Compilation of Maintenance Schedules based on their Key Performance Indicators for Fast Iterative Interaction

Dara Curran, Roman van der Krogt, James Little, Nic Wilson
Cork Constraint Computation Centre (4C), University College Cork, Ireland

Abstract

Finding the optimal solution for a scheduling problem is hard, both from a computational perspective and because it is frequently difficult to articulate what the user wants. Often there are a range of possible key performance indicators (such as makespan, resource utilisation, and priority of tasks), and thus there can be many objectives that we want to optimise. However, it will typically be hard for the user to state numerical trade-offs between these objectives. Instead, it can be helpful if the user can explore the solution space themselves, to find which compromises between objectives they prefer. This paper demonstrates the use of Multi-valued Decision Diagrams in consideration of scheduling a real maintenance problem, namely the scheduling of Irish Navy dockyard maintenance. We show how candidate schedules can be compiled into MDDs, based on their associated Key Performance Indicators (KPIs). This representation allows the possible values of KPIs to be restricted by the user, and achievable values of other KPIs can be quickly determined, thus enabling fast iterative interaction with the user in order to achieve a satisfactory balance between the KPIs. We experimentally compare the performance of the MDD with that of a database, showing that the MDD can be considerably faster.

1 Introduction

Generating a good schedule is a complex task, for which highly optimised methods and software have been developed, see, e.g., (Brucker 2004; M.L.Pinedo 2008). This is especially true when there are multiple, possibly conflicting, objectives. There are several approaches for dealing with this problem, ranging from a simple linear combination of the objectives (for example, used by (Berrada, Ferland, and Michelon 1996)) to Pareto-based evaluation (such as employed by, e.g., (Johnston and Giuliano 2011)). However, when building a scheduling support tool, the clients' requirements and priorities are often unclear, even to themselves. As a consequence, it takes significant effort to elicit these from the end-user (van der Krogt, Little, and Simonis 2009). In the past, we have found that prompting the user with a schedule can generate feedback regarding what they want of a schedule. This can be used to produce another schedule, and over the course of several iterations, it is possible to learn the user's preferences.

However, the priorities are often subjective and can change between actual instances of a problem. It is therefore desirable to have ways of allowing the user to explore the possible solution space interactively—in particular, by putting bounds on the key performance indicators (KPIs)—before settling on a workable solution to move forward with. In order to facilitate this idea we need to generate sufficient compact compiled schedules in advance and to store them in such a way that makes navigation through them easy.

This paper proposes a system using Multi-valued Decision Diagrams (MDDs) to achieve this. In particular, it supports the Irish Navy in solving a maintenance scheduling problem. Initially, schedules, in the form of their key performance indicators, are compiled into an MDD. The KPIs can include, for example, such measures as makespan, utilisation of resources and duration of certain tasks. From the MDD, the system can efficiently retrieve the possible ranges of the KPIs, and show what is available in terms of possible schedules and their performance indicators. The user can inspect those, and focus on particular classes of solutions by constraining the KPIs, such as by enforcing that the utilisation of resource X should be at least 60%. The MDD can then be used to update the ranges of the indicators by enforcing the constraints proposed by the user. In parallel, each solution in the form of a set of KPIs has an associated actual schedule which the user can inspect at any time. This gives the user an end view to make a further judgement of choosing this plan or continuing searching.

The benefits of such a system are two-fold: *(i)* it takes away the burden of having to fully specify the desired behaviour; and *(ii)* it allows for a more flexible system where the user can easily vary preferences from one instance to the next. Also, since it puts the user in control, they may more easily accept the outcome of the tool, which does not always happen (see, e.g. (Fagerholt 2004), discussing vessel fleet scheduling).

This paper focuses not on the scheduling model (for that, we refer the reader to (Boyle et al. 2011)), but rather on the system around it. The remainder of this paper is therefore organised as follows. Section 2 briefly describes the maintenance scheduling problem; Section 3 discusses how we use

MDDs for representing achievable combinations of KPI values. The system architecture and experimental evaluation is described in Sections 4 and 5, and Section 6 concludes.

2 Context: Irish Navy Maintenance Scheduling

The context of our work is a maintenance scheduling problem on sea-going vessels for the Irish Naval Services. Their maintenance/refit policy across all their ships is based on a 28 day period (or 20 working days) every year. During that time, a team of specialist fitters, riggers, electricians and plumbers are employed at the dockyard to carry out the majority of tasks associated with the maintenance. Ideally, the schedule will have all the tasks completed within the time window without the need for unplanned outsourcing. Regarding the evaluation of a schedule, there are many criteria of interest, expressed as KPIs, representing utilisation of different resources, durations of activities, and so on.

