Algorithm Selection for Combinatorial Search Problems: A Survey

Lars Kotthoff larsko@4c.ucc.ie

Abstract

The Algorithm Selection Problem is concerned with selecting the best algorithm to solve a given problem instance on a case-by-case basis. It has become especially relevant in the last decade, with researchers increasingly investigating how to identify the most suitable existing algorithm for solving a problem instance instead of developing new algorithms. This survey presents an overview of this work focusing on the contributions made in the area of combinatorial search problems, where algorithm selection techniques have achieved significant performance improvements. We unify and organise the vast literature according to criteria that determine algorithm selection systems in practice. The comprehensive classification of approaches identifies and analyses the different directions from which algorithm selection has been approached. This paper contrasts and compares different methods for solving the problem as well as ways of using these solutions.

1. Introduction

For many years, Artificial Intelligence research has been focusing on inventing new algorithms and approaches for solving similar kinds of problem instances. In some scenarios, a new algorithm is clearly superior to previous approaches. In the majority of cases however, a new approach will improve over the current state of the art for only some problem instances. This may be because it employs a heuristic that fails for instances of a certain type or because it makes other assumptions about the instance or environment that are not satisfied in some cases. Selecting the most suitable algorithm for a particular problem instance aims to mitigate these problems and has the potential to significantly increase performance in practice. This is known as the Algorithm Selection Problem.

The Algorithm Selection Problem has, in many forms and with different names, cropped up in many areas of Artificial Intelligence in the last few decades. Today there exists a large amount of literature on it. Most publications are concerned with new ways of tackling this problem and solving it efficiently in practice. Especially for combinatorial search problems, the application of algorithm selection techniques has resulted in significant performance improvements that leverage the diversity of systems and techniques developed in recent years. This paper surveys the available literature and describes how research has progressed.

Researchers have long ago recognised that a single algorithm will not give the best performance across all problem instances one may want to solve and that selecting the most appropriate method is likely to improve the overall performance. Empirical evaluations have provided compelling evidence for this (e.g. Aha, 1992; Wolpert & Macready, 1997).

The original description of the Algorithm Selection Problem was published in Rice (1976). The basic model described in the paper is very simple – given a space of instances and a space of algorithms, map each instance-algorithm pair to its performance. This mapping can then be used to select the best algorithm for a given instance. The original figure that illustrates the model is reproduced in Figure 1 on the following page. As Rice states,

> "The objective is to determine $S(x)$ [the mapping of problems to algorithms] so as to have high algorithm performance."

Almost all contemporary approaches employ machine learning to learn the performance mapping from problem instances to algorithms using features extracted from the instances. This often involves a *training phase*, where the candidate algorithms are run on a sample of the problem space to

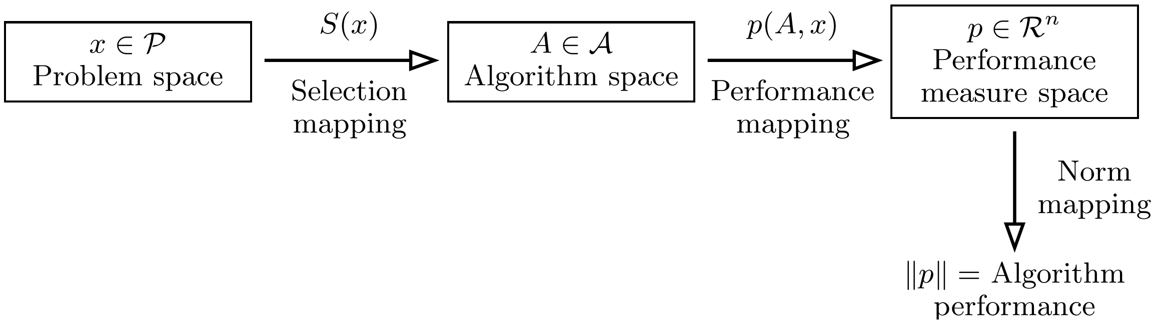

Figure 1: Basic model for the Algorithm Selection Problem as published in Rice (1976).

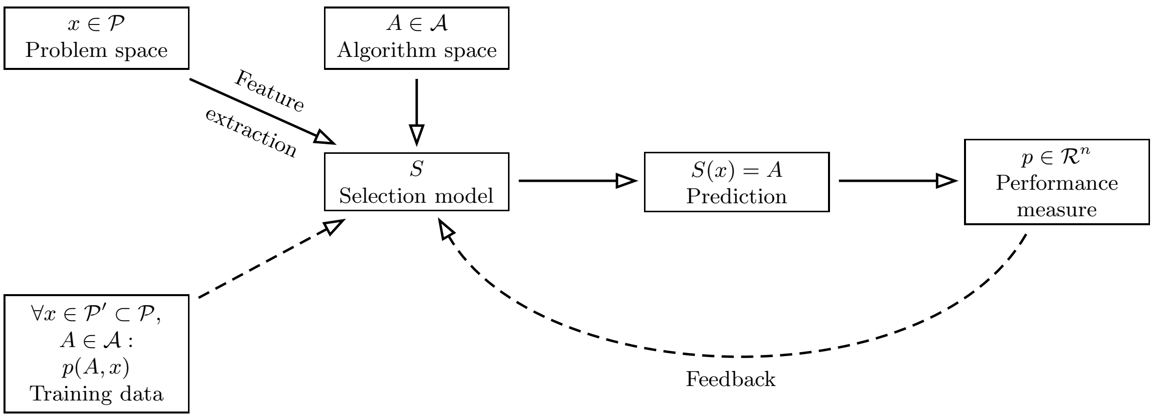

Figure 2: Contemporary algorithm selection model. Dashed lines show optional connections.

experimentally evaluate their performance. This training data is used to create a *performance model* that can be used to predict the performance on new, unseen instances. The term *model* is used only in the loosest sense here; it can be as simple as a representation of the training data without any further analysis.

Figure 2 sketches a contemporary algorithm selection model that corresponds more closely to approaches that use machine learning. At the heart of is the selection model S, which is trained using machine learning techniques. The data for the model comes from the algorithms $A \in \mathcal{A}$ and the problems $x \in \mathcal{P}$, which are characterised by features. S is created either by using training data that contains the performances of the algorithms on a subset of the problems from the problem space, or feedback from executing the chosen algorithm on a problem and measuring the performance. Some approaches use both data sources.

The model S makes the prediction of a specific algorithm A given a problem x. This algorithm is then used to solve the problem. At a high level, this describes the workings of an algorithm selection model, but there are many variations. The figure is meant to give a general idea, not describe every approach mentioned in this paper.

