An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem

Tommy Messelis, Patrick De Causmaecker

CODeS Research Group,
member of ITEC-IBBT-KU Leuven
E. Sabbelaan 53, 8500 Kortrijk, Belgium, Tel: +32 56-24.60.02
e-mail: {tommy.messelis,patrick.decausmaecker}@kuleuven-kulak.be

Abstract

This paper investigates the construction of an automatic algorithm selection tool for the multi-mode resource-constrained project scheduling problem (MRCPSP). The research described relies on the notion of empirical hardness models. These models map problem instance features onto the performance of an algorithm. Using such models, the performance of a set of algorithms can be predicted. Based on these predictions, one can automatically select the algorithm that is expected to perform best given the available computing resources. The idea is to combine different algorithms in a super-algorithm that performs better than any of the components individually. We apply this strategy to the classic problem of project scheduling with multiple execution modes. We show that we can indeed significantly improve on the performance of state-of-the-art algorithms when evaluated on a set of unseen instances. This becomes important when lots of instances have to be solved consecutively. Many state-of-the-art algorithms perform very well on a majority of benchmark instances, while performing worse on a smaller set of instances. The performance of one algorithm can be very different on a set of instances while another algorithm sees no difference in performance at all. Knowing in advance, without using scarce computational resources, which algorithm to run on a certain problem instance, can significantly improve the total overall performance.

Keywords: decision support systems, combinatorial optimization, performance prediction, project scheduling, algorithm portfolio, automatic algorithm selection

1. Introduction

When solving instances of hard combinatorial optimisation problems in practice, we often see that state-of-the-art algorithms show very different behaviour. There is no universal best algorithm for a large class of problem instances. One algorithm is better for some instances, while other algorithms are better for other instances. There is no clear or simple distinction between these groups of instances, nor is there a straightforward explanation known to why one algorithm is better on a set of specific instances (See e.g., Smith-Miles and Lopes, 2011). Consequently, it is hard to compare a set of algorithms based on their performance.

In literature, the average performance over a set of benchmark instances is often used to rank algorithms. A new algorithm is considered good and useful when an improvement on at least one instance over the current state-of-the-art is achieved, or when the average overall performance over the benchmark set is better, or at least as good as the current state-of-the-art (See e.g., the survey of Kolisch and Hartmann, 2006). This new algorithm might do better on certain specific instances, but it is very well possible that it performs worse than the state-of-the-art on (many) other instances. It is therefore not generally the case that the algorithm that scores best on average over the benchmark set, performs best on each and every instance of that set. In practical applications where new instances need to be solved, it is therefore very useful to know which state-of-the-art algorithm will perform best on a given instance. This actually boils down to the algorithm

selection problem, which was already formulated by Rice (1976). The combination of several state-of-the-art algorithms can lead to an improved average performance over each of the components individually.

In this paper, we build such a portfolio consisting of two state-of-the-art algorithms for the multi-mode resource-constrained project scheduling problem. We design an automatic algorithm selection component deciding which algorithm to run on a given instance. This strategy indeed leads to a better overall performance than when each of the algorithms are used separately.

The motivation lies in the successful application of such ideas in other domains. One important example is within the field of propositional satisfiability problems (SAT). Xu et al. (2008) build a portfolio of several state-of-the-art solvers. They basically predict the running time of each of these solvers and then select the solver with minimal predicted running time. The resulting super-algorithm called SATzilla won several medals in subsequent SAT competitions.[1]

Our algorithm selection approach relies on the concept of empirical hardness models. Empirical hardness is the apparent complexity of an instance, when solved by a particular algorithm. This hardness is measured by some performance criterion of the algorithm. The idea is to build a model for predicting the performance of an algorithm on a certain problem instance, using only readily available characteristics of this instance. This has been successfully done for combinatorial auctions, SAT problems and scheduling problems. Smith-Miles (2009) presents a survey on meta-learning for algorithm selection. She reviews literature from different fields like the machine learning community, operational research and artificial intelligence, to name a few.

A crucial precondition for building an accurate empirical hardness model is the existence of a good set of instance features. It is important to capture the internal structure of the instances and to find features relating to the performance of an algorithm. Such a set of features is not in all cases evident, and does not always generalise to other problems. Smith-Miles and Lopes (2012) discuss this problem of measuring instance difficulty for a variety of combinatorial optimisation problems. The authors give a literature review of work relevant for this task, including a review of a wide collection of problem specific features.

Such resulting empirical hardness models allow building automatic algorithm selection tools in several different ways. At first we use separate models to predict the performance of different algorithms. Comparing these predictions leads to a choice of the algorithm to run. A second approach is to build a model that maps instance features onto an algorithm choice, instead of a performance measure. Using such predictions to choose an algorithm is straightforward.

Research into selecting heuristics for (single-mode) project scheduling goes back to the nineties. Kolisch and Drexl (1996) discuss the choice of a scheduling scheme and a heuristic priority rule, based on one characteristic (the resource strength) of the instance. Their study is limited to a subset of the (single-mode) PSPlib instances.[2] Schirmer and Riesenberg (1998) investigate the influence of both the resource strength and resource factor[3] in the choice of scheduling scheme, sampling scheme and priority rule. Schirmer (2000) further extends this approach by integrating a case-based reasoning component, which results in an early approach to an automatic heuristic selection tool for (single-mode) project scheduling. In this paper, we consider algorithms as black boxes and employ a much larger set of instance characteristics. Doing so, we build models able to predict an algorithm's individual performance.

The remainder of this paper is structured as follows. In Section 2, we present a formal definition of the resource-constrained project scheduling problem and its extension to multiple modes. We also review a number of characteristics that relate to the complexity of such problems. Section 3 focusses on building accurate empirical hardness models, and using them in automatic algorithm selection tools. In Section 4, we discuss the related fields of algorithm configuration and hyper-heuristics and compare our approach to the study of algorithm footprints. We conclude and give some directions for further research in Section 5.

[1] See `http://www.satcompetition.org/` for more information on these competitions.

[2] The PSPlib benchmark instances can be found at `http://129.187.106.231/psplib/`.

[3] These characteristics are explained in more detail in the following sections.

2. The multi-mode resource-constrained project scheduling problem

2.1. Problem definition

This section describes the multi-mode resource-constrained project scheduling problem in detail. We start with a definition of the basic resource-constrained project scheduling problem (RCPSP), which is then extended to include multiple execution modes and a new resource type to form the multi-mode resource-constrained project scheduling problem (MRCPSP).

The resource-constrained project scheduling problem is a classic problem formulation for project scheduling. In its basic form, it can be formulated as follows. The RCPSP considers a project with J activities labelled $j = 1, \ldots, J$. Each activity j has a processing time (or duration), denoted as p_j. The order in which the activities of the project should be executed is determined by a set of precedence relations. These relations are given as sets of immediate predecessors P_j, indicating that the execution of activity j can not be started before the execution of each of the predecessors $i \in P_j$ has finished. The precedence relations can be represented by a directed graph. Each activity is represented by a node and there are directed arcs between each activity and its immediate predecessors. It is assumed that this graph is acyclic. There are K renewable resources, labelled $k = 1, \ldots, K$. The capacity of each resource is denoted as R_k, which is the maximum number of units of this resource that can be used concurrently at any given time. Each activity j requires r_{jk} units of resource k for its execution. The fact that resources are renewable implies that their capacity is constant over time. After an activity has finished, the required resource units are released and can be reused by another activity. A schedule is an assignment of a start time S_j to each activity j. For a schedule to be feasible, the precedence constraints, as well as the resource constraints should be satisfied. The objective is to find a feasible schedule with minimal makespan, i.e. a schedule for which the last activity finishes as early as possible. It is common to add two dummy activities $j = 0$ and $j = J + 1$, representing the start and finish of a project respectively. Additional precedence relations state that activity $j = 0$ is the predecessor of all activities that do not have other predecessors, and that activity $j = J + 1$ has as immediate predecessors all activities that are not an immediate predecessor to any other activity. These activities have a duration of 0 time units and require no resource units. A schedule with minimal makespan then corresponds to a schedule with minimal start time S_j for activity $j = J + 1$. Blazewicz et al. (1983) have shown that the RCPSP is a strongly NP-hard problem.

While this definition is already a rich model, several extensions have been formulated for coping with different situations occurring in practice. One commonly used extension is the multi-mode resource-constraint project scheduling problem. The MRCPSP is like a regular RCPSP, with the additional constraint that each activity j should be executed in one of several modes labelled $m = 1, \ldots, M_j$. Each mode m corresponds to a combination of a processing time p_{jm} and a set of resource requirements r_{jmk} for each resource k. When considering multiple modes, a new type of resource is introduced: the non-renewable resource. Non-renewable resources have a specified capacity. When an activity requires a number of units of such a resource during its execution, these units are consumed and hence not available any more for any of the following activities. At any time, the required number of non-renewable resource units should not exceed the available capacity. A schedule for the MRCPSP is an assignment of a start time S_j and a mode selection m_j to each activity j. The objective is to find a schedule with minimal makespan. Other objectives are also possible (see e.g. Hartmann and Briskorn, 2010), but minimal makespan is most commonly used. Note that there may not be a feasible solution in the presence of non-renewable resources. Moreover, as shown by Kolisch and Drexl (1997), the related feasibility problem is NP-complete when at least two non-renewable resources are present. The MRCPSP can be classified as $m, 1T|cpm, disc, mu|C_{max}$ in the classification scheme of Herroelen et al. (1998) or as $MPS|prec|C_{max}$ following the classification of Brucker et al. (1999).

We refer to Hartmann and Briskorn (2010) for a survey on solution methods for variants and extensions of the resource-constrained project scheduling problem. Węglarz et al. (2011) survey single-project, single-objective, deterministic project scheduling problems where activities can be scheduled in a finite or infinite number of modes. Since the publication of these surveys, some more recent approaches to solving the multi-mode resource-constrained project scheduling problem appeared. Wauters et al. (2011) use reinforcement learning agents to solve the problem. Van Peteghem and Vanhoucke (2011) employ a scatter search algorithm and depending on resource scarceness characteristics, one of three improvement methods is selected. Coelho

and Vanhoucke (2011) split the problem into a mode assignment and a single-mode project scheduling step. The mode assignment problem is solved by a SAT solver and the subsequent single-mode project scheduling problem is solved by a state-of-the-art solver from the literature.

2.2. Instance complexity factors for project scheduling

This subsection reviews literature concerning a number of complexity factors for project scheduling. In order to build accurate empirical hardness models, we need to find a set of instance features relating to the apparent hardness of the problem instances (as experienced by the algorithms).

In the last decades of the previous century, investigating such characteristics for problems that are represented by an activity network was a very lively topic. Several factors were proposed and the relationships between these indicator values and the complexity of the instances were empirically determined. In the resulting literature, researchers mainly considered the intrinsic complexity of problem instances, independent of the algorithm that is used. This contrasts to the empirical hardness in which we are interested in this paper. However, it is not clear whether these complexity indicators will also be of importance for the empirical hardness of project scheduling problems. We will first review a number of such indicators, most of these are included in our feature set. Their importance for building empirical hardness models will be investigated in the next section.

Davis (1975) and Patterson (1976) were already studying the effects of problem structure on the performance of algorithms for problems using an activity network representation. Elmaghraby and Herroelen (1980) found a relationship between complexity measures and the algorithm that is used. A network considered complex for one algorithm might be easy for another algorithm. While, at that time, researchers were mostly interested in the algorithm independent complexity of the problems, Elmaghraby and Herroelen (1980) state that it is unlikely that the complexity of a network can be captured by only one measure or indicator. Consequently, a number of complexity indicators have been proposed in the literature. The following list summarizes a number of these indicators. They concern the size of the network, the topological structure and the availability of resources.

