Automated Algorithm Selection: Survey and Perspectives

Pascal Kerschke kerschke@uni-muenster.de
Information Systems and Statistics, University of Münster, 48149 Münster, Germany

Holger H. Hoos hh@liacs.nl
Leiden Institute of Advanced Computer Science, Leiden University, 2333 CA Leiden, The Netherlands

Frank Neumann frank.neumann@adelaide.edu.au
Optimisation and Logistics, The University of Adelaide, Adelaide, SA 5005, Australia

Heike Trautmann trautmann@uni-muenster.de
Information Systems and Statistics, University of Münster, 48149 Münster, Germany

https://doi.org/10.1162/evco_a_00242

Abstract

It has long been observed that for practically any computational problem that has been intensely studied, different instances are best solved using different algorithms. This is particularly pronounced for computationally hard problems, where in most cases, no single algorithm defines the state of the art; instead, there is a set of algorithms with complementary strengths. This performance complementarity can be exploited in various ways, one of which is based on the idea of selecting, from a set of given algorithms, for each problem instance to be solved the one expected to perform best. The task of automatically selecting an algorithm from a given set is known as the *per-instance algorithm selection problem* and has been intensely studied over the past 15 years, leading to major improvements in the state of the art in solving a growing number of discrete combinatorial problems, including propositional satisfiability and AI planning. Per-instance algorithm selection also shows much promise for boosting performance in solving continuous and mixed discrete/continuous optimisation problems. This survey provides an overview of research in automated algorithm selection, ranging from early and seminal works to recent and promising application areas. Different from earlier work, it covers applications to discrete and continuous problems, and discusses algorithm selection in context with conceptually related approaches, such as algorithm configuration, scheduling, or portfolio selection. Since informative and cheaply computable problem instance features provide the basis for effective per-instance algorithm selection systems, we also provide an overview of such features for discrete and continuous problems. Finally, we provide perspectives on future work in the area and discuss a number of open research challenges.

Keywords

Automated algorithm selection, automated algorithm configuration, combinatorial optimisation, continuous optimisation, machine learning, metalearning, feature-based approaches, exploratory landscape analysis, data streams.

1 Introduction

It has long been observed that for well-studied computational problems for which several high-performance algorithms are available, there is typically no single algorithm

Manuscript received: 17 October 2018; revised: 13 November 2018 and 18 November 2018; accepted: 19 November 2018.

that dominates all others on all problem instances. Instead, different algorithms perform best on different types of problem instances—a phenomenon also known as *performance complementarity*, which is often incorrectly attributed to an interesting theoretical result known as the *no-free-lunch (NFL) theorem* (Wolpert and Macready, 1995, 1997). In the context of search problems, the NFL theorem strictly applies only if arbitrary search landscapes are considered, while the instances of basically any search problem of interest have compact descriptions and therefore cannot give rise to arbitrary search landscapes (Culberson, 1998). Performance complementarity has been observed for practically all NP-hard decision and optimisation problems; these include propositional satisfiability, constraint satisfaction, a wide range of planning and scheduling problems, mixed integer programming, and the travelling salesperson problem, as well as a broad range of continuous optimisation, machine learning and important polynomial-time-solvable problems (e.g., sorting and shortest path finding). For these and many other problems, theoretical results stating which algorithmic strategies work best are restricted to very limited classes of problem instances, so that it is generally unknown a priori which of several algorithms should be used to solve a given instance.

This gives rise to the increasingly prominent *per-instance algorithm selection problem*: given a computational problem, a set of algorithms for solving this problem, and a specific instance that needs to be solved, determine which of the algorithms can be expected to perform best on that instance. This problem has already been considered in the seminal work by Rice (1976), but it took several decades before practical per-instance algorithm selection methods became available (see, e.g., Cook and Varnell, 1997; Leyton-Brown et al., 2003; Xu et al., 2008). Since then, the problem and algorithms for solving it have steadily gained prominence, and by now have given rise to a large body of literature. Indeed, per-instance algorithm selection techniques have produced substantial improvements in the state of the art in solving a large range of prominent computational problems, including propositional satisfiability (SAT) and the travelling salesperson problems (TSP) (Xu et al., 2008, 2012; Kerschke et al., 2017).

It is important to note that there are several concepts that are quite closely related to that of per-instance algorithm selection, notably, per-set algorithm selection, algorithm configuration, algorithm schedules, and parallel algorithm portfolios, which are all discussed in further detail in Section 2. Unfortunately, there is some potential for confusion, especially between per-instance algorithm selection and parallel algorithm portfolios, since in the literature, the term *portfolio* is sometimes used to refer to algorithm selectors. Furthermore, some of the most prominent and successful algorithm selection approaches from the literature, such as SATzilla (Xu et al., 2008) and AutoFolio (Lindauer, Hoos, Hutter, and Schaub, 2015), implement combinations of algorithm scheduling and per-instance selection. While we will briefly discuss these more complex systems, along with approaches that select more than one algorithm to be run on a given problem instance, the focus of this survey is on pure per-instance algorithm selection, as outlined above and defined formally in Section 2.

We note that per-instance algorithm selection can be applied to optimisation problems, where the goal is to find an optimal (or best possible) solution according to a given objective function, as well as to decision problems, where one wants to determine, as quickly as possible, whether a solution satisfying certain conditions exists. Furthermore, it is useful to distinguish between continuous problems, where the components of a possible solution are real numbers (possibly constrained to a given interval), and discrete problems, where candidate solutions are discrete objects, such as graphs, permutations, or vectors of integers.

Several surveys on algorithm selection have been published over the last decade. In the first extensive survey in this area, Smith-Miles (2009) summarised developments in the metalearning, artificial intelligence, and operations research communities. Adopting a cross-disciplinary perspective, she combined contributions from these areas under the umbrella of a "(meta)learning" framework, which permitted her to identify parallel and closely related developments within these rather well-separated communities. However, this survey was published a decade ago and therefore does not cover recent developments and improvements to the state of the art in this fast-moving research area.

A more recent overview on algorithm selection was published by Kotthoff (2014). His survey presents an extensive, valuable guide to the automated algorithm selection literature up to 2014 and provides answers to several important questions, such as (i) what are the differences between static and dynamic portfolios, (ii) what should be selected (single solver, schedule, different candidate portfolios), (iii) what are the differences between online and offline selection, (iv) how should the costs for using algorithm portfolios be considered, (v) which prediction type (classification, regression, etc.) is most promising when training an algorithm selector, and (vi) what are differences between static and dynamic, as well as low-level and high-level features. Unfortunately, Kotthoff's survey is restricted to algorithm selection for discrete problems and does not cover in any detail problem instance features, which provide the basis for per-instance algorithm selection.

Those two limitations where—at least partially—addressed by Muñoz Acosta et al. (2013). Although the title ("The Algorithm Selection Problem on the Continuous Optimization Domain") appears to suggest otherwise, their survey mostly addresses the paucity of work on algorithm selection for continuous optimisation problems and the challenges arising in this context. Rather than providing an overview of algorithm selection approaches in this area, Muñoz Acosta et al. (2013) summarise promising results on discrete problems and hint at the possibility of achieving similar results in continuous optimisation. In their follow-up survey, Muñoz Acosta, Sun et al. (2015) provide further insights into the existing ingredients for algorithm selection in the domain of continuous optimisation: benchmarks, algorithms, performance metrics, and problem characteristics obtained by exploratory landscape analysis. Still, they do not cover any work describing automated algorithm selection in this domain.

Our goal here is to not only update, but also to complement and extend these previous surveys. Firstly, we cover work on algorithm selection for discrete *and* continuous problems; as a result, we can compare the difficulties, challenges, and solutions found in those domains. Secondly, one of the most important ingredients for successful algorithm selection approaches are informative (problem-specific) features. We therefore provide an overview of several promising feature sets and discuss characteristics that have been demonstrated to provide a strong basis for algorithm selection. Thirdly, we discuss several problems closely related to (and sometimes confused with) algorithm selection, such as automated algorithm configuration, algorithm schedules, and parallel portfolios, pointing out differences, similarities, and synergies. Of course, in light of the considerable and fast-growing body of literature on and related to algorithm selection, we cannot provide comprehensive coverage; instead, we selected contributions based on their impact, promise, and conceptual contributions to the area.

The remainder of this survey article is structured as follows. In Section 2, we formally define the per-instance algorithm selection problem and situate it in the context of related problems, such as automated algorithm configuration. Next, Section 3 provides an overview of instance features for discrete and continuous optimisation problems

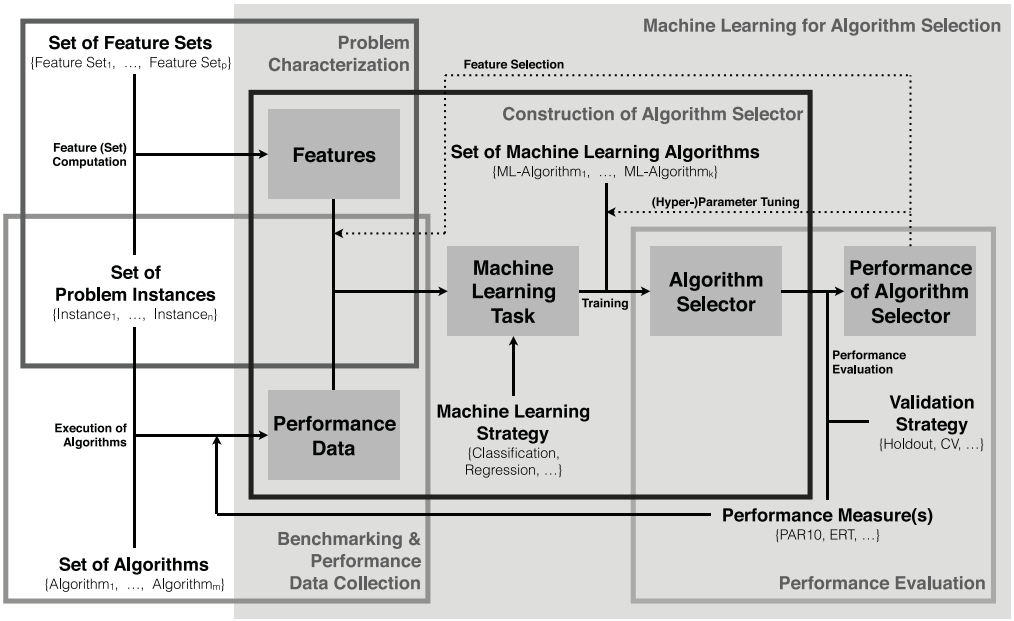

Figure 1: Schematic overview of interplay between problem instance features (top left), algorithm performance data (bottom left), selector construction (center) and the assessment of selector performance (bottom right).

that provide the basis for automated algorithm selection. Successful applications of algorithm selection in discrete and continuous optimisation are discussed in Sections 4 and 5, respectively. Finally, Section 6 provides additional perspectives on algorithm configuration and outlines several open challenges.

2 Algorithm Selection and Related Problems

We consider the selection of algorithms for a given decision or optimisation problem P. Specifically, the per-instance algorithm selection problem can be formulated as follows (see also Rice, 1976): Given a set I of instances of a problem P, a set $\mathbf{A} = \{A_1, \ldots A_n\}$ of algorithms for P and a metric $m : \mathbf{A} \times I \to \mathbb{R}$ that measures the performance of any algorithm $A_j \in \mathbf{A}$ on instance set I, construct a selector S that maps any problem instance $i \in I$ to an algorithm $S(i) \in \mathbf{A}$ such that the overall performance of S on I is optimal according to metric m.

Of course, in general, we cannot hope to efficiently find perfect solutions to the per-instance algorithm selection problem, and instead, we strive to find selectors whose performance is as close as possible to that of a perfect selector on instance set I. This is typically achieved by making use of informative and cheaply computable features $\mathbf{f}(i) = (f_1(i), \ldots, f_k(i))$ of the given problem instance i. A general overview on the interplay of instance features, algorithm performance data, and algorithm selection is shown in Figure 1. The features are of key importance, and we will discuss them in more detail in the following section of this article.

We note that the performance of a (hypothetical) perfect per-instance algorithm selector, often also referred to as an *oracle selector* or *virtual best solver (VBS)*, provides

a lower bound on the performance of any realistically achievable algorithm selector (where we assume, w.l.o.g., that the given performance measure is to be minimised), and is often used in the context of assessing selector performance.

Another useful concept is that of the *single best solver (SBS)*, which is the algorithm A' with the best performance among all the $A_j \in \mathbf{A}$. The SBS is the solution to the closely related *per-set algorithm selection problem*, and its performance provides a natural upper bound on the performance of any reasonable per-instance algorithm selector. Furthermore, the difference or ratio between the performance of the SBS and VBS, also known as the *VBS-SBS gap*, gives an indication of the performance gains that can be realised, in the best case, by per-instance algorithm selection, and the fraction of the VBS-SBS gap closed by any per-instance algorithm selector S provides a measure of its performance (see, e.g., Lindauer et al., 2017b). State-of-the-art per-instance algorithm selectors for combinatorial problems have demonstrated to close between 25% and 96% of the VBS-SBS gap (see, e.g., Lindauer, van Rijn et al., 2015b). It is important to note that the VBS-SBS gap is large when the given set $\mathbf{A}$ of algorithms shows high performance complementarity on instance set I, i.e., when different $A_j \in \mathbf{A}$ perform best on different $i \in I$, and those algorithms that are best on some instances perform quite poorly on others. Generally, per-instance algorithm selection can be expected to achieve large performance gains over the single best algorithm if there is high performance complementarity within $\mathbf{A}$ and there is a set of sufficiently cheaply computable and informative instance features that can be leveraged in learning a good mapping from instances to algorithms.

It is very important to distinguish between per-set algorithm selection and per-instance algorithm selection. The former does not require any instance features and is typically done by exhaustive evaluation of all given algorithms on a set of problem instances deemed to be representative for those to be solved later. The connection between per-instance algorithm selection and related problems is shown in Figure 2. In many ways, algorithm competitions, such as the international SAT and planning competitions (see, e.g., Järvisalo et al., 2012; Vallati et al., 2015), can be seen as identifying solutions to per-set algorithm selection problems for broad sets of interesting instances, and competition winners are often seen as the single best algorithm for the respective problem. Sometimes, to reduce the computational cost for per-set algorithm selection, racing methods are used. These run candidate algorithms on an increasing number of instances, eliminating those from consideration whose performance is significantly below that of others, based on a statistical test (see, e.g., Maron and Moore, 1994; Birattari et al., 2002).

