Journal of Global Optimization manuscript No.
(will be inserted by the editor)

A case study of algorithm selection for the traveling thief problem

**Markus Wagner · Marius Lindauer ·
Mustafa Mısır · Samadhi Nallaperuma ·
Frank Hutter**

Received: date / Accepted: date

Abstract Many real-world problems are composed of several interacting components. In order to facilitate research on such interactions, the Traveling Thief Problem (TTP) was created in 2013 as the combination of two well-understood combinatorial optimization problems.

With this article, we contribute in four ways. First, we create a comprehensive dataset that comprises the performance data of 21 TTP algorithms on the full original set of 9720 TTP instances. Second, we define 55 characteristics for all TPP instances that can be used to select the best algorithm on a per-instance basis. Third, we use these algorithms and features to construct the first algorithm portfolios for TTP, clearly outperforming the single best algorithm. Finally, we study which algorithms contribute most to this portfolio.

Keywords Combinatorial optimization · instance analysis · algorithm portfolio

M. Wagner was supported by the Australian Research Council (DE160100850) and by a Priority Partner Grant by the University of Adelaide, Australia. M. Lindauer and F. Hutter were supported by the DFG (German Research Foundation) under Emmy Noether grant HU 1900/2-1. M. Mısır was supported by the Nanjing University of Aeronautics and Astronautics Starter Research Fund.

Markus Wagner
Optimisation and Logistics Group, School of Computer Science, The University of Adelaide, Australia, E-mail: markus.wagner@adelaide.edu.au

Marius Lindauer
Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany
E-mail: lindauer@cs.uni-freiburg.de

Mustafa Mısır
Institute of Machine Learning and Computational Intelligence, Nanjing University of Aeronautics and Astronautics, China, E-mail: mmisir@nuaa.edu.cn

Samadhi Nallaperuma
Department of Computer Science, University of Sheffield, UK
E-mail: s.nallaperuma@sheffield.ac.uk

Frank Hutter
Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany
E-mail: fh@cs.uni-freiburg.de

1 Introduction

The complexity of operations is increasing in most companies, with several interacting components having to be addressed at once. For example, the issue of scheduling production lines (e.g., maximizing the efficiency or minimizing the cost) has direct relationship with inventory costs, transportation costs, delivery-in-full-on-time to customers, and hence should not be considered in isolation. In addition, optimizing one component of the operation may negatively impact activities in other components.

The academic traveling thief problem (TTP) (Bonyadi et al 2013) is quickly gaining attention as an NP-hard combinatorial optimization problem that combines two well-known subproblems: the traveling salesperson problem (TSP) and the knapsack problem (KP). These two components have been merged in such a way that the optimal solution for each single one does not necessarily correspond to an optimal TTP solution. The motivation for the TTP is to allow the systematic investigation of interactions between two hard component problems, to gain insights that eventually help solve real-world problems more efficiently (Bonyadi et al 2016).

Since the introduction of the TTP, many algorithms have been introduced for solving it. While the initial approaches were rather generic hill-climbers, researchers incorporated more and more domain knowledge into the algorithms. For example, this resulted in deterministic, constructive heuristics, in restart strategies, and also in problem-specific hill-climbers that try to solve the TTP holistically. While the use of insights typically resulted in an increase in the objective scores, the computational complexity also increased. Consequently, which one of the algorithms performs best is highly dependent on the TTP instance at hand. To exploit this complementarity of existing algorithms, here we study the applicability of algorithm selection (Rice 1976) to this problem.

Specifically, after describing the TTP (Section 2) and the algorithm selection problem (Section 3), we make the following contributions:

- We analyze the performance of 21 TTP algorithms on the original set of 9720 instances created by Polyakovskiy et al (2014a) (Section 4);
- We describe characteristics of TTP instances that can be used as “features” for determining the best algorithm for the instance (Section 5);
- We create the first algorithm portfolios for TTP, substantially improving performance over the best single TTP algorithm (Section 6); and
- We analyze how complementary the algorithms in the portfolio are and which algorithms are most important for achieving good performance (Section 7).

2 The travelling thief problem (TTP)

The traveling thief problem (TTP) (Bonyadi et al 2013) is a recent attempt to provide an abstraction of multicomponent problems with dependency among components. It combines two problems and generates a new problem with two components. In particular, it combines the traveling salesperson problem (TSP) and the knapsack problem (KP), as both problems are well known and have been studied for many years in the field of optimization.

In this section, we motivate the TTP as an academic problem that addresses an important gap in research and then define it formally.

2.1 Motivation

In contemporary business enterprises the complexity of real-world problems has to be perceived as one of the greatest obstacles in achieving effectiveness. Even relatively small companies are frequently confronted with problems of very high complexity. Some researchers investigated features of real-world problems that served to explain difficulties that Evolutionary Algorithms (EAs) experience in solving them. For example, Weise et al (2009) discussed premature convergence, ruggedness, causality, deceptiveness, neutrality, epistasis, and robustness, which make optimization problems hard to solve. However, it seems that these reasons are either related to the landscape of the problem (such as ruggedness and deceptiveness) or to the optimizer itself (such as premature convergence and robustness) and do not focus on the nature of the problem. Michalewicz and Fogel (2004) discussed a few different reasons behind the hardness of real-world problems, including problem size, presence of noise, multi-objectivity, and presence of constraints. Most of these features have been captured in different optimization benchmark sets, such as TSPlib (Reinelt 1991), MIPlib (Koch et al 2011) and OR-library (Beasley 1990).

Despite decades of research efforts and many articles written on Evolutionary Computation (EC) in dedicated conferences and journals, still it is not that easy to find applications of EC in the real-world. Michalewicz (2012) identified several reasons for this mismatch between academia and the real world. One of these reasons is that academic experiments focused on single component (single silo) benchmark problems, whereas real-world problems are often multi-component problems. In order to guide the community towards this increasingly important aspect of real-world optimization (Bonyadi et al 2016), the traveling thief problem was introduced (Bonyadi et al 2013) in order to illustrate the complexities that arise by having multiple interacting components.

A related problem is the vehicle routing problem (VRP, for an overview see Bell and McMullen (2004); Rizzoli et al (2007)). The VRP is concerned with finding optimal routes for a fleet of vehicles delivering or collecting items from different locations (Dantzig and Ramser 1959; Laporte 1992). Over the years, a number of VRP variants have been proposed, such as variants with multiple depots or with capacity constraints. However, the insights gained there do not easily carry over to the academic TTP, as we consider in addition to the routing problem not only a load-dependent feature, but also the NP-hard optimisation problem of deciding which items are to be stolen by the thieves. For discussions on how the TTP differs from the VRP, we refer the interested reader to Bonyadi et al (2013, 2014).

Despite being a challenging problem, it is often disputed whether the TTP is realistic enough because it only allows a single thief to travel across hundreds or thousands of cities to collect (steal) items. In addition, the thief is required to visit all cities, regardless of whether an item is stolen there or not. Chand and Wagner (2016) discussed the shortcomings of the current formulation and presented a relaxed version of the problem which allows multiple thieves to travel across different cities with the aim of maximizing the group's collective profit. A number of fast heuristics were also proposed for solving the newly proposed multiple travelling thieves problem (MTTP). It was observed that having a small number of additional thieves could yield significant improvements of the objective scores in many cases.

2.2 Formal Definition

We use the definition of the TTP by Polyakovskiy et al (2014a). Given is a set of cities $N = \{1, \ldots, n\}$ and a set of items $M = \{1, \ldots, m\}$ distributed among the cities. For any pair of cities $i, j \in N$, we know the distance d_{ij} between them. Every city i, except the first one, contains a set of items $M_i = \{1, \ldots, m_i\}$, $M = \underset{i \in N}{\cup} M_i$. Each item k positioned in city i is characterized by its profit p_{ik} and weight w_{ik}, thus the item $I_{ik} \sim (p_{ik}, w_{ik})$. The thief must visit all cities exactly once starting from the first city and returning back to it in the end. Any item may be selected in any city as long as the total weight of collected items does not exceed the specified capacity W. A renting rate R is to be paid per each time unit taken to complete the tour. υ_{max} and υ_{min} denote the maximal and minimum speeds that the thief can move. The goal is to find a tour, along with a packing plan, that results in the maximal profit.