- The *coefficient of network complexity* (CNC), introduced by Pascoe (1966) is defined as the ratio of the number of arcs over the number of nodes of an activity network. It was first used for activity-on-the-arc networks and later also for activity-on-the-node networks. In the latter context, Kolisch et al. (1995) observe that a more complex problem instance has more connections in the network, leading to a higher value of the CNC. However, subsequent studies seem to confirm that instances become easier as the CNC increases, because there is less freedom to decide on an order to schedule the activities in. Although the CNC clearly contains some information on the complexity of project scheduling instances, it is not sufficient to discriminate between networks that have an equal number of nodes and arcs, but very different degrees of complexity.

- The *order strength* (OS), defined as the number of precedence relations divided by the theoretical maximum number of such relations $n(n-1)/2$ (with n the number of activities), was first introduced by Mastor (1970). It is also referred to as the *density* by Kao and Queyranne (1982) and, as observed by Elmaghraby and Herroelen (1980), is equal to 1 minus the *flexibility ratio* of Dar-El (1973).

- The *reduction complexity* as defined by Bein et al. (1992) is the minimum number of node reductions to reduce a two-terminal acyclic network to a single edge. De Reyck and Herroelen (1996) adopted the reduction complexity as the definition of the *complexity index* (CI). They conclude that the CI is more informative than the CNC, but it is still not sufficient to accurately discriminate between hard and easy RCPSP instances. This is somewhat intuitive, as these measures do not take into account any information on the requirements and availabilities of the resources. Computing this indicator is not straightforward. We refer to De Reyck and Herroelen (1996) for a detailed description of the CI and an algorithm to compute it.

- The *resource factor* (RF), introduced by Pascoe (1966), reflects the degree to which activities request units of the different resource types. It is defined as the average over the activities of the number of

resource types for which units are requested by the activity over the total number of resource types present in the problem definition. If all activities request units of all resource types, than the RF equals 1.

- The *resource strength* (RS_k) of a (renewable) resource type k was first introduced by Cooper (1976) as the capacity of a resource type k divided by the average number of units requested by the activities. Kolisch et al. (1995) redefined this indicator due to some limitations of the earlier definition. In its original formulation, the resource strength is not normalized to the interval $[0, 1]$. It is also easy to generate two problem instances with equal RS, but with very different complexity (Kolisch et al., 1995). In its later definition (Kolisch et al., 1995; Kolisch, 1996), the resource strength is a normalized value that expresses the relationship between the resource requirements and the resource availability and takes into account some information on the precedence constraints. We refer the reader to Kolisch et al. (1995) for a detailed description of the resource strength in their definition.

- The *resource constrainedness* (RC_k) of a (renewable) resource type k was introduced by Patterson (1976) and is defined as the average demand of k (i.e. the average number of units requested by all activities) divided by the availability (i.e. the capacity of the resource type). It is thus related to the earlier definition of the resource strength (Cooper, 1976) and can be seen as a normalized version of this resource strength.

In their study on phase transitions in project scheduling, Herroelen and De Reyck (1999) conclude that there is not yet a totally unambiguous resource availability measure. There exist arguments for using the resource constrainedness instead of the resource strength. First, the RC is a more 'pure' availability measure as it does not incorporate information on the precedence constraints. Second, there are occasions where the RS can no longer discriminate between easy and hard instances while the RC continues to do so (Patterson, 1976). In our feature set (in Appendix A), we propose a set of features which, when combined, cover a wide set of statistics on the availability of the resources. We consequently choose to exclude the resource strength (as defined by Kolisch et al. (1995)) from our feature set, partly because of the computational effort (i.e. it requires building a critical path schedule), but mainly because we believe that representing the availability of resources as a vector of many values is a more detailed approach than only one aggregate value, and that this will (hopefully) allow for better predictions. Additionally, the complexity index is excluded from the feature set, due to its rather complex computation. All other measures (including the original formulation of the resource strength by Cooper (1976)) are (in some form) included in our feature set. We refer the reader to Appendix A for a detailed description of this feature set.

The above mentioned factors are static, they are independent of a solution method and can be calculated a priori. It is also possible to measure related characteristics of partial solutions. For example, when a number of activities are already scheduled, one can calculate resource scarceness measures that relate to the complexity of the task of scheduling the remaining activities. These indicators can be used to guide an algorithm in completing a partial schedule. Buddhakulsomsiri and Kim (2007) propose a moving resource strength, which helps their priority rule-based heuristic for the MRCPSP where activity splitting is allowed. Van Peteghem and Vanhoucke (2011) use a resource scarceness matrix to decide which improvement step will be taken by their scatter search algorithm. This matrix looks at the scarceness of both renewable and non-renewable resources, given the mode assignments of an individual. The idea is that when resource scarceness is low, the final schedule will be mostly determined by the precedence constraints and hence, the algorithm should focus on this. When resources are scarce however, the resource constraints will become more important and the algorithm should focus on these constraints.

These characteristics can only be calculated for (partial) solutions, and can help a procedure *while* it is solving an instance. It is however not possible to calculate such indicators a priori, which is necessary for an algorithm selection tool as the one we will present in the current paper.

3. Building an automatic algorithm selection tool

In this section, we describe an automatic algorithm selection tool for the MRCPSP. Such a tool can be built using prediction models for the performance of a set of algorithms (empirical hardness models).

We start with a discussion on the usefulness of algorithm selection tools in Section 3.1, and present a framework for evaluating whether or not building an algorithm selection could be interesting. In Section 3.2, we discuss how empirical hardness models can be built, after which in Section 3.3, we explain how these models can be used for building an automatic algorithm selection tool.

3.1. On the usefulness of building an algorithm selection tool

Deciding on which algorithms are to be considered in an algorithm selection tool is mostly done in an ad-hoc manner. In this section, we will present a number of properties that must be satisfied such that the resulting tool can effectively improve over the best single-algorithm approach.

The choice of algorithms is in fact arbitrary; but for an algorithm selection approach to be useful, the algorithms should be competitive on the considered instance set and their difference in performance should be large.

We say that a set of algorithms is competitive, when each algorithm outperforms the other algorithm(s) on a subset of instances; while at the same time, it is outperformed by at least one other algorithm on other instances and these subsets should furthermore be sufficiently large. In case of two algorithms (here denoted as `algA` and `algB`), we could e.g. define the following ratio as a measure for the competitiveness between the algorithms: Let A be the set of instances on which `algA` outperforms `algB`, and B vice versa. Let T be the total set of instances. We can then define the *competitiveness ratio* c as $c = 2\min(|A|/|T|, |B|/|T|)$. The normalization factor 2 was introduced to make sure that $0 \leq c \leq 1$. This ratio denotes how competitive a set of two algorithms is. The lower c, the less competitive the algorithms are with respect to each other. This competitiveness depends on two factors: the *equipotency* of the algorithms and their *reach* in the instance set. The *equipotency* e is defined as $e = 2\min(|A|/|A|+|B|, |B|/|A|+|B|)$. The equipotency is a real number $0 \leq e \leq 1$ and reflects how evenly distributed the algorithms individual dominance over the other algorithm is. If $e = 0$, this indicates that one algorithm is never performing worse than the other; if $e = 1$, this indicates that both algorithms each outperform the other one on an equal number of instances (i.e. that A and B are of equal size).The *reach* r of a set of two algorithms is defined as $r = |A|+|B|/|T|$. The reach represents the portion of the instances on which the algorithms perform differently. It thus gives an indication of the relative size of the subset of instances on which an algorithm selection tool can improve. The reach also equals 1 minus the fraction of instances on which both algorithms perform equally. The product of the equipotency and the reach equals the competitiveness ratio $c = e.r$. A high competitiveness ratio c thus indicates that the two algorithms differ on many instances, and that they are not unevenly dominated by one another. The competitiveness ratio of a set of algorithms thus indicates how accurate an algorithm selection tool must be in order to produce a higher number of best choices than any single-algorithm strategy would. For highly competitive algorithms, the best single-algorithm strategy will be the best choice for a little more than half of the instances. For poorly competitive algorithms however, the best single-algorithm strategy will be the best choice for far more instances, up to nearly all instances. In fact, $1 - c/2$ is the fraction of best choices produced by the best single-algorithm strategy and a natural lower bound for any acceptable selection tool.

Competitiveness is not the only criterion for the usefulness of an algorithm selection tool. The actual difference in performance between both algorithms is also an important factor. Even for highly competitive algorithms, the difference in performance might be small, leading to only marginal performance improvements. Alternatively, consider the case where one algorithm is the best choice for most instances and the actual performance difference is small on these instances, while on the instances where the other algorithm is better, the differences are significantly larger. In such a situation, the best single-algorithm strategy is not the algorithm that is best for most instances. Indeed, the overall absolute performance improvement on the small set of instances could outweigh the absolute performance improvement of the other algorithm on the larger set. When comparing algorithms, it is thus also important to look at the absolute performance improvements, apart from the number of wins.

Before formally introducing a factor reflecting the size of the actual performance differences, we need a little more notation. Let $m_{S,\texttt{algX}}$ denote the sum of the performance achieved by `algX` on the set S of instances. The total performance of the optimal selection tool on the instance set T is thus $m_{T,\texttt{opt}} = m_{A,\texttt{algA}} + m_{(T-A),\texttt{algB}}$. We can then introduce the *potential impact* i as $i = \min(|m_{A,\texttt{algB}} - m_{A,\texttt{algA}}|/m_{T,\texttt{opt}}, |m_{B,\texttt{algA}} - m_{B,\texttt{algB}}|/m_{T,\texttt{opt}})$. The potential impact is a fraction indicating an upper bound for the relative overall relative performance improvement that could be made by using an algorithm selection tool. It in fact represents the actual relative improvement that a perfect algorithm selection tool would achieve over the best single-algorithm strategy. When the potential impact is multiplied with the average performance over the instance set, an indication of the maximal average absolute performance improvement per instance is found. The potential impact thus indicates the relative size of the possible performance improvements. A low potential impact can only lead to small relative performance improvements, while a high potential impact allows an accurate algorithm selection tool to result in large overall performance improvements. The product of the potential impact and the average absolute performance results in an indication of the maximal absolute performance improvement per instance.

Whereas the competitiveness ratio indicated how accurate an algorithm selection tool must be in order to produce a higher number of best choices, the potential impact now indicates how large the possible performance improvement of such a tool can be. The choice of algorithms to be considered in an automatic algorithm selection tool is thus dependent on both the competitiveness and the potential impact. Highly competitive algorithms can already benefit from moderately accurate selection tools; while for poorly competitive algorithms, these tools should be highly accurate. Furthermore, the potential impact indicates the boundaries of the overall performance gains of such a tool. The potential impact can also be used to get an indication of the absolute performance improvements.

The definitions of these indicators are restricted to competitiveness and potential impact in the case of two algorithms. In the case of more algorithms, these concepts require generalization. Pairwise comparison, leading to a set of such indicators, may provide a ranking, but is not informative on the individual contributions as part of the whole. Another option is to compute indicators crosswise, denoting how competitive each algorithm is with respect to the combination of the other algorithms.

3.2. Empirical hardness models

In this section, we review a systematic approach towards hardness analysis. The key concept is the notion of empirical hardness. What is seen in the literature is that many different algorithms exist for a variety of problems in computer science. While most state-of-the-art algorithms have an overall good performance, there is no single algorithm dominating all others. Instead, many algorithms perform well on a majority of benchmark instances, while performing worse on a smaller set of instances. One instance may appear hard to solve for one algorithm, while another algorithm is able to solve the instance quickly. It is difficult to put a complexity label on such instances (i.e. to label them hard or easy). In this case, it makes more sense to use the notion of empirical hardness. Empirical hardness denotes the complexity or hardness of an instance, when solved with a particular algorithm or configuration. This complexity is measured in terms of a performance criterion. The running time is a commonly used criterion for exhaustive search solvers.