Per-set algorithm configuration can be seen as a special case of *algorithm configuration*, a practically very important problem that can be described as follows: Given an algorithm A whose performance (but not semantics) is affected by the settings of parameters $\mathbf{p} = (p_1, \ldots, p_k)$, a set C of possible values for $\mathbf{p}$ (called *configurations* of A), a set of problem instances I and a performance metric m, find a configuration $c^* \in C$ of A that achieves optimal performance on I according to m. Note that the set C of configurations can be seen as corresponding to a set of algorithms, of which we wish to select the one that performs best. The key difference to algorithm selection is that this set can be very large, since it arises from combinatorial combinations of values of the individual parameters p_l, which, in some cases, can take continuous values, leading to (potentially) uncountably infinite sets of algorithm configurations over which we have to optimise. Realistic algorithm configuration scenarios typically involve tens to hundreds of parameters (see, e.g., Hutter et al., 2009, 2011; López-Ibáñez et al., 2016;

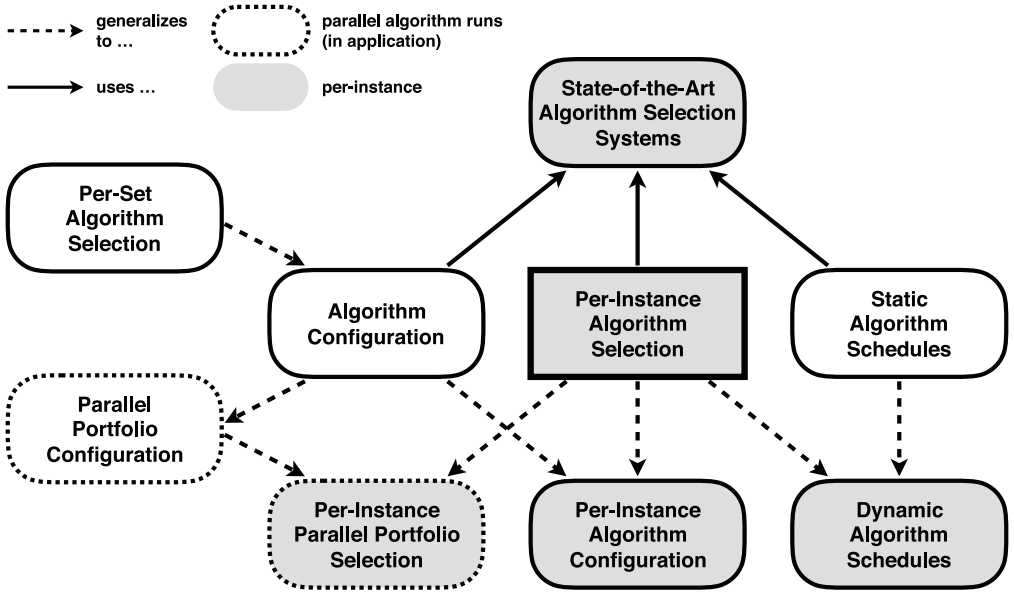

Figure 2: Connections between per-instance algorithm selection and related problems.

Ansótegui et al., 2015; Thornton et al., 2013; Kotthoff et al., 2017). Therefore, per-set algorithm selection techniques are typically not directly applicable to algorithm configuration, although racing techniques can be extended to work well in this case (see, e.g., López-Ibáñez et al., 2016; Pérez Cáceres et al., 2017). Per-set algorithm configuration is closely related to hyperparameter optimisation in machine learning; the main difference is that in algorithm configuration, performance is to be optimised on a possibly diverse set of problem instances, which often requires trading off performance on some instance against that achieved on others. In a sense, the typical hyperparameter optimisation problem encountered in machine learning is analogous to configuring a parameterised algorithm for performance on a single problem instance.

Since typical procedures used for building per-instance algorithm selectors have design choices that can be exposed as parameters, algorithm configuration techniques can be applied to optimise their performance on specific (sets of) selection scenarios. This has been done, with considerable success, in the recent AutoFolio selection system by Lindauer, Hoos, Hutter, and Schaub (2015), which we will discuss in further detail in Section 4.

The per-instance variant of the algorithm configuration problem, which can be seen as a generalisation of per-instance algorithm selection, largely remains an open challenge (see, e.g., Hutter et al., 2006; Belkhir et al., 2016, 2017), and we briefly discuss it further in Section 6.

Performance complementarity within a set of algorithms can be leveraged in ways that differ from per-instance algorithm configuration. One prominent approach is that of a *parallel algorithm portfolio*, where each algorithm from a given set $\mathbf{A}$ is run in parallel on a given problem instance i (see, e.g., Huberman et al., 1997; Gomes and Selman, 2001; Fukunaga, 2000). When applied to a decision problem, all runs are terminated as soon as one of the component algorithms has solved the given instance i; for optimisation problems, the best solution achieved by any of the component algorithms at any given time

is returned as the solution of the entire portfolio. Parallel algorithm portfolios are conceptually similar to ensemble methods in machine learning (see, e.g., Dietterich, 2000; Rokach, 2010). The key difference is that ensemble methods aggregate the results from the various component algorithms, for example, by weighted or unweighted averaging.

When run on parallel hardware, algorithm portfolios typically achieve performance very close to that of the VBS in terms of wall-clock time, at the price of parallelism of degree equal to the number n of algorithms in $\mathbf{A}$. Of course, parallel portfolios can be run at lower actual degrees of parallelism, and even fully sequentially, using task-switching, as provided, for example, by the operating system; in that case, the wall-clock time is typically close to that of n times the performance of the VBS in terms of the time required to solve a given instance of a decision problem, or to achieve a certain solution quality in case of an optimisation problem. Most of this overhead, which for large sets of algorithms can be very substantial, can be avoided by using per-instance algorithm selectors instead of parallel portfolios. Nevertheless, especially in the area of evolutionary computation, the concept of parallel algorithm portfolios has given rise to a growing body of research, in which the basic concept is often combined with additional techniques to achieve improved performance (see, e.g., Tang et al., 2014; Yuen and Zhang, 2015).

Although the term *algorithm portfolio* is sometimes used in the literature to refer to per-instance algorithm selectors and other techniques that leverage performance complementarity within a set of algorithms, we discourage this broad use as it easily leads to confusion between conceptually very different approaches. This potential confusion is easily avoided by restricting the term *algorithm portfolio* to parallel algorithm portfolios, consistent with the seminal work by Huberman et al. (1997). Recently, the concepts of per-instance algorithm selection and parallel portfolios have been combined, by selecting, on a per-instance basis, several algorithms to be run in parallel (Lindauer, Hoos, and Hutter, 2015). While this would not make sense in the context of a perfect algorithm selector, it can limit the impact of the poor selection decisions sometimes made in practice.

Algorithm schedules provide another way of exploiting performance complementarity (see, e.g., Lindauer, 2014; Lindauer et al., 2016). The key idea is to run a sequence of algorithms from a given set $\mathbf{A}$, one after the other, each for a given (maximum) time. Those cut-off times can differ between the stages of the schedule, and some algorithms may not be run at all. Static algorithm schedules, i.e., schedules that have been determined in a per-set fashion and are applied uniformly to any given problem instance i, can be quite effective and are typically much easier to implement than per-instance algorithm selectors (see, e.g., Roussel, 2012). They are also used in state-of-the-art algorithm selection systems during a so-called pre-solving phase, in order to solve easy problem instances quickly and without the need for computing the instance features required for per-instance selection (see, e.g., Xu et al., 2012; Lindauer, Hoos, Hutter, and Schaub, 2015). Unless stated explicitly otherwise, we will in the following, when discussing per-instance algorithm selectors, always refer to the pure per-instance algorithm selection problem, as defined above, without pre-solving schedules and other extensions found in cutting-edge algorithm selection systems.

Finally, the problem of predicting the performance of an algorithm A on a given problem instance i is closely related to per-instance algorithm selection. If computationally cheap and accurate performance prediction were possible, evidently, we could use performance predictors for our given algorithms $A_1, \ldots, A_n$ and simply select the one predicted to perform best on i. In practice, sufficient accuracy can be achieved for many problems using state-of-the-art regression techniques from machine learning, at

moderate computational cost (Hutter, Hoos et al., 2014), and performance predictors form the basis for one of the main approaches to per-instance algorithm selection. At the same time, other approaches, such as cost-based classification, exist and also find use in state-of-the-art algorithm selection systems, as explained in the following sections.

3 Features for Discrete and Continuous Problems

Linking algorithm performance on an instance i to instance characteristics forms a central part of automated algorithm selection and several related problems. For this purpose, automatically computable features $\mathbf{f}(i) = (f_1(i), \ldots, f_k(i))$ are required, ideally with the following properties: Firstly, features should be *informative*, in that they allow for a sufficient distinction between different instances; they should also be *interpretable*, so that feature values enable an expert to gain maximum insight into instance properties. Furthermore, features should be *cheaply computable*, so that the advantages gained by selecting an algorithm based on them is not outweighed by the cost of feature computation. Features should also be *generally applicable*; that is, they should be effectively and efficiently computable for a broad range of problem instances, rather than being restricted, for example, to small instance sizes. Finally, the features f_j should be *complementary*, in that redundant sets of features are not only computationally wasteful, but can also cause problems when used by certain machine learning algorithms as a basis for algorithm selection and related problems.

In the following, we provide an overview of commonly used instance features for several prominent discrete and continuous problems—not only to illustrate what kind of features are useful in the context of per-instance algorithm selection, but also to draw attention to an important and somewhat underrated research topic of significant importance to tasks beyond algorithm selection. In particular, informative instance features can provide important insights into strengths and weaknesses of a given algorithm, and hence play a crucial role in devising improvements. We generally distinguish between problem-specific features that are closely based on particular aspects of the problem to be solved, such as the number of clauses in instances of propositional satisfiability problems, and generic features that are more broadly applicable, such as high-level statistics over information gleaned from short "probing" runs of a solver for the given problem.

3.1 Discrete Problems

To give concrete examples, and in light of the importance of problem-specific features, we will focus on three of the most prominent and well-studied discrete combinatorial problems: propositional satisfiability (and related problems), AI planning, and the travelling salesperson problem (TSP).

Propositional satisfiability and related problems. The *propositional satisfiability problem (SAT)* is to determine whether for a given formula F in propositional logic, containing Boolean variables $X_1, \ldots, X_N \in \{true, false\}$, there exists an assignment of logical values to the variables such that F evaluates to $true$; such a variable assignment is said to satisfy F. Typically, the problem is restricted to formulae F in *conjunctive normal form (CNF)*, i.e., F consists of conjunctions ($\wedge$) of so-called clauses, which are disjunctions ($\vee$) of Boolean variables X_j and their negations $\neg X_j$. A CNF-formula F evaluates to $true$, if each of its clauses is satisfied simultaneously. SAT is one of the most prominent and intensely studied combinatorial decision problems and has important applications in hard- and software verification (see, e.g., Biere et al., 2009). Given the ties to other combinatorial problems, improvements in SAT often also impact widely studied related

problems, such as *maximum satisfiability (MaxSAT) problem*, in which the objective is to find a variable assignment that maximises the number of satisfied CNF clauses.

The first large collection of features for SAT (and thus also MaxSAT) instances was provided by Nudelman, Leyton-Brown, Hoos et al. (2004). Despite the rather simple structure of SAT instances, the authors devised nine different feature sets and a total of 91 features, which characterise a given CNF formula from a multitude of perspectives. Eleven *problem size features* describe SAT instances based on summary statistics of their numbers of clauses and variables. A set of *variable-clause graph (VCG) features* comprises ten node degree statistics based on a bipartite graph over the variables and clauses of a given instance. Interactions between the variables are captured by four *variable graph (VG) features*; these are the minimum, maximum, mean, and coefficient of variation of the node degrees for a graph of variables, in which edges connect pairs of variables that jointly occur in at least one clause. Similarly, the set of *clause graph (CG) features* contains seven node degree statistics of a graph whose edges connect clauses that have at least one variable in common, as well as three features based on weighted clustering coefficients for the clause graph. Thirteen *balance features* capture the balance between negated and unnegated variables per clause, their overall occurrences across all clauses, as well as fractions of unary, binary, and ternary clauses, whereas six further features quantify the degree to which the given F resembles a Horn formula (a restricted type of CNF formula, for which SAT can be decided efficiently). The solution of a linear program representing the given SAT instance provides the basis for six *LP-based features*. Finally, there are two sets of so-called *probing features*, which are based on performance statistics over short runs of several well-known SAT algorithms (based on DPLL and stochastic local search, two prominent approaches to solving SAT) and capture the degree to which these make early progress on the given instance.

Some of the feature sets—specifically, the CG, VG, and LP-based features, as well as some of the VCG, balance and DPLL-probing features—are computationally quite expensive (see, e.g., Xu et al., 2008; Hutter, Hoos et al., 2014) and consequently not always useful in the context of practical algorithm selection approaches. Similarly, the algorithm runs for probing features are limited to a very small part of the overall time budget for solving a given instance, to make sure that sufficient time remains available for running the selected SAT solver.

A decade later, Hutter, Hoos et al. (2014)—building on the work by Nudelman, Leyton-Brown, Hoos et al. (2004)—introduced a set of 138 SAT features. While they removed some features from the earlier sets, much of the set remained the same. The most significant changes were an extension of the CG and VG feature sets by five new features each, as well as three new feature sets accounting for an additional 48 features. The VG feature set was extended by so-called *diameter* features, which capture statistics based on the set of longest shortest paths from one variable to any other one in the graph. Also, instead of the weighted clustering coefficients based on the CG (as done by Nudelman, Leyton-Brown, Hoos et al., 2004), Hutter, Hoos et al. (2014) used a set of *clustering coefficients* that measure the CG's "local cliqueness." Furthermore, they introduced 18 novel *clause learning* features, which summarise information gathered during short runs of a prominent SAT solver, zchaff_rand, that learns conflict clauses during its search for a satisfying assignment (Mahajan et al., 2004). Another 18 features are derived from estimates of variable bias obtained from the SAT solver VARSAT (Hsu and McIlraith, 2009); these features essentially capture statistics over estimates for the probability for variables to be *true*, *false*, or *unconstrained* in every satisfying assignment. Finally, Hutter, Hoos et al. (2014) proposed to use the actual feature costs,

in terms of the running time required for computing each of the 11 feature sets; they noted that the diameter and survey propagation features tend to be expensive to compute and may thus be of limited usefulness in the context of per-instance algorithm selection.

A well-known generalisation of SAT is the problem of *answer set programming* (ASP; see, e.g., Baral, 2003), which deals with determining so-called "answer sets," that is, stable models for logic programs. Many combinatorial problems can be presented in ASP in a rather straightforward way and solved, at least in principle, using general-purpose ASP solvers. Because of the close relationship between ASP and SAT, many features for ASP instances are closely related to the SAT features outlined above. One of the most widely used collection of ASP features has been proposed by Maratea et al. (2012); it is comprised of 52 features, which can be grouped into four sets. Three of these feature sets closely correspond to well-known SAT features (Nudelman, Leyton-Brown, Hoos et al., 2004) and contain eight *problem size*, three *balance* and two *proximity to Horn* features. In addition, Maratea et al. (2012) proposed 39 ASP-specific features, such as the numbers of true and disjunctive facts, the fraction of normal rules and constraints, and several combinations of the latter.

AI planning. *Automated planning* (also known as *AI planning*) is one of the most prominent challenges in artificial intelligence (see, e.g., Ghallab et al., 2004). While there are many variants of AI planning problems, the basic setting (also known as classical planning) involves a set of actions with associated pre-conditions, deterministic effects, and sometimes costs, an initial state and one or more goal states. The objective in satisficing planning is to find a valid plan, that is, a sequence or partially ordered set of actions that, when applied to the initial state, reach a goal state, or to determine that no valid plan exists. In the optimisation variants of planning problems, the objective is to find plans of minimal length or cost. Most variants of AI planning are at least NP-hard, and satisficing classical planning is known to be PSPACE-complete. AI planning algorithms have important applications, for example, in robotics, gaming, logistics, and software test case generation; they are also used for the operation and management of traffic, energy grids, and fleets of shared vehicles.

In classical planning, there is an important distinction between a problem instance and a so-called *planning domain*. This distinction arises from the fact that states and actions are specified in an abstract way, using so-called *predicates* and *operators* that can be instantiated to yield specific properties of *states* and specific *actions*, respectively (see, e.g., Ghallab et al., 2004). For example, in a planning problem that involves moving goods using a fleet of trucks, there might be a predicate stating that a specific truck is in a given location, and an operator that moves the truck from one location to another. A planning domain is a class of a planning instances with the same set of specific predicates and operators. Planning domains and instances can be concisely described in a widely used, uniform language called PDDL (Planning Domain Definition Language; see, e.g., Gerevini and Long, 2005).

Howe et al. (1999) were among the first to characterise AI planning instances by simple features, namely, the number of actions, predicates, objects, goals, as well as the number of predicates used to specify the initial state. A decade later, Roberts et al. (2008) introduced a substantially extended set of 41 features, which includes summary statistics of the domain and instance files (16 and three features, respectively), but also captures 13 high-level features of the given planning instance in terms of its PDDL requirements. They also considered nine features based on the so-called *causal graph (CG)*

(i.e., a graph capturing causal dependencies between states), such as the number of vertices in the CG and their average degree, as well as various metrics computed from the edges of the CG.

