper bound $Z^{\mathrm{ub}}(u)$ is an important additional indicator that can be seen to somehow summarize important information about the state of node u. In summary, two nodes are only merged in our A*-based construction if (a) the common resource 0 is used to the same extent, (b) the last used secondary resource is the same, and (c) the values of the problem-specific upper bounds coincide.

Note that a merged node will have the same t_0 value as the original nodes according to Eq. (10). Since each job requires the common resource 0 for a positive time, each transition from a node to a successor node increases the corresponding t_0 value. From this follows the important property that the t_0 values strictly increase along any path in our MDD. Consequently, it holds that cycles cannot occur and that the open list gets empty in a finite number of iterations (since when the t_0 values strictly increase along any path, the set of jobs that might be scheduled will decrease due to the deadline of the jobs). Hence, the increase of t_0 in each state transition helps to guarantee that the algorithm terminates with a complete relaxed MDD.

In Section 7 we will experimentally investigate also the following simpler labeling functions: $L^1(u) = t_0(u)$, $L^2(u) = (t_0(u), r(u))$, and $L^3(u) = (t_0(u), Z^{\mathrm{ub}}(u))$.

Besides the argued theoretical convergence, it might be the case that the practical running time of the algorithm is still too large due to the not strongly limited domain size of the labels: Values $t_0(u)$ as well as $Z^{\mathrm{ub}}(u)$ may be continuous and in the worst case, exponentially many different values may emerge in the course of our algorithm, leading to a potentially exponential number of collector nodes. In our experiments in Section 7, this situation did not occur. In case that it does, discretizing these values in the labeling function by appropriate rounding can be a solution.

5.4. Dominated Merging

Algorithm 2 does not merge already expanded nodes since, in general, the operations of re-evaluating and updating the expanded sub-graphs would be too expensive. However, sometimes it is possible to merge nodes with already expanded collector nodes without further evaluations and updates. Let $v \in Q$ be a not yet expanded node and $u \in V$ be an already expanded node. If $\sigma(v) \oplus \sigma(u) = \sigma(u)$, $Z^{\mathrm{lp}}(v) \leq Z^{\mathrm{lp}}(u)$, and $t_0(v) = t_0(u)$ holds, then it is possible to merge v into u without changing the state of u and without increasing the length of the currently longest path to it. The last two conditions are important to (a) safely omit the re-expansion of node u and (b) to efficiently identify such possible merges by additionally indexing all so far encountered nodes $u \in V$ by their $t_0(u)$ values.

After each node expansion, each new or changed node in Q is considered for this type of merge by checking the condition in conjunction with all other nodes in V that have the same t_0 value. If a pair of nodes u and v that fulfills this condition is found, we remove v from the open list and merge v into u by redirecting all incoming arcs from v to u. Since this kind of merge does not introduce any relaxation loss, we perform this procedure after every node expansion even if $|Q| \leq \phi$.

5.5. Tie Breaking in the Priority Function

The nodes in the open list Q are sorted according to the value of the priority function f, given in Eq. (2). It is not uncommon that different nodes have the same f-value, and we therefore use the following two-stage tie breaking in order to further guide the algorithm in a promising way. First, if two nodes have the same f-value, we always prefer *exact* nodes over *non-exact* nodes. We call a node exact when it has a longest path from the root node that does not contain any merged node where the merging induced a relaxation loss. In other words, an exact node is guaranteed to have a feasible solution that corresponds to this longest path. Such nodes are considered more promising to expand than non-exact nodes with the same f-value. In case of a remaining tie, we prefer nodes where the corresponding state has fewer jobs that may still be scheduled, i.e., we prefer nodes u with smaller $|P(u)|$.

6. Construction of a Restricted MDD Based on a Relaxed MDD

A restricted MDD represents only a subset of all feasible solutions. It is primarily used to obtain feasible solutions and corresponding lower bounds. The construction usually follows a layer-by-layer top-down approach (Bergman et al. 2014a, 2016b). As for relaxed MDDs, the size of a restricted MDD is typically limited by imposing a maximum width β for each layer. Whenever the allowed width is exceeded, nodes are selected from the current layer according to a greedy criterion and removed together with their incoming arcs. Note that this approach for obtaining promising feasible solutions closely corresponds to the well-known *beam search* metaheuristic (Ow and Morton 1988).

So far, we are only aware of previous approaches that construct restricted DDs independently of relaxed DDs. However, an earlier construction of a relaxed DD will, in general, have already collected substantial information. We propose to exploit this information in a successive construction of a restricted DD. The goal is to speed up the construction of the restricted DD and/or to obtain a stronger restricted DD representing better solutions.

Throughout this section, we denote all elements of restricted MDDs with primed symbols, while corresponding symbols of relaxed MDDs are not primed. Our approach applies the common top-down compilation principle. Each node $u' \in V'$ in the restricted MDD always has a *corresponding node* $u \in V$ in the relaxed MDD M in the sense that a path from $\mathbf{r}'$ to u' represents a feasible partial solution that is also represented in M by a path from $\mathbf{r}$ to node u. In other words, the node $u' \in V'$

Algorithm 3: Construction of a restricted MDD based on a relaxed MDD

```
Input: relaxed MDD $M = (V, A)$, maximum width $\beta$
Output: restricted MDD $M' = (V', A')$
1  $V'_1 \leftarrow \{\mathbf{r}'\}$; $A' \leftarrow \emptyset$;
2  $l \leftarrow 1$;
3  while $V'_l \neq \emptyset$ do
4      $V'_{l+1} \leftarrow \{\}$;
5      foreach node $u' \in V'_l$ do
6          let $u \in V$ be the node corresponding to $u'$ w.r.t. the path from the root;
7          foreach outgoing arc $a = (u, v)$ of node $u$ do
8              if $\tau(\sigma(u'), \mathrm{val}(a)) = \hat{0}$ then
9                  continue with next arc;
10             if $|V'_{l+1}| = \beta \wedge$ node $v$ would be removed from $V'_{l+1} \cup \{v\}$ then
11                 continue with next arc;
12             $\Sigma \leftarrow \tau(\sigma(u'), \mathrm{val}(a))$; strengthen $\Sigma$;
13             if $\nexists v' \in V'_{l+1} \mid \sigma(v') = \Sigma$ then
14                 add new node $v'$ to $V'_{l+1}$ and set $\sigma(v') = \Sigma$;
15             add new arc $a' = (u', v')$ to $A'$;
16             if $|V'_{l+1}| > \beta$ then
17                 select and remove a node from $V'_{l+1}$ with its incoming arcs according to a greedy
                     criterion;
18     $l \leftarrow l + 1$;
19 return $M' = (V', A')$ with $V' = V'_1 \cup \ldots \cup V'_{l-1}$;
```

that corresponds to a node $u \in V$ is the node that can be reached by the same sequence of scheduled jobs. For each newly created node in the restricted MDD, we keep track of its corresponding node in the relaxed MDD.

When expanding node u', this corresponding node u will allow us to skip certain transitions in the restricted MDD without evaluating them, i.e., we avoid to introduce the corresponding arcs and successor nodes. In this way, a vast amount of arcs and nodes for states that cannot lead to an optimal solution may be omitted.

Algorithm 3 shows this compilation of a restricted MDD M' that utilizes the relaxed MDD M. We starts with the first layer that consists of the root node $\mathbf{r}'$. Then, each successive layer V'_{l+1} is built from the preceding layer V'_l by creating nodes and arcs for feasible transitions from the states associated with the nodes in V'_l.

Here comes the first novel aspect: For each node u' in layer V'_l we consider only state transitions corresponding to outgoing arcs of the respective node u in the relaxed MDD. Other potentially feasible state transitions do not need to be considered since we know from the relaxed MDD that they cannot lead to an optimal feasible solution. Note, however, that the relaxed node u might have outgoing arcs representing transitions that are actually infeasible for node u' in the restricted MDD. This may happen since the states of u' and u do not need to be the same but u' may dominate u due to merged nodes on the path from $\mathbf{r}$ to u in the relaxed MDD. In Line 8, our algorithm therefore checks the feasibility of the respective transition (remember that $\hat{0}$ represents the infeasible state) and skips infeasible ones. For PC-JSOCMSR, this feasibility check simply corresponds to testing if $\mathrm{val}(a) \in P(u)$.

When we have reached the maximum allowed width at the current layer, we can make an efficient pre-check if the node v' that would be created next would be removed later when the set V'_{l+1} is greedily reduced to β nodes. To this end, we evaluate the criterion that is used to decide which nodes are removed from the current layer for the corresponding node v in the relaxed MDD in conjunction with the so far obtained set V'_{l+1}. If this criterion is chosen in a sensible way, the evaluation for v will never indicate a removal of node v when v' would not be removed, since either the associated states are identical or the state of v' dominates the state of v. In our algorithm, Line 10 realizes this pre-check and correspondingly skips the respective transitions.

For the remaining transitions, Line 12 calculates the obtained new state Σ and creates the corresponding node v' if no node in V' exists yet for Σ. Then, a new arc (u', v') representing the transition in the restricted MDD is added to A'. Finally, if V'_{l+1} has grown to more than β nodes, a node is removed according to the used greedy criterion.