Arguably, one of the most prominent systems to do algorithm selection is and has been SATzilla (Xu, Hutter, Hoos, & Leyton-Brown, 2008). There are several reasons for this. It was the first system to really bring home the point of algorithm portfolios in combinatorial search by dominating

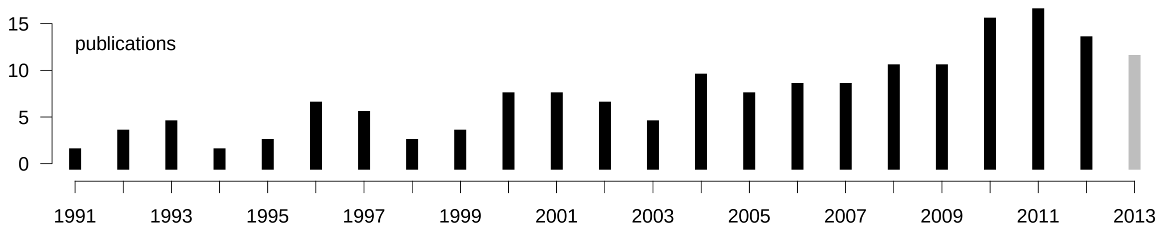

Figure 3: Publications by year. Data taken from `http://4c.ucc.ie/~larsko/assurvey/`. Note that the number for 2013 is not final.

the SAT competition[1] for years. Furthermore, it is probably the only system that has been developed over a period of several years, continuously improving its performance. Its authors have not limited themselves to scientific advancements, but also implemented a number of techniques that make it viable to run the system in practice.

Figure 3 shows that over the last two decades, there has been an increasing interest in algorithm selection, as witnessed by the number of publications in the field. In particular, the number of publications has increased significantly after the success of SATzilla in the SAT competition in 2007. This is probably the best indication of its impact on the research community.

Despite this prominence, other approaches rarely use SATzilla's ideas directly. Another set of prominent systems that have been developed over a number of years start with ISAC (Kadioglu, Malitsky, Sellmann, & Tierney, 2010) and continue with 3S (Kadioglu, Malitsky, Sabharwal, Samulowitz, & Sellmann, 2011) and CSHC (Malitsky, Sabharwal, Samulowitz, & Sellmann, 2013). Despite the commonality of the goal, and indeed application domain, there is no explicit cross-fertilisation between these approaches and SATzilla.

In other research areas, innovations implemented by one system usually make it into other systems after some time. This is not the case in algorithm selection. This is partly because of the sheer number of different approaches to solving the problem – a technique may simply not be applicable in a different context and completely different methods may be able to achieve similar performance. In addition, there is no algorithm selection community in the same sense in which there is for example a SAT community. As a result, publications are fragmented and scattered throughout different areas of AI. This makes it very hard to get a big picture overview of what the best techniques are or indeed to simply become aware of other approaches. As a result, many approaches have been reinvented with small variations.

Because there is no clear single line of research, this survey will look at the different aspects of algorithm selection and the techniques that have been used to tackle them. For a different take from a machine learning point of view, the interested reader is referred to Smith-Miles (2008).

2. Algorithm portfolios

For diverse sets of problem instances, it is unlikely that a single algorithm will be the most suitable one in all cases. A way of mitigating this restriction is to use a *portfolio* of algorithms. This idea is closely related to the notion of algorithm selection itself – instead of making an up-front decision on what algorithm to use, it is decided on a case-by-case basis for each instance individually.

The idea of algorithm portfolios was first presented by Huberman, Lukose, and Hogg (1997). They describe a formal framework for the construction and application of algorithm portfolios and evaluate their approach on graph colouring problems. However, the work cited more frequently in

1. `http://www.satcompetition.org/`

the Artificial Intelligence community is the later paper by Gomes and Selman (1997), which draws on the ideas presented by Huberman et al., 1997. The technique itself however had been described under different names by other authors at about the same time in different contexts (e.g. "algorithm family", Allen & Minton, 1996).

Beyond the simple idea of using a set of algorithms instead of a single one, there is a lot of scope for different approaches. There are two main types of portfolios. *Static portfolios* are constructed offline before any problem instances are solved. While solving an instance, the composition of the portfolio and the algorithms within it do not change. *Dynamic portfolios* change in composition, configuration of the constituent algorithms, or both during solving.

2.1 Static portfolios

Static portfolios are the most common type. They have been used in the earliest papers, e.g. Huberman et al. (1997). The number of algorithms or systems in the portfolio is fixed. This approach is used for example in the systems SATzilla (Xu et al., 2008), AQME (Pulina & Tacchella, 2007), and CPhydra (O'Mahony, Hebrard, Holland, Nugent, & O'Sullivan, 2008).

As the algorithms in the portfolio do not change, their selection is crucial for its success. Ideally, the algorithms will complement each other such that good performance can be achieved on a wide range of different problem instances. Hong and Page (2004) report that portfolios composed of a random selection from a large pool of diverse algorithms outperform portfolios composed of the algorithms with the best overall performance. In the same spirit, Samulowitz and Memisevic (2007) use a portfolio of heuristics for solving quantified Boolean formulae problems that have specifically been crafted to be orthogonal to each other. Xu, Hoos, and Leyton-Brown (2010) automatically engineer a portfolio with algorithms of complementary strengths. In Xu, Hutter, Hoos, and Leyton-Brown (2012), the authors analyse the contributions of the portfolio constituents to the overall performance and conclude that not algorithms with the best overall performance, but with techniques that set them apart from the rest contribute most.

Most approaches make the composition of the portfolio less explicit. Many systems use portfolios of solvers that have performed well in solver competitions with the implicit assumption that they have complementing strengths and weaknesses and the resulting portfolio will be able to achieve good performance. This is true for example for SATzilla. Part of the reason for composing portfolios in this manner may be that publications indicating that this may not be the best way are little known. Furthermore, it is of course very easy to take an existing set of algorithms.

2.2 Dynamic portfolios

Static portfolios are necessarily limited in their flexibility and diversity. Being able to modify the portfolio algorithms or create entirely new ones is an idea that emerged soon after the first applications of portfolios, leveraging earlier ideas on modifying algorithms dynamically.

The SAGE system (Strategy Acquisition Governed by Experimentation) (Langley, 1983) specialises generic building blocks for the instance to solve. It starts with a set of general operators that can be applied to a search state. These operators are refined by making the preconditions more specific based on their utility for finding a solution. The Multi-tac (Multi-tactic Analytic Compiler) system (Minton, 1996) similarly specialises a set of generic heuristics for the constraint problem instance to solve.