The objective function uses a binary variable $y_{ik} \in \{0, 1\}$ that is equal to one when the item k is selected in the city i, and zero otherwise. Also, let W_i denote the total weight of collected items when the thief leaves the city i. Then, the objective function for a tour $\varPi = (x_1, \ldots, x_n)$, $x_i \in N$ and a packing plan $P = (y_{21}, \ldots, y_{nm_i})$ has the following form:

$$Z(\varPi, P) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R \left(\left(\sum_{i=1}^{n-1} \frac{d_{x_i x_{i+1}}}{\upsilon_{max} - \nu W_{x_i}} \right) + \frac{d_{x_n x_1}}{\upsilon_{max} - \nu W_{x_n}} \right)$$

where $\nu = \frac{\upsilon_{max} - \upsilon_{min}}{W}$ is a constant value defined by input parameters. The first term is the sum of all packed items' profits and the second term is the amount that the thief pays for the knapsack's rent (equal to the total traveling time along $\varPi$ multiplied by R). Within the knapsack's rent term, the first addend is the cost for traveling from city i to $i + 1$ and the second addend is the cost for traveling from the last city back to the city with ID 1.

Note that different values of the renting rate R result in different TTP instances that might be "harder" or "easier" to solve. For example, for small values of R (relative to the profits), the overall rent contributes little to the final objective score. In the extreme case $R = 0$, the best solution for a given TTP instance is equivalent to the best solution of the KP component, which means that there is no need to solve the TSP component at all. Similarly, high renting rates reduce the effect of the profits, and in the extreme case the best solution of the TTP is the optimum solution for the given TSP component.

We provide a brief example in the following (see Figure 1); full details are given by Polyakovskiy et al (2014a). Each city but the first has an assigned set of items, e.g., city 2 is associated with item I_{21} of profit $p_{21} = 20$ and weight $w_{21} = 2$, and with item I_{22} of profit $p_{22} = 30$ and weight $w_{22} = 3$. Let us assume that the maximum weight $W = 3$, the renting rate $R = 1$ and υ_{max} and υ_{min} are set as 1 and 0.1, respectively. Then the optimum objective value is $Z(\varPi, P) = 50$ when to tour is $\varPi = (1, 2, 4, 3, 1)$ and when items I_{32} and I_{33} are picked up (total profit of 80). As the thief's knapsack has a weight of 2 on the way from city 3 back to city 1, this reduces the speed and results in an increased cost of 15. Consequently, the final objective value is $Z(\varPi, P) = 80 - 1 \cdot (5 + 6 + 4) - 1 \cdot \left(\frac{6}{1 - \frac{1.0 - 0.1}{3} \cdot 2} \right) = 80 - 15 - 15 = 50$.

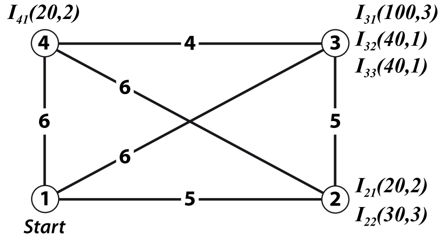


Fig. 1: Illustrative example for a TTP instance (taken from Polyakovskiy et al (2014a), with permission)

2.3 Algorithms for TTP

In the following, we provide a historical overview of approaches to the TTP. As we shall later see, none of these algorithms dominates all others.

In the original article in which the TTP is defined, Bonyadi et al (2013) used exhaustive enumeration on instances with four cities and six items in order to demonstrate the interconnected components. A year later, Polyakovskiy et al (2014a) created a set of instances with up to almost 100,000 cities and 1,000,000 items, rendering exhaustive enumeration no longer feasible.

It were also Polyakovskiy et al (2014a) who proposed the first set of heuristics for solving the TTP. Their general approach was to solve the problem using two steps. The first step involved generating a good TSP tour by using the classical Chained Lin-Kernighan heuristic (Applegate et al 2003). The second step involved keeping the tour fixed and applying a packing heuristic for improving the solution. Their first approach was a simple heuristic (SH) which constructed a solution by processing and picking items that maximized the objective value according to a given tour. Items were picked based on a *score* value that was calculated for each item to estimate how good it is according to the given tour. They also proposed two iterative heuristics, namely the Random Local Search (RLS) and (1+1)-EA, which probabilistically flipped a number of packing bits. After each iteration the solution was evaluated and if an improvement was noted, the changes were kept; otherwise they were ignored.

Bonyadi et al (2014) experimentally investigated the interdependency between the TSP and knapsack components of the TTP. They proposed two heuristic approaches named Density-based Heuristic (DH) and CoSolver. DH is again a two-phased approach similar to SH from Polyakovskiy et al (2014a), and it also ignores any dependencies between the TSP and Knapsack components. In contrast to this, CoSolver is a method inspired by coevolution based approaches. It divides the problem into sub-problems where each sub-problem is solved by a different module of the CoSolver. The algorithm revises the solution through negotiation between its modules. The communication between the different modules and sub-problems allows for the TTP interdependencies to be considered. A comparison across several benchmark problems showed the superiority of CoSolver over DH. This was especially evident for larger instances.

Mei et al (2014b) also investigated the interdependencies between the TSP and knapsack components. They analysed the mathematical formulation to show that the TTP problem is not additively separable. Since the objectives of the TSP and knapsack components are not fully correlated, one cannot expect to achieve competitive results by solving each component in isolation. The authors used two separate approaches for solving the TTP: a cooperative coevolution based approach similar to CoSolver, and a memetic algorithm called MATLS which attempts to solve the problem as a whole. The memetic algorithm, which considered the interdependencies in more depth, outperformed cooperative coevolution. Both works by Bonyadi et al (2014) and Mei et al (2014b) highlight the importance of considering interdependencies between the TTP components as this will allow for the generation of more competitive solutions.

Faulkner et al (2015) investigated multiple operators and did a comprehensive comparison with existing approaches. They proposed a number of operators, such as Bitflip and PackIterative, for optimising the packing plan given a particular tour. They also proposed Insertion for iteratively optimising the tour given a particular packing. They combined these operators in a number of simple (S1–S5) and complex (C1–C6) heuristics that outperformed existing approaches. The main observation was that there does not yet seem to be a single best algorithmic paradigm for the TTP. Their individual operators, however, were quite beneficial in improving the quality of results. While the proposed operators seem to have certain benefits, the simple and complex heuristics did not consider the interdependencies between the TTP components, since all of these approaches were multi-step heuristics. Surprisingly, their best approach was a rather simple restart approach name S5 that combines good TSP tours with the fast PackIterative.

Wagner (2016) recently investigated the use of swarm intelligence approaches with the so-called Max-Min Ant System (MMAS, by Stützle and Hoos (2000)). Wagner investigated the impact of two different TSP-specific local search (ls) operators and of “boosting” TTP solutions using TTP-specific local search. The resulting approaches focus less on short TSP tours, but more on good TTP tours, which can be longer. This allowed them to outperform the previous best approaches MATLS and S5 on relatively small instances with up to 250 cities and 2000 items.

El Yafrani and Ahiod (2016) studied and compared different approaches for solving the TTP from a metaheuristics perspective. Two heuristic algorithms were proposed, including a memetic algorithm (MA2B) and one using simulated annealing (CS2SA). The results show that the new algorithms were competitive to S5 and MATLS on a range of larger TTP instances.

Lastly, we would like to mention that no efficient complete solver for the TTP is known. One of the reasons for this appears to be the fact that even when the tour is kept fixed, packing is NP-hard (Polyakovskiy and Neumann 2015).

Note that most articles so far used the single-objective TTP formulation “TTP1” from Bonyadi Bonyadi et al (2013), however, multi-objective considerations of problems with interconnected components are becoming increasingly popular. For example, Blank et al (2017) investigated a variant of the multi-objective “TTP2”, however, they neglected the value drop effect defined in the original TTP2 and they used their study was limited to self-constructed problem instances. Yafrani et al (2017) created an approach that generates diverse sets of TTP/TTP1 solutions, while being competitive with the state-of-the-art single-objective algorithms. A more general discussion of a multi-objective approach to interconnected problems can be found in Klamroth et al (2017).