A typical way to select the nodes for removal at each layer is to take the nodes with the smallest lengths of their longest paths from the root node $\mathbf{r}'$, i.e., the nodes with the smallest $Z^{\mathrm{lp}}(v')$, $v' \in V'_{l+1}$ (Bergman et al. 2014a, 2016b). As already observed in Maschler and Raidl (2018a) this strategy is not beneficial for PC-JSOCMSR since it disregards the advances in the time line. Instead, we remove nodes with the smallest $Z^{\mathrm{lp}}(v')/t_0(v')$ ratios in our implementation for the PC-JSOCMSR. When applying this removal criterion to the corresponding node v of the relaxed MDD in Line 10, it holds that $Z^{\mathrm{lp}}(v)/t_0(v) \geq Z^{\mathrm{lp}}(v')/t_0(v')$ as $Z^{\mathrm{lp}}(v) \geq Z^{\mathrm{lp}}(v')$ and $t_0(v) \leq t_0(v')$ since state $\sigma(v')$ is equal to or dominates state $\sigma(v)$. We can even sharpen this estimation by using $(Z^{\mathrm{lp}}(u') + z(a))/t_0(v)$ and thus take advantage of our knowledge of $Z^{\mathrm{lp}}(u')$ and the respective transition cost $z(a)$ to reach node v.

The benefits of exploiting the relaxed MDD in the compilation of the restricted MDD depends on how closely the exact states in the restricted MDD are approximated by the corresponding states in the relaxed MDD as well as the size of the solution space encoded in the relaxed MDD. Various filtering techniques, as for example described by Cire and van Hoeve (2013) for sequencing problems, can substantially reduce relaxed MDDs, and consequently, their application to the relaxed MDD before its exploitation in the construction of the restricted MDD may be advantageous.

7. Computational Results

The A*-based construction of a relaxed MDD for the PC-JSOCMSR, which we abbreviate in the following by A*C, as well as the approach from the last section to further derive a restricted MDD were implemented in C++ using GNU g++ 5.4.1. All tests were performed on a cluster of machines with Intel Xeon E5-2640 v4 processors with 2.40 GHz in single-threaded mode with a memory limit of 16 GB per run.

We use two non-trivial sets of benchmark instances from **?** and, applying the same randomized construction scheme, further extended these to include instances with up to 500 jobs. The first set is based on characteristics from the particle therapy application scenario and denoted here as set P (referred to as set B in the former work), whereas the second instance set is based on the avionic system scheduling scenario and called set A; cf. Section 3.3.

The two instance sets differ substantially in their structure. For the particle therapy ones, the number of jobs that can approximately be scheduled grows linearly with the instance size, and the prize for each job is correlated to the time the job needs the common resource. In contrast, for the avionic instances, the number of jobs that can be scheduled stays in the same order of magnitude irrespective of the instance size and the prize does not depend on a job's processing time but instead on a priority. The number of secondary resources is $m \in \{2, 3\}$ in set P, which corresponds to the available rooms at a real particle therapy center in Austria, and $m \in \{3, 4\}$ in set A; note that in general more secondary resources make the problem significantly easier to solve as the common resource tends to become the sole bottleneck. As we will see in the following results, the structural differences in the instance sets also impact the obtained relaxed MDDs. In particular, the height of relaxed MDDs compiled for the particle therapy instances grows with the problem size, whereas the relaxed MDDs obtained for avionic instances typically have a height of the same magnitude. All instances are available at `https://www.ac.tuwien.ac.at/research/problem-instances` and are described in

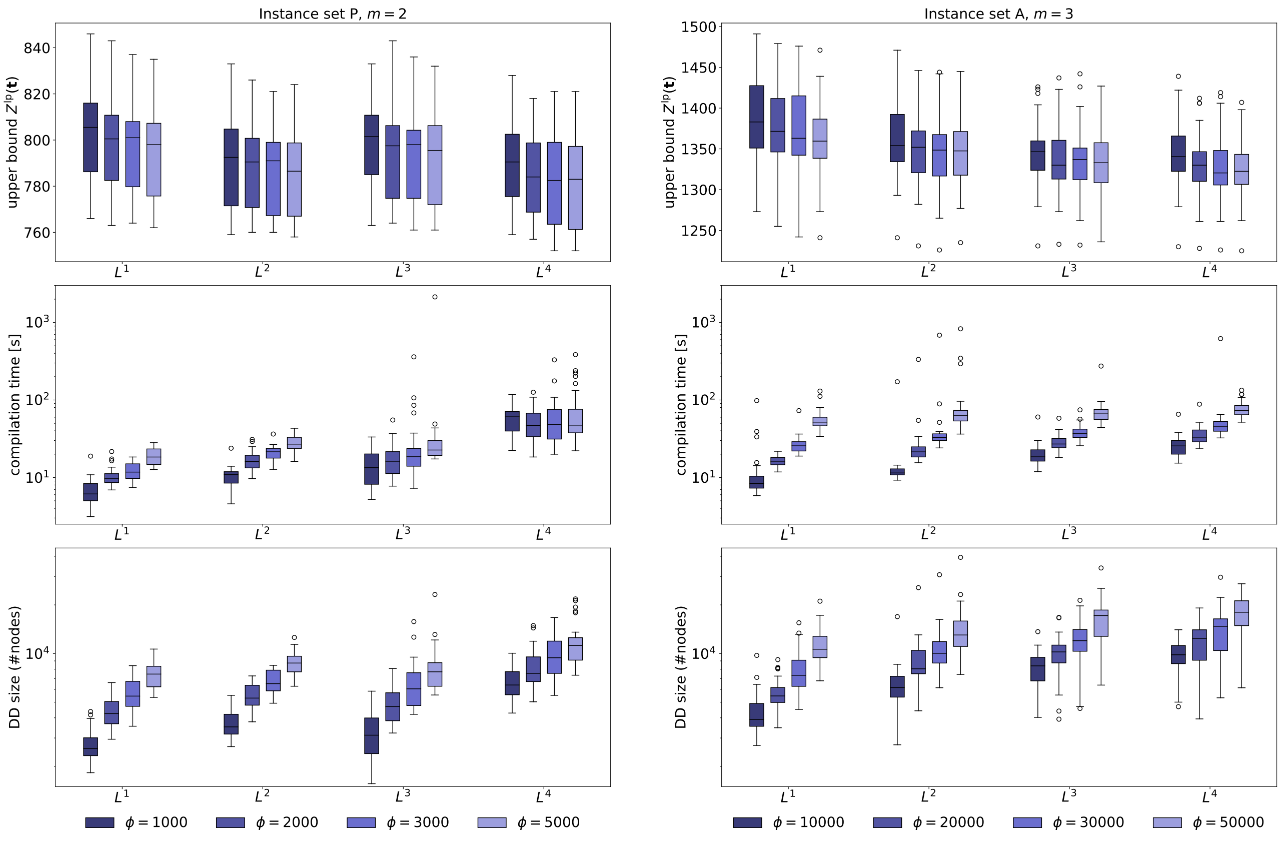


Figure 6: Comparison of open list size limits ϕ and labeling functions L^i, $i = 1, \ldots, 4$, for instances of sets P and A with 250 jobs and $m = 2$ and $m = 3$ secondary resources, respectively.

more detail in D.

This section is structured such that first, Section 7.1 presents results from studying the impact of different values for the open list size limit ϕ and of different choices for the labeling function $L(u)$ in the compilation of relaxed MDDs with A*C. Thereafter, in Section 7.2, we compare the quality of upper bounds obtained from A*C to those from other approaches. Section 7.3 finally compares primal bounds obtained from the derived restricted MDDs to those from other heuristic and exact approaches for PC-JSOCMSR.

7.1. Impact of Open List Size Limit ϕ and Different Labeling Functions

We tested A*C with different open list size limits ϕ and four different variants of the labeling function $L(u)$ used for mapping nodes to collector nodes. The considered labeling function variants are $L^1(u) = t_0(u)$, $L^2(u) = (t_0(u), r(u))$, $L^3(u) = (t_0(u), Z^{\text{ub}}(u))$ and $L^4(u) = L(u) = (t_0(u), r(u), Z^{\text{ub}}(u))$, as proposed in Section 5.3. Figure 6 illustrates the impact of the different choices for ϕ and the labeling function on middle-sized instances with 250 jobs of set P with $m = 2$ and of set A with $m = 3$, respectively.

For each combination of value for ϕ and labeling function variant, there is a box plot drawn that summarizes the obtained results over all 30 instances of the corresponding category. The diagrams at the top show the lengths $Z^{\text{lp}}(\mathbf{t})$ of the longest paths from the obtained relaxed MDDs, whereas the diagrams in the middle show the corresponding CPU-times for compiling the MDDs. Moreover, the diagrams at the bottom state the size of the relaxed MDDs in terms of the number of nodes. The diagrams to the left in Figure 6 show the results for instance set P using the values $\phi \in \{1000, 2000, 3000, 5000\}$. As one could expect, we see that with increasing ϕ, the lengths of the longest paths of the obtained relaxed MDDs in general get smaller, i.e., the obtained upper bounds become stronger, while the MDD sizes and computation times naturally increase. Thus, parameter ϕ indeed allows to control the MDD's size, although not in such a direct linearly related fashion as the width-limit in a classical top-down construction. This effect can be observed for all labeling functions. Concerning the different labeling functions, $L^1(u) = t_0(u)$ yields relaxed MDDs with in general the smallest sizes, but also the weakest bounds. This is, however, also achieved in the shortest times. The reason for this is that labeling function L^1 does only consider the time from which on the common resource is available and has therefore the smallest domain among the four considered labeling functions. Hence, when using L^1, there are in general far more node merges than with one of the other more complex labeling functions which provide larger domains and therefore a finer differentiation of nodes. It can clearly be seen that the additional consideration of $r(u)$ or $Z^{\mathrm{ub}}(u)$ in the labeling function in general significantly improves the obtained bounds $Z^{\mathrm{lp}}(\mathbf{t})$, and the combination of all these aspects in L^4 provides the best results. This, however, at the cost of larger MDDs and higher running times. The smallest median longest path length of 783 for instances with two secondary resources were obtained when limiting the size of the open list to $\phi = 5000$ nodes and using L^4. In more detail, note that parameter ϕ has more impact when using labeling function L^1 and less when using labeling function L^4. This can again be explained by the domain sizes of the labeling functions, but also the fact that the bounds obtained with L^4 are in general already closer to the optimal solution values and it becomes more and more difficult to find better bounds. When comparing the results of L^2 and L^3, we can see that L^2 yields mostly slightly better results, but this again at the cost of longer running times.