The same principle of combining algorithmic building blocks can be applied to algorithm portfolios. An example of this is the Adaptive Constraint Engine (ACE) (Epstein & Freuder, 2001). The building blocks are so-called advisors, which characterise variables of the constraint problem instance and give recommendations as to which one to process next. ACE combines these advisors into more complex ones. Also based on these ideas is CLASS (Fukunaga, 2002), which combines heuristic building blocks to form composite heuristics for solving SAT problems.

In these approaches, there is no strong notion of a portfolio – the algorithm or strategy used to solve a problem instance is assembled from lower level components and never made explicit.

3. Problem solving with portfolios

Once an algorithm portfolio has been constructed, the way in which it is to be used has to be decided. There are different considerations to take into account. We need to decide what to select and when to select it.

Given the full set of algorithms in the portfolio, a subset has to be chosen for solving the problem instance. This subset can consist of only a single algorithm that is used to solve the instance to completion, the entire portfolio with the individual algorithms interleaved or running in parallel or anything in between.

The selection of the subset of algorithms can be made only once before solving starts or continuously during search. If the latter is the case, selections can be made at well-defined points during search, for example at each node of a search tree, or when the system judges it to be necessary to make a decision.

3.1 What to select

A common and the simplest approach is to select a single algorithm from the portfolio and use it to solve the problem instance completely. This single algorithm has been determined to be the best for the instance at hand. For example SATzilla (Xu et al., 2008), ArgoSmArT (Nikolić, Marić, & Janičić, 2009), SALSA (Demmel, Dongarra, Eijkhout, Fuentes, Petitet, Vuduc, Whaley, & Yelick, 2005) and Eureka (Cook & Varnell, 1997) do this. The disadvantage of this approach is that there is no way of mitigating a wrong selection. If an algorithm is chosen that exhibits bad performance on the instance, the system is "stuck" with it and no adjustments are made, even if all other portfolio algorithms would perform much better.

An alternative approach is to compute schedules for running (a subset of) the algorithms in the portfolio. In some approaches, the terms portfolio and schedule are used synonymously – all algorithms in the portfolio are selected and run according to a schedule that allocates time slices to each of them. The task of algorithm selection becomes determining the schedule rather than to select algorithms. In the simplest case, all of the portfolio algorithms are run at the same time in parallel. This was the approach favoured in early research into algorithm selection (Huberman et al., 1997).

More sophisticated approaches compute explicit schedules. Roberts and Howe (2006) rank the portfolio algorithms in order of expected performance and allocate time according to this ranking. Pulina and Tacchella (2009) investigate different ways of computing schedules and conclude that ordering the algorithms based on their past performance and allocating the same amount of time to all algorithms gives the best overall performance.

Gomes and Selman (1997) also evaluate the performance of different candidate portfolios, but take into account how many algorithms can be run in parallel. They demonstrate that the optimal schedule (in this case the number of algorithms that are being run) changes as the number of available processors increases. Gagliolo and Schmidhuber (2008) investigate how to allocate resources to algorithms in the presence of multiple CPUs that allow to run more than one algorithm in parallel. Yun and Epstein (2012) take this approach a step further and craft portfolios with the specific aim of running the algorithms in parallel.

None of these approaches has emerged as the prevalent one so far – contemporary systems select both single algorithms as well as compute schedules. Part of the reason for this diversity is that neither has been shown to be inherently superior to the other one so far. Computing a schedule adds robustness, but is more difficult to implement and more computationally expensive than selecting a single algorithm.

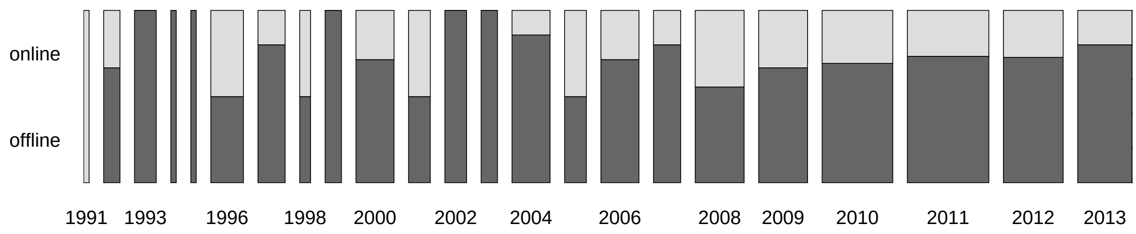

Figure 4: Publications by time of prediction by year, distinguishing between offline (dark grey) and online (light gray). The width of each bar corresponds to the number of publications that year (cf. Figure 3). Data taken from `http://4c.ucc.ie/~larsko/assurvey/`. Note that the number for 2013 is not final.

3.2 When to select

In addition to whether they choose a single algorithm or compute a schedule, existing approaches can also be distinguished by whether they operate before the problem instance is being solved (offline) or while the instance is being solved (online). The advantage of the latter is that more fine-grained decisions can be made and the effect of a bad choice of algorithm is potentially less severe. The price for this added flexibility is a higher overhead, as algorithms need to be selected more frequently.

Both approaches have been used from the very start of research into algorithm selection. The choice of whether to make predictions offline or online depends very much on the specific application and performance improvements have been achieved with both.

Examples of approaches that only make offline decisions include Xu et al. (2008), Minton (1996), O'Mahony et al. (2008). In addition to having no way of mitigating wrong choices, often these will not even be detected. These approaches do not monitor the execution of the chosen algorithms to confirm that they conform with the expectations that led to them being chosen. Purely offline approaches are inherently vulnerable to bad choices. Their advantage however is that they only need to select an algorithm once and incur no overhead while the instance is being solved.

Moving towards online systems, the next step is to monitor the execution of an algorithm or a schedule to be able to intervene if expectations are not met. Much of this research started as early as algorithm portfolio research itself. Fink (1997) investigates setting a time bound for the algorithm that has been selected based on the predicted performance. If the time bound is exceeded, the solution attempt is abandoned. More sophisticated systems furthermore adjust their selection if such a bound is exceeded. Borrett, Tsang, and Walsh (1996) try to detect behaviour during search that indicates that the algorithm is performing badly, for example visiting nodes in a subtree of the search that clearly do not lead to a solution. If such behaviour is detected, they propose switching the currently running algorithm according to a fixed replacement list.