Leyton-Brown et al. (2002) present a general framework for building empirical hardness models and demonstrate its use on the winner determination problem within the field of combinatorial auctions. The idea is to predict the empirical hardness of an instance (i.e. the running time of a certain algorithm) based on a number of efficiently computable features of the instance. An empirical hardness model is thus a mapping from the feature space of problem instances onto an algorithm's performance criterion. There is no straightforward or automatic way of finding a set of useful features for a given problem domain. Instead, one usually builds on expert knowledge for constructing a set of properties that intuitively may be expected to have some influence on the complexity of the problem instances. This framework for building empirical hardness models that predict running time can be summarized as the following six-step procedure:

Step 1: Identification of a problem instance distribution
Step 2: Selection of one or more algorithms
Step 3: Selection of a set of inexpensive, distribution independent features

Step 4: Sampling of the instance distribution for the generation of a training set; for each instance, all the feature values and the running time(s) of the algorithm(s) are determined

Step 5: Elimination of redundant and uninformative features

Step 6: Building of a model for each algorithm that maps the feature space onto the running time

Nudelman et al. (2004) apply this strategy to the more abstract problem of propositional satisfiability problems. They are able to find accurate prediction models for two classes of random 3-SAT problems.

We will apply the strategy of Leyton-Brown et al. (2002) to the MRCPSP in order to build prediction models for the performance of a set of state-of-the-art algorithms. It will be necessary to adapt or slightly modify the steps to make the strategy fit our needs. The most important adaptation is caused by the application of the strategy to meta-heuristics for an optimisation problem. This implies that other criteria than running time will become important. Many heuristic algorithms use time as a stopping criterion. Performance can in that case be measured in terms of the quality of the produced solution. This and other adaptations to the strategy are mentioned in the following subsections, each of which describes the application of one step of the above framework.

Step 1: Identification of a problem instance distribution

The first step is crucial for the strategy as it sets the scope for the research. It is important to clearly indicate the boundaries of the problem instance distribution, as the resulting empirical hardness models are only expected to be valid within this scope.

A commonly used set of benchmark instances for project scheduling is the publicly available library PSPlib (Kolisch and Sprecher, 1996). The library originally contained benchmark sets for the RCPSP and MRCPSP and has thereafter been extended with benchmark sets for other extensions of the problem. Throughout the years, researchers have been using these benchmarks to evaluate their algorithms. Unfortunately, due to advances in both algorithmic design and computer infrastructure, the instances can now be considered rather small. Current state-of-the-art algorithms produce optimal or near-optimal solutions for most benchmark instances. Even in the worst case, the solutions are never far from optimal (i.e. the optimality gap is below one or two percent). In terms of performance, there is little room for improvement left for current state-of-the-art algorithms.

Van Peteghem (2010) observes that many algorithms solve the PSPlib benchmark instances to (near-) optimality. Moreover, the analysis of the instances in terms of order strength and resource strength shows that there is not much variation in the set. In order to overcome these shortcommings, Van Peteghem (2010) proposes a new benchmark library called MMlib.[4] This benchmark set contains multi-mode instances that are larger and more diverse, in terms of order strength, resource factor and resource strength. The set consists of three subsets. The first two subsets, MMlib50 and MMlib100, each contain 540 instances. These instances represent projects consisting of 50 and 100 activities respectively. There are always three possible execution modes. The order strength, resource strength and resource factor are varied within these sets. The last subset, MMlib+, is larger and contains 3240 instances with projects consisting of 50 or 100 activities (uniformly distributed). Activities have either three, six or nine execution modes with varying order strength and resource strength. For these instances, the resource factor always equals 1.

Current state-of-the-art algorithms still perform well on average on these instances, as is shown by Van Peteghem (2010). However, as will be demonstrated later, when looking at the per-instance performance, there are much larger differences between the algorithms compared to the PSPlib benchmark set. The algorithms we consider in this study are highly competitive on these instances, and the potential impact is sufficiently large. This makes the MMlib benchmark set particularly interesting for the construction of an automatic algorithm selection approach. We will thus focus on the MMlib benchmark set in this study and will only mention the PSPlib benchmark instances when interesting results are found.

[4]The MMlib benchmark instances can be found at `http://www.projectmanagement.ugent.be/mrcpsp.html`.

Step 2: Selection of one or more algorithms and performance criteria

In this study, we will consider two state-of-the-art meta-heuristic algorithms. Whereas Leyton-Brown et al. (2002) used only running times when building empirical hardness models, we extend this second step and include the selection of the performance criterion. When comparing meta-heuristic algorithms, in contrast to exhaustive search methods, it is not possible to look at running times. Many meta-heuristics use running time as a stopping criterion. The performance of such meta-heuristics is measured on the quality of the solutions they produce. This quality may be considered in relation to some known upper or lower bound on the quality of the optimal (or best known) solution. For such a comparison between two different algorithms to be fair, one must ensure that both algorithms can use more or less the same computational effort. In many cases this effort is measured as the time an algorithm is allowed to run. To avoid the influence of personal programming skills, programming language or infrastructure, and following Kolisch and Hartmann (2006), we will use the number of generated schedules as a stopping criterion. As in the work of Hartmann and Kolisch (1998) and Kolisch and Hartmann (2006), among others, the stopping criterion in this study is set at 5000 generated schedules. Additionally, we will perform experiments with a stopping criterion of 25000 schedules as well. The performance of an algorithm is defined as the quality of the best obtained solution after meeting the stopping criterion. The algorithms we consider in this study aim at minimizing the makespan of the resulting schedule, hence performance is in our case measured in terms of this makespan. The shorter the makespan, the better an algorithm performs. We will look at the average makespan over five independent runs of the algorithms. Such averages are only representative for the real performance of an algorithm when the variation over the runs is relatively low. For the algorithms and stopping criteria considered in this study, this is indeed the case.

The first algorithm, `Algorithm A`, is the tabu-search algorithm of Nonobe and Ibaraki (2002). This algorithm was not specially designed for the MRCPSP and can handle more general resource-constrained project scheduling problems with other extensions as well. Solutions are represented by the commonly used activity list representation. Each solution is represented by an activity list and a mode list. The activity list defines an order in which the activities are to be scheduled. The mode list specifies in which mode each activity is to be scheduled. The algorithm uses a method very similar to the standard serial scheduling scheme to build a schedule from an activity list. The search space is defined by several moves that generate neighbours of the current solution. The tabu tenure is adaptively controlled throughout the search phase.

The second algorithm, `Algorithm B`, is a hybrid genetic algorithm implemented by ourselves, based on the algorithm described by Lova et al. (2009). This genetic algorithm uses the same activity list representation but adds two additional bits to the gene string: one bit indicates whether a serial or parallel scheduling scheme will be used for transforming the activity list into a schedule, the other bit indicates whether this scheme will be applied in a forward or backward manner. Doing so, the genetic algorithm incorporates some form of learning while solving an instance. This concept was first introduced by Kolisch and Drexl (1996) and was also used by Hartmann (2002). Based on intermediate results, the algorithm automatically favours the serial or parallel scheduling scheme for certain types of activity lists. The genetic algorithm is further enhanced by adding an efficient mode improving method, making it a hybrid approach. This method takes the generated schedule of each individual and tries to shorten the makespan by changing modes of certain activities. This step, as shown by Lova et al. (2009), is very effective in reducing the makespan of the resulting schedules.

Since the algorithms are very different in nature (a tabu search algorithm versus a (hybrid) genetic algorithm), their performance is expected to differ over the benchmark sets. Different algorithms are however not the only good candidates. Even very similar algorithms that differ only in details or parameter configurations could be considered. These small differences could still lead to significant differences in performance (i.e. potential impact) and to competitiveness among the algorithms. As we will show, these algorithms are competitive on the MMLIB benchmark set and their potential impact is relatively large. This indicates that an algorithm selection approach could possibly lead to substantial improvements. The choice for these algorithms is based on the availability of executable code and supported by their competitiveness and potential impact.

Both algorithms are shown to be state-of-the-art on the PSPLIB benchmark set. Table 1 summarizes

		instance set of PSPLIB							
		j10	j12	j14	j16	j18	j20	j30	all
`Algorithm A`	avg. rel. deviation from best (%)	1.19	1.17	1.31	1.48	1.78	2.55	3.40	1.84
	% best solutions found	87.0	84.6	78.8	74.9	64.1	62.2	48.0	71.3
`Algorithm B`	avg. rel. deviation from best (%)	0.07	0.14	0.45	0.76	1.01	1.50	2.97	0.99
	% best solutions found	99.5	96.7	90.6	84.3	74.1	70.8	57.8	81.9

Table 1: Comparison of both algorithms on PSPLIB benchmark instances, given a stopping criterion of 5000 schedules.

		instance set of MMLIB			
		MMLIB50	MMLIB100	MMLIB+	all
`Algorithm A` 5000 schedules	avg. rel. deviation from best (%)	0.87	1.19	2.38	2.09
	avg. abs. deviation from best	0.28	0.64	3.69	2.99
	% best solutions found	65.6	70.9	58.0	60.2
`Algorithm B` 5000 schedules	avg. rel. deviation from best (%)	2.75	3.92	3.16	3.20
	avg. abs. deviation from best	1.08	1.68	2.70	2.41
	% best solutions found	55.1	46.2	44.8	46.1
`Algorithm A` 25000 schedules	avg. rel. deviation from best (%)	0.63	0.21	0.32	0.34
	avg. abs. deviation from best	0.17	0.08	0.30	0.26
	% best solutions found	75.3	89.8	90.8	89.0
`Algorithm B` 25000 schedules	avg. rel. deviation from best (%)	3.39	2.77	9.46	8.41
	avg. abs. deviation from best	1.37	5.87	10.20	8.43
	% best solutions found	48.9	33.3	11.1	17.6

Table 2: Comparison of both algorithms on MMLIB benchmark instances.

their performance on this benchmark set. It shows both the average relative deviation from the optimal (or best known) solution and the percentage of optimal solutions found. Both algorithms were stopped at 5000 generated schedules, allowing for a fair comparison. Figure 1 shows a plot of the average difference between `Algorithm A` and `Algorithm B` on the PSPLIB benchmark set, given a stopping criterion of 5000 schedules. Figure 1(a) shows the absolute difference in makespan, Figure 1(b) the relative difference. Both graphs are sorted on this difference.[5] The competitiveness ratio of the considered algorithms on this benchmark set is limited ($c = 0.2194$) and the potential impact is relatively small ($i = 0.41\%$). Given the average makespan in this set (25.325 time units), the maximal average performance improvement of an optimal algorithm selection tool on a problem instance in this set will only be 0.1039 time units, which is very low. Other, more recent approaches such as the genetic algorithm presented by Van Peteghem and Vanhoucke (2010), perform just a little bit better on the PSPLIB benchmark instances. We can conclude that this benchmark set is not well-suited for an algorithm selection approach. Most state-of-the-art algorithms perform very well. There is only limited competitiveness between algorithms but more importantly, the actual performance differences are small. Combined with a short average makespan, this leads to only limited absolute potential improvements.

When both algorithms are run on the MMLIB benchmark set, we get a much more interesting setting. Table 2 summarizes the performance of both algorithms on the MMLIB benchmark set. The average performance over five independent runs is shown for both algorithms, given two different stopping criteria: 5000 and 25000 schedules. The first line for each algorithm displays the average relative deviation from the best solution found by any of these two algorithms. The second line shows the average absolute deviation from the best solution. The last line shows the percentage of instances on which the corresponding algorithm performed better than or equal to the other algorithm. We see that the algorithms perform differently on

[5] In Figure 1(a), this difference is simply the difference in makespan; in Figure 1(b), the difference is measured as the relative deviation from the best found solution which is positive (negative) in case `Algorithm A` (`Algorithm B`) has a longer makespan.

the different subsets.