The diagrams to the right in Figure 6 shows the results for instance set A using the values $\phi \in \{10000, 20000, 30000, 50000\}$. Note that, since the time horizon in this case never exceeds $T = 1000$, larger values of ϕ were considered than in the experiments for instance set P. This implies that also the MDDs' heights are restricted correspondingly, and larger values for ϕ can be used to utilize roughly comparable computation times. Again, we can see that parameter ϕ allows to control the quality of the obtained relaxed MDDs. Hence, with increasing ϕ, the lengths of the longest paths of the obtained relaxed MDDs in general get smaller, while the computation times and MDD sizes increase.

Structurally similar results are obtained for instances of set P with three secondary resources as well as for instances of set A with four secondary resources, cf. E.

For all further experiments, we went for a compromise between expected quality of the relaxed MDD and compilation time and fixed the following settings. Instance set P: labeling function $L^3(u)$ and $\phi = 1000$; instance set A: labeling function $L^4(u)$ and $\phi = 20000$.

7.2. Upper Bound Comparison

The five types of upper bounds to be compared are the following. The first two are from A*C, namely $Z^{\mathrm{ub}}_{\mathrm{min}}$, which is obtained when the target node is chosen for expansion, and $Z^{\mathrm{lp}}(\mathbf{t})$, which is the longest path length of the completely constructed relaxed MDD. A third one is obtained by solving a MIP model with a commercial solver, while the last two come from MDDs built with traditional TDC and IR algorithms. Remember that $Z^{\mathrm{lp}}(\mathbf{t})$ may be larger than $Z^{\mathrm{ub}}_{\mathrm{min}}$ due to additionally performed merging operations after having found $Z^{\mathrm{ub}}_{\mathrm{min}}$.

The MIP approach to which we compare is the compact order-based formulation from **?**, and we applied Gurobi Optimizer 8.1[3] in single-threaded mode with a CPU time limit of 900 seconds. The TDC and IR methods are those from Maschler and Raidl (2018a). The latter is performed with a CPU time limit of 900 seconds and TDC is executed with two different width limits β which were chosen in a way so that the average running times are roughly in the same order of magnitude and usually not smaller than those of A*C: $\beta \in \{300, 500\}$ for set P and $\beta \in \{3000, 5000\}$ for set A.

Figure 7 documents the results of this comparative study for instance sets P and A. The diagrams at the top show the obtained upper bounds, the middle diagrams the computation times, and the diagrams at the bottom the sizes of obtained relaxed MDDs in terms of the number of nodes. Each

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

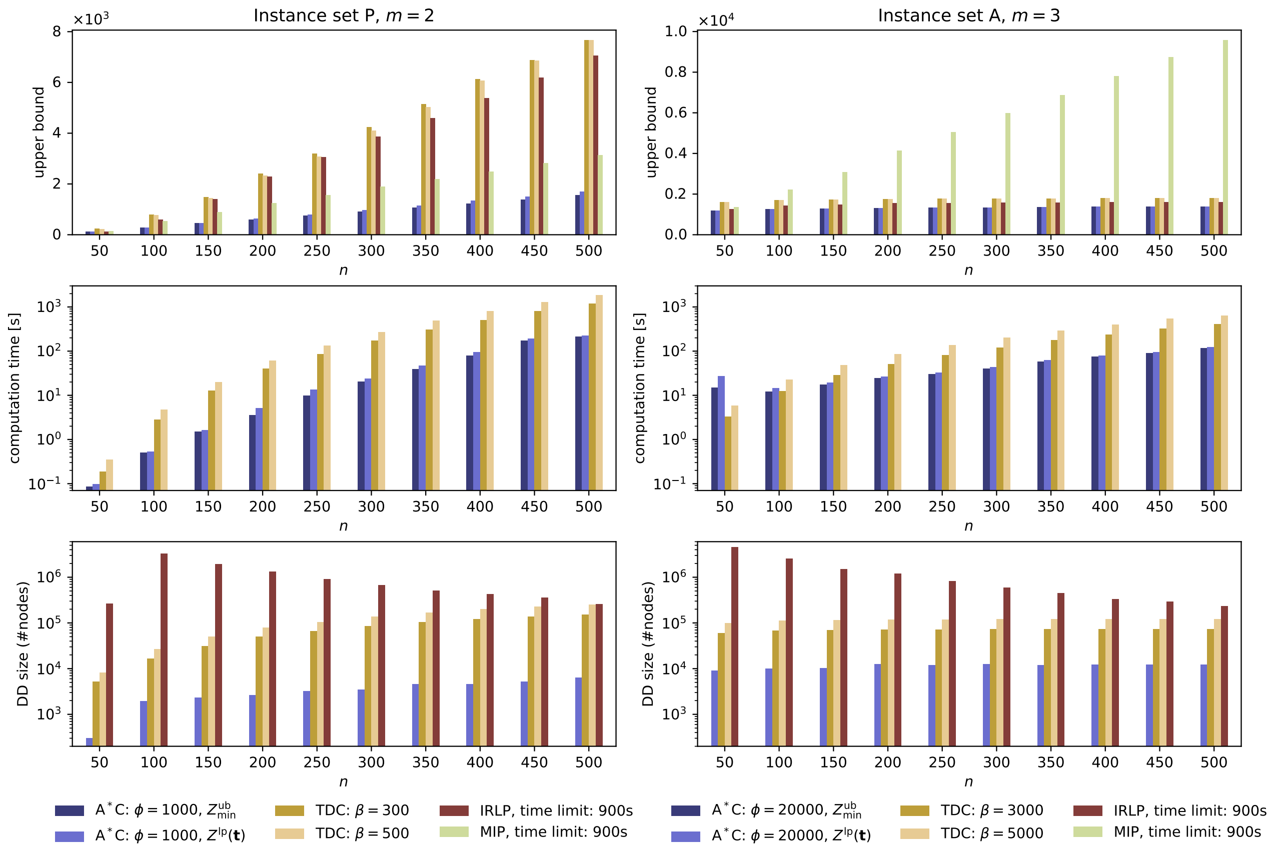


Figure 7: Instance sets P and A with two and three secondary resources, respectively, average values of: upper bounds obtained from A*C, the classical TDC, the IR, and the order-based MIP approach; respective computation times; and the sizes of the obtained relaxed MDDs.

group of bars on the horizontal axes corresponds to a specific instance class with the stated number of jobs, and each bar indicates the average value over all 30 instances of the corresponding instance class and the respective approach.

Concerning the depicted computation times, each first bar shows A*C's average time to obtain $Z^{\mathrm{ub}}_{\min}$, i.e., when the construction would stop according the classical A* termination criterion, while the second bar shows the average time required for the construction of the complete relaxed MDD. Since the MIP approach as well as IR exhausted the time limit of 900 seconds in almost all runs, we omit corresponding bars. More specifically, the MIP solver could only solve the smallest instances to proven optimality. The percentages of the instances with $n = 50$ jobs are 23.3% and 10% of set P for $m = 2$ and $m = 3$, respectively. The IR approach was not able to solve any instance to proven optimality.

The results for instance set P, shown in Figure 7 on the left side, give a rather clear picture. The average upper bounds $Z^{\mathrm{ub}}_{\min}$ obtained by the A*C algorithm are always the strongest. They are in particular substantially better than those obtained from the TDC variants and the IR approach. The difference is more than a factor of four for the largest instances. Even more dramatic are the differences in the sizes of the respectively obtained MDDs. A*C's MDDs are usually more than an order of magnitude smaller than those compiled with TDC and IR. The A*C algorithm clearly can take advantage from avoiding multiple nodes for the same state at different layers, and the merging strategy we proposed appears to be effective. The bounds obtained from the MIP approach are clearly better than those of TDC and IR, but also significantly worse than those of A*C. Differences between $Z^{\mathrm{ub}}_{\min}$ and $Z^{\mathrm{lp}}(\mathbf{t})$ are in comparison to the bounds from the other approaches not that large, but still significant.

For instance set A, Figure 7 shows remarkable differences. The upper bounds obtained from the MIP approach are far worse than those obtained from A*C as well as TDC and IR. Differences between A*C, TDC, and IR are not that large anymore, but nevertheless, in each case the strongest upper bounds could be obtained by A*C. The better relative performance of the classical approaches TDC and IR on these instances in comparison to set P can be explained by the constant time horizon and the special prize structure, due to which the height of the MDDs is limited in a stronger way. Concerning the size of the obtained MDDs, A*C exhibits again substantial advantages over TDC: A*C's MDDs only have about half the size of TDC's MDDs, and those of IR are even more than three times larger than those of A*C for the smaller instances.

Similar results are obtained for instances of instance set P with three secondary resources as well as for instances of instance set A with four secondary resources, cf. E.

7.3. Lower Bound Comparison to Other Approaches

Finally, we consider the A*C approach to construct a relaxed MDD followed by the construction of a restricted MDD and compare to other heuristic methods to approach larger PC-JSOCMSR instances. Now, our focus is primarily on the quality of obtained heuristic solutions, i.e., lower bounds, but since our approach also yields upper bounds from the relaxed MDD, we will also study resulting gaps. We compare to a conventional TDC of a restricted MDD, a general variable neighborhood search (GVNS) metaheuristic, the MIP approach, and a basic CP formulation.

After a relaxed MDD has been constructed by A*C, it is post-processed by filtering in order to reduce its size and strengthen it before deriving the restricted MDD. This is done as follows.

1. A first lower bound (and heuristic solution) is determined in a quick way by compiling a small restricted MDD in an independent way (maximum width $\beta = 100$ for type P instances and $\beta = 15000$ for type A instances).