The approaches that make decisions during search, for example at every node of the search tree, are necessarily online systems. Brodley (1993) recursively partitions the classification problem to be solved and selects an algorithm for each partition. Similarly, Arbelaez, Hamadi, and Sebag (2009) select the best search strategy at checkpoints in the search tree. In this approach, a lower-level decision can lead to changing the decision at the level above. This is usually not possible for combinatorial search problems, as decisions at a higher level cannot be changed easily.

Figure 4 shows the relative numbers of publications that use online and offline predictions by year. In the first half of the graph, the behaviour is quite chaotic and no clear trend is visible. This is certainly partly because of the relatively small number of relevant publications in these years. In the second half of the graph, a clear trend is visible however. The share of online systems steadily decreases – the overwhelming majority of recent publications use offline approaches.

This may be in part because as solvers and systems grow in complexity, implementing online approaches also becomes more complex. Whereas for offline approaches, existing algorithms can be used unchanged, making online decisions requires information on how the algorithm is progressing. This information can be obtained by instrumenting the system or similar techniques, but requires additional effort compared to offline approaches.

Another explanation is that at the beginning of this trend, SATzilla achieved noteworthy successes in the SAT competition. As SATzilla is an offline system, its success may have shifted attention towards those, resulting in a larger share of publications.

4. Portfolio selectors

Research on *how* to select from a portfolio in an algorithm selection system has generated the largest number of different approaches within the framework of algorithm selection. There are many different ways a mechanism to select from a portfolio can be implemented. Apart from accuracy, one of the main requirements for such a selector is that it is relatively cheap to run – if selecting an algorithm for solving a problem instance is more expensive than solving the instance, there is no point in doing so.

There are several challenges associated with making selectors efficient. Algorithm selection systems that analyse the problem instance to be solved, such as SATzilla, need to take steps to ensure that the analysis does not become too expensive. One such measure is the running of a pre-solver (Xu et al., 2008). The idea behind the pre-solver is to choose an algorithm with reasonable general performance from the portfolio and use it to start solving the instance before starting to analyse it. If the instance happens to be very easy, it will be solved even before the results of the analysis are available. After a fixed time, the pre-solver is terminated and the results of the algorithm selection system are used. Pulina and Tacchella (2009) use a similar approach and run all algorithms for a short time in one of their strategies. Only if the instance has not been solved after that, they move on to the algorithm that was actually selected. A number of additional systems implement a pre-solver in more or less explicit form (e.g. Kadioglu et al., 2011). As such, it is probably the individual technique that has found the most widespread adaptation, even though this is not explicitly acknowledged.

The selector is not necessarily an explicit part of the system. Minton (1996) compiles the algorithm selection system into a Lisp programme for solving a constraint problem instance. The selection rules are part of the programme logic. Fukunaga (2008) evolve selectors and combinators of heuristic building blocks using genetic algorithms. The selector is implicit in the evolved programme.

4.1 Performance models

The way the selector operates is closely linked to the way the performance model of the algorithms in the portfolio is built. In early approaches, the performance model was usually not learned but given in the form of human expert knowledge. Borrett et al. (1996) use hand-crafted rules to determine whether to switch the algorithm during solving. Allen and Minton (1996) also have hand-crafted rules, but estimate the runtime performance of an algorithm. More recent approaches sometimes use only explicitly-specified human knowledge as well.

A more common approach today is to automatically learn performance models using machine learning on training data. The portfolio algorithms are run on a set of representative problem instances and based on these experimental results, performance models are built. This approach is used by Xu et al. (2008), Pulina and Tacchella (2007), O'Mahony et al. (2008), Kadioglu et al. (2010), Guerri and Milano (2004), to name but a few examples. A drawback of this approach is that the time to collect the data and the training time are usually large.

Models can also be built without a separate training phase, but while the instance is solved. This approach is used by Gagliolo and Schmidhuber (2006) for example. While this significantly reduces the time to build a system, it can mean that the result is less effective and efficient. At the beginning, when no performance models have been built, the decisions of the selector might be poor. Furthermore, creating and updating performance models while the instance is being solved incurs an overhead.

The choice of machine learning technique is affected by the way the portfolio selector operates. Some techniques are more amenable to offline approaches (e.g. linear regression models used by Xu et al., 2008), while others lend themselves to online methods (e.g. reinforcement learning used by Gagliolo & Schmidhuber, 2006).

What type of performance model and what kind of machine learning to use is the area of algorithm selection with the most diverse set of different approaches, sometimes entirely and fundamentally so. There is no consensus as to the best technique at all. This is best exemplified by SATzilla. The performance model and machine learning used in early versions (Xu et al., 2008) was replaced by a fundamentally different approach in the most recent version (Xu, Hutter, Hoos, & Leyton-Brown, 2011). Nevertheless, both approaches have excelled in competitions.

Recently, research has increasingly focused on taking into account the cost of making incorrect predictions. Whereas in standard classification tasks there is a simple uniform cost (e.g. simply counting the number of misclassifications), the cost in the algorithm selection context can be quantified more appropriately. If an incorrect algorithm is chosen, the system takes more time to solve the problem instance. This additional time will vary depending on which algorithm was chosen – it may not matter much if the performance of the chosen algorithm is very close to the best algorithm, but it also may mean a large difference. SATzilla 2012 (Xu et al., 2011) and CSHC (Malitsky et al., 2013) are two examples of systems that take this cost into account explicitly.

4.1.1 Per-portfolio models

One automated approach is to learn a performance model of the entire portfolio based on training data. Usually, the prediction of such a model is the best algorithm from the portfolio for a particular problem instance. There is only a weak or no notion of an individual algorithm's performance.

This is used for example by Pulina and Tacchella (2007), O'Mahony et al. (2008), Kadioglu et al. (2010). Again there are different ways of doing this. Lazy approaches do not learn an explicit model, but use the set of training examples as a case base. For new problem instances, the closest instance or the set of n closest instances in the case base is determined and decisions made accordingly. Pulina and Tacchella (2007), O'Mahony et al. (2008) use nearest-neighbour classifiers to achieve this.

Explicitly-learned models try to identify the concepts that affect performance on a given problem instance. This acquired knowledge can be made explicit to improve the understanding of the researchers of the application domain. There are several machine learning techniques that facilitate this, as the learned models are represented in a form that is easy to understand by humans. Carbonell, Etzioni, Gil, Joseph, Knoblock, Minton, and Veloso (1991), Brodley (1993), Vrakas, Tsoumakas, Bassiliades, and Vlahavas (2003) learn classification rules that guide the selector. Vrakas et al. (2003) note that the decision to use a classification rule leaner was not so much guided by the performance of the approach, but the easy interpretability of the result. A classification model can be used to gain insight into what is happening in addition to achieving performance improvements. This is a relatively unexplored area of algorithm selection research – in many cases, the performance models are too complex or simply not suitable for this purpose.