The problem and our solution approach is described in detail in (Boyle et al. 2011); here we present only a short overview to give the user some context to the problem at hand. The constraints present in this problem can be divided into the following categories.

Resource Constraints There are two types of labour/equipment resources identified. The first type is the type one commonly sees in scheduling, which is dedicated to a single task for its entire duration (e.g. a welder replacing a piece of piping). The other type are those which are spread across a number of tasks at the same time in a supporting role such as cranes and foremen. A limit is imposed on how many tasks can be supervised simultaneously. The tasks are constrained generally in their durations, although several can be done in a variable amount of time depending on the number of resources assigned to it.

Space Constraints The restricted space on a ship can mean that it is sometimes difficult for two or more tasks to take place in the same area, or use the same access routes. The Naval Dockyard (NDY) (human) scheduler has already indicated which areas these are and hence which tasks are affected. For the same type of reason, tasks involving gas or welding, even in a large area, may require other tasks to be absent.

Temporal Constraints There are some cases where one task must follow another sequentially for logical reasons. Examples are *Deammunition* before *Magazine Service*, and *Remove Turbo* before *Rebalance Turbo*.

Other Constraints The granularity of time is half a day since this is the minimum the NDY scheduler currently allocates any task to a person. Using a scheduling model, it is easy to change this, and future work could look at the possible merits of adjusting the granularity. Of particular significance to scheduling is the task of engine service which takes the full 20 days to complete and sets a lower bound on completion time.

Objectives The objective is to maximise the number of tasks carried out internally, before any essential work is outsourced within the scheduling window. Beyond that, it is to finish the tasks as early as possible.

3 Background: MDDs

Multi-valued Decision Diagrams (MDDs), which generalise Binary Decision Diagrams (BDDs) (Bryant 1986; 1995) to non-Boolean values, have been studied for example in (Amilhastre, Fargier, and Marquis 2002; Wilson 2005; Andersen, Hadzic, and Pisinger 2010). An MDD is a directed graph with a unique *source* (i.e., initial) node and a unique *sink* (i.e., final) node. Each edge is associated with an assignment to a variable. Paths from Source to Sink correspond to assignments to a set of variables, and so the MDD represents a set of complete assignments via its set of paths. This can be a very compact representation, since the number of complete assignments represented can even be exponential in the number of nodes in the MDD.

We use MDDs as a compact representation of KPI values achievable by a consistent schedule for our problem. We have one variable for each KPI; the possible values of the variable represent small ranges of possible values for the KPI. An illustration of such an MDD is given in Figure 1. Here, the first variable might represent the KPI "Utilisation of Plumbers", with four possible value ranges. The second variable could represent the duration of some particular task. A path in the MDD now corresponds to a feasible assignent to all KPI variables, which corresponds to one or more schedules.

From such an MDD we can efficiently compute a number of things. Firstly, an MDD can efficiently return the number of possible designs remaining at any time, given a number of choices having already been made, simply by counting the remaining paths in the graph. For example in Figure 1, there are six paths left. This means that the user is informed of the size of the remaining search space at any time, helping the user in understanding the impact of their decisions. We can also use this information to guide the user to those variables whose choice makes the biggest impact.

Secondly, MDDs can invoke propagation between categories. When a particular choice/value is made for a schedule KPI, then all other associated edges of that choice in the MDD are removed. All paths going through those edges are therefore also no longer present in the MDD, thus removing some values in others choices. The remaining edges therefore represent the possible values for decisions within this new solution space. For example, suppose the user restricted the value of the first (topmost) variable to be one of the two edges on the left. This leaves only four paths in the MDD, none of which goes through the second value of the last (bottom) variable. Thus, this value can be removed from the set of possible values for this variable.

Thirdly, optimisation of any numerical decision simply becomes one of choosing the lowest (in the case of minimising) value and eliminating all the other edges for that particular KPI before propagating. This will leave at least one single path through the network representing the optimal set of decisions around the best value.