2. Using the obtained lower bound, cost-based filtering (see, e.g., Cire and van Hoeve (2013)) is applied in order to get rid of many arcs and nodes that cannot be part of a path representing a better solution.

3. For each node u in the relaxed MDD, we have the upper bound for the cost-to-go obtained from the auxiliary upper bound function $Z^{\mathrm{ub}}(u)$, but also the length of the longest u-$\mathbf{t}$ path provides an upper bound, which we denote by $Z^{\mathrm{lp}\uparrow}(u)$. We keep the better of these bounds and check if further arcs and nodes may be removed due to it by cost-based filtering. Note that $Z^{\mathrm{lp}\uparrow}(u)$ can be determined for all $u \in V$ efficiently by a single bottom-up traversal of the MDD.

4. When removing some ingoing arc of a node, we always re-determine the state of the node, and if the state changes, the auxiliary upper bound $Z^{\mathrm{ub}}(u)$. Changes are always propagated to successor nodes as far as they are affected.

After the relaxed MDD has been filtered, it is used to compile the main restricted MDD. Experiments showed that on average 51.57% and 88.90% of all arcs can be removed from the relaxed MDD over all instance sizes for type P and type A instances, respectively. This substantial reduction leads, in particular for type A instances, to shorter computation times when compiling the main restricted MDD.

The conventional TDC of a restricted MDD uses the same greedy criterion from Section 6 to select nodes for removal as our compilation method based on the relaxed MDD. Figure 8 shows for different maximum widths β a comparison between the conventional TDC and the TDC when utilizing a previously compiled and filtered relaxed MDD by A*C. The relaxed MDDs were compiled with different values of ϕ and used labeling function $L^3(u)$ and $L^4(u)$ for instance set P and A, respectively. Although the choice of ϕ has an impact on the quality of the obtained relaxed MDD, as shown in Section 7.1, the plots in Figure 8 indicate that ϕ does not significant influence the finally obtained objective values from a subsequently applied TDC for the considered instance classes. Regarding

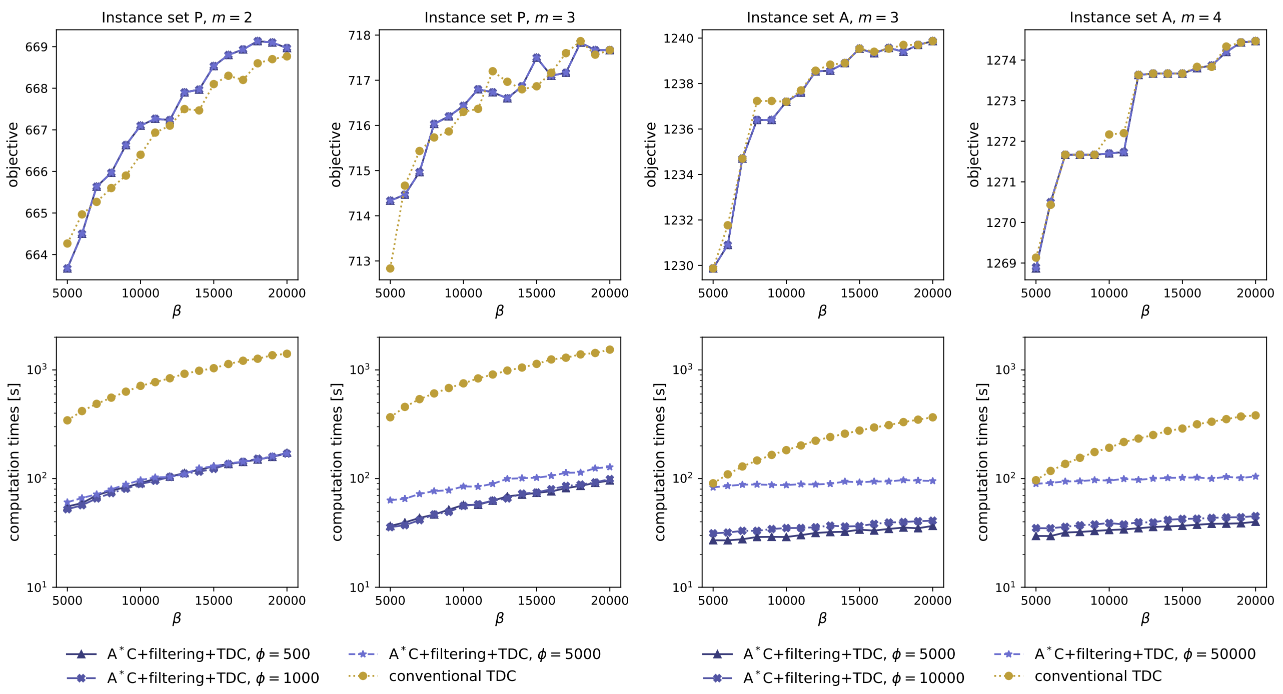


Figure 8: Comparison of A*C+filtering+TDC and conventional TDC for instances of sets P and A with 250 jobs: average objective values and median computation times in dependence of β.

computation times, we can see that larger values of ϕ will result in larger computation times. While the obtained objective values are not substantially different compared to those from the conventional TDC, the diagrams at the bottom row indicate substantial time savings when a relaxed MDD is used to compile a restricted MDD. For example for instances with 250 jobs and two secondary resources of instance set P the conventional TDC needs for $\beta = 20000$ 1407 seconds to terminate whereas the A*C+filtering+TDC approach only needs 170 seconds. This time saving of frequently almost an order of magnitude is the benefit of using the structural information of a previously compiled and filtered relaxed MDD.

Moreover, we also tried to create restricted MDDs by a variant of A*C in which nodes are removed from the open list instead of merging them in Line 29 of Algorithm 1. This, however, only yielded restricted MDDs with substantially worse objective values than the conventional TDC. Increasing parameter ϕ to allow a larger open list size also did not help much in this case but just led to larger computation times.

Note that the computational experiments reported in Figure 8 were done on a somewhat different cluster environment at a later time than all other experiments of this article. After a thorough analysis we concluded that computation times of our algorithms differed by a factor of 2.2 between the different environments, and we have scaled the reported times in Figure 8 accordingly to make them directly comparable.

For the main results in Table 1 we compile restricted MDDs with $\beta = 2000$ and $\beta = 12000$ for benchmark sets P and A, respectively. Moreover, the A*C+filtering+TDC approach compiles restricted MDDs with $\beta = 12000$ and $\beta = 45000$ for benchmark sets P and A, respectively. These values have been selected so that the TDC terminates for the largest instances in about 900 CPU-seconds.

The GVNS is the one from Maschler and Raidl (2018a). It applies a job permutation encoding, starts with a random initial solution, and combines a classic exchange and insertion neighborhood search for intensification. For diversification (shaking), up to four random insertion moves are performed. The GVNS terminates when reaching a time limit of 900 seconds.

Moreover, we compare to the objective values of the best feasible solutions provided by the order-based MIP formulation from **?**, solved again by Gurobi using a single thread with a CPU time limit