Also based on the idea of using graph properties, Cenamor et al. (2013) proposed a total of 47 features, which capture the information contained in causal and *domain transition graphs*. The latter represent the permissible transitions between states. The causal graph features of Cenamor et al. (2013) can be categorised into four different sets: (i) four general graph properties, (ii) four features based on various ratios of graph properties, (iii) 12 statistical aggregations over the entire graph, and (iv) six additional, high-level statistics for states with defined values in the goal specifications. The remaining 21 domain transition graph features include (i) three general graph properties (number of edges and states, sum of edge weights) and (ii) 18 statistical features similar to those of the causal graph.

The most recent and extensive collection of AI planning instance features was prepared by Fawcett et al. (2014). It contains 12 sets with 311 features in total, covering most of the features from earlier work, as well as a broad range of new ones. The first three sets extend the 16 domain, three problem and 13 language requirement features from Roberts et al. (2008) by two, four, and eleven new features, respectively. Four further feature sets are based on a translation of the given PDDL instance into a *finite domain representation* (FDR), by means of a well-known AI planning system, Fast Downward (Helmert, 2006). This FDR representation, as well as information collected during the translation and preprocessing, gives rise to sets of 19, 19 and eight features, respectively. Building on the work of Cenamor et al. (2013), Fawcett et al. (2014) also provide a set of 41 causal and domain transition graph features. Six features are computed from information gathered during the preprocessing phases of LPG-td (Gerevini et al., 2003), another well-known planning system, while 10 further features capture information produced by the Torchlight local search analysis tool (Hoffmann, 2011); another 16 features are determined based on the trajectories of one-second probing runs of Fast Downward. Furthermore, the 115 SAT features from Xu et al. (2012) are included, based on a SAT representation of the given planning instance (in form of a CNF with a planning horizon of 10). The final feature set introduced by Fawcett et al. (2014) contains information on whether the previously outlined sets were computed successfully and additionally captures the respective computation times.

Travelling salesperson problem. The *travelling salesperson problem (TSP)* is one of the most intensely studied combinatorial optimisation problems. For decades, it has been the subject of a large body of work and continues to be highly relevant for theoretical analyses, design of algorithms and practical applications ranging from logistics to manufacturing (see, e.g., Applegate et al., 2007). In the TSP, given an edge-weighted graph, whose vertices are often called *cities* and whose edges represent the cost of travelling from one city to another, the objective is to find a Hamiltonian cycle with minimum total weight, that is, a minimum-cost trip that passes through every city exactly once. Most work on the TSP focusses on the special case of the two-dimensional Euclidean TSP, where cities are locations in the Euclidean plane, and costs correspond to the Euclidean distances between cities.

The development of features for TSP instances has been initiated by Smith-Miles and van Hemert (2011), who proposed the following features for characterising a given TSP instance: (1) coordinates of the instance's centroid, (2) average distance from all cities to the centroid, (3 & 4) standard deviation of distances, as well as fraction of

distinct distances within distance matrix, (5) size of the rectangle enclosing the instance's cities, (6 & 7) standard deviation, as well as coefficient of variation of the normalised nearest neighbour distances, (8) ratio of number of clusters found by GDBSCAN (Sander et al., 1998) to the number of all cities, (9) variance of number of cities per cluster, (10) ratio between number of outliers and all cities, and (11) the number of cities. Furthermore, Kanda et al. (2011) and Kovárik and Málek (2012) proposed features derived from the distance matrix of a given TSP instance.

Nearly all features from these earlier studies were combined by Mersmann et al. (2013) and further extended, leading to a collection of six TSP feature sets with a total of 68 features, many of which are derived from the distance matrix and from the spatial distribution of the cities in the Euclidean plane. More precisely, the distance matrix is condensed into *distance* and *mode features*, and the distribution of cities is captured by a set of *cluster features*, based on multiple runs of GDBSCAN, as well as *convex hull features*, which quantify the spread of the cities in the Euclidean plane. The closeness of neighbouring cities is measured by various *nearest neighbour statistics*, and a final set of features is comprised of the depth and edge costs of the *minimum spanning tree* for the given TSP instance.

Hutter, Hoos et al. (2014) developed a set of 64 TSP features that includes some of the previously outlined instance characteristics as well as new *probing features*. The latter are based on 20 short runs of a well-known local search solver (LK; Lin and Kernighan, 1973), as well as single short runs of the state-of-the-art exact TSP solver, Concorde (Applegate et al., 2007). Probing features were also used by Kotthoff et al. (2015).

The most comprehensive collection of TSP features was provided by Pihera and Musliu (2014). Their set of 287 features builds on the earlier work by Hutter, Hoos et al. (2014), but additionally includes instance characteristics derived from the distances of the cities to the convex hull, the number of intersections of locally optimal tours, and statistics of disjoint tour segments. The largest group of new features is based on strongly and weakly connected components of so-called *k-nearest neighbour graphs*, for many values of k.

Finally, we see significant potential for new features based on recent work on funnel-structures in the search landscapes associated with TSP instances (Ochoa et al., 2015; Ochoa and Veerapen, 2016). Considering that highly related aspects of global search space structure have shown to play an important role in algorithm selection for continuous optimisation problems (Bischl et al., 2012; Kerschke and Trautmann, 2018), features of this nature may also prove to be useful for discrete optimisation problems, such as the TSP.

Other combinatorial problems. There is a sizeable body of work on instance characteristics for other combinatorial problems, including the epistasis measures by Davidor (1991) and Fonlupt et al. (1998), indicators for the hardness of quadratic assignment (QAP, Angel and Zissimopoulos, 2002) or constraint satisfaction problems (CSP, Boukeas et al., 2004), features for so-called orienteering problems, which generalise the TSP by distinguishing between static and dynamic locations (Bossek et al., 2018), and variable interaction measures for combinatorial optimisation problems (Seo and Moon, 2007). A detailed discussion of these problems and the respective instance features (some of which are quite generic and can be applied to a range of discrete combinatorial problems) is beyond the scope of this survey; however, we note that these features, like the problem-specific features described earlier in this section, provide a good basis for per-instance algorithm selection and related tasks.

3.2 Continuous Problems

We now turn our attention to the optimisation of continuous *fitness landscapes* (Wright, 1932; Kauffman, 1993). In contrast to discrete optimisation problems, which differ very substantially from each other (consider, for example, SAT vs. TSP) and require problem-specific features for characterising instances, the general idea of continuous optimisation problems can be expressed uniformly, in a rather straightforward way: (w.l.o.g.) find the global minimum of an objective or fitness function $f : \mathcal{X} \to \mathcal{Y}$, which maps vectors of variables, $\mathbf{x} = (x_1, \ldots, x_d)$, from a d-dimensional decision space $\mathcal{X} \subseteq \mathbb{R}^d$ (whose values may be subject to additional constraints) to p-dimensional objective or fitness values $\mathbf{y} = (y_1, \ldots, y_p) := f(\mathbf{x}) \in \mathcal{Y} \subseteq \mathbb{R}^p$ (Jones, 1995; Stadler, 2002). Unfortunately, in most real-world scenarios, the exact mathematical representation of the fitness function f is unknown. Thus, its optimisation often has to be handled as a black-box problem, and consequently becomes difficult and expensive. In light of this, it is especially useful to characterise a specific problem by means of (informative) features, based on which it is possible to select a suitable optimisation algorithm.

As there only exist preliminary studies on the characterisation of multi-objective ($p \geq 2$) problems (see, e.g., Kerschke, Wang et al. 2016)—which we will discuss later—we will in the following mainly focus on the manifold of characterisation approaches for single-objective ($p = 1$) continuous optimisation problems.

Single-objective continuous problems. Overviews on the early works related to this problem class can be found in Pitzer and Affenzeller (2012), Malan and Engelbrecht (2013), Sun et al. (2014), and Muñoz Acosta, Sun et al. (2015). However, in contrast to recent studies, the majority of the studies covered by those surveys proposed measures for characterising white-box problems—that is, problems whose landscapes are entirely known upfront—rather than cheap, informative, and automatically computable landscape features as needed for black-box problems. Obviously, only the latter are beneficial for automated algorithm selection (or related problems). Nevertheless, we briefly discuss some noteworthy contributions to the characterisation of white-box problems, as they form the basis for recent developments.

In the 1990s, landscapes were classified into easy and hard problems—from an optimisation algorithm's point of view. While Jones and Forrest (1995) (and later Müller and Sbalzarini, 2011) proposed *fitness distance correlation* as a key characteristic, Rosé et al. (1996) suggested the *density of states* for solving the binary classification task. Furthermore, so-called *epistasis* measures (Naudts et al., 1997; Rochet et al., 1997) quantify the influence of single variables (or bits, in case of a bit-representation of $\mathbf{x}$) on the problem's fitness, which in turn can be used to rank the landscapes according to their difficulty. The *information content* measures from Vassilev et al. (2000) provide another basis for quantifying hardness.

In the following decade, attention shifted from characterising problem difficulty to *ruggedness*. Depending on *fitness evolvability portraits* (Smith et al., 2002), *autocorrelation coefficients*, number and *distribution of optima* (Brooks and Durfee, 2003), or *entropy* (Malan and Engelbrecht, 2009), landscapes were categorised into rugged, neutral and smooth problems. During that time, researchers also focused on problem multimodality (Preuss, 2015), that is, the analysis of the problems' landscapes w.r.t. multiple local and/or global optima. For instance, the *barrier trees* of Flamm et al. (2002) provided means for identifying basins of attractions, local optima and saddle points within the landscapes, and the *dispersion* metrics from Lunacek and Whitley (2006) enabled an approximate estimation of the degree of multimodality of a given landscape.

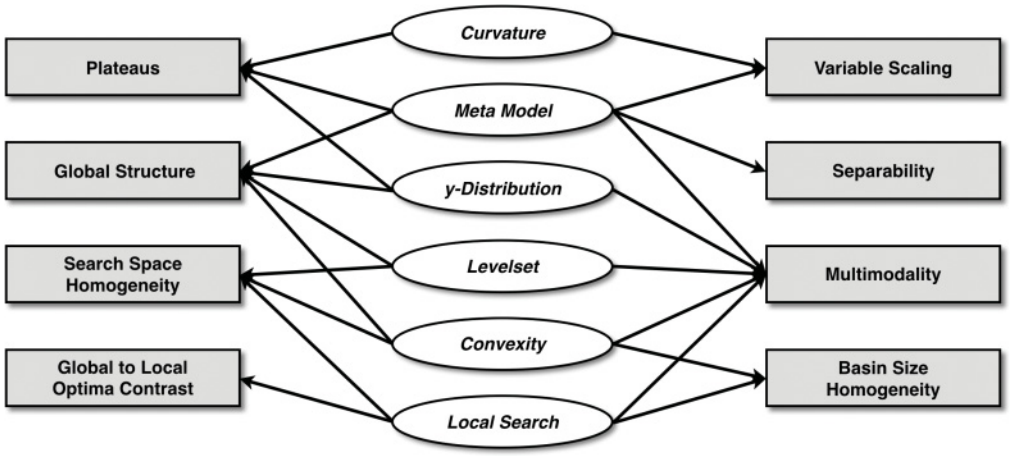

Figure 3: Overview of connections between "high-level" properties (grey rectangles) and "low-level" features (white ellipses), taken from Mersmann et al. (2011).

Pitzer and Affenzeller (2012) combined these approaches under the term *fitness landscape analysis (FLA)*. However, as the majority of those characteristics was proposed for white-box settings, they are not useful for automated algorithm selection. With the introduction of *Exploratory Landscape Analysis (ELA)*, Mersmann et al. (2011) were the first to explicitly develop landscape features for use in black-box optimisation (BBO)—and hence for algorithm selection in continuous optimisation. As shown in Figure 3, they introduced six feature sets (*curvature*, *convexity*, *levelset*, *local search*, *meta models*, and *y-distribution*), combining a total of 50 features, and used these to predict eight different problem characteristics, such as the global structure (none, weak, strong, deceptive) and the degree of multimodality (none, low, mediocre, high). We note that the attributes of the latter "high-level" properties can be assigned only by someone with knowledge of the entire problem, whereas the former "low-level" features can be automatically computed based on a small, but representative sample of points—the so-called *initial design*.

Following the idea of automatically computable numerical features, some of the earlier white-box landscape characteristics have been adapted to the black-box context. For instance, Muñoz Acosta et al. (2012) and Muñoz Acosta, Kirley et al. (2015) refined the *information content features* and extended them by *basin of attraction features*. Similarly, Abell et al. (2013) proposed *hill climbing* and *random point features*, whose general idea stems from the local search and fitness distance correlation methods, respectively, and enhanced them by *problem definition measures*.

In addition, several new feature sets have been introduced in recent years: Kerschke et al. (2014) discretised the continuous search space into a grid of cells and used a Markov-chain-inspired cell-mapping approach to obtain features that measure, amongst others, the sizes of the basins of attraction. However, due to the curse of dimensionality, those features are only practically applicable to low-dimensional problems. A much more scalable measure for the basins of attraction and the global structure of a landscape are the *nearest better clustering features* (Kerschke et al., 2015), which provide the means to distinguish funnel-shaped from rather random global structures. The most recently published features based on aggregated information of neighbouring points are

the *bag of local landscape features* by Shirakawa and Nagao (2016). Furthermore, the *length scale features* of Morgan and Gallagher (2015) measure the variable scaling based on the ratio between the change in objective space and distance in decision space for pairs of distinct observations from the initial design.

We note that work in this area is not restricted to the development of features or problem characteristics. For instance, Malan et al. (2015) and Bagheri et al. (2017) investigated constrained optimisation problems, whereas Kerschke, Preuss et al. (2016) showed that landscape features already possess sufficient information if they are computed based on rather small samples of $50 \cdot d$ observations (where d is the dimensionality of the given optimisation problem).

Until recently, the simultaneous use of feature sets from different research groups has been cumbersome and hence rarely practiced. However, with the development of `flacco` (Kerschke, 2017b, c), an `R`-package that provides source code for most of the previously listed ELA features, this obstacle has been overcome. Since then, the complementarity of the various features and their potential usefulness as a basis for algorithm selection has been demonstrated in several studies. We provide a detailed overview of this work later (see Section 5). Note that by using a platform-independent web-application of the `flacco` package[1] (Hanster and Kerschke, 2017), researchers and practitioners who are unfamiliar with `R` can also benefit from this extensive collection of more than 300 landscape features. Belkhir et al. (2016, 2017) were among the first to leverage the ELA features provided by `flacco` for per-instance algorithm configuration.

Multiobjective continuous problems. While the characterisation of single-objective continuous optimisation problems has been studied for over two decades, only preliminary studies have been conducted with respect to informative features of multiobjective problems. For instance, Kerschke and Trautmann (2016) used features that have been developed for single-objective problems and used them to cluster some well-known and frequently used multiobjective benchmark problems. However, those features do not capture characteristics that are especially important to multiobjective problems, such as interaction effects between the objectives. Eventually, techniques that were originally aimed at measuring variable interactions (see, e.g., Reshef et al., 2011; Sun et al., 2017) could help to overcome these limitations.

Kerschke, Wang et al. (2016, 2018) investigated locally efficient sets and the corresponding locally optimal fronts (i.e., the multiobjective equivalents of local optima) and proposed measures—on the basis of those sets and fronts—which enable the distinction of multiobjective problems according to their degree of multimodality. Interestingly, those studies revealed that even in the case of rather simple multiobjective problems, strong interaction effects between the objectives exist. Thus, in order to improve the understanding of multiobjective problems, Kerschke and Grimme (2017) introduced new techniques for visualising them. Their approach, dubbed *gradient field heatmaps* (see, e.g., Figure 4), is based on visualising the decision space, but nevertheless clearly reveals interactions between the objectives—in the form of rugged landscapes with several basins of attractions. Unexpected findings of this kind (see, e.g., Grimme et al., 2018) indicate that research in this area is still at an early stage, leaving many open challenges for future work (see Section 6 for further details).