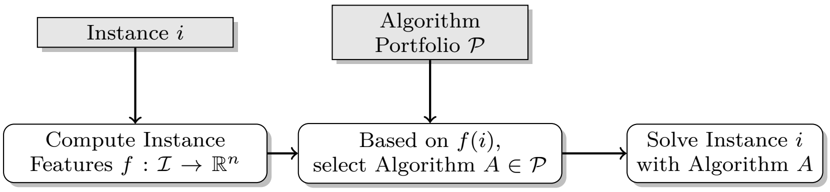

Instance i
Algorithm Portfolio $\mathcal{P}$
Compute Instance Features $f : \mathcal{I} \rightarrow \mathbb{R}^n$
Based on $f(i)$, select Algorithm $A \in \mathcal{P}$
Solve Instance i with Algorithm A

Fig. 2: Workflow of Algorithm Selection

3 Algorithm Selection

As we shall see in Section 4, no algorithm dominates all other algorithms on all instances. One way to exploit this complementarity of the algorithms is to use algorithm selection (Rice 1976; Huberman et al 1997) to select a well-performing algorithm on a per-instance base.

3.1 Problem Statement

The algorithm selection problem is to find a mapping from problem instances $\mathcal{I}$ to algorithms $\mathcal{P}$. This is realized by computing numerical characteristics – so-called *instance features* $f(i)$ – that describe a problem instance $i \in \mathcal{I}$, and then learning a mapping from the resulting feature space to algorithms. Figure 2 shows the general workflow of algorithm selection.

We will describe instance features for the TTP later (in Section 5), but a simple feature is, e.g., the number of cities. Based on these instance features, we will select an algorithm from a portfolio of the 21 TTP algorithms we described in Section 4.2 to solve the instance at hand.

The selection step is typically realized with machine learning methods. Based on gathered training data (i.e., instance features and performance data on training instances), we learn a machine learning model that maps from instance features to a well-performing algorithm.

3.2 Popular Algorithm Selection Approaches

One of the first successful algorithm selection procedures for satisfiability problems (Biere et al 2009) was *SATzilla* (Xu et al 2008). It mainly used two concepts: (i) Learning an empirical performance model (Leyton-Brown et al 2002; Hutter et al 2014) to predict the performance of an algorithm for a given instance and select the algorithm with the best predicted performance; and (ii) Static algorithm schedules (Xu et al 2008; Kadioglu et al 2011; Hoos et al 2015), which run a sequence of algorithms with a runtime budget each. *SATzilla* uses such a static schedule for "pre-solving", to solve easy instances without the overhead of computing features.

Other approaches include

- classification models (e.g., *ME-ASP* by Maratea et al (2014), *3S* by Kadioglu et al (2011), and *CSHC* by Malitsky et al (2013)) that directly learn a mapping from instance features to good algorithms;

- pairwise classification models (e.g., the more recent version of *SATzilla* (Xu et al 2011)), which learns a binary classifier for each pair of algorithms, weighting each training instance by the performance difference between the two algorithms (and thereby emphasizing instances for which the two algorithms' performances differ a lot);
- unsupervised clustering (e.g., *ISAC* by Kadioglu et al (2010)) to partition instances in the feature space into homogeneous subsets and then select the best-performing algorithm of the cluster a new instance is closest to; and
- recommender systems (e.g., Mısır and Sebag (2013)) to recommend an algorithm given only partial training data.

For a thorough overview on algorithm selection procedures, we refer the interested reader to Smith-Miles (2008); Kotthoff (2014).

As was shown in the 2015 ICON challenge on algorithm selection[1], there currently exist two state-of-the-art algorithm selection approaches. The first is the pairwise classification version of *SATzilla* (Xu et al 2011), which won the ICON Challenge. The second is the automatic algorithm selection method *AutoFolio* system (Lindauer et al 2015). *AutoFolio* uses the flexible *FlexFolio* framework (Hoos et al 2014), which combines several different algorithm selection methods, and searches for the best suited algorithm selection approach (and its hyperparameter settings) for a given algorithm selection scenario using algorithm configuration (Hutter et al 2009) via the model-based configurator *SMAC* (Hutter et al 2011). For example, *AutoFolio* determines whether classification or a regression approach will perform better and in case of classification, how to set the hyperparameters of a random forest classifier (Breimann 2001). As shown by Lindauer et al (2015), *AutoFolio* often chooses the pair-wise classification approach of *SATzilla*, but it is more robust than other algorithm selection approaches since it can also switch to other approaches if necessary. As a result, *AutoFolio* established state-of-the-art performance on several different domains in the algorithm selection library (Bischl et al 2016) and performed best on two out of three tracks of the ICON challenge.

In this section, we focused on algorithm selection for hard combinatorial problem solving, since TTP is also a hard combinatorial problem. However, there also exists work on algorithm selection in other fields, such as meta-learning for machine learning algorithms (Vilalta and Drissi 2002; Brazdil et al 2008; Smith-Miles 2008; van Rijn et al 2015).

4 Benchmarking of TTP Algorithms

An important step toward the creation of algorithm portfolios is the conduct of experiments where one determines the performance of algorithms on the available problem instances. To this end, we introduce in this section the originally defined set of TTP instances, and we outline the experimental setup and the results.

[1] `http://challenge.icon-fet.eu/`

4.1 Introduction of Benchmark Instances

For our investigations, we use the set of TTP instances defined by Polyakovskiy et al (2014a).[2] In these instances, the two components of the problem have been balanced in such a way that the near-optimal solution of one sub-problem does not dominate over the optimal solution of another sub-problem.

The characteristics of the original 9,720 instances vary widely. We outline the most important ones in the following:[3]

- The instances have 51 to 85,900 cities, based on instances from the TSPlib by Reinelt (1991).
- For each TSP instance, there are three different types of knapsack problems: *uncorrelated*, *uncorrelated with similar weights* and *bounded strongly correlated* types, where the last type has been shown to be difficult for different types of knapsack solvers by Martello et al (1999); Polyakovskiy et al (2014a). In the uncorrelated case, weights w_{ik} and profits p_{ik} of item k are uniformly distributed random integer values in $\left[1, 10^3\right]$. In the uncorrelated with similar weights case, w_{ik} and p_{ik} are integer values within $\left[10^3, 10^3 + 10\right]$ and $\left[1, 10^3\right]$. In the correlated case, w_{ik} are integer values within $\left[1, 10^3\right]$ and the corresponding profit is $p_{ik} = w_{ik} + 100$.
- For each TSP and KP combination, the number of items per city (referred to as an *item factor*) is $F \in \{1, 3, 5, 10\}$. Note that all cities of a single TTP instance have the same number of items, except for the first city (which is also the last city), where no items are available.
- For each instance, the renting rate R that links both subproblems is chosen in such a way that at least one TTP solution with an objective value of zero exists. This is achieved by the instances' authors Polyakovskiy et al (2014a) by setting each instance's individual renting rate $R = \frac{Z\left(P^{OPT}\right)}{TIME\left(\Pi^{linkern}, P^{OPT}\right)}$, where $Z\left(P^{OPT}\right)$ corresponds to the optimal profit of the KP component and $TIME\left(\Pi^{linkern}, P^{OPT}\right)$ denotes the total traveling time along the near-optimal TSP tour $\Pi^{linkern}$ obtained via the Chained Lin-Kernighan heuristic while picking the items according to the optimal KP component's solution P^{OPT}.
- Lastly, for each TTP configuration of the above-mentioned characteristics 10 different instances exist where the knapsack capacity is varied.

The sheer size of this original TTP instance set makes comprehensive experimental evaluations computationally expensive and the high-dimensional space of characteristics further complicates comparisons. For this reason, different researchers have selected different subsets, with each subset having (intentionally or unintentionally) a particular bias. For example, only the very first article by Polyakovskiy et al (2014a) considered the entire set of 9720 instances. Mei et al (2014a) focused on 30 larger instances with 11849 to 33810 cities. Faulkner et al (2015) covered a wider range using 72 instances with 195 to 85900 cities, and Wagner (2016) used 108 instances with 51 to 1000 cities. Based on these individual and incomplete glimpses at algorithm performance, it is difficult to grasp the full picture.

[2] As available at the TTP project page: `http://cs.adelaide.edu.au/~optlog/research/ttp.php`

[3] For a more detailed description, we refer the interested reader to Polyakovskiy et al (2014b,a).

4.2 Benchmark Results

In order to establish a reliable data set for the subsequent analyses, we run existing TTP algorithms on all 9720 instances. This has the benefit of creating the complete picture using the same hardware and other conditions for the experiments.