			A*C+filtering+TDC						TDC			GVNS		MIP		CP	
set	m	n	$\overline{obj}$	$\sigma(obj)$	%-gap	$t^f[s]$	$t^c[s]$	$t[s]$	$\overline{obj}$	$\sigma(obj)$	$t[s]$	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$	$\overline{obj}$	$\sigma(obj)$
P	2	50	**123.3**	10.3	2.2	<1	<1	<1	**123.3**	10.3	<1	123.2	10.4	122.9	10.7	**123.3**	10.3
P	2	100	**259.9**	11.7	9.4	<1	5	6	259.2	11.7	5	259.0	11.9	238.5	13.3	200.7	20.0
P	2	150	**401.2**	18.9	11.5	<1	17	19	398.8	19.3	20	396.6	17.7	328.9	22.3	261.0	23.5
P	2	200	**530.0**	18.7	12.3	<1	43	50	526.1	19.3	54	527.1	19.0	383.4	29.5	273.6	61.2
P	2	250	**667.2**	21.1	12.6	1	84	100	661.7	20.0	108	660.8	22.4	475.6	29.7	281.0	52.9
P	2	300	**797.4**	16.8	13.1	2	145	170	792.0	16.8	193	790.0	17.5	570.8	32.1	308.1	78.5
P	2	350	**931.3**	25.6	13.7	3	231	282	923.9	25.2	313	923.0	26.7	626.5	53.8	326.7	122.2
P	2	400	**1061.6**	21.3	13.9	5	338	442	1054.4	21.4	481	1055.1	19.4	661.1	55.0	329.6	110.9
P	2	450	**1197.0**	28.7	13.5	7	485	679	1187.5	28.6	704	1180.9	27.4	704.7	48.5	348.6	152.4
P	2	500	**1339.1**	25.4	14.6	8	613	845	1330.3	25.0	949	1324.3	28.0	711.6	199.3	403.4	351.5
P	3	50	**140.3**	10.4	1.3	<1	<1	<1	**140.3**	10.4	<1	**140.3**	10.4	139.3	10.2	140.2	10.5
P	3	100	**289.9**	14.5	5.1	<1	5	6	288.4	14.6	7	288.5	14.3	268.7	16.0	240.0	16.4
P	3	150	**437.4**	15.1	7.1	<1	13	15	433.5	14.8	24	437.0	17.1	362.0	21.3	331.3	19.3
P	3	200	581.8	18.3	8.4	<1	27	33	576.9	18.2	59	**582.9**	16.3	460.8	26.7	367.2	44.4
P	3	250	716.7	13.6	9.5	1	44	63	712.0	14.4	117	**721.9**	16.5	573.4	22.2	380.6	89.3
P	3	300	850.3	16.0	11.2	2	71	107	846.0	15.3	210	**864.0**	19.8	675.1	27.6	419.5	109.8
P	3	350	988.0	26.8	11.7	3	107	171	983.3	27.7	341	**1008.2**	29.3	716.4	61.0	405.7	188.8
P	3	400	1124.6	24.5	12.3	5	152	296	1119.1	23.6	530	**1142.5**	24.1	757.8	61.9	527.5	161.1
P	3	450	1266.3	19.7	11.9	7	198	433	1257.7	20.9	751	**1283.5**	26.1	846.6	59.8	524.5	207.3
P	3	500	1397.8	25.3	12.3	10	269	672	1392.1	23.7	1082	**1418.0**	27.1	900.2	53.6	589.3	238.6
A	3	50	**1130.3**	39.8	5.0	7	20	57	1127.8	38.4	6	**1130.3**	39.8	1114.5	41.9	892.0	45.8
A	3	100	1201.1	36.5	5.6	9	81	110	1196.6	36.4	23	**1201.4**	37.0	1108.2	52.0	712.7	44.1
A	3	150	**1215.5**	26.3	6.0	11	141	169	1208.9	28.7	58	**1215.5**	27.0	936.7	58.0	643.4	41.9
A	3	200	1228.9	21.8	6.7	14	219	261	1220.4	25.6	109	**1229.4**	21.4	842.6	132.5	544.5	149.4
A	3	250	1244.7	28.6	6.6	18	279	331	1238.5	30.8	180	**1245.4**	28.7	703.2	79.3	575.3	48.0
A	3	300	**1243.9**	23.4	7.5	22	379	448	1234.4	22.9	265	1243.7	23.5	675.2	80.8	553.8	41.3
A	3	350	**1256.2**	22.6	7.4	30	440	542	1245.6	24.8	370	1255.5	23.6	683.8	84.4	536.5	50.3
A	3	400	**1269.7**	19.1	8.0	34	529	646	1262.5	19.1	493	1267.2	19.6	714.1	57.8	525.2	42.8
A	3	450	**1268.4**	19.2	8.3	41	609	748	1257.5	24.5	647	1268.2	18.3	730.1	79.1	516.6	48.6
A	3	500	**1271.7**	19.0	8.2	46	676	869	1260.4	22.7	799	1271.2	17.9	680.7	63.8	527.5	24.0
A	4	50	1141.8	35.5	3.4	7	2	45	1138.7	35.0	6	**1142.4**	36.2	1127.6	40.0	882.5	42.0
A	4	100	1218.8	40.6	4.0	8	57	82	1215.7	41.5	25	**1218.9**	40.6	1137.3	62.5	708.9	119.8
A	4	150	**1253.8**	30.6	4.4	10	118	146	1248.5	30.4	60	1253.5	30.9	963.7	77.7	655.6	48.5
A	4	200	1259.5	31.3	4.9	15	180	228	1253.6	32.0	118	**1259.6**	31.7	881.0	145.7	576.5	129.5
A	4	250	**1280.4**	27.0	5.9	20	277	339	1273.6	25.7	191	**1280.4**	28.2	702.8	60.0	541.4	134.8
A	4	300	**1293.8**	25.1	5.9	27	346	441	1282.6	28.0	287	1292.3	24.9	691.6	64.0	565.0	46.0
A	4	350	**1298.9**	23.9	5.8	32	394	504	1289.9	24.5	396	1297.2	22.9	684.6	75.4	548.8	38.1
A	4	400	**1304.4**	42.0	6.0	37	477	617	1298.1	41.2	533	1299.6	42.9	700.5	69.8	541.4	27.9
A	4	450	**1308.4**	25.2	6.9	45	551	752	1298.9	29.2	672	1304.7	26.7	696.4	65.2	546.5	56.6
A	4	500	**1315.6**	28.0	6.9	48	644	842	1304.0	25.8	858	1312.5	29.1	715.4	73.1	526.8	33.1

Table 1: Comparison of the subsequent application of A*C, filtering, and the construction of restricted MDDs to the conventional top-down construction of restricted MDDs, the GVNS, and MIP and CP approaches.

of 900 seconds.

Last but not least, we also consider the CP model from Horn et al. (2018). The model was implemented with MiniZinc 2.1.7[4] and we apply the backbone solver Chuffed with a time limit of 900 seconds. Results with Chuffed consistently dominated those obtained with the alternative backbone solvers Gecode and G12 LazyFD. Note that we further performed tests with the newer MiniZinc version 2.3.2, but obtained results were inconsistent and mostly worse than those from version 2.1.7.

The results of all approaches are presented in Table 1. Each row shows the aggregated results over the 30 benchmark instances with the characteristics given in the first three columns. For all approaches columns $\overline{obj}$ and $\sigma(obj)$ state the mean objective values of obtained heuristic solutions and corresponding standard deviations. For the MDD-based approaches these values correspond to the lengths of the longest paths in the restricted MDDs. Moreover, we list for the MDD-based approaches median total computation times in seconds in the $t[s]$ columns, and for the A*C based approach more specifically in column $t^f[s]$ median times just for filtering the relaxed MDDs including

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

the times for determining the required lower bound and in column $t^{c}[s]$ median times for compiling the final restricted MDDs. For GVNS, MIP, and CP timing information is omitted as they were always terminated with the time limit of 900 seconds. The only exceptions are MIP and CP runs for the smallest instances with 50 jobs, which finished in some cases earlier with proven optimality. In addition, we list for the A*C based approach average optimality gaps, where %-gap $= 100\% \cdot (Z^{\mathrm{ub}}_{\min} - obj)/Z^{\mathrm{ub}}_{\min}$.

If we disregard the results from the benchmark instances of type P with three secondary resources for now, Table 1 gives a clear picture. A*C+filtering+TDC provides in general the best solutions, followed by the GVNS and the TDC. While the TDC performs, on the P instances with two secondary resources, in most cases better than the GVNS, the GVNS is superior to the TDC on the other instances. The weakest solutions have on average been obtained by the MIP and CP approaches, which are only competitive for type P instances with 50 jobs. Especially, for the medium to large instances, the A*C based method typically requires less time than TDC. A*C+filtering+TDC is superior to the conventional TDC in two ways. Not only are we able to construct much larger restricted MDDs, usually yielding better solutions in less time, but since we first also determine the relaxed MDDs, our approach has the additional bonus of providing upper bounds. Average gaps never exceed 15% and are in particular for instance set A usually not larger than 8%.

On benchmark instances of set P with three secondary resources, GVNS typically provides the best solutions when more than 150 jobs are considered. The relative differences between the obtained objective values from GVNS and our A*C-based approach are typically about one to two percent. We believe that in these cases, the GVNS's local search is particularly effective. Clearly, an option would be to finally "polish" the solutions of the MDD-based methods by applying a local search. Another particularity of the results for set P with $m = 3$ are the required times t^{c} for constructing the restricted MDDs. Although the same maximum width is used as for the instances with two secondary resources, these median times are considerably shorter for the case with three secondary resources than for two. This indicates an even better exploitation of the relaxed MDD and underlines the consistent performance improvements of A*C+filtering+TDC over the classical TDC of restricted MDDs.

The optimality gaps increase with the problem size on all instance sets, as one might expect for a compilation of relaxed and restricted MDDs with fixed parameter values. In comparison to instance set A, we obtain smaller optimality gaps on type P instances with few jobs but get larger optimality gaps for the instances with many jobs. This can be explained by the problem size independent time horizon of set A instances, which implies a certain maximal number of jobs that can be scheduled independently of the number of available jobs.

Last but not least, for some instances the optimality gap has been closed, i.e., they could be solved to proven optimality. This was the case for nine of the type P instances with two secondary resources and 50 jobs. For type P instances with three secondary resources we could optimally solve ten instances that consider 50 jobs. Furthermore, for a single benchmark instance with 50 jobs and four secondary resources of type A, the lower and upper bound coincided.

8. Conclusions and Future Work

We considered the PC-JSOCMSR problem, a prize-collecting scheduling problem, where a subset of jobs must be selected from a ground set of jobs and sequenced to form a feasible solution. By a simple extension, MDDs that are traditionally used for sequencing become suitable to represent the search space of the PC-JSOCMSR problem, where the solutions are of variable length.

By applying the principles of A* search, we proposed a new way of compiling relaxed MDDs for large instances of the PC-JSOCMSR that are challenging to solve to proven optimality. The suggested method has the advantage that it does not rely on a layer-to-variable correspondence, and consequently, multiple nodes for the same states at different layers are efficiently avoided. In contrast, traditional layer-oriented TDC and IR approaches would, for the PC-JSOCMSR, typically lead to relaxed MDDs with a substantial amount of redundant isomorphic substructures. Note further that also the merging of nodes is done across layers.

Moreover, our A*-based method utilizes an auxiliary heuristic function to estimate the cost-to-go from each reached node. This function guides the A* search, and thus the relaxed MDD may be constructed in a more meaningful way. As in any A* search, the better this heuristic function estimates the real cost-to-go, the more efficient the approach becomes.

We propose to restrict the number of nodes in the open list instead of restricting the width at each layer. If merging becomes necessary, a node that appears less promising to be part of a finally longest path is selected first and a similar partner node is efficiently determined by the proposed collector node concept. To this end, not yet expanded nodes are labeled in a state-space-relaxation fashion and maintained in a dictionary for efficient lookup. Choosing a proper labeling is important both to obtain a strong relaxation but also to prevent cycles in the construction of the relaxed MDD and to ensure termination of the construction. Our experiments confirmed that substantially smaller and stronger relaxed MDDs could be obtained in the same or shorter times than with traditional TDC and IR methods.