When given a limit of 5000 schedules, `Algorithm A` turns out to be the overall better choice on the MMLIB50 and the MMLIB100 subsets. It is better in terms of the the number of best solutions found and it achieves a lower average relative and absolute deviation from the best found solution than `Algorihtm B`. However, on the MMLIB+ subset (and also on the combination of all benchmark sets), `Algorithm B` achieves the lowest absolute deviation from the best solution. While `Algorithm A` is the best choice for more instances, the absolute difference in performance seems to be larger for those instances where `Algorithm B` is the best choice. Figure 2 confirms this statement. It is similar to Figure 1 but shows the difference between both algorithms on the MMLIB benchmark set, given the stopping criterion of 5000 schedules. Both algorithms are very competitive on these instances ($c = 0.7957$). This is a result of a high equipotency ($e = 0.8492$) and a high reach ($r = 0.9370$). The differences in performance are furthermore large. This leads to a high potential impact $i = 2.52\%$, indicating that relatively large performance improvements could be achieved by an accurate algorithm selection approach based on these algorithms. Indeed, given the average best makespan in the instance set of 96.003 time units, this results in a maximal average possible improvement of 2.4166 time units per instance.

When both algorithms are allowed to construct 25000 schedules, we get a different view. Figure 3 shows a similar plot as Figure 2, but for a stopping criterion of 25000 schedules. `Algorithm A` has become more dominant over the other one, being the best choice on 89% of the instances. The equipotency has dropped to $e = 0.2354$ resulting in a much lower competitiveness ratio $c = 0.2198$ than for the 5000 schedules case. It seems that `Algorithm A` benefits more from longer calculation times than `Algorithm B`. The differences in performance between both algorithms are still large, but the potential impact is relatively small: $i = 0.29\%$. Given the average makespan in this set (90.656 time units), the maximal average performance improvement is 0.6212 time units per instance. In this setting, it will probably be much harder for an algorithm selection approach to improve over the best single-algorithm strategy.

Step 3: Selection of a feature set

For the empirical hardness models to be accurate, we need to capture those properties of the instances that influence the empirical hardness. We have constructed a feature set for general MRCPSP instances. This set contains 686 instance features that can be roughly categorised into four groups:

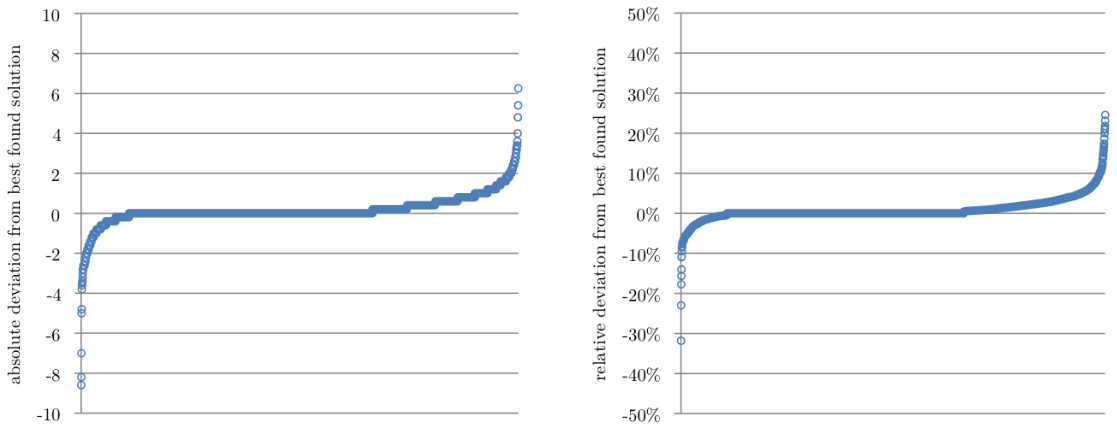
- size related features (35 features)
- resource constraint related features (486 features)
- precedence constraint related features (21 features)
- activity duration related features (144 features)

These features are easily computable properties of the problem instances. The MRCPSP has a rather compact formulation. Most features can be calculated by simply iterating over the constraints and counting certain values. A detailed description of the feature set can be found in Appendix A. The set includes a number of (variations of) the complexity measures introduced in Section 2.2.

Step 4: Data generation

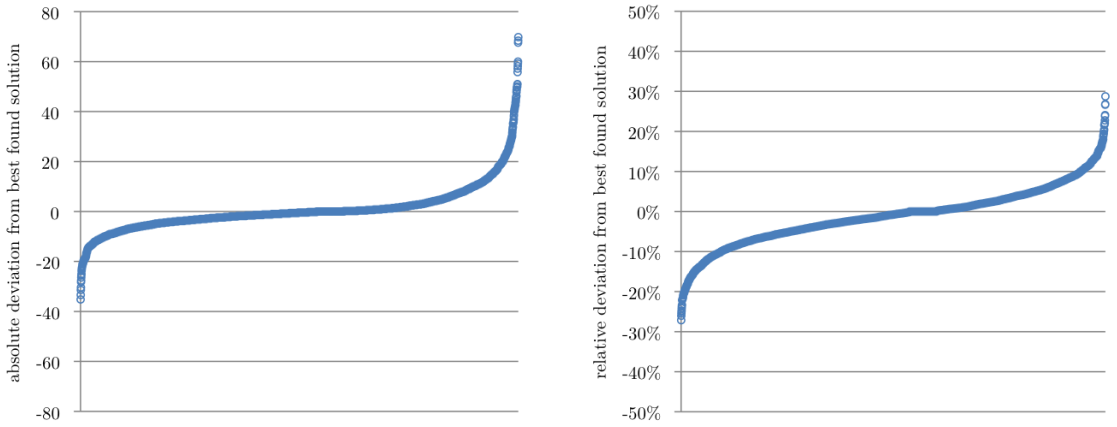
The instance distribution is sampled to construct a training set of instances. All algorithms are run to determine their performance, and all feature values are calculated.

We use the complete MMLIB benchmark set consisting of 4320 instances. On some of these instances, one or both algorithms were unable to find a feasible solution. These instances are filtered out. A set of 4140 instances remains, from which we have randomly selected 3140 instances as a training set. The remaining 1000 instances serve as a validation set for the empirical hardness models and the automatic algorithm selection tools. We generate two separate datasets based on running the algorithms with a different stopping criterion (5000 schedules and 25000 schedules). On all instances, we ran both algorithms with both stopping criteria five times each and recorded the average performance. We have parallellized this task and employed



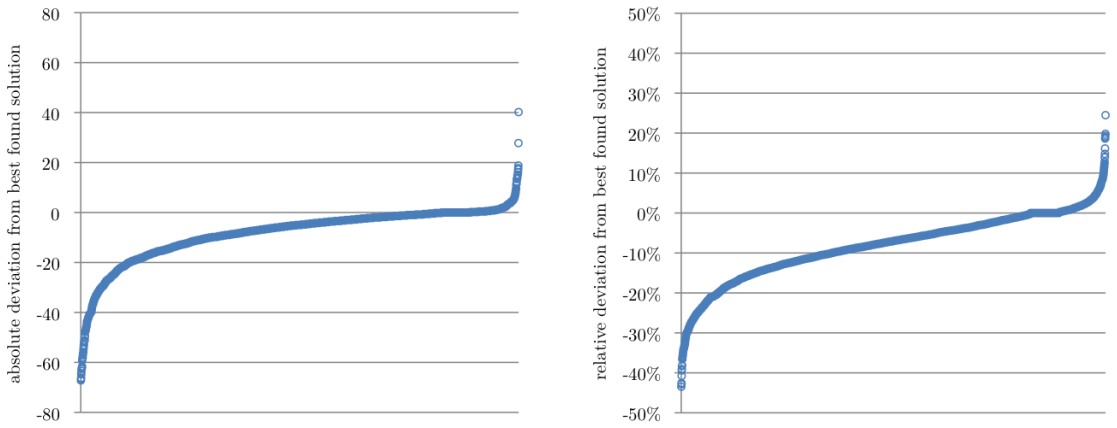
(a) absolute difference (b) relative difference

Figure 1: Difference in performance between `Algorithm A` and `Algorithm B` on the PSPLib benchmark set, given 5000 schedules.



(a) absolute difference (b) relative difference

Figure 2: Difference in performance between `Algorithm A` and `Algorithm B` on the MMLib benchmark set, given 5000 schedules.



(a) absolute difference (b) relative difference

Figure 3: Difference in performance between `Algorithm A` and `Algorithm B` on the MMLib benchmark set, given 25000 schedules.

the infrastructure of the VSC - Flemish Supercomputer Center, funded by the Hercules foundation and the Flemish Government - department EWI.[6]

Step 5: Feature elimination

The fifth step in the procedure reduces the data by eliminating useless and uninformative features. Due to the structure of the instances, some features are uni-valued, meaning that they have the same value for all instances. Examples of such features are the ones related to doubly constrained resources.[7] In total, 103 uni-valued features are removed.

Another type of uninformative features are correlated features. We carry out a correlation analysis and remove 242 features perfectly correlated to at least one other feature. (The choice of which features are deleted and which feature stays is arbitrary.) The remaining set contains 341 features.

Step 6: Model construction

In this step, we use machine learning techniques to find mappings from the feature space onto the performance measures. In the studies on combinatorial auctions (Leyton-Brown et al., 2002) and propositional satisfiability problems (Nudelman et al., 2004), statistical regression techniques are used and reported to be able to produce accurate models. Smith-Miles and Lopes (2011) employ self-organising maps to find such mappings for the job shop scheduling problem. In our study, we experimented with several standard machine learning techniques made available through the WEKA software tool (Hall et al., 2009). This tool offers a large set of machine learning and data mining techniques and enables us to experiment with many different techniques and settings. In order to overcome problems of over-fitting, the models are evaluated using 10-fold cross-validation on the training set. The best models are then evaluated on the validation set of unseen instances in order to assess their true performance. Due to space restrictions, we report only on the best results for each algorithm.

For both algorithms, the M5-based models (M5P-Tree and M5-Rule models) appear to be among the most accurate ones. An M5P-Tree model is a decision tree with regression models at the leafs. Based on the properties of the instance, one of several linear regression models is used to predict the algorithm performance (i.e. the achieved makespan). A fragment of an M5P-Tree model is shown in Figure 4. An M5-Rule model is similar in nature but builds a set of rules. These rules state that when some properties are fulfilled, the makespan is to be calculated by some specific regression model. If no rules apply, then a 'backup' model is used. A fragment of an M5-Rule model is shown in Figure 5. (We refer to Quinlan (1992) and Wang and Witten (1997) for more details on these machine learning techniques.) The correlation coefficients of both techniques, using cross-validation on the training set, are $R = 0.99$ for both algorithms and both stopping criteria. High correlation coefficients indicate good predictive power. Indeed, when the models are evaluated on the validation set of unseen instances, the correlation coefficients remain $R = 0.99$. Figure 6(a) shows the real quality of solutions achieved by `algorithm A` (5000 schedules) versus the predicted quality of the M5P-Tree model on the validation set. Figure 7(a) shows a similar graph for `algorithm B` (5000 schedules).