As code for most of the TTP algorithms outlined in Section 2.3 is available online, we can consider a wide range of different algorithms, which include constructive heuristics, hill-climbers, problem-agnostic and problem-specific heuristics, single-solution heuristics and cooperative coevolutionary approaches. In the following, we briefly list (in chronological order) the 21 considered algorithms with their original names (and, where applicable, abbreviated names in parentheses):

- Polyakovskiy et al (2014a): SH, RLS, EA
- Bonyadi et al (2014): DH
- Mei et al (2014b): MATLS
- Faulkner et al (2015): S1, S2, S3, S4, S5, C1, C2, C3, C4, C5, C6
- El Yafrani and Ahiod (2016): CS2SA
- Wagner (2016): MMASls3 (M3), MMASls4 (M4), MMASls3boost (M3B), MMASls4boost (M4B).

We run all algorithms for a maximum of 10 minutes per instance. All computations are performed on machines with Intel Xeon E5430 CPUs (2.66GHz) and Java 1.8.

We would like to note that the computation budget is motivated by a real-world scenario. For a real-world decision maker who is interested in what-if analyses, 10 minutes correspond to about a cup of coffee. After 10 minutes, the next results are available and the decision maker can make the next change(s) to the system to investigate alternatives. While it would be possible to deviate from the 10 minutes in the present study, this time limit is very often used in TTP research. This includes the development of specialized approaches to subsets of instances, as mentioned in Section 2.3.

As the encountered objective scores cover several orders of magnitude, as well as positive and negative scores, we assess the quality of the algorithms using the following approach. For each TTP instance, we determine the best and the worst objective scores of the final solutions obtained by all the compared algorithms; these two values define the boundaries of the interval of observed objective scores for each instance. We then map actual scores from this interval linearly to $[0, 1]$, where the highest score is equivalent to 1. In case an algorithm did not produce a score for a particular instance, e.g. due to a time-out or crash, we assign to it the score of -1.

First, we report a performance overview across all 9720 instances in Figure 3, where we ignore runs which did not produce a solution given the time limit. At first sight, it appears that many algorithms perform comparably, since 18 of 21 algorithms achieve a median scaled performance of > 0.80. However, this is largely because DH and SH often performed rather poorly and thus skew the scale. Figure 4 shows the full results, which now include the consideration of unsuccessful runs, and we average the results. As we can immediately see, several algorithms produce good results if they produce solutions at all given the time limit. Still, 14 algorithms have an average rescaled performance of 0.8 or higher. In particular, the following algorithms did not always produce solutions given the time limit: MATLS (204 unsolved instances), M3 (721), M4 (720), M3B (1342), M4B (1316), and CS2SA (6284). To the best of our

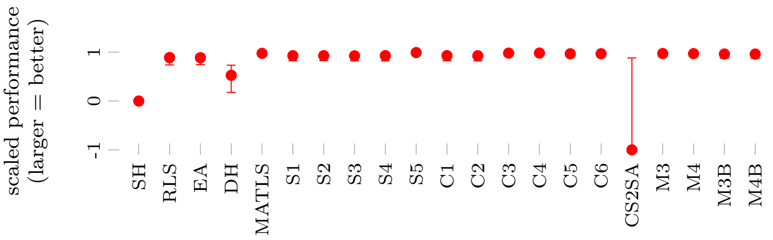


Fig. 3: Scaled median performance (± 1st/3rd quartile) of all 21 algorithms on all 9720 instances, not considering unsuccessful runs that were assigned a score of −1.

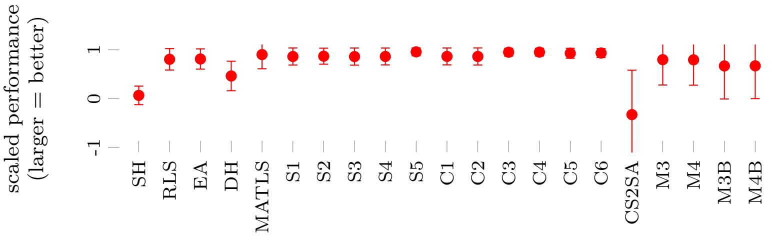


Fig. 4: Scaled average performance (± standard deviation) of all 21 algorithms on all 9720 instances, considering unsuccessful runs that were assigned a score of −1.

knowledge, the first five of these suffer from long subroutines that keep them from stopping after the time limit is reached, while CS2SA crashes on these instances. We also note that the algorithms starting with M dominate on smaller instances, and that S5 performs well on larger instances.

These figures provide only a first indication, since the instance set they are based on contains many small instances, which biases this performance comparison such that algorithms performing well on small instances are favored. We also note that the algorithms starting with M dominate on smaller instances, and that S5 performs well on larger instances.

Since none of our algorithms has a perfect score of 1 on average, we have already a first indication that the algorithms are complementary and algorithm selection could potential improve the performance further. More detailed analyses will be presented later.

We have made the performance data set publicly available: CSV format at `http://cs.adelaide.edu.au/~optlog/research/ttp.php`.

5 Instance Features for the TTP

For our approach to algorithm portfolio generation, in addition to algorithm performance data (see previous section) we also need data that describes problem instances. In total we consider 55 TTP instance features. Of these, 47 are TSP features from previous studies on TSP (Mersmann et al 2012, 2013; Nallaperuma et al 2013a,b, 2014, 2015). The R package tspmeta Mersmann et al (2013)[4] is used to calculate these features.

The features fall into 10 groups, which we outline in the following, with the running feature IDs in parentheses. Throughout the following description we refer to basic statistical features with their common names. For a distribution of n variables x_i the mean $\overline{x}$ is the average value of a distribution described by $1/n \cdot \sum_{i=1}^{n} x_i$. The mode represents the value with maximum frequency. The median describes the $\frac{(n+1)}{2}^{th}$ value in a sorted sequences and the standard deviation is calculated using $\sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^{n}(x - \overline{x})^2}$.

Distance Features (1-11). These are based on summary statistics of the edge cost distribution. Here, we consider the lowest, highest, mean and median edge costs, as well as the proportion of edges with distances shorter than the mean distance, the fraction of distinct distances (i.e. different distance levels), and the standard deviation of the distance matrix. Also, we consider the mode frequency, quantity and mean. Mode frequency is calculated by counting the number of occurrences. The mode quantity describes the ratio of number of mode values to the total number of edges. For this we simply count the number of occurrences. Finally, we used the expected tour length for a random tour, given by the sum of all edge costs multiplied by $2/(N-1)$.

Mode Features (12). As an additional feature characterizing the distribution of edge costs, we also include its number of modes as a feature. Here, the mode is estimated from the real valued distribution of edge weights by considering the edge cost values having a probability mass above 0.01.

Cluster Features (13-18). The clusters describe the sets of cities gathered together in the Euclidean space. GDBSCAN is used for clustering where reachability distances of 0.01, 0.05 and 0.1 are chosen. The reachability distance describes the radius of the neighborhood to be considered for each clusterization. Derived features are the number of clusters and the mean distances to the cluster centroids.

Nearest Neighbor Distance Features (19-24). Nearest-neighbor features describe the distribution of distances between each city and its nearest neighbour. Uniformity of an instance is reflected by the minimum, maximum, mean, median, standard deviation and the coefficient of variation of the normalized nearest-neighbor distances (nnd) of each node.

Centroid Features (25-29). The centroid defines the arithmetic mean of the positions of the points (cities) on the plane. The x- and y-coordinates of the instance centroid together with the minimum, mean and maximum distance of the nodes from the centroid.

[4] `https://cran.r-project.org/web/packages/tspmeta/`

MST Features (30-40). A spanning tree for a graph $G = (V, E)$ is the subset of edges $E\prime \subset E$ such that spans across all vertices $(\exists u \in V : (u, v) \in E\prime V(v, u) \in E\prime \forall v \in V)$ and has no cycles $(|E\prime| \leq |V| - 1)$. The tree which has minimum weight across all such trees is the minimum spanning tree (MST). We first calculate the MST for the complete graph induced by cities and the distances of the TSP instance. Statistics which characterize the depth and the distances of the MST comprise the MST features. These include MST depth distribution consists of the depth of the nodes of MST. The minimum, mean, median, maximum and the standard deviation of the depth and distance values of the MST as well as the sum of the distances on the MST (which we normalize by diving it by the sum of all pairwise distances).