While a relaxed MDD yields an upper bound on the optimal solution value for a maximization problem and encodes much useful information, it does in general not directly yield a promising heuristic solution and lower bound. For obtaining heuristic solutions, restricted MDDs are suitable. In previous works, they have been constructed independently of the relaxed MDD. We showed how the construction of a restricted MDD can be improved by constructing a relaxed MDD first and then exploiting the encoded knowledge. Again, our experiments for the PC-JSOCMSR confirmed the advantages: The main benefit is a substantial speedup in the construction of the restricted MDD. We even showed that the total time for constructing the relaxed MDD, filtering it, and deriving a restricted MDD of a certain size based on the relaxed MDD can take less time than the classical independent construction of a restricted MDD of the same size. Thus, one might say that in our combined approach, one gets the upper bound from the relaxed MDD and thus a quality guarantee in addition to a promising heuristic solution "for free".

We compared this overall approach to an order-based MIP model solved by Gurobi, to a GVNS metaheuristic, and to a basic CP approach solved by MiniZinc. The MIP model only produced rather weak lower and upper bounds for all instances except the smallest. For most cases, our approach yielded the best solutions. An exception are the larger instances of the particle therapy benchmark set with three secondary resources, where the GVNS outperformed the other methods.

Naturally, it is interesting to test the proposed methods in future work also on other problems that include both the selection and sequencing aspects, like those referred to in Section 1. Although not a strong limitation, an important property of a suitable problem may be the order-invariance of the objective function, which is exploited by the proposed approach. Moreover, note that the idea of exploiting relaxed DDs in the construction of a successive restricted DD is more generally applicable. For some problems, the proposed way of finding similar partner nodes for merging may also be useful in the context of a classical layer-wise TDC of relaxed MDDs, where so far a simpler bulk merging is primarily used.

For some applications of relaxed DDs, an important aspect is incrementability, i.e., that a once obtained complete DD can be further refined, for example, to strengthen the obtained bound. Naturally, known iterative refinement methods based on node splitting and filtering can also directly be applied to relaxed DDs obtained from the A*-based construction. Moreover, the A*-based construction may be iteratively applied with increasing open list size limits, yielding stronger and stronger DDs over the time. Hereby, information contained in one DD can always be exploited to speed up the construction of a successive DD in a similar way as we derived a larger restricted DD on the basis of a relaxed DD. An interesting research question is if a completely constructed DD can also be effectively updated in-place by a following A*-based refinement pass.

Last but not least, it is of relevance to investigate the proposed A*-based MDD construction also from a more theoretical side. Unfortunately, a constant open list size limit ϕ does in general not necessarily imply that the obtained relaxed MDD has polynomial size, and therefore also the algorithm's runtime is not necessarily polynomially bounded. Note, however, that similarly no better performance guarantees can be given for classical A* search without considering a more specific problem setting and a concrete heuristic function. In fact, the resulting MDD's actual size strongly depends on the interplay of ϕ, the problem-specific heuristic function for the cost-to-go, the labeling function for the merging, and how the merging is performed.

One extension to guarantee a termination with a complete relaxed MDD when reaching a certain time limit or MDD size is to reduce ϕ to a very small value from this point onward. However, this naive completion may in general degrade the strength of the obtained MDD substantially. Studying more advanced methods, possibly by adaptively adjusting ϕ over the whole run, or developing an entirely different way of deciding when to merge which nodes, is desirable.

Acknowledgments

This project is partially funded by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Foundation (FWF) Project No. W1260-N35. The work of Elina Rönnberg is supported by the Center for Industrial Information Technology (CENIIT), Project-ID 16.05.

A. Strengthening of States

When constructing a MDD, a state can be replaced by a dominating state if it is ensured that the latter still allows for the same feasible solutions. This dominance relation is defined as follows. A state $(P'(u), t'(u))$ *dominates* a state $(P(u), t(u))$, denoted by $(P'(u), t'(u)) \rhd (P(u), t(u))$, when $P'(u) \subseteq P(u)$, $t'_r(u) \geq t_r(u)$ for all $r \in R_0$, and $(P'(u), t'(u)) \neq (P(u), t(u))$. The feasible extensions from $(P'(u), t'(u))$ towards complete solutions can then only be a subset of those from $(P(u), t(u))$.

To possibly strengthen a state $(P(u), t(u))$, let $P'(u) = \{j \in P(u) \mid \mathrm{s}((P(u), t(u)), j) \neq T^{\max}\}$ include only the jobs from $P(u)$ that can actually be scheduled. Then, set the times

$$t'_0(u) = \min_{j \in P'(u)} \left(\mathrm{s}((P(u), t(u)), j) + p_j^{\mathrm{pre}} \right), \tag{11}$$

$$t'_r(u) = \begin{cases} \min_{j \in P'(u) \mid q_j = r} \ \mathrm{s}((P(u), t(u)), j), & \text{if } \{j \in P'(u) \mid q_j = r\} \neq \emptyset, \\ T^{\max}, & \text{else,} \end{cases} \qquad r \in R, \tag{12}$$

such that they correspond to the earliest possible time when the corresponding resource can actually be used considering the jobs in $P'(u)$. Here, $t'_r(u)$ is set to $T^{\max}$ if no job that requires resource r remains in $P'(u)$. This strengthening also ensures that any state for which no feasible extension exists anymore is mapped to the unique target state $\mathbf{t} = (\emptyset, (T^{\max}, \ldots, T^{\max}))$.

B. Calculation of Upper Bound $Z^{\mathrm{ub}}(u)$

We adopt the upper bound calculation for the cost-to-go for a node u from Horn et al. (2018). An upper bound can be calculated by solving the following linear programming (LP) relaxation of a multi-constrained 0–1 knapsack problem.

$$Z^{\mathrm{ub}}_{\mathrm{KP}}(u) = \max \quad \sum_{j \in P(u)} z_j x_j \tag{13}$$

$$\text{s.t} \quad \sum_{j \in P(u)} p_j^0 x_j \leq W_0(P(u), t(u)) \tag{14}$$

$$\sum_{j \in P(u) \cap J_r} p_j x_j \leq W_r(P(u), t(u)) \qquad r \in R \tag{15}$$

$$0 \leq x_j \leq 1 \qquad j \in P(u) \tag{16}$$

Variables x_j indicate if job j is scheduled (=1) or not (=0), $j \in P(u)$. The right-hand-sides of the knapsack constraints are

$$W_0(P,t) = \left| \bigcup_{\substack{j \in P, \\ k=1,\dots,\omega_j | \\ W^{\text{end}}_{jk} - p^{\text{post}}_j \geq t_0 + p^0_j}} \left[\max\left(t_0, W^{\text{start}}_{jk} + p^{\text{pre}}_j\right), W^{\text{end}}_{jk} - p^{\text{post}}_j \right] \right| \tag{17}$$

and

$$W_r(P,t) = \left| \bigcup_{\substack{j \in P \cap J_r, \\ k=1,\dots,\omega_j | \\ W^{\text{end}}_{jk} \geq t_r + p_j}} \left[\max\left(t_r, W^{\text{start}}_{jk}\right), W^{\text{end}}_{jk} \right] \right|, \tag{18}$$

where the union of intervals is defined as $\bigcup_{i=1,\dots,k}[\alpha_i, \beta_i] = \{\gamma \in \mathbb{R} \mid \exists i : \gamma \in [\alpha_i, \beta_i]\}$, and function $|\cdot|$ denotes the sum of the lengths of the resulting disjoint continuous intervals of this union. Thus, $W_0(P,t)$ and $W_r(P,t)$ represent the total amount of still available time for resource 0 and resource r, $r \in R$, respectively, considering the current state and the time windows.

To solve this upper bound calculation problem for each state in the A* search turned out to be computationally too expensive already for small instances (**?**). Instead, simpler upper bounds are determined by solving two types of further relaxations. The first one is obtained by relaxing inequalities (15).

$$Z^{\text{ub}}_0(u) = \max \quad \sum_{j \in P(u)} z_j x_j \tag{19}$$

$$\text{s.t} \quad \sum_{j \in P(u)} p^0_j x_j \leq W_0(P(u), t(u)) \tag{20}$$

$$0 \leq x_j \leq 1 \qquad j \in P(u) \tag{21}$$

The second relaxation is obtained by performing a Lagrangian relaxation of inequality (14), where $\lambda \geq 0$ is the Lagrangian dual multiplier associated with this inequality.

$$h^{\text{ub}}(u,\lambda) = \max \quad \sum_{j \in P(u)} z_j x_j + \lambda \left(W_0(P(u), t(u)) - \sum_{j \in P} p^0_j x_j \right) \tag{22}$$

$$\text{s.t} \quad \sum_{j \in P(u) \cap J_r} p_j x_j \leq W_r(P(u), t(u)), \qquad r \in R \tag{23}$$

$$0 \leq x_j \leq 1 \qquad j \in P(u) \tag{24}$$

Both, $Z^{\text{ub}}_0(u)$ and $h^{\text{ub}}(u,\lambda)$, are computed by solving LP relaxations of simple knapsack problems. In the latter case, this is possible since the problem separates over the resources and for each resource, the resulting problem is an LP relaxation of a knapsack problem. An LP relaxation of a knapsack problem can be efficiently solved by a greedy algorithm that packs items in decreasing prize/time-ratio order; the first item that does not completely fit is packed partially so that the capacity is exploited as far as possible, see Kellerer et al. (2004).

It follows from weak duality (see, e.g., Nemhauser and Wolsey (1988), Prop. 6.1) that $h^{\text{ub}}(u,\lambda)$ yields an upper bound on $Z^{\text{ub}}_{\text{KP}}(u)$ for all $\lambda \geq 0$, but the quality of this upper bound depends on the choice of λ. We have chosen to consider $h^{\text{ub}}(u,\lambda)$ for the values $\lambda = 0$ and $\lambda = z_{\bar{j}}/p^0_{\bar{j}}$, where $\bar{j}$ is the last, and typically partially, packed item in an optimal solution to the problem solved to obtain $Z^{\text{ub}}_0(u)$.