[1] https://flacco.shinyapps.io/flacco/

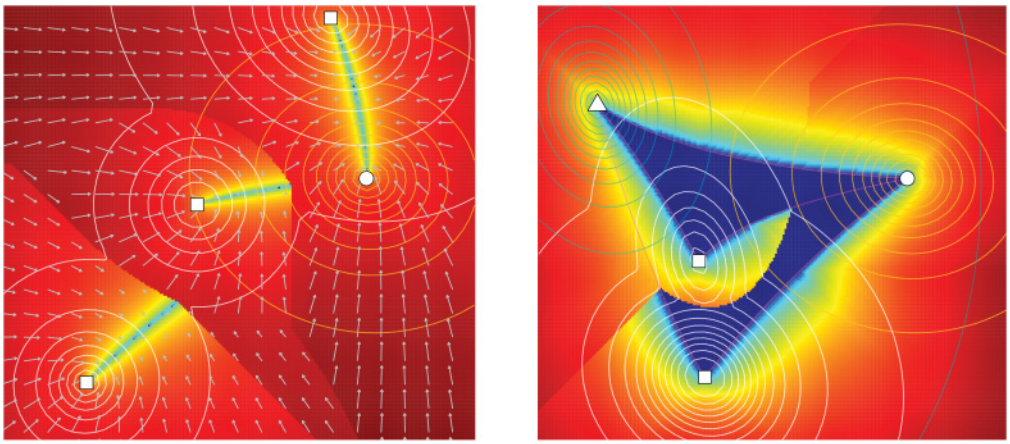

Figure 4: Exemplary visualisations of the *gradient field heatmaps* proposed by Kerschke and Grimme (2017). The images show the decision spaces of continuous optimisation problems with two (left) and three objectives (right), respectively. As a result of the interactions between the local optima of the different objectives (indicated by circles, squares, and triangles), the landscapes show multiple basins of attraction.

4 Algorithm Selection for Discrete Problems

Over the years, algorithm selection techniques have achieved remarkable results in several research areas, especially for discrete combinatorial problems (see, e.g., Smith-Miles, 2009; Kotthoff, 2014). However, due to the significant differences between various problems, not only the respective instance features, but also solvers and algorithm selectors vary considerably. Therefore, it is impossible to cover all work in this area; instead, in the following, we focus on a small number of particularly well-known problems: *propositional satisfiability (SAT)* and related problems, *AI planning*, and the *travelling salesperson problem (TSP)*.

Propositional satisfiability and related problems. Historically, some of the first and most widely known successes of per-instance algorithm selection have been achieved in the context of solving the *propositional satisfiability problem (SAT)*. We note that several approaches described in the following can be applied to closely related problems (such as MaxSAT) in a rather straightforward way.

SATzilla2003 and SATzilla2007. Within the highly contested area of SAT, the first AS system to outperform stand-alone SAT solvers was SATzilla. While its first version, denoted SATzilla2003 (Nudelman, Leyton-Brown, Devkar et al., 2014), still showed (minor) weaknesses—it ranked "only" second (twice) and third in the 2003 SAT Competitions—its (enhanced) successor, SATzilla2007 (Xu et al., 2008), won multiple prizes (3x first, 2x second and 2x third) in the 2007 SAT Competition. Despite the differences with respect to their success, the general working principles underlying both systems are quite similar. During the actual construction phase, pre- and backup solvers are identified, based on the performances of the given solvers on training data. All instances that have not been solved by the respective pre-solvers are then used for training

separate regression models per solver, which in turn are used for selecting a promising subset of "main solvers."

The main difference between SATzilla2003 and SATzilla2007 lies in the regression models used for the main algorithm selection phase: While the former used *empirical hardness models* (Leyton-Brown et al., 2002) based on *ridge regression* (see, e.g., Bishop, 2006), the latter employed *hierarchical hardness models* (Xu et al., 2007), more precisely *sparse multinomial logistic regression* (SMLR, see, e.g., Krishnapuram et al., 2005). The latter version of SATzilla sequentially runs up to two manually selected pre-solvers; if these fail to solve the given instance within a user-specified running time budget, instance features are computed and used to predict (and sequentially run) the best solver(s) from the given set **A**. The system either terminates once the given SAT instance has been solved, or when a user-specified cutoff time is reached.

Xu et al. (2008) compared different versions of SATzilla2007 on 2 300 random (RAND), 1 490 crafted/handmade (HAND), and 1021 application/industrial (INDU) instances, using the same setup as the 2007 SAT Competition. For this purpose, they used a total of 48 features and identified between one and two pre-solvers, one backup solver, as well as three to five main solvers from a given set **A** (which differed between RAND, HAND, and INDU). Notably, their pre-solvers solved between 32 and 62% of the instances—despite running for only seven CPU seconds at most. Ultimately, SATzilla2007 ranked first (inter alia) in the SAT & UNSAT (RAND and HAND) categories of the competition, as well as in UNSAT (HAND).

3S. Kadioglu et al. (2011) proposed a hybrid approach, denoted *semi-static solver schedules* (3S), which combines algorithm selection with solver scheduling. Since it can be very expensive to determine schedules over all solvers from a given set, Kadioglu et al. (2011) devised a different approach, in which they partitioned the (normalised) instance feature space for a given training set by means of g-means (Hamerly and Elkan, 2003). Then, the best value of k to be used by the k-nearest neighbour algorithm (Hastie et al., 2009) was identified for each cluster and directly used for training semi-static solver schedules. 3S was demonstrated to perform very well; for example, it reduced the gap between the (back then) state-of-the-art selector on RAND instances, SATzilla2009_Rand, and the VBS by 57%. Additionally, this general-purpose selector performed very well at the 2011 SAT Competition, winning seven medals (including two gold) without training separate selectors for different competition tracks (as had been done for previous versions of SATzilla).

SATzilla2012. Building on the success of earlier versions of SATzilla, Xu et al. (2012) developed SATzilla2012, which showed outstanding performance in multiple SAT competitions. SATzilla2012 uses cost-sensitive pairwise classification as the basis for per-instance algorithm selection; these penalise incorrect predictions according to the loss in performance caused by them. More precisely, one cost-sensitive decision tree (Ting, 2002) is used for every pair of solvers in the given set **A** to predict which of the two solvers will perform better on the given instance. Simple voting over these pairwise predictions is used to determine the solver to be run. Like earlier versions of SATzilla, SATzilla2012 uses pre- and backup-solvers in addition to the main algorithm selection stage. Additionally, a decision forest (Hastie et al., 2009) based on the number of clauses and variables in the given SAT instance is used to predict whether the computation of further instance features is sufficiently cheap to proceed to the main selection stage (otherwise, a statically chosen backup solver is run). In the 2012 SAT Competition,

SATzilla2012 used a total of 31 SAT solvers and 138 features and ended up winning multiple prizes.

CSHC. Building on the static scheduling approach underlying 3S, Malitsky et al. (2013) introduced an algorithm selection system based on the core concept of *cost-sensitive hierarchical clustering* (CSHC). During its training phase, CSHC iteratively partitions the instance feature space by means of hyperplanes, and occasionally undoes splits if that leads to improvements in overall performance. When given a new instance, CSHC first runs the 3S static algorithm schedule for 10% of the overall running time allotted for solving a given SAT instance F and, in case F remains unsolved, runs the SAT solver that performed best on the partition to which F belongs. The resulting CSCH algorithm selection system has been reported to achieve even better performances than SATzilla2012 on a slightly modified version of the 2011 SAT competition (for details, see Malitsky et al., 2013).

SNNAP. An approach dubbed *solver-based nearest neighbour for algorithm portfolios* (SNNAP; Collautti et al., 2013) successfully combines clustering with per-instance algorithm selection. It uses *random forests* (Hastie et al., 2009) to predict the running times of individual solvers. However, instead of directly selecting a solver based on these predictions, SNNAP uses them to identify SAT instances from the given training set that are similar to the instance to be solved. Specifically, instance similarity is quantified by means of the *Jaccard distance* – whose distance between two sets A and B is defined as $d(A, B) = 1 - |A \cap B|/|A \cup B|$ – applied to binary vectors indicating a (small) fixed number of best solvers per instance. SNNAP then selects the solver that performed best on the k nearest neighbours of the given instance, where k is a user-defined constant. According to results reported by Collautti et al. (2013), despite the simplicity of the approach, SNNAP closes around 50% of the VBS-SBS gap on a broad set of well-known SAT instances.

AutoFolio. In light of the many design choices encountered in the development of state-of-the-art algorithm selection systems, Lindauer, Hoos et al. (2015, 2017) proposed a powerful combination of per-instance algorithm selection and automated algorithm configuration: AutoFolio. In a nutshell, AutoFolio applies the automated algorithm configurator SMAC (Hutter et al., 2011) to the highly parametric algorithm selection framework claspfolio2 (Hoos et al., 2014). The space is structured in layers, starting with parameters for pre-solving schedules (including their allocated budgets), pre-processing procedures (transformations, filtering, etc.) and the algorithm selection systems (resembling a broad range of approaches, including SATzilla2007, SATzilla2012, 3S, and SNNAP). Subsequent layers specify additional design decisions and (hyper)parameters. As demonstrated for multiple algorithm selection scenarios from the ASLib, AutoFolio indeed achieves results that are highly competitive with those of the best-performing selection systems for a broad range of algorithm selection scenarios, without the need for manual choice of the selection mechanism or selector parameters (Lindauer, Hoos et al., 2015, 2017).

AI planning. The notion of algorithm selection can be applied to domains as well as to instances of AI planning problems. In the first case, a planner is selected for a specific domain and then applied for solving arbitrary instances in that domain. This is conceptually closely related to per-set algorithm selection, as discussed in Section 2. In the second case, a planner is selected for a specific instance of a planning problem, such

that even within the same domain, different planners may be chosen, depending on the characteristics of the specific problem instance. Unlike per-domain selection of planners, this is an instance of per-instance algorithm selection and hence will be our focus in the following.

Per-domain selection approaches. Because per-domain selection of AI planners has been prominently studied in the literature, we briefly discuss some well-known approaches. The PbP planning system and its successor, PbP2, are based on the idea of statistically analysing the performance of several domain-independent planning algorithms on a set of training instances from a given planning domain in order to select a set of planners and associated running times (Gerevini et al., 2009, 2011). When solving new instances from the same domain, these planners are run one after the other, using round-robin scheduling with the pre-determined running times for each planner. PbP and PbP2 also make use of macro-actions, sequences of actions whose judicious use can considerably improve planner performance. PbP was the overall winner of the learning track of the 6th International Planning Competition, and PbP2 brought further improvement through the integration of automated algorithm configuration to better exploit the performance potential of parameterised planners.

The ASAP planning system is based on similar ideas (Vallati et al., 2013, 2014). In addition to macro-actions, ASAP also exploits so-called entanglements, which reflect causal relationships that are characteristic for a given domain of planning problems. Different from PbP and PbP2, ASAP selects only a single representation (set of macro-actions and entanglements) and planner for a given domain. On standard benchmarks, ASAP has been demonstrated to outperform PbP2 (as well as all the component planners it uses) in terms of the quality or cost of the plans found (Vallati et al., 2014).

IBaCoP2. To the best of our knowledge, the first successful application of per-instance algorithm selection to AI planning was demonstrated by IBaCoP2 (Cenamor et al., 2014). IBaCoP2 uses 12 component planners, which were selected based on their performance on a large and diverse set of problem instances from past international planning competitions, applying Pareto efficiency analysis to the solution quality of the best plan found by a given planner, and the time required to find the first valid plan. A random forest model (see, e.g., Hastie et al., 2009), learned from performance data of the component planners on a set of training instances using WEKA (Witten et al., 2016), forms the core of the algorithm selection strategy. This model is used to predict whether a component planner will solve a given problem instance within a fixed time limit, based on a set of 35 cheaply computable, domain-specific features, some of which are derived from heuristics used in state-of-the-art planning algorithms. To hedge against the consequences of poor predictions, IBaCoP2 selects the five component planners with the highest estimated probability p for solving the given instance i and runs these, in the order of decreasing values of p, one after the other, each for one fifth of the overall time given to the selector for solving i. We note that, because of this latter strategy, IBaCoP2 is not a pure per-instance algorithm selector, but rather combines per-instance algorithm selection with a simple algorithm scheduling approach (cf. Section 2). IBaCoP and IBaCoP2 showed strong performance in the 8th International Planning Competition, with IBaCoP2 winning the sequential satisficing track (Vallati, 2015).

Planzilla. A second per-instance algorithm selection approach for AI planning has been considered by Rizzini et al. (2015, 2017). Their Planzilla system can be seen as an application of the previously outlined *zilla approach (Cameron et al., 2017) to AI planning.

Based on the default configuration of *zilla, Planzilla is comprised of four sequential stages: (1) a static pre-solving schedule, (2) feature computation, (3) per-instance algorithm selection, and (4) a backup solver. The pre-solving schedule is obtained by greedy selection from the given set of component planners and allocated 1/90 of the overall time budget for solving the given instance i. Training instances solved during the pre-solving stage are not considered for constructing the per-instance selector, nor for selecting the backup solver. Per-instance algorithm selection makes use of a comprehensive set of 311 features that includes a broad range of properties of instance i, as well as features derived from encoding i into propositional satisfiability. Based on this set of features, Planzilla uses cost-sensitive classification forests for each pair of component planners in combination with a voting procedure to determine the planner to be run for the remainder of the given time budget. Before computing the complete feature set, which can be somewhat costly, Planzilla uses a simple model to predict whether feature computation can be completed within the remaining time, t'; if not, feature computation and per-instance algorithm selection are skipped, and a backup solver is run instead. This backup solver is also run if the component planner selected in stage 3 terminates early without producing a valid plan; it is determined as the solver with the best performance for running time t' on the set of problem instances used to train Planzilla (excluding any instances solved during pre-solving).

Using all planners that participated in the optimal track of the 2014 International Planning Competition (IPC-14), Planzilla was found to substantially outperform these individual planners and achieve performance close to that of the VBS (Rizzini et al., 2015, 2017). However, when evaluated on a set of testing instances dissimilar from those used for training, it was found that dynamic algorithm scheduling approaches performed better than Planzilla; these approaches dynamically construct an algorithm schedule by performing multiple stages of per-instance algorithm selection, using not only features of the planning instance i to be solved, but also taking into account which component planners have already been run on i, without success, in earlier stages of the schedule.

Travelling salesperson problem. The potential for per-instance algorithm selection for the TSP differs markedly between *exact TSP solvers*, which are guaranteed to find provably optimal solutions for given TSP instances, and *inexact solvers*, which may find optimal solutions, but cannot produce a proof of optimality. In exact TSP solving, there is a single algorithm, Concorde (Applegate et al., 2007), that has defined state-of-the-art performance for more than a decade. In contrast, for inexact TSP solving, there is no single algorithm that clearly dominates all others (across all types of instances). In fact, several studies (Pihera and Musliu, 2014; Kotthoff et al., 2015; Kerschke et al., 2017) have shown that at least three TSP solvers, EAX (Nagata and Kobayashi, 1997, 2013), LKH (Helsgaun, 2000, 2009), and MAOS (Xie and Liu, 2009), define state-of-the-art performance on different kinds of TSP instances. In addition, two enhanced versions of EAX and LKH (denoted EAX+restart and LKH+restart), which employ additional restart mechanisms to overcome stagnation in the underlying search process, often (but not always) outperform EAX and LKH, respectively (Dubois-Lacoste et al., 2015). Further modifications of these algorithms—for example, based on the alternative crossover operator proposed by Sanches et al. (2017a, b), which recently was integrated into LKH (Tinós et al., 2018)—might achieve even better performance; however, to this date, we are not aware of conclusive evidence to this effect. Instead, using automated algorithm selection techniques, the performance complementarity between existing solvers has

been leveraged, leading to very substantial performance improvements over the single best solver (Kotthoff et al., 2015; Kerschke et al., 2017).