Angle Features (41-45). This feature group comprises statistics of the distribution of angles between a node and its two nearest neighbor nodes: the minimum, mean, median, maximum and standard deviation.

Convex Hull Features (46-47). For set of points S in Euclidean space the convex hull is defined as the set of all convex combinations of points S. In a convex combination, each point $x_i \in s$ is assigned a weight or coefficient α_i in such a way that the coefficients are all non-negative and sum to one, and these weights are used to compute a weighted average of the points. Mathematically, this can be expressed as $Conv(S) = \left\{ \sum_{i=1}^{|S|} \alpha_i x_i | (\forall i : \alpha_i \geq 0) \wedge \sum_{i=1}^{|S|} \alpha_i = 1 \right\}$. The area of the convex hull of the instance reflects the "spread" of the instance in the plane. Additionally, we compute the fraction of nodes which define the convex hull.

In addition to these 47 existing TSP-specific features, we considered the following eight new features.

Number of Cities (48). We also consider this obvious feature, which is not computed by the tspmeta T package.

Knapsack Features (49-52). These include the capacity of the knapsack, the knapsack type, the total number of items, and the number of items per city.

Traveling Thief Features (53-55). Lastly, as TTP-specific features we have the renting ratio R, the minimum travel speed v_{min} and the maximum travel speed v_{max}.

It is important to note that these eight new features do not require any processing, as they are part of the definition of the instances.

Note that we are not considering the values of min/max/average/standard deviation/... of the items' properties, as they are effectively identical across all instances due to the way the knapsacks were created (see Section 4) and due to the large number of items. However, if there was variation in these properties we would expect such summary statistics to be very useful.

Future investigations should include additional TTP-specific features. A first step towards this has been taken by Polyakovskiy and Neumann (2015) with their concept of "profitable/unprofitable" items for the special case when tours are fixed. However, since we are not considering this restriction, their concept does not easily carry over.

Note that we have made the full set of feature values available online: `http://cs.adelaide.edu.au/~optlog/research/ttp.php`.

To investigate whether our instance features can discriminate our instances well and will enable algorithm selection later on, we visualized the instances in the instance feature space by using a principal component analysis (PCA) to 2 dimensions, and we study the algorithms' footprints and instance hardness as motivated by Smith-Miles et al (2014). In Figures 5a and 5b, we marked all instances with red that can be solved with a score of at least 0.99 (i.e., 1% gap to the optimal score). As expected by the results shown in Figure 4, S5 performs very well on most of the instances, but it does not perform well on two clusters of instances (in the upper part of Figure 5a). In contrast, CS2SA can solve these instances reasonable well, see Figure 5b. Furthermore, in Figure 5c we encoded the instance hardness, i.e., how many algorithms obtained at least a score of 0.99 on an instance. We see that many instances can be solved equally well by many algorithms (colors green to red). However, we have some instance clusters on which only a few algorithms perform well—these will be the crucial instances in the following algorithm selection study.

6 Experimental Study of Algorithm Selection on TTP

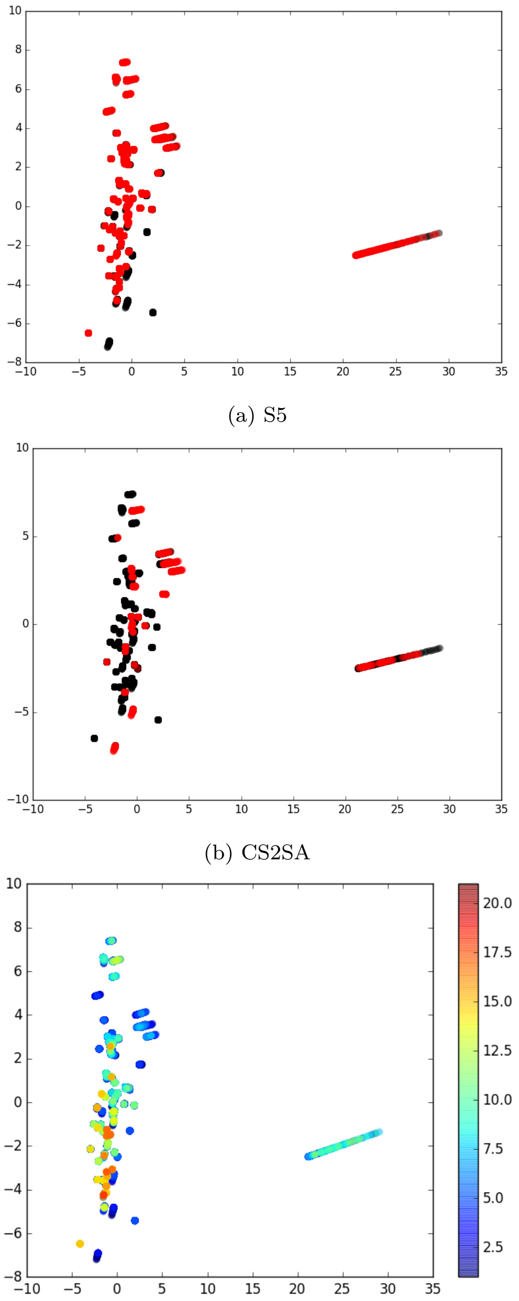
We follow the approach of Hoos et al (2014) by studying the performance of different, well-known algorithm selection approaches. In detail, we ran *FlexFolio*[5] (using Python 2.7.6 and sklearn 0.14.1) with various approaches which *simulate* the behavior of existing systems: *SATzilla'09* (regression approach), *SATzilla'11* (cost-sensitive pairwise-classification), *ISAC* (clustering) and *3S* (direct classification with $k = 32$ nearest neighbors). In contrast to the original implementations of *SATzilla'11* and *3S*, we do not use static (pre-solving) schedules in our experiments, because their schedule computation implicitly assumes that the performance metric to be optimized is runtime. Since our performance metric is not runtime, we cannot compute such schedules and hence, we focus on the classical algorithm selection approach of selecting one algorithm per instance.

To this end, we created an algorithm selection benchmark scenario in the format of the algorithm selection library (ASlib; Bischl et al (2016)) from our TTP benchmark data.[6] This ASlib scenario includes the performance values for all our algorithms and the instance features for each instance. Furthermore, it also provides the splits for a 10-fold cross validation to obtain an unbiased performance estimate (i.e., the instances are randomly split into 10 equally sized sets and in each iteration, one of the splits is used as a test set to validate our algorithm selection procedure and all others are used to train the algorithm selector; the overall performance of an algorithm selection procedure is then the average performance across all iterations). With all this information saved, our ASlib scenario allows for hardware-independent reproducibility of our experiments.

Table 1 shows the performance of the different approaches on TTP, demonstrating that existing algorithm selection approaches work very well for this benchmark. Our baseline is the performance of the *single best* algorithm, i.e., always using the algorithm that performs best across all instances. The *single best* algorithm with a performance of 0.959 is S5 (as previously shown in Section 4). Due to the scaling of the objective scores, the best possible score on each instance is 1. Therefore, the

[5] `http://www.ml4aad.org/flexfolio/`

[6] See `TTP-2016` at `http://www.aslib.net`.



(a) S5

(b) CS2SA

(c) Instance Hardness

Fig. 5: Visualizing the 9720 TTP instances in the feature spaces using a PCA to 2 dimensions. Figures 5a and 5b show all instances in red that can be solved with a score of at least 0.99 by S5 and CS2SA, respectively. Figure 5c encodes the number of algorithms that obtain a score of at least 0.99 (i.e., 1% gap to the optimal score).

Simulated System	Approach	Performance
Single Best (S5)	Baseline	0.959
Oracle	Theoretical Optimum	1.0
SATzilla'09-like	Regression (Lasso-Regression)	0.966
SATzilla'11-like	Pairwise Classification (RF)	0.993
ISAC-like	Clustering (k-means)	0.989
3S-like	Classification (k-NN)	0.992

Table 1: Comparing different algorithm selection approaches on TTP

theoretical optimal performance of a perfect algorithm selection procedure (the so-called *oracle* or virtual-best solver) is also 1 here. Formally, this oracle performance is defined as

$$\text{Oracle}(\mathcal{P}, \mathcal{I}) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \max_{A \in \mathcal{P}} m(A, i), \qquad (1)$$

where $A \in \mathcal{P}$ are all algorithms in our portfolio of TTP algorithms, $i \in \mathcal{I}$ are TTP instances, and m is our performance metric (which has to be maximized here).