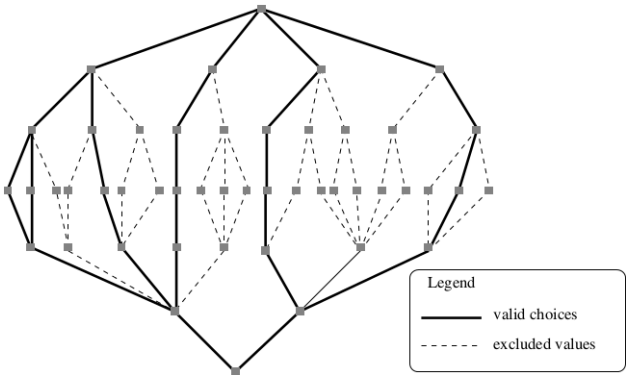


Figure 1: An example MDD

All of the above operations can be performed in time linearly dependent on the size of the MDD; indeed some of the results in Section 5 make this apparent.

4 System Architecture

The system described in this paper is comprised of five interconnected components:

1. a *schedule generator* to generate the initial set of schedules;
2. a *database* to store the schedules found;
3. an *MDD server* that stores the MDD generated for the set of solutions;
4. the *user interface* to allow the user to interact with the MDD server; and
5. a *local search module* to improve upon the schedules retrieved from the DB.

The following subsections describe each component in turn.

Schedule Generator The role of the schedule generator is to generate a large sample of schedules from a list of tasks, resources, resource assignments and schedule constraints provided by the user. In addition, the schedule generator creates an MDD from the values of KPIs derived from the generated schedules. Since our aim was to explore the usefulness of the MDD structure, the generator is very straightforward.

It creates schedules by iteratively altering the duration of each task and finding valid schedules for each task duration combination (if one exists). The schedule generator begins by calculating the minimum and maximum duration that is possible for each task, given the resources available. It then breaks up the minimum/maximum range into a series of segments, resulting in a list of possible durations for each task. For each of these durations, the generator first checks to ensure that a schedule is possible for a given task duration. If it is not, the task duration in question is blacklisted and not employed further. The generator then iterates through all possible combinations of task durations and attempts to generate schedules for each.

For each combination of task durations, the generator may find a large number of solutions. Many of these solutions will represent schedules that are very similar to one another which is not desirable from a user perspective. To limit this effect, each generated schedule is checked against previous schedules to ensure that it is significantly different. For simplicity, we use task start time as a symmetry breaking strategy. To be considered for inclusion, a schedule must have task start times that are different from previous schedules for the same set of task durations. To achieve this, we only include a candidate schedule if its task start times differences from other schedules are above a given threshold.

If the schedule is considered sufficiently different, an XML representation of the schedule along with the values for its KPIs are stored in the database (see the next section). A GANTT chart is also generated using GNUplot for each schedule and stored.

Once all the schedules have been stored, the generator builds an MDD from the KPI values in the following manner. For each KPI, its minimum and maximum values are obtained from the schedules. The minimum/maximum range is then broken into intervals and each interval for each KPI is added to the MDD and the MDD is stored on disk.

Database The database stores information on tasks, resources, the generated schedules and their corresponding KPI value assignments. The database is a MySQL database comprised of 6 tables. The schedules table holds the number of valid schedules (solutions) that have been found for a given task duration combination. These solutions are stored in the solutions table (each set of task durations may produce a number of valid schedule solutions). The KPI Assignment table holds the KPI values for each solution.

MDD Server The MDD server application loads the MDD from disk and listens for requests from the user interface to provide a number of functions:

1. Get the list of KPIs and their current value ranges;
2. Get the list of available schedules;
3. Select a value range for KPIs and update the MDD; and
4. Fetch schedule data from the database.

User Interface The user interface (see Figure 2 for a snapshot of the relevant part) allows the user to interact with the