Ideally, a performance model could be analysed and the knowledge that enables the portfolio to achieve performance improvements made explicit and be leveraged in creating new algorithms or adapting existing ones. In practice, this is yet to be tackled successfully by the algorithm selection community. A system that allowed to do this would certainly be a major step forward and take algorithm selection to the next level.

4.1.2 Per-algorithm models

A different approach is to learn performance models for the individual algorithms in the portfolio. The predicted performance of an algorithm on a problem instance can be compared to the predicted performance of the other portfolio algorithms and the selector can proceed based on this. The advantage of this approach is that it is easier to add and remove algorithms from the portfolio – instead of having to retrain the model for the entire portfolio, it suffices to train a model for the new algorithm or remove one of the trained models. Most approaches only rely on the order of predictions being correct. It does not matter if the prediction of the performance itself is wildly inaccurate as long as it is correct relative to the other predictions. In practice, the predictions themselves will indeed be off by orders of magnitude in at least some cases, but the overall system will still be able to achieve good performance.

Models for each algorithm in the portfolio are used for example by Allen and Minton (1996), Xu et al. (2008), Gagliolo and Schmidhuber (2006). A common way of doing this is to use regression to directly predict the performance of each algorithm. Xu et al. (2008), Leyton-Brown, Nudelman, and Shoham (2002), Haim and Walsh (2009) do this, among others. The performance of the algorithms in the portfolio is evaluated on a set of training instances, and a relationship between the characteristics of an instance and the performance of an algorithm derived. This relationship usually has the form of a simple formula that is cheap to compute at runtime.

Silverthorn and Miikkulainen (2010) on the other hand learn latent class models of unobserved variables to capture relationships between solvers, problem instances and run durations. Based on the predictions, the expected utility is computed and used to select an algorithm. Weerawarana, Houstis, Rice, Joshi, and Houstis (1996) use Bayesian belief propagation to predict the runtime of a particular algorithm on a particular instance. Bayesian inference is used to determine the class of a problem instance and the closest case in the knowledge base. A performance profile is extracted from that and used to estimate the runtime.

One of the main disadvantages of per-algorithm models is that they do not consider the interaction between algorithms at all. It is the interaction however that makes portfolios powerful – the entire idea is based on the fact that a combination of several algorithms is stronger than an individual one. Despite this, algorithm selection systems that use per-algorithm models have demonstrated impressive performance improvements. More recently however, there has been a shift towards models that also consider interactions between algorithms.

4.1.3 Hybrid models

A number of recently-developed approaches use performance models that draw elements from both per-portfolio models and per-algorithm models. The most recent version of SATzilla (Xu et al., 2011) uses models for pairs of algorithms to predict one which is going to have better performance. These predictions are aggregated as votes and the algorithm with the overall highest number of votes is chosen. This type of performance model allows to explicitly model how algorithms behave with respect to other algorithms and addresses the main conceptual disadvantage in earlier versions of SATzilla.

An orthogonal approach is inspired by the machine learning technique stacking (Kotthoff, 2012). It combines per-algorithm models at the bottom layer with a per-portfolio model at the top layer. The bottom layer predicts the performance of each algorithm individually and independently and the top layer uses those predictions to determine the overall best algorithm.

Hierarchical models are a similar idea in that they make a series of predictions where the later models are informed by the earlier predictions. Xu, Hoos, and Leyton-Brown (2007) use sparse multinomial logistic regression to predict whether a SAT problem instance is satisfiable and, based on that prediction, use a logistic regression model to predict the runtime of each algorithm in the portfolio. Here, the prediction of whether the instance is satisfiable is only used implicitly for the next performance model and not as an explicit input.

In all of these examples, a number of performance models are combined into the overall performance model. Such hybrid models are similar to per-portfolio models in the sense that if the composition of the portfolio changes, they have to be retrained. They do however also encapsulate the notion of an individual algorithm's performance.

4.2 Types of predictions

The way of creating the performance model of a portfolio or its algorithms is not the only choice researchers face. In addition, there are different predictions the performance model can make to inform the decision of the selector of a subset of the portfolio algorithms. The type of decision is closely related to the learned performance model however. The prediction can be a single categorical value – the best algorithm. This type of prediction is usually the output of per-portfolio models and used for example in Guerri and Milano (2004), Pulina and Tacchella (2007), Gent, Jefferson, Kotthoff, Miguel, Moore, Nightingale, and Petrie (2010). The advantage of this simple prediction is that it determines the choice of algorithm without the need to compare different predictions or derive further quantities. One of its biggest disadvantages however is that there is no flexibility in the way the system runs or even the ability to monitor the execution for unexpected behaviour.

A different approach is to predict the runtime of the individual algorithms in the portfolio. This requires per-algorithm models. For example Horvitz, Ruan, Gomes, Kautz, Selman, and Chickering (2001), Petrik (2005), Silverthorn and Miikkulainen (2010) do this. Allen and Minton (1996) estimate the runtime by proxy by predicting the number of constraint checks. Lobjois and Lemaître (1998) estimate the runtime by predicting the number of search nodes to explore and the time per node. Xu, Hutter, Hoos, and Leyton-Brown (2009) predict the *penalized average runtime score*, a measure that combines runtime with possible timeouts used in the SAT competition. This approach aims to provide more realistic performance predictions when runtimes are censored.

Some types of predictions require online approaches that make decisions during search. Borrett et al. (1996), Sakkout, Wallace, and Richards (1996), Carchrae and Beck (2004) predict when to switch the algorithm used to solve a problem instance. Horvitz et al. (2001) predict whether a run will complete within a certain number of steps to determine if to restart the algorithm. Lagoudakis and Littman (2000) predict the cost to solve a sub-instance. However, most online approaches make predictions that can also be used in offline settings, such as the best algorithm to proceed with.

The choice for type of prediction depends on the application in many scenarios. In a competition setting, it is usually best to use the scoring function used to evaluate entries. If the aim is to utilise a machine with several processors as much as possible, predicting only a single algorithm is unsuitable unless that algorithm is able to take advantage of several processors by itself. As the complexity of the predictions increases, it usually becomes harder to make them with high accuracy. Kotthoff, Gent, and Miguel (2012) for example report that using statistical relational learning to predict the complete order of the portfolio algorithms on a problem instance does not achieve competitive performance.