A common understanding is that smaller models should be preferred. Using fewer features further avoids the problem of over-fitting, and offers an opportunity for domain experts to interpret the models. (I.e. it is easier to draw conclusions from a low-dimensional decision tree than it is from a high-dimensional tree.) WEKA offers several methods for feature selection. The approaches can be categorized into two main groups: learner-independent techniques and learner-dependent techniques. The main difference between these methods is that learner-independent techniques select a feature set prior to the actual model construction and based on data properties (like e.g. feature correlation) or models built by other learners (e.g. selecting the features with high coefficients in a regression model); while learner-dependent techniques do feature selection iteratively while constructing the model. This can be done in a forward or a backward manner, depending on whether features are iteratively added or deleted. A bi-directional approach is also possible. We have experimented with several learner-dependent and learner-independent techniques offered by WEKA.

[6] More information on the supercomputer can be found at `https://vscentrum.be/`.

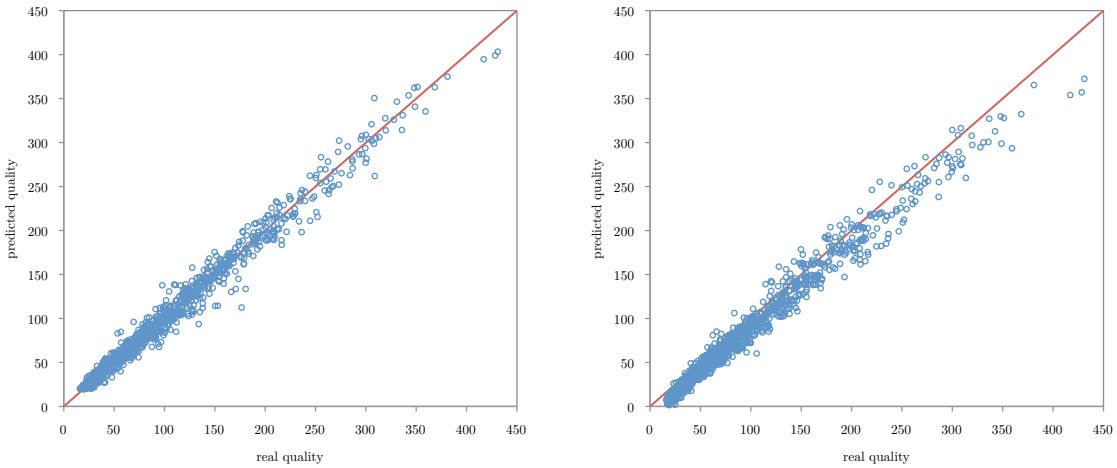
[7] Such resource types can be represented by a the combination of one renewable and one non-renewable resource type.

```
resourceUnitsPerMode_stddev_PerAct_stddev <= 1.8 :
|   successorsPerAct_variation <= 0.558 :
|   |   ratio_meanNonRenewableOverAvailability_mean <= 0.984 :
|   |   |   ratio_minRequestedOverAvailability_mean <= 5.3 :
|   |   |   |   nrPrecedenceConstraintsOverTheoreticalMaximum <= 0.117 :
|   |   |   |   |   durationPerMode_minOverMax_minOverMax <= 0.112 : LM1
|   |   |   |   |   durationPerMode_minOverMax_minOverMax >  0.112 :
|   |   |   |   |   |   ratio_minNonRenewableOverAvailability_mean <= 0.717 : LM2
|   |   |   |   |   |   ratio_minNonRenewableOverAvailability_mean >  0.717 : LM3
|   |   |   |   nrPrecedenceConstraintsOverTheoreticalMaximum >  0.117 :
|   |   |   |   |   precedenceConstraintsPerAct_variation <= 0.287 :
|   |   |   |   |   |   ratio_maxNonRenewableOverAvailability_variation <= 0.009 :
|   |   |   |   |   |   |   ratio_minRequestedOverAvailability_mean <= 2.858 : LM4
|   |   |   |   |   |   |   ratio_minRequestedOverAvailability_mean >  2.858 : LM5
|   |   |   |   |   |   ratio_maxNonRenewableOverAvailability_variation >  0.009 : LM6
|   |   |   |   |   precedenceConstraintsPerAct_variation >  0.287 :
|   |   |   |   |   |   ratio_minRequestedOverAvailability_mean <= 3.785 : LM7
|   |   |   |   |   |   ratio_minRequestedOverAvailability_mean >  3.785 :
|   |   |   |   |   |   |   ratio_minRequestedOverAvailability_mean <= 3.987 : LM8
|   |   |   |   |   |   |   ratio_minRequestedOverAvailability_mean >  3.987 : LM9
| <rest of the decision tree omitted>


LM num: 1
makespan =
        -  76.2618 * renewableResourceUnitsPerMode_variation_PerAct_stddev
        -  18.0078 * renewableResourceUnitsPerMode_minOverMax_PerAct_mean
        -   1.5174 * renewableResourceUnitsPerMode_minOverMax_PerAct_minOverMax
        +   4.6245 * resourceUnitsPerMode_stddev_PerAct_stddev
        -  46.1537 * resourceUnitsPerMode_minOverMax_PerAct_minOverMax
        +   2.0108 * ratio_minRequestedOverAvailability_mean
        +  12.46   * ratio_minNonRenewableOverAvailability_mean
        -   6.5218 * ratio_minNonRenewableOverAvailability_stddev
        +   8.2114 * ratio_maxRequestedOverAvailability_minimum
        +   0.5591 * ratio_maxRenewableOverAvailability_stddev
        -  22.659  * ratio_maxNonRenewableOverAvailability_variation
        +   1.0825 * ratio_maxNonRenewableOverAvailability_maximum
        -  16.9874 * ratio_meanRequestedOverAvailability_minimum
        +   9.1355 * ratio_meanNonRenewableOverAvailability_mean
        - 118.876  * nrPrecedenceConstraintsOverTheoreticalMaximum
        +   3.7608 * precedenceConstraintsPerAct_variation
        -   1.0581 * predecessorsPerAct_variation
        -   1.4417 * successorsPerAct_variation
        -   0.5442 * durationPerMode_stddev_variation
        -   0.1962 * durationPerMode_minimum_variation
        -  10.1374 * durationPerMode_minimum_minOverMax
        -  97.3204 * durationPerMode_maximum_variation
        -   9.8294 * durationPerMode_minOverMax_mean
        -   3.2007 * durationPerMode_minOverMax_variation
        -   6.9194 * durationPerMode_minOverMax_minimum
        - 146.7138 * durationPerMode_minOverMax_minOverMax
        -  10.4505 * durationPerRenewableType_variation_maximum
        + 128.2112

<other linear models omitted>
```

Figure 4: Fragment of the M5P-Tree model for `Algorithm A` (5000 schedules, reduced feature set).

```
Number of Rules : 25

Rule: 1
IF
        durationPerMode_minOverMax_minimum <= 0.092
        ratio_minRequestedOverAvailability_mean > 3.752
        ratio_meanNonRenewableOverAvailability_mean <= 0.995
        nrPrecedenceConstraintsOverTheoreticalMaximum > 0.109
THEN
makespan =
            64.1914 * renewableResourceUnitsPerMode_minOverMax_PerAct_mean
        +    2.1678 * nonRenewableResourceUnitsPerMode_minOverMax_PerAct_mean
        +   10.5603 * resourceUnitsPerMode_stddev_PerAct_stddev
        -   15.7521 * resourceUnitsPerMode_minOverMax_PerAct_minOverMax
        +   15.4853 * ratio_minRequestedOverAvailability_mean
        +   67.1617 * ratio_minRequestedOverAvailability_minimum
        -  192.8276 * ratio_minNonRenewableOverAvailability_mean
        +    0.03   * ratio_maxRenewableOverAvailability_stddev
        -  228.1945 * ratio_maxNonRenewableOverAvailability_mean
        - 2400.7993 * ratio_maxNonRenewableOverAvailability_stddev
        +  215.1461 * ratio_maxNonRenewableOverAvailability_maximum
        - 1411.1295 * ratio_maxNonRenewableOverAvailability_minOverMax
        +   88.2701 * ratio_meanRequestedOverAvailability_minimum
        +  347.036  * ratio_meanNonRenewableOverAvailability_mean
        -  319.2553 * nrPrecedenceConstraintsOverTheoreticalMaximum
        -    5.7718 * precedenceConstraintsPerAct_variation
        -    4.49   * precedenceConstraintsPerAct_minOverMax
        -   39.389  * predecessorsPerAct_variation
        -   25.3079 * successorsPerAct_variation
        +   15.3748 * durationPerMode_stddev_stddev
        -   32.9514 * durationPerMode_stddev_variation
        -  142.3609 * durationPerMode_variation_mean
        -   20.8415 * durationPerMode_minimum_minOverMax
        -   55.2811 * durationPerMode_minOverMax_mean
        +    0.6625 * durationPerMode_minOverMax_variation
        -  377.7354 * durationPerMode_minOverMax_minimum
        +   29.6102 * durationPerMode_minOverMax_minOverMax
        -   98.4044 * durationPerRenewableType_variation_maximum
        + 1302.1617

<Rule 2-23 omitted>

Rule: 24
IF
        ratio_maxNonRenewableOverAvailability_mean <= 1.768
THEN
makespan =
             3.8366 * ratio_minRequestedOverAvailability_mean
        +    3.9176 * ratio_maxRenewableOverAvailability_stddev
        -   22.4956 * ratio_maxNonRenewableOverAvailability_mean
        +   71.9196

Rule: 25 (Backup rule)
makespan =
        +   32.36
```

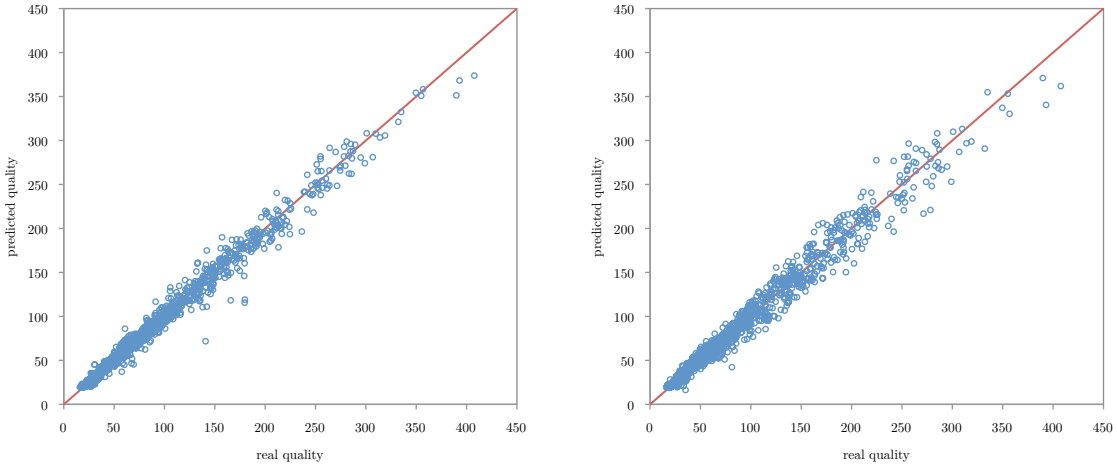
Figure 5: Fragment of the M5-Rule model for `Algorithm B` (5000 schedules, reduced feature set).



(a) using all 341 features

(b) after selecting 28 features

Figure 6: Real quality versus predicted quality of the solutions of `Algorithm A` on the validation set.



(a) using all 341 features

(b) after selecting 29 features

Figure 7: Real quality versus predicted quality of the solutions of `Algorithm B` on the validation set.

It appears that the different techniques come up with more or less the same subsets of features. There is some variation but mainly the same features are present in all the reduced sets. In what follows, we discuss the construction of smaller models using the bi-directional correlation-based feature selection procedure in WEKA, which is a learner-independent method.