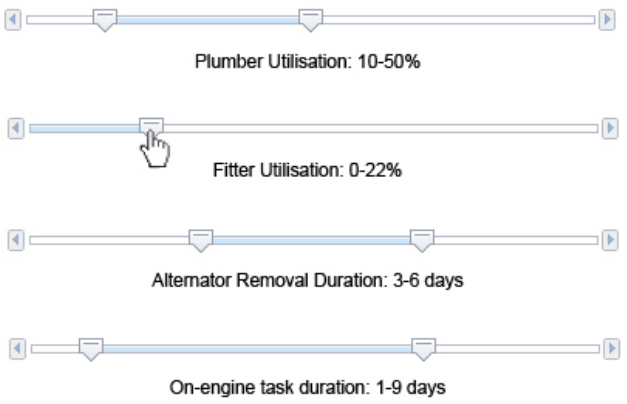


Figure 2: Part of example User Interface

MDD Server by selecting ranges of values for each KPI, by moving the associated min and/or max sliders. The user can alter the desired range of a KPI by moving the sliders. For example, by dragging the left-most slider of the top range to the right, the user can specify that their desired value for the minimum "Plumber Utilisation" is more than the current value of 10%. Each time the user alters the range of a KPI, the MDD Server updates its MDD and returns a list of available KPI values remaining for this and other KPIs. These are then updated in the user interface, by automatically moving the min and max sliders to their new minima and maxima. For example, dragging the minimum plumber utilisation to 20% may result in the Alternator Removal duration changing to 4–6, which is indicated by the relevant slider moving. The user can then go on to restrict other KPIs. Maximising (minimising) a particular KPU can be achieved by moving the *min* slider (*max* slider) to the position of the other slider.

It is important to realise that although each of the values of a particular KPI can be achieved, this does not hold for arbitrary combinations of values for KPIs. When a choice in range alters the available options on another KPI, these are highlighted, so the user can clearly see the impact of their choices. An undo-mechanism allows the user to retract choices again (even out-of-order).

"Local Search" for tweaking schedules Once the user has set KPIs to their satisfaction and obtained a set of schedules from the system, they may find that while some schedules fit their constraints, they are prepared to relax certain KPI values in order to achieve better results elsewhere. For example, though they might like the KPIs of a particular schedule, they would prefer to increase the utilisation of fitters. This is similar to solution critiquing in recommender systems (McGinty and Reilly 2011; Ricci et al. 2011). However, adjusting KPI values manually at this point would become a trial-and-error process and so we devised a simple local search mechanism to perform this type of search automatically. Once the user has selected a schedule of interest, they have the option of selecting a KPI that they would like to either increase or decrease in value. To continue the example, the system might present the user with a list of choices such as including a different set of tasks or decreasing the duration of certain tasks by employing more fitters.

Again, the MDD data structure is helpful in this task. Given a particular new value or range that we want to achieve, the question essentially boils down to finding a new path through a particular edge (or one of a set of edges, as the same range or value may occur multiple times). This can be done as follows. For a particular edge, perform a best-first search both "upwards" and "downwards" to find nodes in the tree that are included in the current solution set. We define a cost function for each of the potential paths, with the cost being the number of edges whose values are not currently in the solution set. In this way, the cost reflects the number of changes we have to make to existing choices. Having found a new path, we can propose it to the user as a way of improving the desired KPI.

The procedure is illustrated in Figure 3. This shows only the upper part of an MDD for clarity. The user wants to include a value or range for the third variable. This occurs in the graph in three places, denoted by the numbers 1–3:

(1) This occurrence can be reached by expanding the restrictions on the second variable to allow the value or range represented by edge a, a cost of one change;

(2) This occurrence requires relaxing the restrictions on both the first and second variable, corresponding to edges b and c, a cost of two changes; and

(3) This occurrence can be included by increasing the possible values of the first variable (edge d). This comes at a cost of one change, as it makes use of edge e, which represents a value for the second variable that is still included in some other valid path.

5 Experimental Evaluation

The justification for building an MDD to store possible user choices about schedules is motivated primarily by the efficiency at which such choices can be made and the model updated. In order to test whether our MDD approach has been successful in this regard, we conducted three experiments to compare the approach with the most obvious alternative of employing a database directly to search for solutions each time a user makes an adjustment to a KPI value.

The experiments are designed to simulate user interactions with the system in the form of repeated KPI min and max value adjustments. We compare our approach with a pure database system which uses SQL queries to retrieve the new minimum and maximum value ranges for each KPI as each choice is made. The experiments use data taken from Irish Navy ship maintenance schedules and consist of 64 tasks taking place over a fixed makespan utilising 47 resources. For our experiment we consider 2 types of KPI: task duration and resource utilisation (i.e., the amount of time that a resource is utilised over the entire makespan). This gives a total of 111 KPIs and the experiments employ 1329 schedules. (In practice, a human scheduler may choose to focus on a smaller set of KPIs.)