The value $\lambda = z_{\bar{j}}/p^0_{\bar{j}}$ is chosen since it is an optimal LP dual solution associated with inequality (20) and therefore has a chance to be a good estimate of a value for λ that gives a strong upper bound.

By solving the relaxations introduced above, the strongest bound on $Z^{\mathrm{ub}}_{\mathrm{KP}}(u)$ we can obtain, and the one that we use in our A*-based construction of a relaxed MDD, is

$$Z^{\mathrm{ub}}(u) = \min\left\{Z^{\mathrm{ub}}_0(u), h^{\mathrm{ub}}(u,0), h^{\mathrm{ub}}(u, z_{\bar{j}}/p^0_{\bar{j}})\right\}. \tag{25}$$

C. Validity of the Merging Operation

This section details the validity of the merge operator. In line with the common definition of a relaxation, Bergman et al. (2016a) defines a relaxed DD as follows.

Definition 1. *A weighted DD is relaxed for an optimization problem $\mathcal{P}$ if*

(i) the DD represents a superset of the feasible solutions to $\mathcal{P}$ and

(ii) each path that represents a feasible solution to $\mathcal{P}$ has a length that is an upper bound on the objective value of this solution.

Given an exact DD formulation and a merge operator, this operator is considered valid if (repeatedly) applying it to the DD will result in a DD that is relaxed with respect to the original problem. In order to show this, it is sufficient to show that if the merge operator is applied to a DD that complies with (i) and (ii), so will the resulting DD, and the result will follow by induction. (For the initial step of the induction, we assume the operator is applied to an exact DD, which trivially complies with (i) and (ii)).

Proposition 1. *Given a relaxed MDD constructed for the PC-JSOCMSR according to Sections 5.1 and 5.2 that complies with (i) and (ii) in Definition 1. When the merge operator defined in Equation* (10) *is applied to this MDD, then the resulting MDD will also comply with (i) and (ii).*

Proof. A state $\sigma(u) = (P(u), t(u))$ carries the following information. The set $P(u) \subseteq J$ of jobs that can still be feasibly scheduled, and the vector $t(u) = (t_r(u))_{r\in R_0}$ of the earliest times from which each resource r is available for performing a next job. When the merge operator $\sigma(u) \oplus \sigma(v)$ is applied to the two states $\sigma(u) = (P(u), t(u))$ and $\sigma(v) = (P(v), t(v))$, the resulting state is $\left(P(u) \cup P(v), (\min(t_r(u), t_r(v)))_{r\in R_0}\right)$. For the merged state, the set of jobs that can be feasibly scheduled is a superset of both the original sets of jobs, and no feasible solutions are omitted due to the merge. As for the earliest times, since the merged state gets the component-wise earliest time from each of the original states, no feasible solution is lost. Because of this, (i) holds after the merge operation is applied. Note that after the merge operation, paths from the merged state that were not feasible with respect to neither $\sigma(u)$ nor $\sigma(v)$ might become feasible for $\sigma(u) \oplus \sigma(v)$. Condition (ii) follows because the longest path from $\sigma(u) \oplus \sigma(v)$ is selected from a superset of the paths that existed before the merge and that the cost of the arcs are the same. Moreover, note that the DD stays acyclic and therefore feasible thanks to the node selection mechanism. □

D. Benchmark Instances

For the experiments in this work, we adopted the set on "balanced particle therapy instances", here denoted by P, and "avionic instances", denoted by A, from **?** and extended them with larger instances with up to 500 jobs using the same construction scheme. Note that in the previous work also a third set of "skewed particle therapy instances has been considered, in which the usage of the secondary resources is not uniform. However, especially for large instances the differences between the balanced and skewed instances turned out to be less interesting, and we therefore do not consider the skewed instances here.

Both instance sets contain 30 instances for each combination of $n \in \{50, 100, \dots, 500\}$ jobs and $m \in \{2, 3\}$ secondary resources for instance set P and $m \in \{3, 4\}$ secondary resources for instance set A.

Particle therapy instances (P) For these instances, the following values are sampled for each job $j \in J$: (a) the secondary resource q_j from the discrete uniform distribution $\mathcal{U}\{1, m\}$, (b) the pre-processing times p_j^{pre} and the post-processing times p_j^{post} from $\mathcal{U}\{0, 8\}$, (c) the times p_j^0 from $\mathcal{U}\{1, 8\}$, and (d) the prize z_j from $\mathcal{U}\{p_j^0, 2p_j^0\}$ (such that this prize correlates to the usage of the common resource of job j). Time windows are chosen such that, on average, roughly 30 % of the jobs can be scheduled. For this purpose let the time horizon be $T = \lfloor 0.3\, n\, \mathrm{E}(\mathrm{p}^0) \rfloor$, where $\mathrm{E}(\mathrm{p}^0)$ is the expected value of the distribution for p_j^0. In the first step, the number of time windows ω_j of job j is sampled from $\mathcal{U}\{1, 3\}$, i.e., a job can have up to three time windows. Second, for time window k, $k = 1, \dots, \omega_j$, its start time W_{jk}^{start} is sampled from $\mathcal{U}\{0, T - p_j\}$ and its end time from W_{jk}^{end} from $W_{jk}^{\text{start}} + \max\,\{p_j, \mathcal{U}\{\lfloor 0.1\, T/\omega_j \rfloor, \lfloor 0.4\, T/\omega_j \rfloor\}\}$. Overlapping time windows are merged.

Avionic instances (A) For details on the motivation and background see **?**. A fixed time horizon of $T = 1000$ is considered, and there are 20% communication jobs, 40% partition jobs, and 40% regular jobs. The time p_j^0 is for partition jobs and regular jobs sampled from the discrete uniform distribution $\mathcal{U}\{36, 44\}$ and for communication jobs $p_j^0 = 40$. Each partition job is assigned a secondary resource and each secondary resource has the same probability to be selected. For partition jobs, the total processing time p_j is sampled from $\mathcal{U}\{5p_j^0, 8p_j^0\}$ and then, with equal probability, p_j^{pre} or p_j^{post} is set to 0 and the respective other value is set to $p_j - p_j^0$. Since the communication jobs and regular jobs do not use a secondary resource in the real scenario, an artificial secondary resource is introduced and assigned to all of these jobs, and $p_j = p_j^0$. The prize z_j is for five of the partition jobs and ten of the communication jobs set to the high value 70 to give these jobs a higher priority, while for the remaining partition jobs and communication jobs the prize is sampled from $\mathcal{U}\{10, 50\}$. For all regular jobs, the prize is sampled from $\mathcal{U}\{10, 25\}$. For partition jobs and regular jobs, the number of time windows and the length of the time windows are computed as in the particle therapy case, but for the communication jobs the structure is different. The communication jobs can only be scheduled at certain points in time when the communication can be performed; these time points are $0, 80, 160, \dots, 880$. Each time window of a communication job corresponds to one such time point and a job's total set of time windows corresponds to a number of consecutive such time points. The number of time windows for a communication job is obtained by sampling a value from the uniform distribution $\mathcal{U}\{1, 3\}$ and multiplying it by three.

E. Further Results

Figure 9 shows additional results for instances of sets P and A with three and four secondary resources, respectively. As in Section 7.1 the impact of the open list size limit ϕ and different labeling functions are analyzed and conclusions are similar: In general, with increasing ϕ the lengths of the longest paths of the obtained relaxed MDDs from A*C get smaller, while the computation times and the MDD sizes increase. The smallest relaxed MDDs with the weakest bounds could in general be obtained from labeling function L^1 whereas labeling function L^4 typically provides the largest MDDs with the strongest bounds. Regarding the comparison of upper bounds obtained from different approaches, Figure 10 shows in addition to the results presented in Section 7.2 corresponding results for instances of set P and A with three and four secondary resources, respectively. Average values of upper bounds, computation times, and the sizes of relaxed MDDs, obtained from A*C, the classical TDC, the IR, and the order based MIP approach are shown. Again, we observe remarkable differences between results obtained from instances of set P and A. Nevertheless, in each case the strongest average upper bounds could be obtained by A*C thereby creating as well the smallest obtained relaxed MDDs.