We applied the feature selection procedure to the dataset for both algorithms (5000 schedules) and the resulting feature sets contain 28 and 29 features for `Algorithm A` and `Algorithm B` respectively. The remaining features can be categorized into four groups:

- statistics on the number of resource units required per mode over the activities (5 and 4 features for `Algorithm A` and `Algorithm B` respectively)
- statistics on the ratio of required resource units over the available units (9 and 11 features, related to the resource constrainedness)
- statistics on the number of precedence constraints (4 and 5 features, including the order strength)
- statistics on the duration of the execution modes (10 and 9 features)

As for the large feature sets, we have built several models for both algorithms using different techniques in WEKA. We report only on the most accurate ones.

It appears that the same techniques are again among the most accurate ones. Additionally, the Multilayer Perceptron technique achieves similar results. This model is a feed-forward artificial neural network that maps the feature values onto the quality (i.e. the makespan) of the solutions. Applying this model to the large set of features was not feasible due to the computational effort of training the network. The correlation coefficients of the M5P-Tree model, the M5-Rule model and the Multilayer Perceptron model, using 10-fold cross-validation on the training set, are $R = 0.98$ for both `Algorithm A` and `Algorithm B` (5000 schedules). When evaluated on the test set of unseen instances, the correlation coefficients remain $R = 0.98$. Figures 6(b) and 7(b) show the real quality versus the predicted quality of the M5P-Tree models based on the reduced feature sets and evaluated on the validation set, for `Algorithm A` and `Algorithm B` respectively (5000 schedules). Although the feature selection process has significantly reduced the feature set, the prediction accuracy is only marginally lowered and not significantly worse. The correlation coefficients and the graphs in Figures 6 and 7 show that there is no clear distinction between large and small models in terms of accuracy, the smaller models should thus be preferred. Similar results were obtained for the case where the algorithms are allowed to construct 25000 schedules. The feature selection approach drastically reduced the feature sets to 24 and 27 features for `Algorithm A` and `Algorithm B` respectively. The correlation coefficients using these reduced feature sets were also $R = 0.98$ for both algorithms, both using 10-fold cross-validation on the training set and when evaluated on the validation set of unseen instances.

It is important to note that the reduced feature sets still include a large portion of features related to the complexity measures mentioned in Section 2.2. It appears that these features (mainly related to the coefficient of network complexity, the order strength and the resource constrainedness) are of high importance. In order to validate this conjecture, we performed the experiment excluding these features from the dataset (for the stopping criterion of 5000 schedules). The correlation coefficients using 10-fold cross-validation on the training set are only $R = 0.74$ and $R = 0.78$ for `Algorithm A` and `Algorithm B` respectively. When evaluated on the validation set of unseen instances, these coefficients further drop to $R = 0.70$ and $R = 0.76$ for `Algorithm A` and `Algorithm B` respectively. We see a significant drop in accuracy when leaving these features out, indicating the importance of the complexity measures for predicting the empirical hardness of these two state-of-the-art algorithms.

In this section, we have shown that accurate prediction models can be built for two state-of-the-art algorithms for the MRCPSP instance distribution under study. Using these empirical hardness models, we will construct an automatic algorithm selection tool in the next section.

3.3. An automatic algorithm selection tool

Smith-Miles (2009) gives a cross-disciplinary survey of meta-learning for algorithm selection. The algorithm selection problem is formulated along the lines of the model presented by Rice (1976). The four key

ingredients, which are considered the meta-data of the problem, are:

- the problem space P, which is the set of problem instances of interest
- the feature space F, characterizing the problem instances
- the algorithm space A, which is the set of considered algorithms
- the performance space Y, that contains performance measures

The algorithm selection problem can then be formally stated as follows: For a given problem instance $x \in P$ with features $f(x) \in F$, find the selection mapping $S(f(x)) = \alpha \in A$ such that the selected algorithm α maximizes the performance $y(\alpha(x)) \in Y$.

The empirical hardness models developed in the previous section can be used to construct a mapping of instance features onto an algorithm choice. It is straightforward to construct such an automatic algorithm selection tool and employ this in a solution strategy that automatically runs the chosen algorithm from a portfolio on a given problem instance. It simply compares performance predictions for all algorithms in the portfolio and chooses the one with the best predicted performance. When the empirical hardness models are sufficiently accurate, one expects such an automatic algorithm selection tool to select the right algorithm for most instances. In what follows, we denote this approach as `AS1`. Table 3 summarizes the overall performance of `AS1` on the validation set of unseen instances (the same one as used in the previous section), and compares the results to the case where no algorithm selection is employed (i.e. always selecting `Algorithm A` (denoted as `always-A`) and always selecting `Algorithm B` (denoted as `always-B`)). The optimal case (where the best algorithm is always chosen, denoted as `AS*`) is also included. The results for both considered stopping criteria are displayed. Looking at the single-algorithm strategies for the stopping criterion of 5000 schedules, we see that `always-A` is better than `always-B`, both in terms of the number of correctly classified instances and in terms of the average relative deviation from the best solution. However, as already mentioned in *Step 2* of the previous section, the absolute difference in performance between both algorithms is larger for those instances where `Algorithm B` is better. When we thus look at the average makespan on the validation set, we see that `always-B` is better than `always-A`. This shows that it is not sufficient to just check the number of correct decisions. It is important to look at the overall performance of the selection strategy as well. For the stopping criterion of 5000 schedules (where the algorithms have a competitiveness ratio of $c = 0.7957$), Table 3 shows that `AS1` is able to effectively improve on the results of the best single-algorithm strategy in terms of all three considered measures (the percentage of correctly classified instances, the average relative deviation from the best found solution, and the average makespan). For the stopping criterion of 25000 schedules on the other hand, `AS1` can not improve over the results of `always-A`. In this setting, the algorithms are far less competitive ($c = 0.2198$) due to the increased dominance of `Algorithm A` over `Algorithm B`. As a consequence, the algorithm selection tool has to be considerably more accurate to be able to beat the best single-algorithm strategy. `AS1` is in this case unable to achieve the required level of accuracy. While we have shown that the proposed strategy (`AS1`) works for highly competitive algorithms, it seems to be unable to achieve the required level of accuracy for the 25000 schedules case. The `AS1` strategy bases its algorithm choice on the predicted qualities, both of which can be erroneous. Comparing such predictions can lead to wrong decisions. We will now investigate how we eliminate this drawback in order to further improve on these results. Based on the ideas of `AS1`, we will develop a new strategy that is able to improve over the best single-algorithm strategy, even when the algorithms are only poorly competitive.

Instead of building models to predict the performance of one algorithm, we will build classification models to predict the best algorithm choice directly. This approach is along the paradigms described by Smith-Miles (2009). We have labelled the training set of instances with an extra class-attribute: the algorithm that performed best on this instance, given the considered stopping criterion (instances where both algorithms performed equally were not labelled). Using classification tools in WEKA, we built models that map the instance features onto this class value. Constructing an automatic algorithm selection tool using such models is straightforward. Given a new instance, it determines the relevant feature values, uses the model to predict to which class the instance belongs and finally runs this predicted algorithm. We will denote this algorithm selection approach as `AS2`. Table 4 shows the performance of `AS2` for three of the best performing

		always-A	always-B	AS1	AS*
5000 schedules	% correctly classified instances	58.1	49.9	69.0	100
	avg. relative deviation from best (%)	2.30	2.85	1.32	0
	avg. makespan	100.68	99.40	98.51	97.34
25000 schedules	% correctly classified instances	89.6	18.0	80.1	100
	avg. relative deviation from best (%)	0.36	8.01	1.04	0
	avg. makespan	92.25	99.98	92.66	91.90

Table 3: Performance of the AS1 algorithm selection strategy on the validation set.

models learnt in WEKA, evaluated on the validation set of unseen instances. AS2a denotes the algorithm selection tool using a REP-Tree model, which is a decision tree that is pruned using a reduced-error-pruning scheme. AS2b denotes the tool using a DecisionTable model. This model builds a table with a number of decision rules (in this case 240 and 219 rules for the algorithms using the stopping criteria of 5000 and 25000 schedules respectively). AS2c denotes the tool using a RandomForest model. This is a collection of (in our case 500) decision trees for which the feature sets contain 30 randomly selected features. The model outputs the class that was the most predicted by the random trees. We see that for both considered stopping criteria, the RandomForest model performs best. For the stopping criterion of 5000 schedules, we see that AS2 significantly improves over AS1. Comparing Tables 3 and 4, we see that AS2 further reduces the relative deviation from the best found solution from 1.32% to 0.51%. The average makespan is very close to the optimal average makespan. While the AS1 approach led to good results for these highly competitive algorithms, the AS2 approach even further improves on these results, leading to near-optimal performance. For the stopping criterion of 25000 schedules, where the algorithms are only poorly competitive, we see that AS2 is able to effectively improve over the results of always-A. The relative deviation from the best found solution is reduced by half (from 0.36% to 0.18%). The AS2 strategy is thus able to achieve much higher accuracy levels than the AS1 approach. The resulting prediction accuracy is even sufficient to be able to achieve improvements on the 25000 schedules case, where the algorithms are only poorly competitive.

Combining the algorithms in a portfolio, and applying an automatic selection approach based on classification techniques can thus effectively improve over the results of each of the components individually. The results are near-optimal for both considered stopping criteria. Even when there is only limited competitiveness among the algorithms, there are still significant performance improvements possible as longs as the prediction accuracy is sufficiently high and the potential impact is not too small.

So far, the AS2 tools are built using all available features. As we have argued before, smaller models should be preferred if they do not affect the overall performance much. We have therefore applied a correlation-based bi-directional feature selection procedure to the datasets for both stopping criteria. The remaining feature sets contain 17 and 13 features for the stopping criteria of 5000 and 25000 respectively. These feature sets are further on used to build a series of AS2' tools. Note that we are now building a model to predict the class of an instance instead of the algorithm performance (i.e. makespan). Evidently, the correlation-based feature selection procedure selects other features. Nevertheless, the remaining sets again contain a considerable amount of features related to the complexity measures of Section 2.2. The remaining features can be categorised into the following groups: (for the stopping criterion of 5000 and 25000 schedules respectively)

- features related to the number of available resource units (1 feature and 1 feature)
- statistics on the number of resource units required per mode over the activities (4 features and 1 feature)
- statistics on the number of available resource units per resource type (3 features and 3 feature)
- statistics on the ratio of required resource units over the available units (8 features and 7 features)
- statistics on the duration of the modes (1 feature and 0 features)

		AS2a	AS2b	AS2c	AS*
5000 schedules	% correctly classified instances	80.2	77.1	83.2	100
	avg. relative deviation from best (%)	0.67	0.79	0.51	0
	avg. makespan	97.93	98.05	97.74	97.34
25000 schedules	% correctly classified instances	92.2	91.4	93.3	100
	avg. relative deviation from best (%)	0.24	0.27	0.18	0
	avg. makespan	92.05	92.08	92.02	91.90

Table 4: Performance of the AS2 algorithm selection strategy on the validation set.

		AS2a'	AS2b'	AS2c'	AS2d'	AS*
5000 schedules	% correctly classified instances	79.4	77.4	81.7	79.5	100
	avg. relative deviation from best (%)	0.69	0.84	0.57	0.67	0
	avg. makespan	97.92	98.10	97.85	97.87	97.34
25000 schedules	% correctly classified instances	91.7	91.6	93.4	91.2	100
	avg. relative deviation from best (%)	0.26	0.27	0.18	0.30	0
	avg. makespan	92.07	92.06	92.03	92.09	91.90

Table 5: Performance of the AS2' algorithm selection strategy on the validation set.