5. Features

The different types of performance models described in the previous sections usually use features to inform their predictions. Features are an integral part of systems that do machine learning. They characterise the inputs, such as the problem instance to be solved or the algorithm employed to solve it, and facilitate learning the relationship between these inputs and the outputs, such as the time it will take the algorithm to solve the problem instance.

A side effect of the research into algorithm selection has been that researchers have developed comprehensive feature sets to characterise problem instances. This is especially true in SAT. Whereas before SAT instances would mostly be described in terms of number of variables and clauses, the authors of SATzilla developed a large feature set to characterise an instance with a high level of detail.

Determining features that adequately characterise a new problem domain is often difficult and laborious. Once a good set of features has been established, it can be used by everybody. The determination of features is likely the single area associated with algorithm selection where most of the traditional science cycle of later approaches building on the results of earlier research occurs. The feature set used by SATzilla (Xu et al., 2008) is also used by 3S (Kadioglu et al., 2011) and Silverthorn and Miikkulainen (2010) for example. Gent et al. (2010) include features described in Guerri and Milano (2004) in their feature set.

The selection of the most suitable features is an important part of the design of algorithm selection systems. There are different types of features researchers can use and different ways of computing these. They can be categorised according to two main criteria.

First, they can be categorised according to how much domain knowledge an algorithm selection researcher needs to have to be able to use them. Features that require no or very little knowledge of the application domain are usually very general and can be applied to new algorithm selection problems with little or no modification. Features that are specific to a domain on the other hand may require the researcher building the algorithm selection system to have a thorough understanding of the domain. These features usually cannot be applied to other domains, as they may be non-existent or uninformative in different contexts.

The second way of distinguishing different classes of features is according to when and how they are computed. Features can be computed *statically*, i.e. before the search process starts, or *dynamically*, i.e. during search. These two categories roughly align with the offline and online approaches to portfolio problem solving described in Section 3.2.

5.1 Low and high-knowledge features

In some cases, researchers use a large number of features that are specific to the particular problem domain they are interested in, but there are also publications that only use a single, general feature – the performance of a particular algorithm on past problem instances. Gagliolo and Schmidhuber (2006), Streeter, Golovin, and Smith (2007), Silverthorn and Miikkulainen (2010), to name but a few examples, use this latter approach to build statistical performance models of the algorithms in their portfolios. The underlying assumption is that all problem instances are similar with respect to the relative performance of the algorithms in the portfolio – the algorithm that has done best in the past has the highest chance of performing best in the future.

Other sources of features that are not specific to a particular application domain are more fine-grained measures of past performance or measures that characterise the behaviour of an algorithm during search. Langley (1983) for example determines whether a search step performed by a particular algorithm is good, i.e. leading towards a solution, or bad, i.e. straying from the path to a solution if the solution is known or revisiting an earlier search state if the solution is not known. Gomes and Selman (1997) use the runtime distributions of algorithms over the size of a problem instance, as measured by the number of backtracks.

Most approaches learn models for the performance on particular problem instances and do not use past performance as a feature, but to inform the prediction to be made. Considering instance features facilitates a much more nuanced approach than a broad-brush general performance model. This is the classic supervised machine learning approach – given the correct label derived from the behaviour on a set of training instances, learn a model that allows to predict this label on unseen data.

The features that are considered to learn the model are specific to the application domain or even a subset of the application domain to varying extents. For combinatorial search problems, the most commonly used basic features include the number of variables, properties of the variable domains, i.e. the list of possible assignments, the number of clauses in SAT, the number of constraints in constraint problems, the number of goals in planning, the number of clauses/constraints/goals of a particular type and ratios of several of those features and summary statistics.

5.2 Static and dynamic features

In most cases, the approaches that use a large number of domain-specific features compute them *offline*, i.e. before the solution process starts (cf. Section 3.2). Examples of publications that only use such static features are Leyton-Brown et al. (2002), Pulina and Tacchella (2007), Guerri and Milano (2004). Examples of such features are the number of clauses and variables in a SAT instance, the number of a particular type of constraint in a constraint problem instance, or the number of operators in a planning problem instance.

An implication of using static features is that the decisions of the algorithm selection system are only informed by the performance of the algorithms on past problem instances. Only dynamic features allow to take the performance on the current problem instance into account. This has the advantage that remedial actions can be taken if the instance is unlike anything seen previously or the predictions are wildly inaccurate for another reason.

A more flexible approach than to rely purely on static features is to incorporate features that can be determined statically, but which estimate the performance on the current problem instance. Such features are computed by probing the search space. This approach relies on the performance probes being sufficiently representative of the entire problem instance and sufficiently equal across the different evaluated algorithms. If an algorithm is evaluated on a part of the search space that is much easier or harder than the rest, a misleading impression of its true performance may result.

Examples of systems that combine static features of the instance to be solved with features derived from probing the search space are Xu et al. (2008), Gent et al. (2010), O'Mahony et al. (2008). There are also approaches that use only probing features. We term this *semi-static* feature computation because it happens before the actual solving of the instance starts, but parts of the search space are explored during feature extraction. This approach is used for example by Allen and Minton (1996), Beck and Freuder (2004), Lobjois and Lemaître (1998). The features that can be extracted through probing include the number of search tree nodes visited, the number of backtracks, or the number and quality of solutions found.

Another way of computing features is to do so *online*, i.e. while search is taking place. These dynamic features are computed by an execution monitor that adapts or changes the algorithm during search based on its performance. The type of the dynamic features, how they are computed and how they are used depends on the specific application. Examples include the following. Carchrae and Beck (2004) monitor the solution quality during search. They decide whether to switch the current algorithm based on this by changing the allocation of resources. Stergiou (2009) monitors propagation events in a constraint solver to decide whether to switch the level of consistency to enforce. Caseau, Laburthe, and Silverstein (1999) evaluate the performance of candidate algorithms in terms of number of calls to a specific high-level procedure. They note that in contrast to using the runtime, their approach is machine-independent. Kadioglu, Malitsky, and Sellmann (2012) base branching decisions in MIP search on features of the sub-problem to solve.

6. Summary

Over the years, there have been many approaches to solving the Algorithm Selection Problem. Especially in Artificial Intelligence and for combinatorial search problems, researchers have recognised that using algorithm selection techniques can provide significant performance improvements with relatively little effort. Most of the time, the approaches involve some kind of machine learning that attempts to learn the relation between problem instances and the performance of algorithms automatically. This is not a surprise, as the relationship between an algorithm and its performance is often complex and hard to describe formally. In many cases, even the designer of an algorithm does not have a general model of its performance.