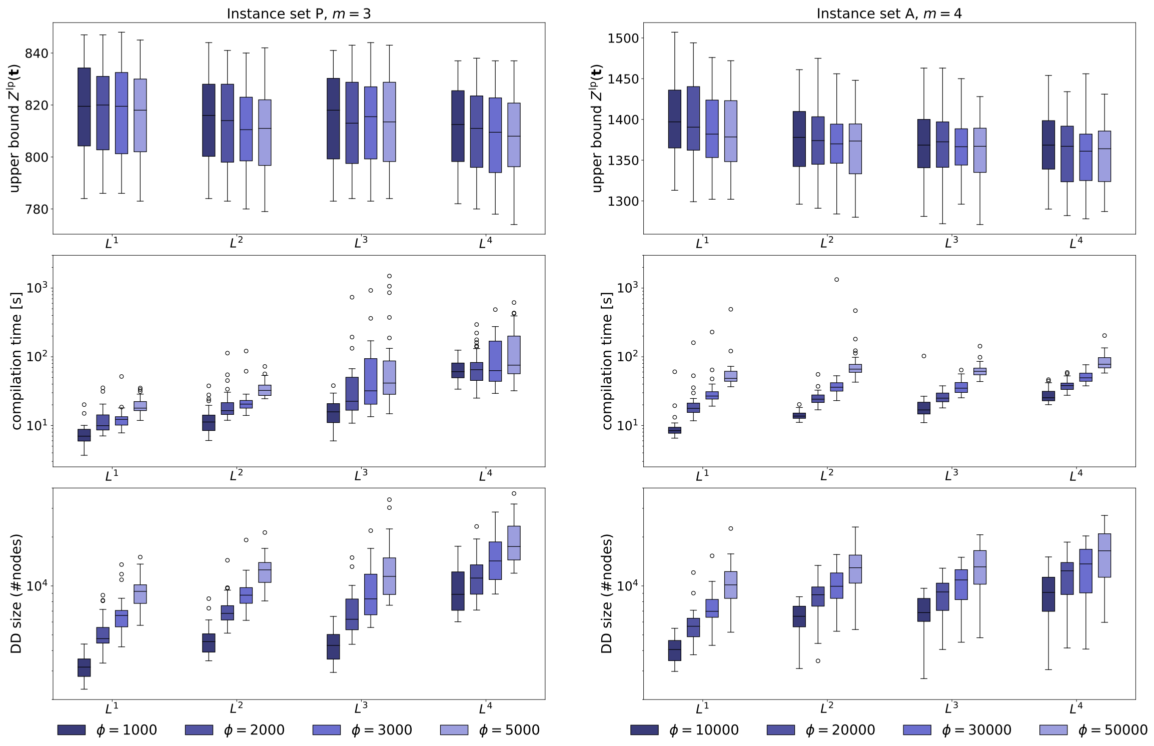


Figure 9: Comparison of open list size limits ϕ and labeling functions $L^i,\ i = 1, \dots, 4$, for instances of sets P and A with 250 jobs and $m = 3$ and $m = 4$ secondary resources, respectively.

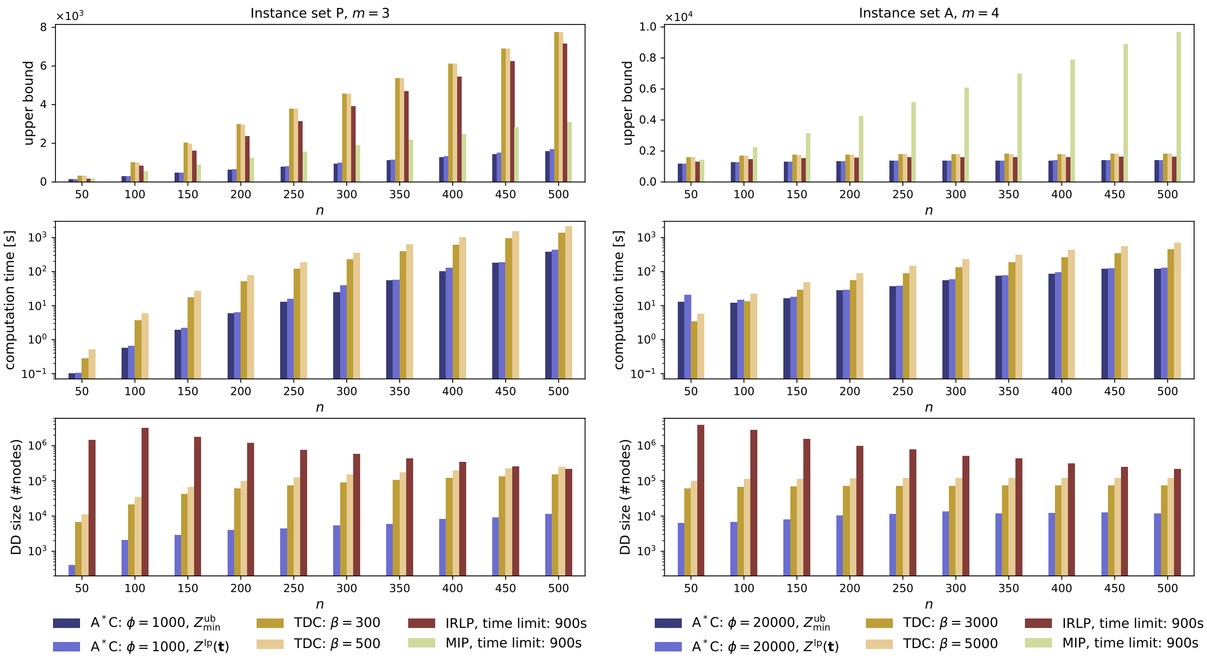


Figure 10: Instance sets P and A with three and four secondary resources, respectively, average values of: upper bounds obtained from A*C, the classical TDC, the IR, and the order-based MIP approach; respective computation times; and the sizes of obtained relaxed MDDs.

References

Andersen, H. R., Hadzic, T., Hooker, J. N., Tiedemann, P., 2007. A constraint store based on multivalued decision diagrams. In: Principles and Practice of Constraint Programming, CP 2007. Vol. 4741 of LNCS. Springer, pp. 118–132.

Bergman, D., Cire, A. A., 2017. On finding the optimal BDD relaxation. In: Integration of Constraint Programming, Artificial Ingelligence and Operations Research, CPAIOR 2017. Vol. 10335 of LNCS. Springer, pp. 41–50.

Bergman, D., Cire, A. A., van Hoeve, W.-J., Hooker, J. N., 2016a. Decision Diagrams for Optimization. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer.

Bergman, D., Cire, A. A., van Hoeve, W.-J., Hooker, J. N., 2016b. Discrete optimization with decision diagrams. INFORMS Journal on Computing 28 (1), 47–66.

Bergman, D., Cire, A. A., van Hoeve, W.-J., Yunes, T., 2014a. BDD-based heuristics for binary optimization. Journal of Heuristics 20 (2), 211–234.

Bergman, D., Cire, A. A., von Hoeve, W.-J., Hooker, J. N., 2014b. Optimization bounds from binary decision diagrams. INFORMS Journal on Computing 26 (2), 253–268.

Blikstad, M., Karlsson, E., Lööw, T., Rönnberg, E., 2018. An optimisation approach for pre-runtime scheduling of tasks and communication in an integrated modular avionic system. Optimization and Engineering 19 (4), 977–1004.

Bryant, R. E., 1986. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35 (8), 677–691.

Christofides, N., Mingozzi, A., Toth, P., 1981. State-space relaxation procedures for the computation of bounds to routing problems. Networks 11, 145–164.

Cire, A. A., van Hoeve, W.-J., 2013. Multivalued decision diagrams for sequencing problems. Operations Research 61 (6), 1411–1428.

Cordone, R., Hosteins, P., Righini, G., 2018. A branch-and-bound algorithm for the prize-collecting single-machine scheduling problem with deadlines and total tardiness minimization. INFORMS Journal on Computing 30 (1), 168–180.

Davis, J. M., Topaloglu, H., Williamson, D. P., 2015. Assortment optimization over time. Operations Research Letters 43 (6), 608–611.

Gunawan, A., Lau, H. C., Vansteenwegen, P., 2016. Orienteering problem: A survey of recent variants, solution approaches and applications. European Journal of Operational Research 255 (2), 315–332.

Hart, P., Nilsson, N., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics 4 (2), 100–107.

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.

Hooker, J. N., 2013. Decision diagrams and dynamic programming. In: Gomes, C., Sellmann, M. (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2013. Vol. 7874 of LNCS. Springer, pp. 94–110.

Hooker, J. N., 2017. Job sequencing bounds from decision diagrams. In: Principles and Practice of Constraint Programming, CP 2017. Vol. 10416 of LNCS. Springer, pp. 565–578.

Horn, M., Raidl, G., Blum, C., 2019. Job sequencing with one common and multiple secondary resources: An A*/Beam Search based anytime algorithm. Artificial Intelligence 277.

Horn, M., Raidl, G., Rönnberg, E., 2020. A* search for prize-collecting job sequencing with one common and multiple secondary resources. Annals of Operations Research.

Horn, M., Raidl, G. R., Rönnberg, E., 2018. 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.

Karlsson, E., Rönnberg, E., Stenberg, A., Uppman, H., 2020. A matheuristic approach to large-scale avionic scheduling. Annals of Operations Research.

Kellerer, H., Pferschy, U., Pisinger, D., 2004. Knapsack Problems. Springer.

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

Kowalczyk, D., Leus, R., 2018. A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching. INFORMS Journal on Computing 30 (4), 768–782.

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

Lee, J. Y., Kim, Y. D., 2012. Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance. Computers and Operations Research 39, 2196–2205.

Maschler, J., Raidl, G. R., 2018a. 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.

Maschler, J., Raidl, G. R., 2018b. Particle therapy patient scheduling with limited starting time variations of daily treatments. International Transactions in Operational Research.

Minato, S., 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. In: 30th ACM/IEEE Design Automation Conference. IEEE, pp. 272–277.

Minato, S., 2011. πdd: A new decision diagram for efficient problem solving in permutation space. In: Theory and Applications of Satisfiability Testing, SAT 2011. Vol. 6695 of LNCS. Springer, pp. 90–104.

Moore, J. M., 1968. An n job, one machine sequencing algorithm for minimizing the number of late jobs. Management Science 15, 102–109.

Nemhauser, G. L., Wolsey, L. A., 1988. Integer and Combinatorial Optimization. Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons.

Ow, P. S., Morton, T. E., 1988. Filtered beam search in scheduling. International Journal of Production Research 26 (1), 35–62.

Ouz, C., Sibel Salman, F., Bilgintürk Yalçin, Z., 2010. Order acceptance and scheduling decisions in make-to-order systems. International Journal of Production Economics 125 (1), 200–211.

Römer, M., Cire, A. A., Rousseau, L.-M., 2018. A local search framework for compiling relaxed decision diagrams. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2018. Vol. 10848 of LNCS. Springer, pp. 512–520.

Silva, Y. L. T., Subramanian, A., Pessoa, A. A., 2018. Exact and heuristic algorithms for order acceptance and scheduling with sequence-dependent setup times. Computers and Operations Research 90, 142–160.

Van der Veen, J. A. A., Wöginger, G. J., 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.