Kotthoff et al. (2015) compared EAX, LKH, and their respective restart variants across four well-known sets of TSP instances: random uniform Euclidean (RUE) instances, problems from the TSP library (TSPLIB), as well as national and VLSI instances.[2] They used the feature sets proposed by Mersmann et al. (2013) and Hutter, Xu et al. (2014) (see Section 3.1) for constructing multiple algorithm selectors. Their best selector, based on multivariate adaptive regression splines (MARS; see, e.g., Friedman, 1991), was trained on a predefined subset of features by Hutter, Xu et al. (2014) and closed the gap between the single best solver from their set (EAX+RESTART) to the VBS by 10%.

In an extended version of this earlier study, Kerschke et al. (2017) considered additional types of TSP instances, feature sets and solvers, and furthermore employed more sophisticated machine learning techniques, including various feature selection strategies, for constructing algorithm selectors. Specifically, the set of TSP instances was extended by clustered and morphed instances (Gent et al., 1999; Mersmann et al., 2012; Meisel et al., 2015); that is, linear combinations of clustered and RUE instances.[3] Furthermore, the basis for algorithm selection was expanded with the feature set of Pihera and Musliu (2014) and the MAOS solver by Xie and Liu (2009). The best algorithm selector under this extended setup was found to be a support vector machine (Karatzoglou et al., 2004), which was constructed on a cheap, yet informative, subset of 16 nearest-neighbour features (Pihera and Musliu, 2014). This particular selector achieved, despite its non-negligible feature computation costs, a PAR10 score of 16.75s and thereby closed the gap between the single best solver (EAX+RESTART; 36.30s) and the virtual best solver (10.73s) by more than 75%.

Further discrete combinatorial problems. Throughout the previous paragraphs, we gave an overview of algorithm selection approaches for some of the most prominent and widely studied discrete combinatorial problems. Of course, per-instance algorithm selection has shown to be effective on several other discrete problems, such as the *travelling thief problem (TTP)*, where Wagner et al. (2017) recently presented the first study of algorithm selection, along with a comprehensive collection of performance and feature data.

In some cases, successful applications of algorithm selection techniques have been described using different terminology. For instance, Smith-Miles (2008) presented her algorithm selector for the *quadratic assignment problem (QAP)* under the umbrella of a "metalearning inspired framework." Similarly, Pulina and Tacchella (2009) demonstrated the successful application of algorithm selection to the problem of solving *quantified Boolean formulae (QBF)*, an important generalisation of SAT; yet, they describe their selector as a self-adaptive multi-engine solver, which "selects among its reasoning engines the one which is more likely to yield optimal results."

Considering the size of the literature on per-instance algorithm selection for discrete combinatorial problems, a comprehensive overview would be far beyond the scope of this survey and produce little additional insight. Instead, we will now shift our attention to continuous problems, which present different challenges and opportunities for algorithm selection.

---

[2] http://www.math.uwaterloo.ca/tsp/index.html

[3] generated using the R-package NETGEN (Bossek, 2015)

5 Algorithm Selection for Continuous Problems

As previously mentioned, the efficacy of algorithm selection methods strongly depends on the performance data used for training them: The more representative the training set is regarding the entire range of possible problem instances, the better performance we can expect on previously unseen instances. For continuous optimisation problems, representativeness of benchmark sets has been a matter of long-standing debate, ranging from early works of De Jong (1975) up to more recent sets, for example, from the CEC competitions (Li et al., 2013) or the black-box optimisation benchmark (BBOB, Hansen et al., 2009) collection. There are several specific function generators, such as a framework for generating test functions with different degrees of multimodality (Wessing, 2015). Some of the most frequently used, and arguably most relevant test functions are included in the `Python`-package `optproblems` (Wessing, 2016) and the `R`-package `smoof` (Bossek, 2017). While all these benchmark sets (and generators) have advantages and drawbacks, a detailed discussion is beyond the scope of this article. However, we note that the construction of representative training sets remains, at least to some degree, an open challenge in the context of algorithm selection for continuous optimisation problems.

Unconstrained single-objective optimisation problems. In single-objective continuous optimisation, only few studies directly and successfully address the algorithm selection problem in an automated way. An initial approach of combining exploratory landscape analysis (ELA) and algorithm selection was presented by Bischl et al. (2012), focusing on the BBOB test suite. The latter consists of 24 functions which are grouped into four classes, mainly based on their multimodality, separability and global structure. Each function is represented by different instances resulting from slightly varied function parametrisations. Within the BBOB competition, 15 algorithm runs had to be conducted per function, equally distributed among five (BBOB 2009) or 15 (BBOB 2010) instances for decision space dimensions $2, 3, 5, 10, 20$, and, optionally, 40. Algorithm performance was then evaluated using expected running time (ERT, Hansen et al., 2009), which reflects the expected number of function evaluations required to reach the global optimum up to a threshold of $\varepsilon > 0$. Subsequently, the ERT was divided by the ERT of the best algorithm for this function within the respective competition to obtain a relative ERT indicator. Within the BBOB setup, accuracies in the range of $\{10^{-3}, 10^{-4}, \ldots, 10^{-8}\}$ are considered.

A representative set of four optimisation algorithms was constructed from the complete list of candidate solvers. Based on the low-level features introduced by Mersmann et al. (2011), Bischl et al. (2012) aimed for an accurate prediction of the best of the four algorithms for each function within the benchmark set. For this purpose, a sophisticated cost-sensitive learning approach, based on one-sided support vector regression with a radial basis function kernel, was used (Tu and Lin, 2010). This complex approach enabled the minimisation of loss (measured by relative ERT) due to incorrect predictions. The median relative ERT of all 600 entries (five runs times five instances for each of the 24 BBOB functions) served as the overall performance measure of the resulting classifier. Two different cross-validation strategies were investigated: cross-validation over the instances of each given function or over the complete set of functions. The latter task can certainly be considered more challenging, due the structure of the BBOB test set, which was designed to comprise 24 functions with maximally diverse characteristics, covering a broad range of continous optimisation problems. While, as expected,

better performance was observed in the first setting, performance was remarkably high in both cases.

Recently, substantial progress has been made in terms of systematically constructing an automated algorithm selector for a joint dataset of all available BBOB test suites (Kerschke, 2017a; Kerschke and Trautmann, 2018), making use of the R-packages `flacco` (Kerschke, 2017b, c), `smoof` (Bossek, 2017), and `mlr` (Bischl, Lang et al., 2016). A total of 480 BBOB instances (instance IDs 1–5 of all 24 BBOB functions, across dimensions 2, 3, 5 and 10) were considered, combined with respective results of all 129 solvers submitted to the COCO platform (Hansen et al., 2016) so far. In order to keep the size of the set of solvers manageable and to focus on the most relevant and high performing solvers, the construction of the algorithm selector was based on a carefully selected subset of 12 solvers: two deterministic methods (variants of the BRENT-STEP algorithm, BSRR and BSQI; see Baudiš and Pošík, 2015), five multi-level approaches—MLSL (Pál, 2013; Rinnooy Kan and Timmer, 1987), FMINCON, FMINUNC, HMLSL (Pál, 2013), and MCS (Huyer and Neumaier, 2009)—as well as four CMA-ES variants: CMA-CSA (Atamna, 2015), IPOP-400D (Auger et al., 2013), HCMA (Loshchilov et al., 2013), and SMAC-BBOB (Hutter et al., 2013). The commercial solver OPTQUEST/NLP (Pál, 2013; Ugray et al., 2007) was also included.

Performance was measured by relative ERT as in Bischl et al. (2012) by normalising the ERT for each solver per problem and dimension based on the best ERT for the respective problem (among the algorithms in the given set). A hybrid version of CMA-ES (HCMA, Loshchilov et al., 2013) turned out to be the single best solver (SBS), with a mean relative ERT score of 30.4 across all considered instances—and thus being the only solver to approximate the optimal objective value of all 96 problems up to the precision level of 10^{-2} used for this study. Various combinations of supervised learning (notably, classification, regression, pairwise regression) methods and feature selection strategies (greedy forward-backward and backward-forward, two genetic algorithm variants) were utilised in combination with leave-one-function-out cross-validation. The best algorithm selector obtained in this manner, a classification-based support vector machine (Vapnik, 1995) combined with a sequence of a greedy forward-backward and a genetic-algorithm-based feature selection approach, managed to reduce the mean relative ERT of the SBS roughly by half, to a value of 14.2, only requiring nine out of more than 300 exploratory landscape features. Specifically, meta model and nearest-better clustering features (see Section 3.2) were used in this context. Feature computation costs were taken into account and accounted for merely $50 \times d$ samples of the objective function (Kerschke, Preuss et al., 2016)—where d denotes the dimensionality of the given decision space—which matches common intitial population sizes of evolutionary optimisation algorithms. Hence, when using such an evolutionary optimisation algorithm, making use of its initial population, which needs to be evaluated in any case, renders feature computation cost negligible.

Constrained single-objective optimisation problems. In the area of constrained continuous optimisation, the performance of popular evolutionary computation techniques—such as differential evolution, evolution strategies, and particle swarm optimisation—has been investigated for problems with linear and quadratic constraints (Poursoltan and Neumann, 2015b, a). Features capturing the correlation of these constraints have been investigated w.r.t. their impact on solver performance. Malan et al. (2015) and Malan and Moser (2018) numerically characterised constraint violations using landscape features. Based on results on the CEC 2010 benchmark problems,

it was demonstrated that this approach produces detailed insights into constraint violation behaviour of continuous optimisation algorithms, indicating its potential usefulness in the context of algorithm selection and related approaches.

Furthermore, constraints have been evolved in different ways to construct instances that can be used for algorithm selection. This includes approaches maximising the performance difference of two given solvers, as well as a multiobjective approach for creating instances that reflect the tradeoffs between two given algorithms observed when varying the constraints. Neumann and Poursoltan (2016) demonstrated that the multiobjective approach leads to an instance set that provides a better basis for algorithm selection than sets obtained by maximising performance differences.

Multiobjective optimisation problems. So far, there are no systematic studies of automated algorithm selection for multiobjective continuous optimisation problems, as feature design is already extremely challenging and the suitability of existing benchmark sets is questionable. However, initial approaches regarding multiobjective features, landscape analysis and multimodality exist and offer promising perspectives for future research (see Section 6).

6 Perspectives and Open Problems

In the previous sections, we have given an overview of the state-of-the-art in algorithm selection for discrete and continuous problems. We have summarised the general approach of per-instance algorithm selection and discussed a number of related problems, including per-set algorithm selection, automated algorithm configuration, algorithm schedules, and parallel algorithm portfolios. An important aspect for any per-instance algorithm selection approach is the design of informative and cheaply computable features that are able to characterise and differentiate problem instances with respect to a given set of algorithms. We have given an overview of feature sets for several prominent discrete problems, as well as features used in continuous black-box optimisation. Based on this, we have summarised prominent algorithm selection approaches for discrete and continuous problems. In the following, we will discuss perspectives and challenges for future research in the area of automated algorithm selection.

Performance measures. A crucial part of any empirical performance evaluation, which provides the basis for constructing algorithm selectors, is the underlying performance measure. While *penalised average running time* (notably, PAR10; Bischl, Kerschke et al., 2016) and *expected running time* (ERT, Hansen et al., 2009) are commonly used in this context, some of its parameters can substantially affect performance measurements. For example, in the case of PAR10, the penalty factor is set to 10 and an arithmetic mean is used to aggregate over multiple runs or problem instances. Both choices can, in principle, be varied (e.g., by replacing the arithmetic mean by different quantiles), with significant effects on the robustness of solvers selected based on them.

Recently, Kerschke, Bossek et al. (2018) presented a structured approach on how to assess the sensitivity of an empirical performance evaluation w.r.t. altered requirements regarding solver robustness across runs, focusing on PAR10 and ERT. As demonstrated within this study, by adjusting the parameters of the performance measures, users are able to adapt the ranking of a given set of algorithms, and the performance characteristics of the algorithm selectors constructed based on that set, according to their preferences, trading off, for example, running time vs. robustness. In an alternative approach, van Rijn et al. (2017) utilise the advantages of two popular performance measures—ERT

and *fixed cost error* (FCE, see, e.g. Bäck et al., 2013)—by combining and standardising them within a joint performance measure. Moreover, a multiobjective perspective on performance measurement shows promise, for example, by enabling direct investigations of the trade-off between the number of failed runs and the average running time of successful runs of a given solver (Bossek and Trautmann, 2018). Concepts such as Pareto-optimality and related multiobjective quality indicators (Coello Coello et al., 2007) could then be used in the context of constructing and assessing the performance of algorithm selectors and related meta-algorithmic techniques.

Evolving / generating problem instances. Naturally, an algorithm selector only generalises to problems which are similar enough to the instances contained in the benchmark set that was used to construct it. Usually, common benchmark sets are considered, which are deemed to be representative of a specific use context. However, one could argue that benchmark sets should be designed specifically with respect ot the given set of candidate solvers, such that they exhibit maximum diversity regarding the challenges posed by the instances for the specific solvers. The idea of evolving instances utilising an evolutionary algorithm dates back to van Hemert (2006) and Smith-Miles et al. (2010), who constructed instances that are extremely hard or easy for specific TSP solvers; these works were followed by more sophisticated approaches of Mersmann et al. (2012, 2013) and Nallaperuma et al. (2013). Bossek and Trautmann (2016a, b) further explored this idea by evolving TSP instances that maximise the performance difference between two given solvers, that is, instances that are extremely hard to solve for one solver, but very easy for the other. This can, in principle, provide insights into links between performance differences and instance characteristics, a topic that is highly relevant for automated algorithm selection.

Recent studies build upon these concepts by explicitly focusing on the diversity of evolved instances (Gao et al., 2015; Neumann et al., 2018), paving the way for a systematic approach to construct most informative and relevant benchmarks specifically tailored to a given set of solvers. The next promising step in this direction will be the complementation of state-of-the-art benchmarks with those specifically designed instances in order to provide most informative benchmark sets tailored to given sets of solvers. Such sets are likely to provide a basis for constructing per-instance algorithm selectors whose performance generalises better to problem instances that differ from those used during their construction. Improvements, configuration and enhancements of the underlying evolutionary algorithm offer extremely promising research perspectives. Interestingly, the overall approach also provides a way for systematically detecting advantages and shortcomings of specific solvers, and thus produce benefits for the analysis and design of algorithms beyond algorithm selection and related approaches.

Online algorithm selection. The work covered in this survey is mainly related to (static) *offline* algorithm selection, where algorithm selectors are constructed based on using a set of training instances *prior* to applying them to new problem instances. Yet, according to Armstrong et al. (2006), Gagliolo and Schmidhuber (2010), and Degroote et al. (2016), in principle, it might be possible to obtain even better results using (dynamic) *online* algorithm selection methods, which adapt an algorithm selector while it is being used to solve a series of problem instances. Although this involves a certain overhead, it can enable better performance and increased robustness, as the selector can react better to changes in the stream of problem instances it is tasked to solve. In order to more easily amortise the overhead involved in online algorithm selection, building on earlier work on probing features (Hutter, Hoos et al., 2014; Kotthoff et al., 2015), the

development of cheap and informative monitoring features—that is, features that extract sufficient instance-specific information without significantly reducing solver or selector performance—is likely to be of key importance.

Another approach for online algorithm selection and the selection of an appropriate algorithm for a given problem instance is provided by so-called *hyper-heuristics*; these are algorithms for selecting or generating solvers for a given problem from a set of given heuristic components (Burke et al., 2013). In many cases, they employ heuristic, rule-based mechanisms and make use of rather simple components, such as greedy construction algorithms for the given problem.

Life-long learning hyper-heuristics are applied in a setting where a series of problem instances is solved. Based on the performance of previously chosen component solvers, a solver for a new problem instance is chosen that is deemed most likely to perform best. Life-long learning hyper-heuristics have achieved good results for well-known decision problems, such as constraint satisfaction (Ortiz-Bayliss et al., 2015) and bin packing (Sim et al., 2015).