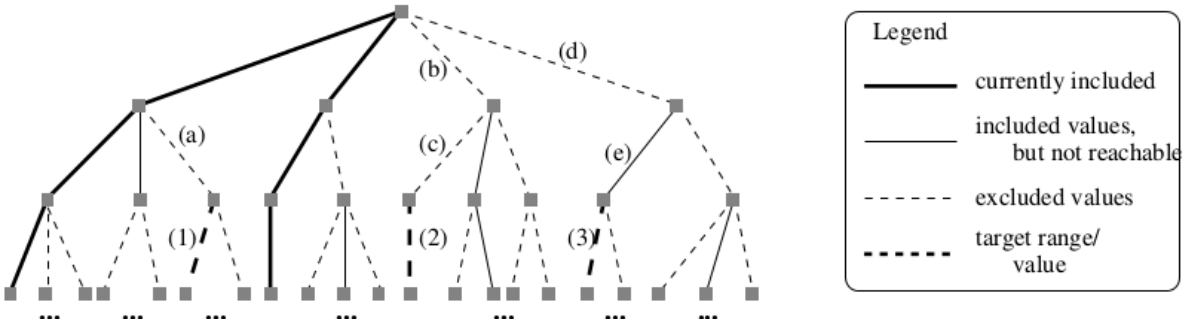


Figure 3: Illustration of the local search procedure. Shown is the upper part of an MDD

Experiment 1 For the first experiment, a number of KPIs (between 1 and 111) are randomly selected for adjustment (with each KPI being chosen only once). Each selected KPI is set to a random range between its currently available minimum and maximum range. The times taken to retrieve the new minimum and maximum KPI ranges for each adjustment were measured for the MDD and database approaches and the experiment was repeated for 19 runs. Figure 4a shows the average time taken for each system to respond to a KPI adjustment. It is clear from this figure that the time taken to make these adjustments is significantly lower for the MDD approach than for the database approach.

Experiment 2 The second experiment initially sets all KPI value ranges to their minimum and maximum. Then, the range of a single KPI is altered, after which each of the remaining 111 KPIs are incrementally altered one by one. In each case, the value of the adjustment is randomly chosen between the current minimum and maximum range of the KPI being altered.

Figure 4b shows the average time taken for each KPI choice to be made for the MDD and database approaches (averaged from 20 independent runs). Initially, the MDD approach requires significantly more time than the database approach due to the initial overhead required when the MDD server loads the MDD. However, once the number of KPI choices reaches 5 the MDD approach begins to significantly outperform the database approach.

The performance difference between the MDD and database approaches is further illustrated by Figure 4c, which shows the time taken for each system to respond as more and more choices are made. As the number of choices increases, the time taken for the database approach to find new minimum and maximum ranges for each KPI increases linearly, while the MDD approach remains almost static.

Experiment 3 The final experiment replicates the setup of Experiments 1 and 2 but for each run, the schedule constraints are randomly perturbed. The goal of the experiment is to examine whether the advantages of the MDD approach are not limited to a single set of schedule constraints.

The results in Figures 4d, 4e and 4f show that the approach maintains its advantage even under different initial constraints.

6 Discussion

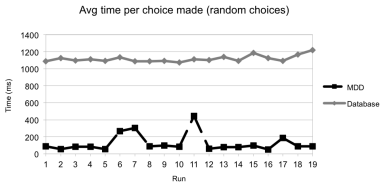
An often occurring problem in real-world scheduling is the fact that there are multiple, conflicting objectives. Often, users find it hard to describe how the system should trade off one objective with another. The context of this work is a navy maintenance scheduling problem in which we ran into exactly this problem.

Configuration and recommender systems, such as described by (Hadzic et al. 2004; Nicholson, Bridge, and Wilson 2006; Andersen, Hadzic, and Pisinger 2010) have used compact compiled representations of sets of solutions. The main purpose of such representations is to allow fast interaction with the user, allowing extra (unary) conditions (including assigning a value to a variable) to be quickly added and retracted, and the consequences made visible to the user. In this paper we explore a similar system for presenting the possible schedules for the navy problem, based on MDDs as the representation, influenced in particular, by the use of solution critiquing (McGinty and Reilly 2011; Ricci et al. 2011). The motivation for this kind of functionality seems even stronger for scheduling problems, because of the computational difficulties of solving scheduling problems, and the exponential number of schedules (even Pareto-optimal ones).

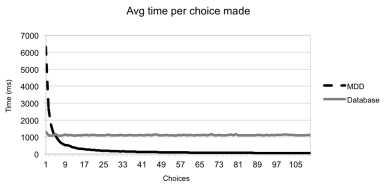
Our results show that the approach can work well on our problem. The MDD approach outperforms a database for the same task, which would be the obvious choice. This holds both for the original problem we faced, but also for random perturbations of that problem.

Future work includes a broader exploration of our methodology. Having shown promising results in one domain, we are keen to explore other types of scheduling problems to see if we can achieve the same results. We are also looking into cleverer ways of finding the initial set of solutions. This is a time-consuming step, so any reduction in the number of schedules we need to compute, while retaining the same or similar coverage would be a big benefit.