The best-performing algorithm selection approaches are the ones of *SATzilla'11* and *3S* with a nearly optimal performance of above 0.99. This closes the performance gap between the single best solver and the oracle by almost 90%. *SATzilla'09* and *ISAC* also outperformed the *single best*. One possible reason for the good performance of algorithm selection for this application is the large instance set, as most other instance sets studied in algorithm selection only consist of hundreds or a few thousand instances (cf. ASlib by Bischl et al (2016)). The resulting availability of more data makes the machine learning problem easier.

We also ran the fully automated *AutoFolio* approach (see Section 3.2) for a day of wallclock time on 4 cores to automatically determine a good portfolio. Since the best *FlexFolio* approach (i.e., *SATzilla'11*) already performed well, *AutoFolio* was only able to improve performance further by a very small margin in the 4th decimal. In fact, *AutoFolio* also decided for the *SATzilla'11* approach and only changed the hyperparameters of the underlying random forest slightly.

A direction for an extension of this portfolio generation, in particular for real-world uses where time budgets are hard constraints or are hard to establish beforehand, could be to consider the actual runtimes, set a budget on the sum of all runtimes and permit more than one algorithm to be executed. For example, van Rijn et al (2015) applied a similar idea in the context of selecting between classification algorithms.

7 Analysis of Algorithm Complementarity

A necessary requirement for algorithm selection to perform well is that the portfolio of algorithms is complementary on the used instances. A first indicator for the complementarity of the portfolio is the difference between the *single best* algorithm and the *oralce* performance (see Section 6). This absolute difference of 0.041 may appear small, but in many optimization problems it is quite easy to achieve 4.1 percent sub-optimality, and only the last few percent are hard to obtain. Furthermore, we note

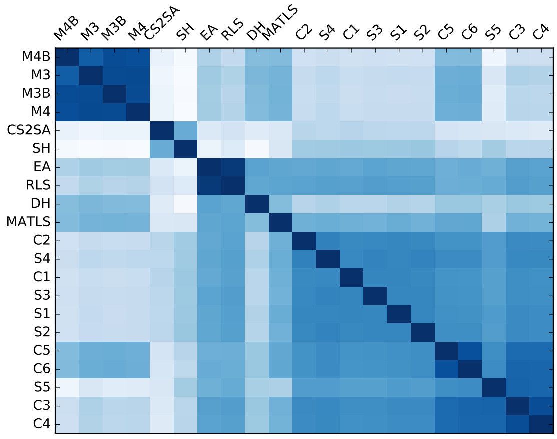


Fig. 6: Spearman rank coefficients in a heatmap (dark fields correspond to large correlation). The algorithms are sorted by hierarchical clustering (see Figure 7).

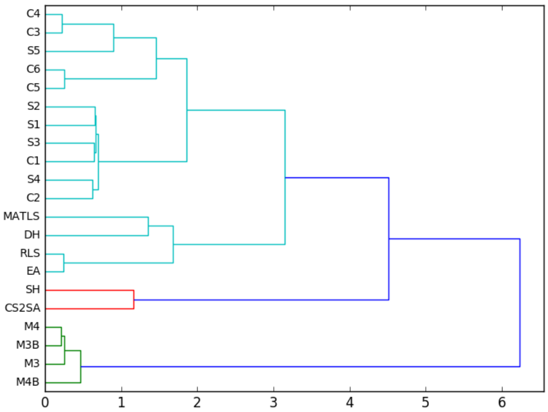


Fig. 7: Dendogram to show hierarchical clustering of algorithms (using Ward's method) according to Spearman correlation coefficient.

that 18 of 21 algorithms achieved median scaled objective scores of > 0.8, and that several algorithms do not produce any solutions on certain TTP instances given the time limit.

Figure 6 shows the performance correlation across instances (based on Spearman correlation coefficients) between all pairs of algorithms. This figure shows that the algorithms form clusters that reflect their historical development. For example, C* and S* fall into one cluster (all use the same fast packing heuristic), the ant-colony approaches M* form one cluster, and early hill-climbers EA with RLS form another one. Figure 7 shows a few more details. For example, C3/4/5/6 and S5 are grouped together at the top; their common features are that they use the same tour generation method, the same packing, and some way to restart the algorithm. Within this cluster, C3/4 are clustered together, since they are virtually identical (with just a minor difference in the search operators), and C5/6 are clustered since they are restart variants of C3/4. The algorithms CS2SA, SH, DH and MATLS are complementary to all other algorithms (which is why the dendogram only groups them with other algorithms further to the right, compared to, for example, the RLS/EA-grouping). We note that this analysis only provides insights about the similarity of algorithms, but it is not a sufficient indicator about the applicability of algorithm selection since one of the algorithms could still dominate all other algorithms.

Another approach of assessing complementarity of algorithms is the *marginal contribution* (MC) to the oracle performance (Xu et al 2012), i.e., how much the oracle performance of an existing portfolio will be improved by adding a new algorithm to it:

$$\mathrm{MC}(A, \mathcal{P}, \mathcal{I}) = \mathrm{Oracle}(\mathcal{P} \cup \{A\}, \mathcal{I}) - \mathrm{Oracle}(\mathcal{P}, \mathcal{I}). \tag{2}$$

This approach has the disadvantage of being strongly dependent on a fixed portfolio. To get a broader overview of an algorithm's contribution, an extension of the marginal contribution analysis consists of using *Shapley values* (SV; Frechette et al (2016)), i.e., the marginal contribution of an algorithm to any subset of the algorithm portfolio[7]:

$$\mathrm{SV}(A, \mathcal{P}, \mathcal{I}) = \frac{1}{2^{|\mathcal{P}|}} \sum_{\mathcal{P}' \in 2^{\mathcal{P}}} \mathrm{MC}(A, \mathcal{P}', \mathcal{I}). \tag{3}$$

Even though Shapley values are defined as a sum over all possible portfolio subsets, they can be efficiently computed in polynomial time by representing them as marginal contribution networks (Chalkiadakis et al 2011; Frechette et al 2016).

Figure 8 shows the ranking of the different algorithms based on their average performance (i.e., running only one algorithm on all instances), the Shapley values, and the marginal contribution to the oracle performance. S5 has the highest standalone performance and the highest Shapley value, but surprisingly it is only ranked second with respect to marginal contribution. Hence, S5 is a very important algorithm as a standalone and in smaller portfolios, but it does not contribute as much on top of the combination of the other algorithms as algorithm CS2SA does (which has the lowest standalone performance and Shapley value but the highest marginal contribution). This demonstrates that CS2SA, despite its poor *average* performance, performs very well on a subset of instances – and reliably enough so for the algorithm portfolio to exploit this. Once the algorithmic or implementation issues of CS2SA are fixed we expect to see significantly better average performance by this algorithm. The complex approaches C1-C6 perform well on average and they can contribute to portfolios (see their ranks in the Shapley values), but their individual (marginal) contributions are low due to their overall similarity. In contrast to this, MATLS is a good contributor to portfolios, not only because it shares hardly any roots with the other algorithms, but also because it performs well on average. The ant-colony approaches M* do not perform too well on average, but they can make useful contributions to algorithm portfolios since they perform very well on small instances. Lastly, on the low end of the performance spectrum are the two constructive heuristics DH/SH and the two uninformed hill-climbers RLS/EA. While these performed

[7] We use the worst possible performance as the performance of an empty portfolio: $\mathrm{Oracle}(\{\}, \mathcal{I}) = -1$.

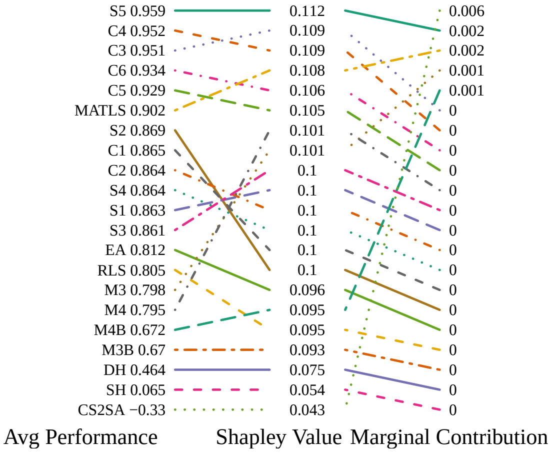


Fig. 8: Standalone performance, Shapley values and contributions to the oracle for all 9720 instances. The lines indicate the ranking of the algorithms using the different metrics. For example, S5 (dark green) has the highest standalone performance and Shapley value, but it is only ranked second in marginal contribution.