Features for mixed (discrete + continuous) problems. Developing good features for a given problem is a challenging task that provides a crucial basis for effective algorithm selection techniques. As discussed previously, rich sets of features have been introduced for well-studied discrete and continuous problems, but many combinatorial problems of practical importance involve discrete and continuous decision variables. Perhaps the best example for this is *mixed integer programming (MIP)*, a problem of great importance in the context of a broad range of challenging real-world optimisation tasks. Hutter, Xu et al. (2014) introduced 95 features for MIP, including problem type and size, variable-constraint graph, linear constraint matrix, objective function, and LP-based features. For the *travelling thief problem (TTP)*, which can be seen as a combination of the TSP and *knapsack problem (KP)*, algorithm selection has been studied by Wagner et al. (2017). They used 48 TSP and four KP features, plus three parameters of the TTP that connect the TSP and KP parts of the problem. Features for the KP that characterise the correlation of weights and profits have so far not been taken into account, although they seem to provide a further improvement in the characterisation of TTP instances.

Algorithm selection for multiobjective optimisation problems. While we have covered numerous studies on algorithm selection in this survey, none of them has dealt with multiobjective optimisation problems. From a practitioner's point of view, this is a significant limitation, as multiple competing objectives arise in many, perhaps most, real-world problems.[4] However, for convenience, these problems are often handled as single-objective problems, for example, by focusing on the most important objective or by applying scalarisation functions. Many prominent benchmarks for continuous multiobjective optimisation algorithms, such as bi-objective BBOB (Tušar et al., 2016), DTLZ (Deb et al., 2005), ED (Emmerich and Deutz, 2007), MOP (van Veldhuizen, 1999), UF (Zhang et al., 2008), WFG (Huband et al., 2006), and ZDT (Zitzler et al., 2000), are entirely artificial, and it is unclear to which degree they resemble real-world problems.

In addition, there is a dearth of research on the characterisation of those optimisation problems, in particular by means of automatically computable features. Of course, one could compute variants of existing features for each of the objectives separately (see,

[4]Consider, for example, the trade-off between the performance improvement and the accompanying costs for finding such an improved solution.

e.g., Kerschke and Trautmann, 2016), but this completely ignores interaction effects between the objectives, which in turn have a strong impact on the landscapes (even for rather simple problems; see, e.g., Kerschke and Grimme, 2017). Hence, there is a significant need and opportunity for research on visualising multiobjective landscapes (see, e.g., da Fonseca, 1995; Tušar, 2014; Tušar and Filipič, 2015; Kerschke and Grimme, 2017), as well as characterising them (numerically)—along the lines of Ulrich et al. (2010), Kerschke, Wang et al. (2016, 2018), or Grimme et al. (2018)—as this will (a) improve the understanding of multiobjective problems and their specific properties, and (b) provide a basis for automated feature computation. We expect the latter to be of key importance for the development of new algorithms and effective per-instance selectors in the area of multiobjective optimisation.

In order to construct a powerful and complementary portfolio of multiobjective optimisers (a priori), as well as for analysing the strengths and weaknesses of the resulting algorithm selector (*a posteriori*), visual approaches such as the *empirical attainment function* (EAF) plots (da Fonseca et al., 2005; da Fonseca and da Fonseca, 2010; López-Ibáñez et al., 2010) provide valuable feedback on location, spread, and inter-point dependence structure of the considered optimisers' Pareto set approximations.

Algorithm selection on streaming data. The importance of automated algorithm selection and related approaches in the context of learning on streaming data should not be neglected. Streaming data (Bifet et al., 2018) pose considerable challenges for the respective algorithms, as (a) data points arrive as a constant stream, (b) the size of the stream is large and potentially unbounded, (c) the order of data points cannot be influenced, (d) data points can typically only be evaluated once and are discarded afterwards, and (e) the underlying distribution of the data points in the stream can change over time (non-stationarity or concept drift).

Few concepts for automated algorithm selection on streaming data exist so far, both for supervised (see, e.g., van Rijn et al., 2014, 2018) and unsupervised learning algorithms. In unsupervised learning, stream clustering is a very active research field. Although several stream clustering approaches exist (see surveys of Amini et al., 2014; Carnein and Trautmann, 2018; Mansalis et al., 2018), these have many parameters that affect their performance, yet clear guidelines on how to set and adjust them over time are lacking. Moreover, different kinds of algorithms are required in the so-called online phase (maintaining an informative aggregated data stream representation, in terms of microclusters) and offline phase (standard clustering algorithm, such as k-means applied to the microclusters), which leads to a huge space of parameter and algorithm combinations. An initial approach on configuring and benchmarking stream clustering approaches based on *irace* (López-Ibáñez et al., 2016) has been presented by Carnein et al. (2017). Building on this work, especially in light of remark (e) above, we expect that algorithm selection will play an important role for robust learning on streaming data, especially when keeping the efficiency regarding real-time capability in mind. This will require informative feature sets, ensemble techniques, as well as a much wider range of suitable benchmarks. Of course, ideally a combination of algorithm selection and (online) configuration is desired and a very promising line of research.

Per-instance algorithm configuration. As previously explained, automatic algorithm configuration involves the determination of parameter settings of a given target algorithm to optimise its performance on a given set of problem instances. *Per-instance algorithm configuration (PIAC)* is a variant of this problem in which parameter settings are

determined for a given problem instance to be solved. It can be seen as a generalisation of per-instance algorithm selection, in which the set of algorithms that form the basis for selection comprises all (valid) configurations of a single, parameterised algorithm. Analogous to per-instance algorithm selection, PIAC involves two phases: an offline phase, during which a per-instance configurator is trained, and an online phase, in which this configurator is run on given problem instances. During the latter, a configuration is determined based on features of the problem instance to be solved. Standard, per-set algorithm configuration, in contrast, is an offline process that results in a single configuration that is subsequently used to solve problem instances presumed to resemble those used during training. PIAC is challenging, because the spaces of (valid) configurations to select from is typically very large (see, e.g., Hutter, López-Ibáñez et al., 2014), and compared to the size of these configuration spaces, any training data used during the offline construction of the per-instance configurator is necessarily sparse. In particular, for typical configuration scenarios, the training data would necessarily cover only a very small number of configurations, which makes it challenging to learn a mapping from instance features to configurations.

We consider PIAC to be a largely open problem, with significant potential for future work. There is some evidence in the literature that it may have significant benefits compared to the more established per-set configuration techniques. Notably, Kadioglu et al. (2010) proposed a PIAC approach based on a combination of clustering and a standard, per-set algorithm configurator and reported promising results on several set covering, mixed integer programming and propositional satisfiability algorithms.

Further challenges. Due to the steady stream of work in algorithm selection and related areas, it is important to keep track of promising developments. Of course, domain-related research networks such as COSEAL[5] might relieve this challenging task to some degree, yet they will not be able to keep up with all developments within this fast-growing and productive community. Instead, comparisons against state-of-the-art methods, which are of special significance in this context, are facilitated by benchmarking platforms and libraries, such as ASLib (Bischl, Kerschke et al., 2016), ACLib (Hutter, López-Ibáñez et al., 2014), and HPOlib (Eggensperger et al., 2013) for algorithm selection, configuration and hyperparameter optimisation, respectively. At the same time, it is important (a) to promote and establish as best practice the use of these libraries, especially in the context of newly proposed methods for algorithm selection and related problems, and (b) to maintain and expand these libraries, in order to ensure their continued relevance, for example, by integrating scenarios for multiobjective and additional real-world problems.

The latter not only applies to the previously mentioned libraries, but also to broader benchmark collections for the underlying specific problems. For example, recent studies have analysed the "footprints" of different continuous optimisation algorithms on common benchmarks; while Muñoz Acosta and Smith-Miles (2017) focused on BBOB (arguably the most prominent benchmark in continuous black-box optimisation), Muñoz Acosta et al. (2018) applied a similar analysis to machine learning problems from the UCI repository (Dheeru and Karra Taniskidou, 2017) and OpenML (van Rijn et al., 2013). Their visual analyses indicate that (a) different "comfort zones" for the various algorithms in question exist across the respective instance spaces, in line with what might be expected based on a liberal interpretation of the NFL theorems by Wolpert

[5]https://www.coseal.net/

and Macready (1997), and (b) the instances from common benchmarks' problems in continuous optimisation are not very diverse, but cover only relatively small areas of the overall problem instance space.

Another important direction for future work is the improvement of problem-specific features in general. Aside from the directions outlined previously (monitoring features as well as features for mixed and multiobjective problems), more informative and cheaper features are always desirable and likely to pave the way towards more effective applications of algorithm selection and related techniques.

An interesting open question regards the trade-off between the performance achieved by algorithm selection approaches, for example, in relation to a hypothetical perfect selector (VBS), and their complexity, including the complexity of the feature sets they operate on. There is recent evidence from an application of algorithm selection to solvers for quantified Boolean formulae (QBF) that suggests that sometimes, a small number of simple features is sufficient for achieving excellent performance (Hoos et al., 2018). However, it is presently unclear to which extent this situation arises in other application scenarios, and to which degree it is contingent on the use of highly sophisticated algorithm selection techniques.

Finally, an intriguing direction for future work is the development of algorithm selection techniques for automated algorithm configurators and selectors. Intuitively, it is clear that different algorithm configuration scenarios would be handled most efficiently using rather different configuration procedures (depending, e.g., on the prevalence of numerical vs. categorical parameters). Likewise, it has been observed in the recent *Open Algorithm Selection Competition* (Lindauer et al., 2018) that different AS techniques work best on different AS scenarios—suggesting that meta-algorithm selection (i.e., AS applied to AS strategies) might be useful for quickly identifying the selection strategy to be used in a particular application context. In both cases, configurator selection and metaselection, the limited amount of training data is likely to give rise to specific challenges, which may well require the development of new AS techniques.

Acknowledgments

Pascal Kerschke, Heike Trautmann, and Holger H. Hoos acknowledge support from the *European Research Center for Information Systems (ERCIS)*. The former two also acknowledge support from the DAAD PPP project No. 57314626. Frank Neumann acknowledges the support of the Australian Research Council through grant DP160102401. The authors gratefully acknowledge useful inputs from Lars Kotthoff and Jakob Bossek.

References

Abell, T., Malitsky, Y., and Tierney, K. (2013). Features for exploiting black-box optimization problem structure. In *Proceedings of the 7th International Conference on Learning and Intelligent Optimization*, pp. 30–36. Lecture Notes in Computer Science, Vol. 7997.

Amini, A., Wah, T. Y., and Saboohi, H. (2014). On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, 29(1):116–141.

Angel, E., and Zissimopoulos, V. (2002). On the hardness of the quadratic assignment problem with metaheuristics. *Journal of Heuristics*, 8(4):399–414.

Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., and Tierney, K. (2015). Model-based genetic algorithms for algorithm configuration. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pp. 733–739.

Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. (2007). *The traveling salesman problem: A computational study*. Princeton: Princeton University Press.

Arik, S., Huang, T., Lai, W. K., and Liu, Q. (Eds.) (2015). *Proceedings Part III of the 22nd International Conference on Neural Information Processing*. Lecture Notes in Computer Science, Vol. 9491.

Armstrong, W., Christen, P., McCreath, E., and Rendell, A. P. (2006). Dynamic algorithm selection using reinforcement learning. In *International Workshop on Integrating AI and Data Mining*, pp. 18–25.

Atamna, A. (2015). Benchmarking IPOP-CMA-ES-TPA and IPOP-CMA-ES-MSR on the BBOB noiseless testbed. In *Proceedings of the 17th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 1135–1142.

Auger, A., Brockhoff, D., and Hansen, N. (2013). Benchmarking the local metamodel CMA-ES on the noiseless BBOB'2013 test bed. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 1225–1232.

Bäck, T., Foussette, C., and Krause, P. (2013). *Contemporary evolution strategies*. Natural Computing Series. New York: Springer.

Bagheri, S., Konen, W., Allmendinger, R., Branke, J., Deb, K., Fieldsend, J., Quagliarella, D., and Sindhya, K. (2017). Constraint handling in efficient global optimization. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 673–680.

Baral, C. (2003). *Knowledge representation, reasoning and declarative problem solving*. Cambridge: Cambridge University Press.

Baudiš, P., and Pošík, P. (2015). Global line search algorithm hybridized with quadratic interpolation and its extension to separable functions. In *Proceedings of the 17th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 257–264.

Belkhir, N., Dréo, J., Savéant, P., and Schoenauer, M. (2016). Feature based algorithm configuration: A case study with differential evolution. In *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature*, pp. 156–166. Lecture Notes in Computer Science, Vol. 9921.

Belkhir, N., Dréo, J., Savéant, P., and Schoenauer, M. (2017). Per instance algorithm configuration of CMA-ES with limited budget. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 681–688.

Biere, A., Heule, M., van Maaren, H., and Walsh, T. (2009). *Handbook of satisfiability*, Vol. 185. Amsterdam: IOS Press.

Bifet, A., Gavaldà, R., Holmes, G., and Pfahringer, B. (2018). *Machine learning for data streams with practical examples in MOA*. Cambridge, MA: MIT Press.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 11–18.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, T. M., Malitsky, Y., Fréchette, A., Hoos, H. H., Hutter, F., Leyton-Brown, K., Tierney, K., and Vanschoren, J. (2016). ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58.

Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., and Jones, Z. M. (2016). mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5.

Bischl, B., Mersmann, O., Trautmann, H., and Preuss, M. (2012). Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 313–320.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Bossek, J. (2015). *netgen: Network generator for combinatorial graph problems*. R-package version 1.0.

Bossek, J. (2017). smoof: Single- and multi-objective optimization test functions. *The R Journal*.

Bossek, J., Grimme, C., Meisel, S., Rudolph, G., and Trautmann, H. (2018). Local search effects in bi-objective orienteering. In *Proceedings of the 20th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 585–592.

Bossek, J., and Trautmann, H. (2016a). Evolving instances for maximizing performance differences of state-of-the-art inexact TSP solvers. In R. Batti, M. Brunato, I. Kotsireas, and P. Pardalos (Eds.), *Learning and intelligent optimization*, pp. 48–59. LION 12 2018. Lecture Notes in Computer Science, Vol. 11353.

Bossek, J., and Trautmann, H. (2016b). Understanding characteristics of evolved instances for state-of-the-art inexact TSP solvers with maximum performance difference. In *Proceedings of the Conference of the Italian Association for Artificial Intelligence*, pp. 3–12. Lecture Notes in Computer Science, Vol. 10037.

Bossek, J., and Trautmann, H. (2019). Multi-objective performance measurement: Alternatives to PAR10 and expected running time. In *Proceedings of the 4th International Conference on Learning and Intelligent Optimization*. Lecture Notes in Computer Science. Publication status: Accepted.

Boukeas, G., Halatsis, C., Zissimopoulos, V., and Stamatopoulos, P. (2004). Measures of intrinsic hardness for constraint satisfaction problem instances. In *SOFSEM: 30th International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 184–195. Lecture Notes in Computer Science, Vol. 2932.

Brooks, C. H., and Durfee, E. H. (2003). Using landscape theory to measure learning difficulty for adaptive agents. In *Adaptive Agents and Multi-Agent Systems*, pp. 291–305. Lecture Notes in Computer Science, Vol. 2636.

Burke, E. K., Gendreau, M., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

Cameron, C., Hoos, H. H., Leyton-Brown, K., and Hutter, F. (2017). OASC-2017: *Zilla submission. In *Proceedings of the Open Algorithm Selection Challenge*, pp. 15–18. *Proceedings of Machine Learning Research*, Vol. 79.

Carnein, M., Assenmacher, D., and Trautmann, H. (2017). An empirical comparison of stream clustering algorithms. In *Proceedings of the Computing Frontiers Conference*, pp. 361–365.