- features related to the precedence constraints (0 features and 1 feature)

Table 5 shows the performance of AS2' for four of the best performing models learnt in WEKA, both for the stopping criteria of 5000 and 25000 schedules. AS2a', AS2b' and AS2c' correspond to the same techniques as AS2a, AS2b and AS2c but using models based on the reduced feature sets. AS2d' denotes the tool using a Multilayer Perceptron model. When compared to the results of AS2 in Table 4, it can be concluded that the feature selection process did not significantly decrease the quality of the results. In some cases, there is even a slight improvement. The results are still near-optimal for both considered stopping criteria. This thus demonstrates that combining several algorithms in a portfolio and using an automatic algorithm selection approach like AS2 can greatly improve the overall performance. This is the case for highly competitive algorithms, but also for poorly competitive algorithms, as long as the difference in performance between both algorithms (and hence the potential impact) is relatively large.

In this section, we have built two automatic algorithm selection strategies using a portfolio of state-of-the-art algorithms for the MRCPSP. We have shown that AS1 can improve over the best single-algorithm approach in the case the algorithms are highly competitive. These results are furthermore improved by the AS2 strategy. This strategy is even capable of improving the overall performance in the case that the algorithms are only poorly competitive.

4. Related work

In this section we mention three domains related to automatic algorithm selection. We introduce the fields of algorithm configuration and hyper-heuristics, and briefly mention the concept of algorithm footprints.

4.1. Algorithm configuration

An automatic algorithm configuration tool tries to find a parameter setting that optimises the performance of an algorithm for a given problem instance set or distribution. Several methods exist and nowadays, algorithms with many numerical and categorical parameters can be tuned automatically.

Recently, Kadioglu et al. (2010) introduced the ISAC framework. The acronym stands for Instance-Specific Algorithm Configuration. The framework's goal is not finding one parameter configuration for the entire instance set or distribution. Instead, the instance distribution is clustered, and the automatic

algorithm configurator GGA (Ansótegui et al., 2009) is used to find a good parameter setting for each cluster. When given a new instance to solve, the framework first determines the cluster to which the instance belongs and then runs the solver with the parameter setting corresponding to that cluster. Another example of an automatic algorithm configurator is called PARAMILS and was introduced by Hutter et al. (2009).

The main difference with our work is that we treat algorithms as black boxes with a predetermined parameter setting. We are not trying to optimise the performance of one algorithm but rather seeking an overall better performance by combining different algorithms. Automatic algorithm configurators can be combined with an automatic algorithm selection tool. Such an application for SAT problems called HYDRA was introduced by Xu et al. (2010). HYDRA automatically builds a set of solvers with complementary strengths by iteratively configuring new algorithms. The resulting set of algorithms forms a portfolio from which an automatic algorithm selection tool picks the best algorithm to solve a new problem instance. Xu et al. (2011) have also applied these ideas to mixed integer programming, resulting in the HYDRA-MIP framework.

4.2. Hyper-heuristics

Recently, hyper-heuristics have drawn significant attention. Hyper-heuristics manage a set of expert-defined heuristics, called the low-level heuristics, to arrive at an algorithm producing better results than each of the low-level heuristics separately. The low-level heuristics are supposed to perform an improvement operation on a current solution while the hyper-heuristic coordinates the choice of which low-level heuristic is to be used. There is a domain barrier present between the low-level heuristics and the hyper-heuristic. The hyper-heuristic is unaware of the actual problem that is to be solved, it only operates on the low-level heuristics, which in their turn build or improve solutions for the actual problem instance. In order to do so, the hyper-heuristic has to evaluate the performance of the low-level heuristics at each point in time and decide which low-level heuristic to launch afterwards. One possibility to arrive at such a management capability is to implement machine learning techniques allowing for the prediction of the behaviour of each low-level heuristic in any eventual configuration of the solution space. Recently, state-of-the-art machine learning techniques have been proven very efficient in this respect (Misir, 2012). This application of machine learning proceeds on-line. Information on the learned predictors is presently not passed on from one run to another and the question as to whether this form of knowledge transfer could be fruitful remains open. One difference with the off-line methodology developed in the current paper is the effect of the speed of the learning mechanism. In on-line learning, learning must be considered as overhead as it takes cpu-time away from the actual algorithm. In off-line learning, such a constraint is not as stringent as the analysis does not interfere directly with the search.

4.3. Algorithm footprints

Corne and Reynolds (2010) have noted that when reporting on the performance of an algorithm, it is important to formulate the boundaries of that performance in the instance space. Many algorithms in the literature are compared based on average performance over a predetermined instance set or distribution. However, the *best* algorithm on the distribution is rarely the best algorithm for individual instances of the distribution or set that is used. The authors introduce the term *algorithm footprint* as a means to indicate how an algorithm's performance generalises in instance space. In their empirical study, the instance set is partitioned in clear and well-defined regions based on simple characteristics of the instances. A large set of algorithms (or configurations) is tested on the regions and it is shown that the best algorithm per region is rarely the algorithm that has an overall best performance over the complete set. The main difference to our approach is that in the work of Corne and Reynolds (2010), the instance space is partitioned a priori and a best method is determined for these well-defined regions. In our work, we investigate the instance space on a per-instance basis and determine for each instance separately which algorithm performs best. Instead of building a map that shows per region the best algorithm, we can use our models to tell us to which class an instance belongs. These classes however, are not as well-defined in terms of feature values as the regions of Corne and Reynolds (2010). A drawback of our approach, with respect to the footprint-approach, is that it yields far less clear or simple rules that can help understand why certain algorithms are better suited for

certain instances. On the other hand, the definition of the regions by Corne and Reynolds (2010) is still rather ad-hoc, and in the case of the MRCPSP considered in this paper, it is not clear how the instance space should be partitioned. A poorly chosen partitioning can make it hard to find a best algorithm for the individual regions. Our approach has the advantage that it does not need an a priori partitioning of the instance set. The models can give a very accurate prediction of which class/region an instance belongs to. What lacks in well-definedness is well compensated in accuracy.

Related to these footprints is the work of Smith-Miles and Lopes (2011). The authors use self-organising maps to visualise how algorithm performance varies over the instance distribution. Self-organising maps were developed by Kohonen (1982) as a way to automatically detect strong features in large datasets. Smith-Miles and Lopes (2011) focus on the real-world timetabling problem that was used in the 2007 International Timetabling Competition (ITC2007). The instances under study are a limited amount of real-world instances from Udine University, together with two sets of generated real-world-like instances. Two algorithms, ranked top 5 in the competition, are compared. The instances are characterised by 32 feature values. Smith-Miles and Lopes show that these self-organising maps can cleary distinguish between the three types of instances (real-world or element of one of the two generated sets). When superposing the algorithm performance onto these maps, and hence revealing the footprints of the algorithms, the boundaries of the regions are far less clear. Both algorithms are competitive across the two-dimensional map. In large portions of the map, there is no clear difference between the algorithms. There are also certain smaller regions where the algorithms do differentiate. However, in these regions, there is still no clear winner. There is only one narrow line going through the map on which one algorithm is consistently better than the other. The authors conclude that it is very difficult to explain this pattern in terms of feature values. This is consistent with our findings. Our algorithm selection approach can predict the best algorithm on a given instance, it does however not provide a simple description of the regions in which each algorithm outperforms the other.

5. Conclusions and further research

In this paper, we have successfully constructed (several) automatic algorithm selection tools for the multi-mode resource-constrained project scheduling problem. The result is a super-algorithm that outperforms all of its components individually. Given a new instance, the super-algorithm selects an algorithm from the portfolio of state-of-the-art algorithms, based on simple characteristics of the instance.

We have introduced a theoretical framework that allows characterising a set of two algorithms in terms of their reciprocal competitiveness and the combined potential impact on an instance set. The competitiveness ratio gives an indication of how accurate an algorithm selection tool must be to be able to achieve performance improvements. The potential impact furthermore represents the relative size of the maximal possible improvement over the instance set. These factors may be informative when deciding which algorithms to use in an algorithm selection tool.

Because of the size and nature of the well-established PSPlib benchmark set, and the quality of state-of-the-art procedures, there is not much room for improvement there. The competitiveness is low and the potential impact indicates that the absolute performance improvements will be limited. The focus of this paper is the MMlib benchmark set, where the considered state-of-the-art algorithms are highly competitive (given a stopping criterion of 5000 schedules). We also included an additional stopping criterion of 25000 schedules, leading to poorly competitive algorithms. We showed that the algorithm selection tools could also improve in such a setting.

We have built a set of automatic algorithm selection tools using two different approaches. Our first approach employs empirical hardness models directly. We have therefore built models able to accurately predict the outcome of an algorithm (i.e. the makespan of the schedule it produces). Producing such accurate predictions is in itself a new and important contribution of this research. This first approach selects the algorithm to run based on the performance predictions of these empirical hardness models. This approach led to good results, but only for the case where the algorithms are highly competitive. Our second approach outperforms our first approach. For this second approach, we built only one classification predicting a class (i.e. the algorithm that will probably perform best). Since only one prediction is made instead of two, we reduce the probability of making wrong selections. We have shown that this approach leads to near-optimal

results for both considered stopping criteria. We have thus shown that this approach is even sufficiently accurate for achieving performance improvements for poorly competitive algorithms.

An important note on our models in particular, but in general on all models built using training data, is that the construction of the training set is crucial for the success of the model. The results are only expected to be valid within the scope of the training data. In this paper, we have built an automatic algorithm selection tool that works (extremely) well on the MMLIB benchmark set. This result however, can not be straightforwardly used in any other practical project scheduling setting. For the method to work, the training set must be representative for the instances that need to be solved in the actual setting. On the other hand, the training set does not necessarily need to be as wide and broad as possible. In practice, the instances will probably always show some internal structure depending on the specific problem domain. It is most important that these structures are represented in the training data. If one knows in advance that there are only three types of resources, and that all jobs will require at least two of them, then there will be no added value in adding training data with many more resource types or requirement patterns. Our study demonstrates the practicality of our approach, it is not a straightforwardly applicable software tool for any practical situation.

The models in our study are built using different machine learning techniques made available through WEKA. In order to build accurate empirical hardness models, we designed an extensive feature set for the problem description. This set is in itself an important contribution to the literature. The feature set can easily be expanded to include other properties of other variants of the RCPSP. The set contains only simple, efficiently computable instance properties. Several complexity measures for project scheduling problems, as reported in the literature and reviewed in Section 2.2, are also included. We have shown that these complexity measures are of high importance for the accuracy of the empirical hardness models. If these measures are not included in the feature set, the accuracy of the models drops considerably.

In this paper, we have successfully constructed automatic algorithm selection tools that choose from a portfolio of state-of-the-art algorithms. Their performance is far better than using any of the components individually. To be able to compare this approach to other state-of-the-art algorithms, we need more information about these other algorithms on the MMLIB benchmark instances. It is considered future work to focus on this task and to hopefully improve the performance even more by including more state-of-the-art algorithms in the portfolio.

While our empirical hardness models have now been shown able to decide where one algorithm will outperform another algorithm, it remains to be investigated *why* this can be done. While the complexity measures in the literature focus on the intrinsic hardness of an instance, we find a clear relationship between properties of an instance and the empirical hardness of that instance (i.e. the performance of a specific algorithm on the instance). It is an open question to try and understand the nature of this relationship and to draw conclusions from this. These conclusions might help the development of new algorithms. An interpretation of the empirical hardness models built in this paper can lead to valuable insights. We plan to undertake such an investigation in the near future.

Ansótegui, C., Sellmann, M., Tierney, K., 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In: Proceedings of the 15th international conference on Principles and practice of constraint programming. CP'09. pp. 142–157.

Bein, W. W., Kamburowski, J., Stallmann, M. F. M., 1992. Optimal reduction of two-terminal directed acyclic graphs. SIAM Journal on Computing 21, 1112–1129.