Despite the theoretical difficulty of algorithm selection, dozens of systems have demonstrated that it can be done in practice with great success. In some sense, this mirrors achievements in other areas of Artificial Intelligence. SAT is formally a problem that cannot be solved efficiently, yet researchers have come up with ways of solving very large instances of satisfiability problems with very few resources. Similarly, some algorithm selection systems have come very close to always choosing the best algorithm.

This survey presented an overview of the algorithm selection research that has been done to date with a focus on combinatorial search problems. A categorisation of the different approaches with respect to fundamental criteria that determine algorithm selection systems in practice was introduced. This categorisation abstracts from many of the low level details and additional considerations that are presented in most publications to give a clear view of the underlying principles. We furthermore gave details of the many different ways that can be used to tackle algorithm selection and the many techniques that have been used to solve it in practice.

This survey can only show a broad and high-level overview of the field. Many approaches and publications are not mentioned at all for reasons of space. A tabular summary of the literature that includes many more publications and is organised according to the criteria introduced here can be found at `http://4c.ucc.ie/~larsko/assurvey/`. The author of this survey hopes to add new publications to this summary as they appear.

6.1 Algorithm selection in practice

Algorithm selection has many application areas and researchers investigating algorithm selection techniques come from many different backgrounds. They tend to publish in venues that are specific to their application domain. This means that they are often not aware of each others' work. Even a cursory examination of the literature shows that many approaches are used by different people in different contexts without referencing the relevant related work. In some cases, the reason is probably that many techniques can be lifted straight from e.g. machine learning and different researchers simply had the same idea at the same time.

Even basic machine learning techniques often work very well in algorithm selection models. If the available algorithms are diverse, it is usually easy to improve on the performance of a single algorithm even with simple approaches. There is no single set of approaches that work best – much depends on the application domain and secondary factors, such as the algorithms that are available and the features that are used to characterise problem instances.

To get started, a simple approach is best. Build a classification model that, given a problem instance, selects the algorithm to run. In doing so, you will gain a better understanding of the relationship between your problem instances, algorithms, and their performance as well as the requirements for your algorithm selection system. A good understanding of the problem is crucial to being able to select the most suitable technique from the literature. Often, more sophisticated approaches come with much higher overheads both in terms of implementation and running them, so a simple approach may already achieve very good overall performance.

Another way to get started is to use one of the systems which are available as open source on the web. Several versions of SATzilla, along with data sets and documentation, can be downloaded from `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`. ISAC is available as MATLAB code at `http://4c.ucc.ie/~ymalitsky/Code/ISAC-Portfolio_v2.zip`. The R package LLAMA (`http://cran.r-project.org/web/packages/llama/index.html`) implements many of the algorithm selection techniques described in this paper through a uniform interface and is suitable for exploring a range of different approaches.[2]

2. Disclaimer: The author of this article is the primary author of LLAMA.

6.2 Future directions

Looking forward, it is desirable for algorithm selection research to become more coordinated. In the past, techniques have been reinvented and approaches reevaluated. This duplication of effort is clearly not beneficial to the field. In addition, the current plethora of different approach is confusing for newcomers. Concentrating on a specific set of techniques and developing these to high performance levels would potentially unify the field and make it easier to apply and deploy algorithm selection systems to new application domains.

Much of algorithm selection research to date has been driven by competitions rather than applications. As a result, many techniques that are known to work well in competition settings are used and systems becoming more and more specialised to competition scenarios. Lacking prominent applications, it remains to be shown which techniques are useful in the real world.

There are many directions left to explore. Using algorithm selection research as a means of gaining an understanding of why particular algorithms perform well in specific scenarios and being able to leverage this knowledge in algorithm development would be a major step. Another fruitful area is the exploitation of today's massively parallel, distributed, and virtualised homogeneous resources.

If algorithm selection techniques were to become widespread in mainstream computer science and software development, a major paradigm shift would occur. It would become beneficial to develop a large number of complementary approaches instead of focusing on a single good one. Furthermore, less time would need to be spent on performance analysis – the algorithm selection system will take care of it.

Acknowledgments

Ian Miguel, Ian Gent and Helmut Simonis provided valuable feedback that helped shape this paper. We also thank the anonymous reviewers whose detailed comments helped to greatly improve it, and Barry O'Sullivan who suggested submitting to AI Magazine. This work was supported by an EPSRC doctoral prize and EU FP7 grant 284715 (ICON).

About the author

Lars Kotthoff is a postdoctoral researcher at Cork Constraint Computation Centre. His PhD work at St Andrews, Scotland, focused on ways of solving the Algorithm Selection Problem in practice.

References

Aha, D. W. (1992). Generalizing from case studies: A case study. In *Proceedings of the 9th International Workshop on Machine Learning*, pp. 1–10, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Allen, J. A., & Minton, S. (1996). Selecting the right heuristic algorithm: Runtime performance predictors. In *The 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 41–53. Springer-Verlag.

Arbelaez, A., Hamadi, Y., & Sebag, M. (2009). Online heuristic selection in constraint programming. In *Symposium on Combinatorial Search*.

Beck, J. C., & Freuder, E. C. (2004). Simple rules for low-knowledge algorithm selection. In *CPAIOR*, pp. 50–64. Springer.

Borrett, J. E., Tsang, E. P. K., & Walsh, N. R. (1996). Adaptive constraint satisfaction: The quickest first principle. In *ECAI*, pp. 160–164.

Brodley, C. E. (1993). Addressing the selective superiority problem: Automatic Algorithm/Model class selection. In *ICML*, pp. 17–24.

Carbonell, J., Etzioni, O., Gil, Y., Joseph, R., Knoblock, C., Minton, S., & Veloso, M. (1991). PRODIGY: an integrated architecture for planning and learning. *SIGART Bull.*, *2*, 51–55.

Carchrae, T., & Beck, J. C. (2004). Low-Knowledge algorithm control. In *AAAI*, pp. 49–54.

Caseau, Y., Laburthe, F., & Silverstein, G. (1999). A Meta-Heuristic factory for vehicle routing problems. In *Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pp. 144–158, London, UK. Springer-Verlag.

Cook, D. J., & Varnell, R. C. (1997). Maximizing the benefits of parallel search using machine learning. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pp. 559–564. AAAI Press.

Demmel, J., Dongarra, J., Eijkhout, V., Fuentes, E., Petitet, A., Vuduc, R., Whaley, R. C., & Yelick, K. (2005). Self-Adapting linear algebra algorithms and software. *Proceedings of the IEEE*, *93*(2), 293–312.