Carnein, M., and Trautmann, H. (2019). Optimizing data stream representation: An extensive survey on stream clustering algorithms. *Business and Information Systems Engineering*, pp. 1–21.

Cenamor, I., de la Rosa, T., and Fernández, F. (2013). Learning predictive models to configure planning portfolios. In *Proceedings of the Fourth Workshop on Planning and Learning at the Twenty-Third International Conference on Automated Planning and Scheduling*, pp. 14–22.

Cenamor, I., de la Rosa, T., and Fernández, F. (2014). IBaCoP and IBaCoP2 planner. *Proceedings of the Eighth International Planning Competition*, pp. 35–38.

Coello Coello, C. A., Lamont, G. B., and van Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems*. New York: Springer.

Collautti, M., Malitsky, Y., Mehta, D., and O'Sullivan, B. (2013). SNNAP: Solver-based nearest neighbor for algorithm portfolios. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 435–450. Lecture Notes in Computer Science, Vol. 8190.

Cook, D. J., and Varnell, R. C. (1997). Maximizing the benefits of parallel search using machine learning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 559–564. Association for the Advancement of Artificial Intelligence.

Culberson, J. C. (1998). On the futility of blind search: An algorithmic view of "no free lunch." *Evolutionary Computation*, 6(2):109–127.

da Fonseca, C. M. M. (1995). Multiobjective genetic algorithms with application to control engineering problems. PhD thesis, Department of Automatic Control and Systems Engineering, University of Sheffield.

da Fonseca, C. M. M., da Fonseca, V. G., and Paquete, L. (2005). Exploring the performance of stochastic multiobjective optimisers with the second-order attainment function. In *Proceedings of the 3rd International Conference on Evolutionary Multi-Criterion Optimization*, pp. 250–264. Lecture Notes in Computer Science, Vol. 3410.

da Fonseca, V., and da Fonseca, C. M. M. (2010). The attainment-function approach to stochastic multiobjective optimizer assessment and comparison. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss (Eds.), *Experimental methods for the analysis of optimization algorithms*, pp. 103–130. New York: Springer.

Davidor, Y. (1991). Epistasis variance: A viewpoint on GA-hardness. In *Foundations of genetic algorithms*, pp. 23–35. Vol. 1. Amsterdam: Elsevier.

De Jong, K. A. (1975). Analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2005). Scalable test problems for evolutionary multiobjective optimization. In A. Abraham, L. Jain, and R. Goldberg (Eds.), *Evolutionary multiobjective optimization*, Advanced Information and Knowledge Processing (AI & KP), pp. 105–145. New York: Springer.

Degroote, H., Bischl, B., Kotthoff, L., and De Causmaecker, P. (2016). Reinforcement learning for automatic online algorithm selection—An empirical study. In *Proceedings of ITAT 2016: Information Technologies—Applications and Theory: Conference on Theory and Practice of Information Technologies*, pp. 93–101. Vol. 1649.

Dheeru, D., and Karra Taniskidou, E. (2017). UCI Machine Learning Repository.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple Classifier Systems*, pp. 1–15. Lecture Notes in Computer Science, Vol. 1857.

Dubois-Lacoste, J., Hoos, H. H., and Stützle, T. (2015). On the empirical scaling behaviour of state-of-the-art local search algorithms for the Euclidean TSP. In *Proceedings of the 17th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 377–384.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H. H., and Leyton-Brown, K. (2013). Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, Vol. 10.

Emmerich, M. T. M., and Deutz, A. H. (2007). Test problems based on Lamé superspheres. In *Proceedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization*, pp. 922–936. Lecture Notes in Computer Science, Vol. 4403.

Fawcett, C., Vallati, M., Hutter, F., Hoffmann, J., Hoos, H. H., and Leyton-Brown, K. (2014). Improved features for runtime prediction of domain-independent planners. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*. Association for the Advancement of Artificial Intelligence.

Flamm, C., Hofacker, I. L., Stadler, P. F., and Wolfinger, M. T. (2002). Barrier trees of degenerate landscapes. *Zeitschrift für Physikalische Chemie. International Journal of Research in Physical Chemistry and Chemical Physics*, 216(2/2002):155–173.

Fonlupt, C., Robilliard, D., and Preux, P. (1998). A bit-wise epistasis measure for binary search spaces. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pp. 47–56. Lecture Notes in Computer Science, Vol. 1498.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, 1–67.

Fukunaga, A. S. (2000). Genetic algorithm portfolios. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1304–1311.

Gagliolo, M., and Schmidhuber, J. (2010). Algorithm selection as a bandit problem with unbounded losses. In *Proceedings of the 4th International Conference on Learning and Intelligent Optimization*, pp. 82–96. Lecture Notes in Computer Science, Vol. 6073.

Gao, W., Nallaperuma, S., and Neumann, F. (2015). Feature-based diversity optimization for problem instance classification. arXiv:abs/1510.08568. *Conference version appeared in Parallel Problem Solving from Nature* (2016).

Gent, I. P., Hoos, H. H., Prosser, P., and Walsh, T. (1999). Morphing: Combining structure and randomness. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pp. 654–660. Vol. 99. Association for the Advancement of Artificial Intelligence.

Gerevini, A., and Long, D. (2005). *Plan constraints and preferences in PDDL3*. Technical Report. Department of Electronics for Automation, University of Brescia.

Gerevini, A., Saetti, A., and Serina, I. (2003). Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research*, 20:239–290.

Gerevini, A., Saetti, A., and Vallati, M. (2009). An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*. Association for the Advancement of Artificial Intelligence (AAAI).

Gerevini, A., Saetti, A., and Vallati, M. (2011). PbP2: Automatic configuration of a portfolio-based multi-planner.

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated planning: Theory and practice*. Amsterdam: Elsevier.

Gomes, C. P., and Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62.

Grimme, C., Kerschke, P., Emmerich, M. T. M., Preuss, M., Deutz, A. H., and Trautmann, H. (2018). Sliding to the global optimum: How to benefit from non-global optima in multimodal multi-objective optimization. In *Proceedings of the International Global Optimization Workshop*. In press.

Hamerly, G., and Elkan, C. (2003). Learning the k in k-means. In *Proceedings of Advances in Neural Information Processing Systems 16*, pp. 281–288.

Hansen, N., Auger, A., Mersmann, O., Tušar, T., and Brockhoff, D. (2016). COCO: A platform for comparing continuous optimizers in a black-box setting. arXiv:abs/1603.08785v3.

Hansen, N., Finck, S., Ros, R., and Auger, A. (2009). Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829. INRIA.

Hanster, C., and Kerschke, P. (2017). flaccogui: Exploratory landscape analysis for everyone. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO) Companion*, pp. 1215–1222.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. 2nd ed. Berlin: Springer.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130.

Helsgaun, K. (2009). General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119–163.

Hoffmann, J. (2011). Analyzing search topology without running any search: On the connection between causal graphs and h+. *Journal of Artificial Intelligence Research*, 41:155–229.

Hoos, H. H., Lindauer, T. M., and Schaub, T. (2014). claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, 14(4-5):569–585.

Hoos, H. H., Peitl, T., Slivovsky, F., and Szeider, S. (2018). Portfolio-based algorithm selection for circuit QBFs. In *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming*, pp. 195–209. Lecture Notes in Computer Science.

Howe, A. E., Dahlman, E., Hansen, C., Scheetz, M., and von Mayrhauser, A. (1999). Exploiting competitive planner performance. In *Proceedings of the Fifth European Conference on Planning*, pp. 62–72. Lecture Notes in Computer Science, Vol. 1809.

Hsu, E. I., and McIlraith, S. A. (2009). VARSAT: Integrating novel probabilistic inference techniques with DPLL search. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, pp. 377–390. Lecture Notes in Computer Science, Vol. 5584.

Huband, S., Hingston, P., Barone, L., and While, L. (2006). A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506.

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

Hutter, F., Hamadi, Y., Hoos, H. H., and Leyton-Brown, K. (2006). Performance prediction and automated tuning of randomized and parametric algorithms. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming*, pp. 213–228. Lecture Notes in Computer Science, Vol. 4204.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2013). An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 1209–1216.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration (extended version). In *Proceedings of 5th International Conference on Learning and Intelligent Optimization*, pp. 507–523. Vol. 6683.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306.

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, T. M., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2014). AClib: A benchmark library for algorithm configuration. In *Proceedings of the 8th International Conference on Learning and Intelligent Optimization*, pp. 36–40. Lecture Notes in Computer Science, Vol. 8426.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111.

Huyer, W., and Neumaier, A. (2009). Benchmarking of MCS on the noiseless function testbed. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO)*.

Järvisalo, M., Le Berre, D., Roussel, O., and Simon, L. (2012). The international SAT solver competitions. *AI Magazine*, 33(1):89–92.

Jones, T. (1995). Evolutionary algorithms, fitness landscapes and search. PhD Thesis, University of New Mexico.

Jones, T., and Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pp. 184–192.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2011). Algorithm selection and scheduling. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming*, pp. 454–469. Lecture Notes in Computer Science, Vol. 6876.

Kadioglu, S., Malitsky, Y., Sellmann, M., and Tierney, K. (2010). ISAC—Instance-specific algorithm configuration. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pp. 751–756. Frontiers in Artificial Intelligence and Applications.

Kanda, J., Carvalho, A., Hruschka, E., and Soares, C. (2011). Selection of algorithms to solve traveling salesman problems using meta-learning. *International Journal of Hybrid Intelligent Systems*, 8(3):117–128.

Karatzoglou, A., Smola, A., Hornik, K., and Zeileis, A. (2004). kernlab—An S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20.

Kauffman, S. A. (1993). *The origins of order: Self-organization and selection in evolution*. Oxford: Oxford University Press.

Kerschke, P. (2017a). Automated and feature-based problem characterization and algorithm selection through machine learning. PhD thesis, University of Münster.

Kerschke, P. (2017b). Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the R-package flacco. arXiv:abs/1708.05258.

Kerschke, P. (2017c). *flacco: Feature-based landscape analysis of continuous and constrained optimization problems*. R-package version 1.6.

Kerschke, P., Bossek, J., and Trautmann, H. (2018). Parameterization of state-of-the-art performance indicators: A robustness study based on inexact TSP solvers. In *Proceedings of the 20th Annual Conference on Genetic and Evolutionary Computation (GECCO) Companion*, pp. 1737–1744.

Kerschke, P., and Grimme, C. (2017). An expedition to multimodal multi-objective optimization landscapes. In *Proceedings of the 9th International Conference on Evolutionary Multi-Criterion Optimization*, pp. 329–343.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., and Trautmann, H. (2017). Leveraging TSP solver complementarity through machine learning. *Evolutionary Computation*, 1–24.

Kerschke, P., Preuss, M., Hernández Castellanos, C. I., Schütze, O., Sun, J.-Q., Grimme, C., Rudolph, G., Bischl, B., and Trautmann, H. (2014). Cell mapping techniques for exploratory landscape analysis. In *EVOLVE—A bridge between probability, set oriented numerics, and evolutionary computation V*, pp. 115–131. New York: Springer.

Kerschke, P., Preuss, M., Wessing, S., and Trautmann, H. (2015). Detecting funnel structures by means of exploratory landscape analysis. In *Proceedings of the 17th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 265–272.

Kerschke, P., Preuss, M., Wessing, S., and Trautmann, H. (2016). Low-budget exploratory landscape analysis on multiple peaks models. In *Proceedings of the 18th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 229–236.

Kerschke, P., and Trautmann, H. (2016). The R-package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 5262–5269.

Kerschke, P., and Trautmann, H. (2018). Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, 1–28. doi:10.1162.

Kerschke, P., Wang, H., Preuss, M., Grimme, C., Deutz, A. H., Trautmann, H., and Emmerich, M. T. M. (2016). Towards analyzing multimodality of multiobjective landscapes. In *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature*, pp. 962–972. Lecture Notes in Computer Science, Vol. 9921.

Kerschke, P., Wang, H., Preuss, M., Grimme, C., Deutz, A. H., Trautmann, H., and Emmerich, M. T. M. (2018). Search dynamics on multimodal multi-objective problems. *Evolutionary Computation*, 0:1–30.

Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI* Magazine, 35(3):48–60.

Kotthoff, L., Kerschke, P., Hoos, H. H., and Trautmann, H. (2015). Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization*, pp. 202–217. Lecture Notes in Computer Science, Vol. 8994.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18(25):1–5.

Kovárik, O., and Málek, R. (2012). *Meta-learning and meta-optimization*. CTU Technical Report KJB2012010501 003, Prague.

Krishnapuram, B., Carin, L., Figueiredo, M. A. T., and Hartemink, A. J. (2005). Sparse multinomial logistic regression: Fast algorithms and generalization bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):957–968.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., and Shoham, Y. (2003). A portfolio approach to algorithm selection. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pp. 1542–1543.

Leyton-Brown, K., Nudelman, E., and 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. Lecture Notes in Computer Science, Vol. 2470.

Li, X., Engelbrecht, A. P., and Epitropakis, M. G. (2013). Benchmark functions for CEC'2013 special session and competition on niching methods for multimodal function optimization. Technical Report. RMIT University, Evolutionary Computation and Machine Learning Group, Australia.

Lin, S., and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516.

Lindauer, T. M. (2014). Algorithm selection, scheduling and configuration of Boolean constraint solvers. PhD thesis, University of Potsdam.

Lindauer, T. M., Bergdoll, R.-D., and Hutter, F. (2016). An empirical study of per-instance algorithm scheduling. In *Proceedings of the 10th International Conference on Learning and Intelligent Optimization*, pp. 253–259. Lecture Notes in Computer Science, Vol. 10079.

Lindauer, T. M., Hoos, H. H., and Hutter, F. (2015). From sequential algorithm selection to parallel portfolio selection. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization*, pp. 1–16. Lecture Notes in Computer Science, Vol. 8994.

Lindauer, T. M., Hoos, H. H., Hutter, F., and Schaub, T. (2015). AutoFolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778.

Lindauer, T. M., Hoos, H. H., Hutter, F., and Schaub, T. (2017). AutoFolio: An automatically configured algorithm selector (extended abstract). In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 5025–5029.

Lindauer, T. M., van Rijn, J. N., and Kotthoff, L. (2017b). Open algorithm selection challenge 2017: Setup and scenarios. In *Proceedings of Machine Learning Research*, pp. 1–7. Vol. 79.

Lindauer, T. M., van Rijn, J. N., and Kotthoff, L. (2018). The algorithm selection competition series 2015-17. arXiv:abs/1805.01214.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

López-Ibáñez, M., Paquete, L., and Stützle, T. (2010). Exploratory analysis of stochastic local search algorithms in biobjective optimization. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss (Eds.), *Experimental methods for the analysis of optimization algorithms*, pp. 209–223. New York: Springer.

Loshchilov, I., Schoenauer, M., and Sebag, M. (2013). Bi-population CMA-ES algorithms with surrogate models and line searches. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 1177–1184.

Lunacek, M., and Whitley, L. D. (2006). The dispersion metric and the CMA evolution strategy. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 477–484.

Mahajan, Y. S., Fu, Z., and Malik, S. (2004). Zchaff2004: An efficient SAT solver. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, pp. 360–375. Lecture Notes in Computer Science, Vol. 3542.

Malan, K. M., and Engelbrecht, A. P. (2009). Quantifying ruggedness of continuous landscapes using entropy. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1440–1447.

Malan, K. M., and Engelbrecht, A. P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163.

Malan, K. M., and Moser, I. (2018). Constraint handling guided by landscape analysis in combinatorial and continuous search spaces. *Evolutionary Computation*, 1–23. doi:10.1162/evco_a_00222

Malan, K. M., Oberholzer, J. F., and Engelbrecht, A. P. (2015). Characterising constrained continuous optimisation problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1351–1358.