Blazewicz, J., Lenstra, J. K., Rinnooy Kan, A. H. G., 1983. Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics 5, 11–24.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112 (1), 3–41.

Buddhakulsomsiri, J., Kim, D. S., 2007. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research 178 (2), 374–390.

Coelho, J., Vanhoucke, M., 2011. Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. European Journal of Operational Research 213 (1), 73–82.

Cooper, D. F., 1976. Heuristics for scheduling resource-constrained projects: An experimental investigation. Management Science 22 (11), 1186–1194.

Corne, D. W., Reynolds, A. P., 2010. Optimisation and generalisation: Footprints in instance space. In: Schaefer, R., Cotta, C., Kolodziej, J., Rudolph, G. (Eds.), PPSN (1). Vol. 6238 of Lecture Notes in Computer Science. Springer, pp. 22–31.

Dar-El, E. M., 1973. MALB - a heuristic technique for balancing large single-model assembly lines. AIIE Transactions 5 (4), 343–356.
Davis, E. W., 1975. Project network summary measures constrained resource scheduling. AIIE Transactions 7 (2), 132–142.
De Reyck, B., Herroelen, W., 1996. On the use of the complexity index as a measure of complexity in activity networks. European Journal of Operational Research 91 (2), 347–366.
Elmaghraby, S. E., Herroelen, W. S., 1980. On the measurement of complexity in activity networks. European Journal of Operational Research 5 (4), 223–234.
Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H., 2009. The WEKA data mining software: an update. SIGKDD Explor. Newsl. 11, 10–18.
Hartmann, S., 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics (NRL) 49 (5), 433–448.
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.
Hartmann, S., Kolisch, R., 1998. Experimental evaluation of state-of-the-art heuristics for the resourc-constrained project scheduling problem. European Journal of Operational Research 127, 394–407.
Herroelen, W., De Reyck, B., 1999. Phase transitions in project scheduling. Journal of the Operational Research Society 50 (2), 148–156.
Herroelen, W., Demeulemeester, E., De Reyck, B., 1998. A classification scheme for project scheduling. In: Weglarz, J. (Ed.), Project Scheduling: Recent Models, Algorithms and Applications. Kluwer, pp. 1–26.
Hutter, F., Hoos, H. H., Leyton-Brown, K., Stutzle, T., 2009. ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306.
Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K., 2010. ISAC - instance-specific algorithm configuration. In: ECAI'10. pp. 751–756.
Kao, E. P. C., Queyranne, M., 1982. On dynamic programming methods for assembly line balancing. Operations Research 30 (2), 375–390.
Kohonen, T., 1982. Self-organized formation of topologically correct feature maps. Biological Cybernetics 43, 59–69.
Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research 90 (2), 320 – 333.
Kolisch, R., Drexl, A., 1996. Adaptive search for solving hard project scheduling problems. Naval Research Logistics (NRL) 43 (1), 23–40.
Kolisch, R., Drexl, A., 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions 29, 987–999.
Kolisch, R., Hartmann, S., 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research 174 (1), 23–37.
Kolisch, R., Sprecher, A., 1996. PSPLIB - a project scheduling problem library. European Journal of Operational Research 96, 205–216.
Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41 (10), 1693–1703.
Leyton-Brown, K., Nudelman, E., Shoham, Y., 2002. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Constraint Programming (CP). pp. 556–572.
Lova, A., Tormos, P., Cervantes, M., Barber, F., 2009. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. International Journal of Production Economics 117 (2), 302–316.
Mastor, A. A., 1970. An experimental investigation and comparative evaluation of production line balancing techniques. Management Science 16 (11), 728–746.
Misir, M., 2012. Intelligent hyper-heuristics: a tool for solving generic optimisation problems. Ph.D. thesis, KU Leuven.
Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In: Ribeiro, C., Celso, P. (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, pp. 557–588.
Nudelman, E., Leyton-Brown, K., Devkar, A., Shoham, Y., Hoos, H., 2004. Understanding random SAT: Beyond the clauses-to-variables ratio. In: International Conference on Principles and Practice of Constraint Programming (CP). pp. 438–452.
Pascoe, T. L., 1966. Allocation of resources - CPM. Revue Francaise Recherche Operationelle (38), 31–38.
Patterson, J. H., 1976. Project scheduling: The effects of problem structure on heuristic performance. Naval Research Logistics Quarterly 23 (1), 95–123.
Quinlan, R. J., 1992. Learning with continuous classes. In: 5th Australian Joint Conference on Artificial Intelligence. World Scientific, Singapore, pp. 343–348.
Rice, J. R., 1976. The algorithm selection problem. Advances in Computers 15, 65–118.
Schirmer, A., 2000. Case-based reasoning and improved adaptive search for project scheduling. Naval Research Logistics (NRL) 47 (3), 201–222.
Schirmer, A., Riesenberg, S., 1998. Class-based control schemes for parameterized project scheduling heuristics. Institute für Betriebswirtschaftlehre der Universität Kiel.
Smith-Miles, K., 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys 41 (1), 1–25.
Smith-Miles, K., Lopes, L., 2011. Generalising algorithm performance in instance space: A timetabling case study. In: Coello Coello, C. A. (Ed.), LION. Vol. 6683 of Lecture Notes in Computer Science. Springer, pp. 524–538.
Smith-Miles, K., Lopes, L., 2012. Measuring instance difficulty for combinatorial optimization problems. Computers & Operations Research 39 (5), 875–889.

Van Peteghem, V., 2010. Multi-mode resource-constrained project scheduling problem: Solution procedures and extensions. Ph.D. thesis, Ghent University.
Van Peteghem, V., Vanhoucke, M., 2010. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 201 (2), 409–418.
Van Peteghem, V., Vanhoucke, M., 2011. Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. J. Heuristics 17 (6), 705–728.
Wang, Y., Witten, I. H., 1997. Induction of model trees for predicting continuous classes. In: Poster papers of the 9th European Conference on Machine Learning. Springer.
Wauters, T., Verbeeck, K., Vanden Berghe, G., De Causmaecker, P., 2011. Learning agents for the multi-mode project scheduling problem. Journal of the Operational Research Society 62 (2), 281–290.
Węglarz, J., Józefowska, J., Mika, M., Waligóra, G., 2011. Project scheduling with finite or infinite number of activity processing modes - a survey. European Journal of Operational Research 208 (3), 177 – 205.
Xu, L., Hoos, H. H., Leyton-Brown, K., 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In: Twenty-Fourth Conference of the Association for the Advancement of Artificial Intelligence (AAAI-10). pp. 210–216.
Xu, L., Hutter, F., Hoos, H. H., Leyton-Brown, K., 2008. Satzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research 32, 565–606.
Xu, L., Hutter, F., Hoos, H. H., Leyton-Brown, K., 2011. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In: RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI).

Appendix A. A feature set for MRCPSP instances

This appendix contains an extensive list of the feature set developed for the construction of empirical hardness models for the (multi-mode) resource-constrained project scheduling problem.

Resource types can be renewable, non-renewable or doubly-constrained. The latter type is however not used in the benchmark sets in this paper, therefore we did not always include doubly-constrained resource types in the features. Nevertheless, the feature set can easily be extended to include similar features regarding doubly-constrained resource types.

For the basic version of the problem (single-mode), the features that take the mean, minimum and maximum over the modes will all have the same values, features that take the standard deviation over the modes will have value 0 and features that take the variation will either be 0 of not defined (NaN). Similarly, when no non-renewable resources are used, a large set of features will also become redundant.

Currently, the feature set consists of 686 instance features for the MRCPSP. These features can be grouped into four categories: size related, resource constraint related, precedence constraint related and activity duration related features.

Note that a subset of the complexity measures of Section 2.2 are also included in this set, albeit under a different name.

The following list sums up all features: (the number of features in each group is also given)

- Size related features (35):
  - the number of activities
  - the total number of resource types, which is the sum of three other features:
    - the number of renewable resource types
    - the number of non-renewable resource types
    - the number of doubly constrained resource types
  - the total number of resource units available, which is the sum of
    - the number of renewable resource units available
    - the number of non-renewable resource units available
    - the number of doubly constrained resource units available
  - the ratio of the total number of resource types over the number of activities
  - the ratio of the total number of resource units available over the number of activities
  - concerning the composition of the resource pool:

    - the fraction of renewable resource types (the number of renewable resource types over the total number of resource types)
    - the fraction of renewable resource units available (the number of renewable resource units available over the total number of resource types available)
    - the fraction of non-renewable resource types
    - the fraction of non-renewable resource units available
    - the fraction of doubly constrained resource types
    - the fraction of doubly constrained resource units available
  - for each resource type, the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of the number of resource units available
    - Additionally, this is done for the renewable and non-renewable resource types separately.

- Resource constraint related features (486):
  - Features which are aggregated over the activities:

    First of all, we look at the number of resource types, as well as the number of resource units each activity requires. As each activity possibly has several modes, we calculate for each activity the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum over the modes of:
    - the number of renewable resource types required
    - the number of renewable resource units required
    - the number of non-renewable resource types required
    - the number of non-renewable resource units required
    - the total number of resource types required
    - the total number of resource units required

    Afterwards, we aggregate this information over the activities by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, resulting in 216 features (6*6*6 = 216). These features include (various statistics on) the resource factor of each resource type as defined by Pascoe (1966).
  - Features which are aggregated over the resource types:

    Again, we look at the number of resource units each activity requires. Each activity has possibly several modes, we consider these modes separately and calculate for each resource type the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of required resource units per mode.

    Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This results in 108 features (3*6*6 = 108).

    We also look at the number of activities needing resources of a certain type. Since activities have several modes, we first calculate for each resource type and each activity the fraction of modes of this activity that requires this resource type. Then the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum over all activities of this fraction is calculated.

    Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This results in 108 features (3*6*6 = 108).

- Ratios of feature values (related to the resource constrainedness as defined by Patterson (1976)).

  We look at the number of resource units each activity requires. We use the mean, minimum and maximum number of units over the modes of each activity (as already calculated above). We add up these values for all activities and divide this by the available resource units of each type.

  Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This leads to another 54 features (3*6*3). Note that this information is related to the (inverse of) the resource strength as originally defined by Cooper (1976).

- Features related to the precedence constraints (21):

  (These features are independent of the existence of modes, hence we include the same features as for the RCPSP)

  - the total number of precedence constraints
  - the number of precendence constraints divided by the theoretical maximum number of precedence constraints (defined as order strength by Mastor (1970))
  - the ratio of the number of precedence constraints over the number of activities (which is the coefficient of network complexity as defined by Pascoe (1966))
  - the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of
    - the total number of precedence constraints per activity
    - the number of predecessors per activity
    - the number of successors per activity

- Features related to the durations of activities (144):

  - Features which are aggregated over the activities:

    Each activity can have several modes, so we first calculate the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of the duration of the modes per activity. Afterwards, we aggregate this information over the activities. We calculate the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values over the activities, resulting in 36 features.

  - Features which are aggregated over the resource types:

    Again, we look at the duration of the activities that require certain resources. Each activity has possibly several modes, we consider these modes separately and calculate for each resource type the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of the duration of modes needing this resource.

    Afterwards, we aggregate this information over the resource types by calculating the mean, standard deviation, variation, minimum, maximum and the ratio of the minimum over the maximum of these values, once only for the renewable resource types, once only for the non-renewable resource types and once for all resource types together. This results in 108 features (3*6*6 = 108).

Note that we do not include the complexity index as defined by De Reyck and Herroelen (1996). The computation of this measure is not as straightforward and efficient as the other features in this set. The resource strength as defined by Kolisch et al. (1995) is also left out for the reasons explained in Section 2.2.