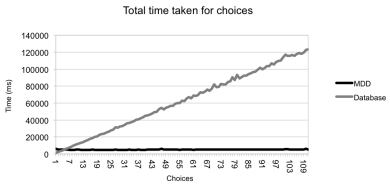
Finally, we are looking into a more advanced User Interface. We want to explore different ways of showing the relationships between KPIs (e.g. indicating which other KPIs are most influenced by making a choice for a particular KPI)



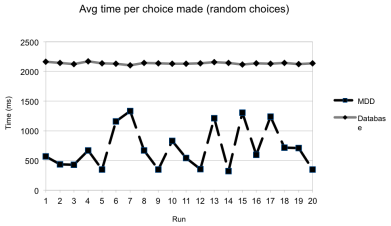
(a) Average time per choice made (random choices)



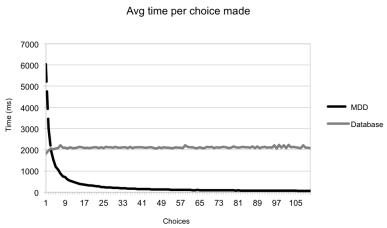
(b) Average time per choice made



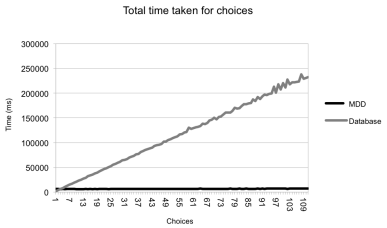
(c) Total time taken for choices



(d) Average time per choice made (random choices)



(e) Average time per choice made



(f) Total time take for choices

Figure 4: Experimental results. Figures (a), (b), and (c) are based on results from the actual problem; (d)–(f) are based on random variations of it

Acknowledgements

This material is partly based on work supported by the Science Foundation Ireland under Grant TIDA I2000.

References

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs—Application to configuration. *Artificial Intelligence* 135:199–234.

Andersen, H. R.; Hadzic, T.; and Pisinger, D. 2010. Interactive cost configuration over decision diagrams. *Journal of Artificial Intelligence Research (JAIR)* 37:99–139.

Berrada, I.; Ferland, J. A.; and Michelon, P. 1996. A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Economic Planning Sciences* 30:183–193.

Boyle, G.; Little, J.; Manning, J.; and van der Krogt, R. 2011. A constraint-based approach to ship maintenance for the irish navy. In *Proceedings of the Irish Transport Research Network Conference (ITRN-11)*.

Brucker, P. 2004. *Scheduling algorithms (4th edition)*. Springer.

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

Bryant, R. E. 1995. Binary Decision Diagrams and beyond: enabling technologies for formal verification. In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, 236–243.

Fagerholt, K. 2004. A computer-based decision support system for vessel fleet scheduling–experience and future research. *Decision Support Systems* 37(1):35–47.

Hadzic, T.; Subbarayan, S.; Jensen, R. M.; Andersen, H. R.; Mueller, J.; and Hulgaard, H. 2004. Fast backtrack-free product configuration using a precompiled solution space representation. In *PETO Conference, DTU-tryk*, 131–138.

Johnston, M. D., and Giuliano, M. E. 2011. Multi-objective scheduling for space science missions. *Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII)* 15(8):1140–1148.

McGinty, L., and Reilly, J. 2011. On the evolution of critiquing recommenders. In Ricci, F.; Rokach, L.; Shapira, B.; and Kantor, P. B., eds., *Recommender Systems Handbook*. Springer. 419–453.

M.L.Pinedo. 2008. *Scheduling: Theory, Algorithms and Systems (3rd edition)*. Springer.

Nicholson, R.; Bridge, D. G.; and Wilson, N. 2006. Decision diagrams: Fast and flexible support for case retrieval and recommendation. In Roth-Berghofer, T.; Göker, M. H.; and Güvenir, H. A., eds., *ECCBR*, volume 4106 of *Lecture Notes in Computer Science*, 136–150. Springer.

Ricci, F.; Rokach, L.; Shapira, B.; and Kantor, P. B., eds. 2011. *Recommender Systems Handbook*. Springer.

van der Krogt, R.; Little, J.; and Simonis, H. 2009. Scheduling in the real world: Lessons learnt. In *Proceedings of the ICAPS'09 Scheduling and Planning Applications workshop*.

Wilson, N. 2005. Decision diagrams for the computation of semiring valuations. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 331– 336.