TTP Algorithm	Selection Frequency	TTP Algorithm	Selection Frequency
S5	0.38	C6	0.02
M4B	0.16	C4	0.02
M3	0.12	C3	0.01
M3B	0.12	M4	0.01
CS2SA	0.09	C5	0.01
MATLS	0.06		

Table 2: Selection frequency of all algorithms that were selected on at least 1% of the instances, using the *SATzilla'11* approach.

reasonably well when they were introduced, they have since then been outclassed by more informed approaches. But even the slightly informed approaches S1/S2/S3/S4, which use good TSP tours and TTP-specific packing operators, are not competitive anymore when being compared to more recent developments.

To verify that our algorithm selectors indeed exploit the complementarity of our TTP algorithms, we also investigated the frequency of how frequently an algorithm is selected using the *SATzilla'11* approach; the results are presented in Table 2. As expected, S5, as the overall best algorithm and the algorithm with the highest Shapley value, was selected most frequently. Since the M* algorithms perform well on small instances (as argued above), these algorithms are also selected quite often (M4B + M3 + M3B = 40%). The last group of frequently-selected algorithms consists of CS2SA and MATLS, which is consistent with our previous analysis.

In summary, we can see that well-performing algorithm portfolios include problem-solving approaches of different complexities in order to deal with the wide range of existing TTP instances: there are swarm-intelligence approaches for small instances, memetic and multi-step heuristics for mid-size instances, and for the large instances the relatively simple restart approach S5 is a good choice.

8 Analysis of Feature Importance and Feature Calculation Time

As the calculation of instance features forms an important step in the application of algorithm portfolios, we review the necessary calculation times in the following. In addition, we analyze which features are the most important ones for algorithm selection and we investigate how subsets of features impact computation time and portfolio performance.

To date, the established computation budget for TTP benchmarking is 10 minutes single-core CPU time per instance. For algorithm selection to be effective, and if only a single algorithm is to be run once, the calculation time of the instance features should not take up a large proportion of these 10 minutes. However, several of the features are computationally costly, for example, because for some of them a complete distance matrix has to be generated, or because a clustering algorithm needs to be run. As a consequence, the calculation time of all 55 features for a single given instance ranges from a few seconds for the smallest TTP instances to hours for the largest ones we considered; for example, the calculations for the eil51* instances take about 2 seconds, those for the pla7397* instances are approaching 10 minutes, and the calculations for the pla33810* instances even exceed 20 hours. While we computed all these features for our analyses, many of them are clearly too expensive for algorithm selection.

To investigate which features are actually needed, we compute the Gini importance (Breimann 2001) for each of the 55 features, averaged across all pair-wise random forests models. The results are shown in Figure 9, revealing that only a small portion of the TTP features actually matter. Interestingly, these are mostly basic knapsack features:

1. CAPACITYOFKNAPSACK: the KP feature defining the knapsack capacity.
2. RENTINGRATIO: the TTP feature that connects the KP and the TSP.
3. NUMBEROFITEMS: the KP feature stating the total number of available items.
4. KNAPSACKDATATYPE: the KP feature stating the knapsack type.
5. DIMENSION: the TSP feature stating the total number of cities.

As the previous portfolio investigations in Section 6 used all 55 features, we now repeat the algorithm selection experiment using only the most important features and our best-performing approach from *SATzilla'11*. The resulting performances are 0.977, 0.980, 0.986, 0.988, and 0.992 (going from using only the most important feature to using the five most important ones). These results show that with just a small subset of the features we can achieve a portfolio performance comparable to the best one from Section 6 (0.993).

Remarkably, all of these five most important features are given in the instance file's header, and are thus “computable” in constant time. Out of these five, CAPACITYOFKNAPSACK and RENTINGRATIO need to be defined by the instance. If NUMBEROFITEMS or DIMENSION are missing, then they can be computed by going through the instance file once and counting the total numbers of items or cities. KNAPSACKDATATYPE is not a computable feature, as it is a parameter that was used in the generation of the instance; for our considered instance set, however, this

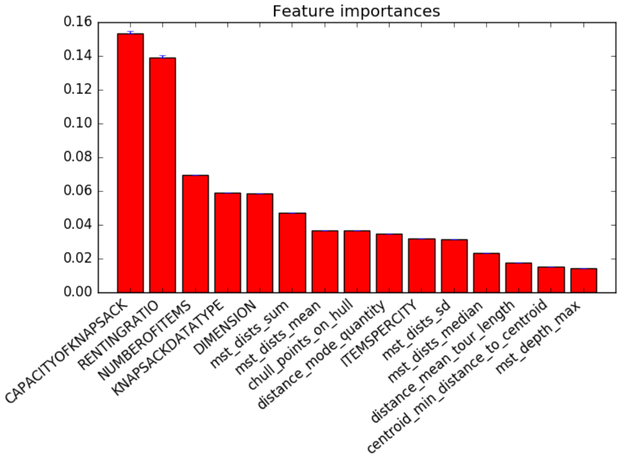


Fig. 9: Average feature importances of the top 15 features based on Gini importance across all pair-wise random forest models. The error-bars indicate the 25th and 75th percentile.

field is always provided. Even if it is not considered, for example when using only the three most important features, we still achieve a performance of 0.986, which is a substantial improvement over the baseline approach S5 (0.959).

From these experiments we see that the exploitation of immediately available instance features results in a substantial average performance increase that is comparable to a significantly more time-consuming one that requires the calculation of all 55 features.

The question now is whether we can learn even more from these outcomes. The visualization and interpretation of the raw outputs of the portfolios is challenging due to the large numbers of instances, features, and randomized algorithms. Nevertheless, let us briefly consider as an example the portfolio when only the feature CAPACITYOFKNAPSACK is used. Let us sort the 9720 instances according to their CAPACITYOFKNAPSACK values, and let us now consider the list of algorithms as they are selected. As expected, this list contains long (but not always continuous) stretches where the same algorithms are selected; in particular, the M* algortihms dominate on the tiny instances, and S5 dominates on mid-sized and large instances. If we do the same ordering for the algorithm selector that uses the five most important features, then the overall picture changes slightly. On the smallest ∼3000 instances, different complex algorithms dominate, and for the tiniest these are often the M* approaches which tend to generate the longest tours. The largest ∼3000 instances are typically assigned to either CS2SA (a fast implementation of search operators) or S5 (resampling solutions), which are two very different approaches.

9 Concluding Remarks

In this article, we presented the first study of algorithm portfolios for the TTP. We first studied the performance of 21 existing TTP algorithms on the full original set of 9720 TTP instances created by Polyakovskiy et al (2014a) and defined 55 instance features for TTP. Then, we studied various different approaches for the resulting algorithm selection problem, showing very substantial improvements over the single best algorithm and closing the gap between it and an omniscient oracle by 90%. Finally, we studied which algorithms contribute most to the portfolio, finding that the algorithms with best average performance (e.g. the complex ones C3–C6 and MATLS, and the swarm-intelligence approaches that start with M) were quite important for the portfolio because of their performance on small and mid-sized TTP instances. Interestingly, the relatively simple heuristic S5 continues to dominate in particular on the large TTP instances and thus is one of the most important contributors to well-performing portfolios. Despite this general trend, the algorithm with the worst average performance, CS2SA, added substantial benefit on top of all other algorithms. An analysis of the feature importance revealed that the values for the five most important features can be extracted from the instance definition in constant time. The resulting portfolio that uses only this subset has a performance comparable to the one that uses all 55 features (which can take hours to compute).

Our future work will largely focus on two directions:

1. *Features.* We aim to create more TTP-specific features, to study which features make TTP instances hard for which algorithms and why, and to explore whether we can identify a smaller representative subset of TTP instances to speed up future benchmarking studies.
2. *Computation Time.* As the time budget of 10 minutes can be seen as rather artificial, we plan to extend our investigations to a wide range of computation budgets, in order to uncover the varying benefits of different algorithms to different portfolios. Also, we plan to further focus on considering the actual runtimes of the algorithms inside the algorithm selection, as more than one algorithm can potentially be run if the computation budget allows it.