Epstein, S. L., & Freuder, E. C. (2001). Collaborative learning for constraint solving. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pp. 46–60, London, UK. Springer-Verlag.

Fink, E. (1997). Statistical selection among Problem-Solving methods. Tech. rep. CMU-CS-97-101, Carnegie Mellon University.

Fukunaga, A. S. (2002). Automated discovery of composite SAT variable-selection heuristics. In *18th National Conference on Artificial Intelligence*, pp. 641–648, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Fukunaga, A. S. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evol. Comput.*, *16*, 31–61.

Gagliolo, M., & Schmidhuber, J. (2006). Learning dynamic algorithm portfolios. *Ann. Math. Artif. Intell.*, *47*(3-4), 295–328.

Gagliolo, M., & Schmidhuber, J. (2008). Towards distributed algorithm portfolios. In *International Symposium on Distributed Computing and Artificial Intelligence, Advances in Soft Computing.* Springer.

Gent, I., Jefferson, C., Kotthoff, L., Miguel, I., Moore, N., Nightingale, P., & Petrie, K. (2010). Learning when to use lazy learning in constraint solving. In *19th European Conference on Artificial Intelligence*, pp. 873–878.

Gomes, C. P., & Selman, B. (1997). Algorithm portfolio design: Theory vs. practice. In *UAI*, pp. 190–197.

Guerri, A., & Milano, M. (2004). Learning techniques for automatic algorithm portfolio selection. In *ECAI*, pp. 475–479.

Haim, S., & Walsh, T. (2009). Restart strategy selection using machine learning techniques. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, pp. 312–325, Berlin, Heidelberg. Springer-Verlag.

Hong, L., & Page, S. E. (2004). Groups of diverse problem solvers can outperform groups of high-ability problem solvers. *Proceedings of the National Academy of Sciences of the United States of America*, *101*(46), 16385–16389.

Horvitz, E., Ruan, Y., Gomes, C. P., Kautz, H. A., Selman, B., & Chickering, D. M. (2001). A bayesian approach to tackling hard computational problems. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pp. 235–244, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Huberman, B. A., Lukose, R. M., & Hogg, T. (1997). An economics approach to hard computational problems. *Science*, *275*(5296), 51–54.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2011). Algorithm selection and scheduling. In *17th International Conference on Principles and Practice of Constraint Programming*, pp. 454–469.

Kadioglu, S., Malitsky, Y., & Sellmann, M. (2012). Non-model-based search guidance for set partitioning problems. In *AAAI*.

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). ISAC Instance-Specific algorithm configuration. In *19th European Conference on Artificial Intelligence*, pp. 751–756. IOS Press.

Kotthoff, L. (2012). Hybrid regression-classification models for algorithm selection. In *20th European Conference on Artificial Intelligence*, pp. 480–485.

Kotthoff, L., Gent, I. P., & Miguel, I. (2012). An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, *25*(3), 257–270.

Lagoudakis, M. G., & Littman, M. L. (2000). Algorithm selection using reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 511–518, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Langley, P. (1983). Learning search strategies through discrimination.. *International Journal of Man-Machine Studies*, 513–541.

Leyton-Brown, K., Nudelman, E., & Shoham, Y. (2002). Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pp. 556–572, London, UK. Springer-Verlag.

Lobjois, L., & Lemaître, M. (1998). Branch and bound algorithm selection by performance prediction. In *Proceedings of the 15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, pp. 353–358, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2013). Algorithm portfolios based on cost-sensitive hierarchical clustering. In *IJCAI*.

Minton, S. (1996). Automatically configuring constraint satisfaction programs: A case study. *Constraints*, *1*, 7–43.

Nikolić, M., Marić, F., & Janičić, P. (2009). Instance-Based selection of policies for SAT solvers. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pp. 326–340, Berlin, Heidelberg. Springer-Verlag.

O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., & O'Sullivan, B. (2008). Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*.

Petrik, M. (2005). Statistically optimal combination of algorithms. In *Local Proceedings of SOFSEM 2005*.

Pulina, L., & Tacchella, A. (2007). A multi-engine solver for quantified boolean formulas. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, CP'07, pp. 574–589, Berlin, Heidelberg. Springer-Verlag.

Pulina, L., & Tacchella, A. (2009). A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, *14*(1), 80–116.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Roberts, M., & Howe, A. E. (2006). Directing a portfolio with learning. In *AAAI 2006 Workshop on Learning for Search*.

Sakkout, H. E., Wallace, M. G., & Richards, E. B. (1996). An instance of adaptive constraint propagation. In *Proc. of CP96*, pp. 164–178. Springer Verlag.

Samulowitz, H., & Memisevic, R. (2007). Learning to solve QBF. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pp. 255–260. AAAI Press.

Silverthorn, B., & Miikkulainen, R. (2010). Latent class models for algorithm portfolio methods. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence.*

Smith-Miles, K. A. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, *41*, 6:1–6:25.

Stergiou, K. (2009). Heuristics for dynamically adapting propagation in constraint satisfaction problems. *AI Commun.*, *22*(3), 125–141.

Streeter, M. J., Golovin, D., & Smith, S. F. (2007). Combining multiple heuristics online. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pp. 1197–1203. AAAI Press.

Vrakas, D., Tsoumakas, G., Bassiliades, N., & Vlahavas, I. (2003). Learning rules for adaptive planning. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, pp. 82–91.

Weerawarana, S., Houstis, E. N., Rice, J. R., Joshi, A., & Houstis, C. E. (1996). PYTHIA: a knowledge-based system to select scientific algorithms. *ACM Trans. Math. Softw.*, *22*(4), 447–468.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.

Xu, L., Hoos, H. H., & Leyton-Brown, K. (2007). Hierarchical hardness models for SAT. In *CP*, pp. 696–711.

Xu, L., Hoos, H. H., & Leyton-Brown, K. (2010). Hydra: Automatically configuring algorithms for Portfolio-Based selection. In *24th 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. *J. Artif. Intell. Res. (JAIR)*, *32*, 565–606.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2009). SATzilla2009: an automatic algorithm portfolio for SAT. In *2009 SAT Competition*.

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)*.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2012). Evaluating component solver contributions to Portfolio-Based algorithm selectors. In *International Conference on Theory and Applications of Satisfiability Testing (SAT'12)*, pp. 228–241.

Yun, X., & Epstein, S. L. (2012). Learning algorithm portfolios for parallel execution. In *Proceedings of the 6th International Conference Learning and Intelligent Optimisation LION*, pp. 323–338. Springer.