Malitsky, Y., Sabharwal, A., Samulowitz, H., and Sellmann, M. (2013). Boosting sequential solver portfolios: Knowledge sharing and accuracy prediction. In *Proceedings of the 7th International Conference on Learning and Intelligent Optimization*, pp. 153–167. Lecture Notes in Computer Science, Vol. 7997.

Mansalis, S., Ntoutsi, E., Pelekis, N., and Theodoridis, Y. (2018). An evaluation of data stream clustering algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 11(4):167–187.

Maratea, M., Pulina, L., and Ricca, F. (2012). Applying machine learning techniques to ASP solving. In *Technical Communications of the 28th International Conference on Logic Programming*, pp. 37–48. Vol. 17.

Maron, O., and Moore, A. W. (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. In *Proceedings of Advances in Neural Information Processing Systems 6*, pp. 59–66.

Meisel, S., Grimme, C., Bossek, J., Wölck, M., Rudolph, G., and Trautmann, H. (2015). Evaluation of a multi-objective EA on benchmark instances for dynamic routing of a vehicle. In *Proceedings of the 17th Annual Conference on Genetic and Evolutionary Computation*, pp. 425–432.

Mersmann, O., Bischl, B., Bossek, J., Trautmann, H., Wagner, M., and Neumann, F. (2012). Local search and the traveling salesman problem: A feature-based characterization of problem hardness. In *Proceedings of the 6th International Conference on Learning and Intelligent Optimization*, pp. 115–129. Lecture Notes in Computer Science, Vol. 7219.

Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., and Rudolph, G. (2011). Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 829–836.

Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., and Neumann, F. (2013). A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence*, 69:151–182.

Morgan, R., and Gallagher, M. (2015). Analysing and characterising optimization problems using length scale. *Soft Computing*, 1–18.

Muñoz Acosta, M. A., Kirley, M., and Halgamuge, S. K. (2012). Landscape characterization of numerical optimization problems using biased scattered data. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1–8.

Muñoz Acosta, M. A., Kirley, M., and Halgamuge, S. K. (2013). The algorithm selection problem on the continuous optimization domain. In *Computational Intelligence in Intelligent Data Analysis*, pp. 75–89. Studies in Computational Intelligence, Vol. 445.

Muñoz Acosta, M. A., Kirley, M., and Halgamuge, S. K. (2015). Exploratory landscape analysis of continuous space optimization problems using information content. *IEEE Transactions on Evolutionary Computation*, 19(1):74–87.

Muñoz Acosta, M. A., and Smith-Miles, K. A. (2017). Performance analysis of continuous black-box optimization algorithms via footprints in instance space. *Evolutionary Computation*, 25(4):529–554.

Muñoz Acosta, M. A., Sun, Y., Kirley, M., and Halgamuge, S. K. (2015). Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317:224–245.

Muñoz Acosta, M. A., Villanova, L., Baatar, D., and Smith-Miles, K. A. (2018). Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147.

Müller, C. L., and Sbalzarini, I. F. (2011). Global characterization of the CEC 2005 fitness landscapes using fitness-distance analysis. In *Proceedings of the European Conference on the Applications of Evolutionary Computation*, pp. 294–303. Lecture Notes in Computer Science.

Nagata, Y., and Kobayashi, S. (1997). Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 450–457.

Nagata, Y., and Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, 25(2):346–363.

Nallaperuma, S., Wagner, M., Neumann, F., Bischl, B., Mersmann, O., and Trautmann, H. (2013). A feature-based comparison of local search and the Christofides algorithm for the travelling salesperson problem. In *Proceedings of the Twelfth Workshop on Foundations of Genetic Algorithms*, pp. 147–160.

Naudts, B., Suys, D., and Verschoren, A. (1997). Epistasis as a basic concept in formal landscape analysis. In *Proceedings of the 7th International Conference on Genetic Algorithms*, pp. 65–72.

Neumann, A., Gao, W., Doerr, C., Neumann, F., and Wagner, M. (2018). Discrepancy-based evolutionary diversity optimization. arXiv:abs/1802.05448.

Neumann, F., and Poursoltan, S. (2016). Feature-based algorithm selection for constrained continuous optimisation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1461–1468.

Nudelman, E., Leyton-Brown, K., Devkar, A., Shoham, Y., and Hoos, H. H. (2004). SATzilla: An algorithm portfolio for SAT. *SAT Competition 2004*.

Nudelman, E., Leyton-Brown, K., Hoos, H. H., Devkar, A., and Shoham, Y. (2004). Understanding random SAT: Beyond the clauses-to-variables ratio. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, pp. 438–452. Lecture Notes in Computer Science, Vol. 3258.

Ochoa, G., and Veerapen, N. (2016). Additional dimensions to the study of funnels in combinatorial landscapes. In *Proceedings of the 18th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 373–380.

Ochoa, G., Veerapen, N., Whitley, L. D., and Burke, E. K. (2015). The multi-funnel structure of TSP fitness landscapes: A visual exploration. In *Proceedings of the 12th International Conference on Artificial Evolution*, pp. 1–13. Lecture Notes in Computer Science, Vol. 9554.

Ortiz-Bayliss, J. C., Terashima-Marín, H., and Conant-Pablos, S. E. (2015). Lifelong learning selection hyper-heuristics for constraint satisfaction problems. In *Proceedings of the 14th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence and Soft Computing, Part I*, pp. 190–201. Lecture Notes in Computer Science, Vol. 9413.

Pál, L. (2013). Comparison of multistart global optimization algorithms on the BBOB noiseless testbed. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 1153–1160.

Pérez Cáceres, L., López-Ibáñez, M., Hoos, H. H., and Stützle, T. (2017). An experimental study of adaptive capping in irace. In *Proceedings of the 11th International Conference on Learning and Intelligent Optimization*, pp. 235–250. Lecture Notes in Computer Science, Vol. 10556.

Pihera, J., and Musliu, N. (2014). Application of machine learning to algorithm selection for TSP. In *Proceedings of the IEEE 26th International Conference on Tools with Artificial Intelligence*, pp. 47–54.

Pitzer, E., and Affenzeller, M. (2012). A comprehensive survey on fitness landscape analysis. In J. Fodor, R. Klempous, and C. P. Suárez Araujo (Eds.), *Recent advances in intelligent engineering systems*, Studies in Computational Intelligence, pp. 161–191. New York: Springer.

Poursoltan S., and Neumann F. (2015a). A feature-based analysis on the impact of set of constraints for ε-constrained differential evolution. In S. Arik, T. Huang, W. Lai, and Q. Liu (Eds.), *Neural Information Processing. ICONIP 2015*, pp. 344–355. Lecture Notes in Computer Science, vol. 9491. Cham, Switzerland: Springer.

Poursoltan S., and Neumann F. (2015b). A feature-based comparison of evolutionary computing techniques for constrained continuous optimisation. In S. Arik, T. Huang, W. Lai, and Q. Liu (Eds.), *Neural Information Processing. ICONIP 2015*, pp. 332–343. Lecture Notes in Computer Science, vol. 9491. Cham, Switzerland: Springer.

Preuss, M. (2015). *Multimodal optimization by means of evolutionary algorithms*. New York: Springer.

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

Reshef, D. N., Reshef, Y. A., Finucane, H. K., Grossman, S. R., McVean, G., Turnbaugh, P. J., Lander, E. S., Mitzenmacher, M., and Sabeti, P. C. (2011). Detecting novel associations in large data sets. *Science*, 334(6062):1518–1524.

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

Rinnooy Kan, A. H. G., and Timmer, G. T. (1987). Stochastic global optimization methods part II: Multi level methods. *Mathematical Programming*, 39(1):57–78.

Rizzini, M., Fawcett, C., Vallati, M., Gerevini, A. E., and Hoos, H. H. (2015). Portfolio methods for optimal planning: An empirical analysis. In *Proceedings of the IEEE 27th International Conference on Tools with Artificial Intelligence*, pp. 494–501.

Rizzini, M., Fawcett, C., Vallati, M., Gerevini, A. E., and Hoos, H. H. (2017). Static and dynamic portfolio methods for optimal planning: An empirical analysis. *International Journal on Artificial Intelligence Tools*, 26(01):1–27.

Roberts, M., Howe, A. E., Wilson, B., and desJardins, M. (2008). What makes planners predictable? In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pp. 288–295. Association for the Advancement of Artificial Intelligence.

Rochet, S., Venturini, G., Slimane, M., and El Kharoubi, E. (1997). A critical and empirical study of epistasis measures for predicting GA performances: A summary. In *European Conference on Artificial Evolution*, pp. 275–285. Lecture Notes in Computer Science, Vol. 1363.

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1–2):1–39.

Rosé, H., Ebeling, W., and Asselmeyer, T. (1996). The density of states—A measure of the difficulty of optimisation problems. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pp. 208–217. Lecture Notes in Computer Science, Vol. 1141.

Roussel, O. (2012). Description of ppfolio. In *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, p. 46.

Sanches, D., Whitley, L. D., and Tinós, R. (2017a). Building a better heuristic for the traveling salesman problem: Combining edge assembly crossover and partition crossover. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 329–336.

Sanches, D., Whitley, L. D., and Tinós, R. (2017b). Improving an exact solver for the traveling salesman problem using partition crossover. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 337–344.

Sander, J., Ester, M., Kriegel, H.-P., and Xu, X. (1998). Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194.

Seo, D.-I., and Moon, B.-R. (2007). An information-theoretic analysis on the interactions of variables in combinatorial optimization problems. *Evolutionary Computation*, 15(2):169–198.

Shirakawa, S., and Nagao, T. (2016). Bag of local landscape features for fitness landscape analysis. *Soft Computing*, 20(10):3787–3802.

Sim, K., Hart, E., and Paechter, B. (2015). A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation*, 23(1):37–67.

Smith, T., Husbands, P., and O'Shea, M. (2002). Fitness landscapes and evolvability. *Evolutionary Computation*, 10(1):1–34.

Smith-Miles, K., van Hemert, J. I., and Lim, X. Y. (2010). Understanding TSP difficulty by learning from evolved instances. In *Proceedings of the 4th International Conference on Learning and Intelligent Optimization*, pp. 266–280. Lecture Notes in Computer Science, Vol. 6073.

Smith-Miles, K. A. (2008). Towards insightful algorithm selection for optimisation using meta-learning concepts. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 4118–4124.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41:1–25.

Smith-Miles, K. A., and van Hemert, J. I. (2011). Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104.

Stadler, P. F. (2002). Fitness landscapes. In *Biological Evolution and Statistical Physics*, pp. 183–204. Lecture Notes in Physics, Vol. 585.

Sun, Y., Halgamuge, S. K., Kirley, M., and Muñoz Acosta, M. A. (2014). On the selection of fitness landscape analysis metrics for continuous optimization problems. In *Proceedings of the 7th International Conference on Information and Automation for Sustainability*, pp. 1–6.

Sun, Y., Kirley, M., and Halgamuge, S. K. (2017). Quantifying variable interactions in continuous optimization problems. *IEEE Transactions on Evolutionary Computation*, 21(2):249–264.

Tang, K., Peng, F., Chen, G., and Yao, X. (2014). Population-based algorithm portfolios with automated constituent algorithms selection. *Information Sciences*, 279:94–104.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855.

Ting, K. M. (2002). An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):659–665.

Tinós, R., Helsgaun, K., and Whitley, L. D. (2018). Efficient recombination in the Lin-Kernighan-Helsgaun traveling salesman heuristic. In *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature*, pp. 95–107. Lecture Notes in Computer Science, Vol. 11101.

Tu, H.-H., and Lin, H.-T. (2010). One-sided support vector regression for multiclass cost-sensitive classification. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1095–1102.

Tušar, T. (2014). Visualizing solution sets in multiobjective optimization. PhD thesis, Jožef Stefan International Postgraduate School.

Tušar, T., and Filipič, B. (2015). Visualization of Pareto front approximations in evolutionary multiobjective optimization: A critical review and the prosection method. *IEEE Transactions in Evolutionary Computation*, 19(2):225–245.

Tušar, T., Brockhoff, D., Hansen, N., and Auger, A. (2016). COCO: The bi-objective black box optimization benchmarking (bbob-biobj) test suite. arXiv:abs/1604.00359.

Ugray, Z., Lasdon, L., Plummer, J., Glover, F., Kelly, J., and Martí, R. (2007). Scatter search and local NLP solvers: A multistart framework for global optimization. *INFORMS Journal on Computing*, 19(3):328–340.

Ulrich, T., Bader, J., and Thiele, L. (2010). Defining and optimizing indicator-based diversity measures in multiobjective search. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature*, pp. 707–717. Lecture Notes in Computer Science, Vol. 6238.

Vallati, M., Chrpa, L., Grześ, M., McCluskey, T. L., Roberts, M., and Sanner, S. (2015). The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98.

Vallati, M., Chrpa, L., and Kitchin, D. (2013). An automatic algorithm selection approach for planning. In *Proceedings of the IEEE 25th International Conference on Tools with Artificial Intelligence*, pp. 1–8.

Vallati, M., Chrpa, L., and Kitchin, D. (2014). ASAP: An automatic algorithm selection approach for planning. *International Journal on Artificial Intelligence Tools*, 23(06):1460032.

van Hemert, J. I. (2006). Evolving combinatorial problem instances that are difficult to solve. *Evolutionary Computation*, 14(4):433–462.

van Rijn, J. N., Bischl, B., Torgo, L., Gao, B., Umaashankar, V., Fischer, S., Winter, P., Wiswedel, B., Berthold, M. R., and Vanschoren, J. (2013). OpenML: A collaborative science platform. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 643–649. Lecture Notes in Computer Science, Vol. 8190.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2014). Algorithm selection on data streams. In *Proceedings of the 17th International Conference on Discovery Science*, pp. 325–336. Lecture Notes in Computer Science, Vol. 8777.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2018). The online performance estimation framework: Heterogeneous ensemble learning for data streams. *Machine Learning*, 107(1):149–176.

van Rijn, S., Wang, H., van Stein, B., and Bäck, T. (2017). Algorithm configuration data mining for CMA evolution strategies. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 737–744.

van Veldhuizen, D. A. (1999). Multiobjective evolutionary algorithms: Classifications, analyzes, and new innovations. PhD thesis, Faculty of the Graduate School of Engineering of the Air Force Institute of Technology, Air University.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. Berlin: Springer.

Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2000). Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1):31–60.

Wagner, M., Lindauer, T. M., Mısır, M., Nallaperuma, S., and Hutter, F. (2017). A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, 1–26.

Wessing, S. (2015). Two-stage methods for multimodal optimization. PhD thesis, Technische Universität Dortmund.

Wessing, S. (2016). *optproblems: Infrastructure to define optimization problems and some test problems for black-box optimization*. `Python`-package version 0.6.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data mining: Practical machine learning tools and techniques*. Sun Francisco: Morgan Kaufmann.

Wolpert, D. H., and Macready, W. G. (1995). *No free lunch theorems for search*. Technical Report, SFI-TR-95-02-010. Santa Fe Institute, Santa Fe, NM.

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

Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the 6th International Congress of Genetics*, pp. 356–366. Vol. 1.

Xie, X.-F., and Liu, J. (2009). Multiagent optimization system for solving the traveling salesman problem (TSP). *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(2):489–502.

Xu, L., Hoos, H. H., and Leyton-Brown, K. (2007). Hierarchical hardness models for SAT. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pp. 696–711. Lecture Notes in Computer Science, Vol. 4741.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Resesearch*, 32:565–606.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*, pp. 228–241. Lecture Notes in Computer Science, Vol. 7317.

Yuen, S. Y., and Zhang, X. (2015). On composing an algorithm portfolio. *Memetic Computing*, 7(3):203–214.

Zhang, Q., Zhou, A., Zhao, S., Suganthan, P. N., Liu, W., and Tiwari, S. (2008). Multiobjective optimization test instances for the CEC 2009 special session and competition. Technical Report. University of Essex, Colchester, UK and Nanyang Technological University, Singapore. Special Session on Performance Assessment of Multi-Objective Optimization Algorithms.

Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, (2):173–195.