References

Applegate D, Cook WJ, Rohe A (2003) Chained Lin-Kernighan for large traveling salesman problems. Journal on Computing 15(1):82–92

Beasley EJ (1990) Or-library: Distributing test problems by electronic mail. Journal of the Operational Research Society 41(11):1069–1072

Bell JE, McMullen PR (2004) Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics 18(1):41 – 48

Biere A, Heule M, van Maaren H, Walsh T (eds) (2009) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol 185. IOS Press

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

Blank J, Deb K, Mostaghim S (2017) Solving the Bi-objective Traveling Thief Problem with Multi-objective Evolutionary Algorithms, Springer, pp 46–60

Bonyadi MR, Michalewicz Z, Barone L (2013) The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In: Congress on Evolutionary Computation, IEEE, pp 1037–1044

Bonyadi MR, Michalewicz Z, Przybylek MR, Wierzbicki A (2014) Socially inspired algorithms for the TTP. In: Genetic and Evolutionary Computation Conference, ACM, pp 421–428

Bonyadi MR, Michalewicz Z, Neumann F, Wagner M (2016) Evolutionary computation for multicomponent problems: opportunities and future directions. CoRR abs/1606.06818, URL http://arxiv.org/abs/1606.06818

Brazdil P, Giraud-Carrier C, Soares C, Vilalta R (2008) Metalearning: Applications to Data Mining, 1st edn. Springer Publishing Company, Incorporated

Breimann L (2001) Random forests. Machine Learning Journal 45:5–32

Chalkiadakis G, Elkind E, Wooldridge M (2011) Computational Aspects of Cooperative Game Theory. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers

Chand S, Wagner M (2016) Fast heuristics for the multiple traveling thieves problem. In: Genetic and Evolutionary Computation Conference (GECCO), ACM, pp 293–300

Dantzig GB, Ramser JH (1959) The truck dispatching problem. Management Science 6(1):80–91

El Yafrani M, Ahiod B (2016) Population-based vs. single-solution heuristics for the travelling thief problem. In: Genetic and Evolutionary Computation Conference (GECCO), ACM, pp 317–324

Faulkner H, Polyakovskiy S, Schultz T, Wagner M (2015) Approximate approaches to the traveling thief problem. In: Genetic and Evolutionary Computation Conference, ACM, pp 385–392

Frechette A, Kotthoff L, Rahwan T, Hoos H, Leyton-Brown K, Michalak T (2016) Using the Shapley Value to Analyze Algorithm Portfolios. In: 30th AAAI Conference on Artificial Intelligence

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

Hoos H, Kaminski R, Lindauer M, Schaub T (2015) aspeed: Solver scheduling via answer set programming. Theory and Practice of Logic Programming 15:117–142

Huberman B, Lukose R, Hogg T (1997) An economic approach to hard computational problems. Science 275:51–54

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

Hutter F, Hoos H, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Coello C (ed) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11), Springer-Verlag, Lecture Notes in Computer Science, vol 6683, pp 507–523

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

Kadioglu S, Malitsky Y, Sellmann M, Tierney K (2010) ISAC - instance-specific algorithm configuration. In: Coelho H, Studer R, Wooldridge M (eds) Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI'10), IOS Press, pp 751–756

Kadioglu S, Malitsky Y, Sabharwal A, Samulowitz H, Sellmann M (2011) Algorithm selection and scheduling. In: Lee J (ed) Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP'11), Springer-Verlag, Lecture Notes in Computer Science, vol 6876, pp 454–469

Klamroth K, Mostaghim S, Naujoks B, Poles S, Purshouse R, Rudolph G, Ruzika S, SayÄśn S, Wiecek MM, Yao X (2017) Multiobjective optimization for interwoven systems. Journal of Multi-Criteria Decision Analysis In print.

Koch T, Achterberg T, Andersen E, Bastert O, Berthold T, Bixby RE, Danna E, Gamrath G, Gleixner AM, Heinz S, Lodi A, Mittelmann H, Ralphs T, Salvagnin D, Steffy DE, Wolter K (2011) MIPLIB 2010. Mathematical Programming Computation 3(2):103–163

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

Laporte G (1992) The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59(3):345 – 358

Leyton-Brown K, Nudelman E, Shoham Y (2002) Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Hentenryck PV (ed) Principles and Practice of Constraint Programming - CP 2002, Springer, Lecture Notes in Computer Science, vol 2470, pp 556–572

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

Malitsky Y, Sabharwal A, Samulowitz H, Sellmann M (2013) Algorithm portfolios based on cost-sensitive hierarchical clustering. In: Rossi F (ed) Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13), pp 608–614

Maratea M, Pulina L, Ricca F (2014) A multi-engine approach to answer-set programming. Theory and Practice of Logic Programming 14:841–868

Martello S, Pisinger D, Toth P (1999) Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science 45(3):414–424

Mei Y, Li X, Yao X (2014a) Improving efficiency of heuristics for the large scale traveling thief problem. In: Simulated Evolution and Learning, LNCS, vol 8886, Springer, pp 631–643

Mei Y, Li X, Yao X (2014b) On investigation of interdependence between sub-problems of the TTP. Soft Computing 20(1):157–172

Mersmann O, Bischl B, Bossek J, Trautmann H, Wagner M, Neumann F (2012) Local search and the traveling salesman problem: A feature-based characterization of problem hardness. In: Hamadi Y, Schoenauer M (eds) Learning and Intelligent Optimization: 6th International Conference (LION 6), Springer, pp 115–129

Mersmann O, Bischl B, Trautmann H, Wagner M, Bossek J, Neumann F (2013) A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. Annals of Mathematics and Artificial Intelligence 69(2):151–182

Michalewicz Z (2012) Ubiquity symposium: Evolutionary computation and the processes of life: The emperor is naked: Evolutionary algorithms for real-world applications. Ubiquity 2012(November):3:1–3:13

Michalewicz Z, Fogel DB (2004) How to solve it - modern heuristics: second, revised and extended edition (2. ed.). Springer

Mısır M, Sebag M (2013) Algorithm selection as a collaborative filtering problem. Tech. rep., INRIA-Saclay, URL `http://hal.inria.fr/hal-00922840`

Nallaperuma S, Wagner M, Neumann F (2013a) Ant colony optimisation and the traveling salesperson problem: Hardness, features and parameter settings. In: Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation, ACM, New York, NY, USA, GECCO '13 Companion, pp 13–14

Nallaperuma S, Wagner M, Neumann F, Bischl B, Mersmann O, Trautmann H (2013b) 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 XII, ACM, New York, NY, USA, FOGA XII '13, pp 147–160

Nallaperuma S, Wagner M, Neumann F (2014) Parameter prediction based on features of evolved instances for ant colony optimization and the traveling salesperson problem. In: Parallel Problem Solving from Nature PPSN XIII, LNCS, vol 8672, Springer, pp 100–109

Nallaperuma S, Wagner M, Neumann F (2015) Analyzing the effects of instance features and algorithm parameters for max min ant system and the traveling salesperson problem. Frontiers in Robotics and AI 2(18)

Polyakovskiy S, Neumann F (2015) Packing while traveling: Mixed integer programming for a class of nonlinear knapsack problems. In: Integration of AI and OR Techniques in Constraint Programming, LNCS, vol 9075, Springer, pp 330–344

Polyakovskiy S, Bonyadi MR, Wagner M, Michalewicz Z, Neumann F (2014a) A comprehensive benchmark set and heuristics for the traveling thief problem. In: Genetic and Evolutionary Computation Conference, ACM, pp 477–484

Polyakovskiy S, Bonyadi MR, Wagner M, Michalewicz Z, Neumann F (2014b) TTP Test Data. See `http://cs.adelaide.edu.au/~optlog/research/ttp.php`

Reinelt G (1991) TSPLIB - A Traveling Salesman Problem Library. ORSA Journal on Computing 3(4):376–384

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

van Rijn J, Abdulrahman S, Brazdil P, Vanschoren J (2015) Fast algorithm selection using learning curves. In: Fromont É, Bie TD, van Leeuwen M (eds) Proceedings of the international symposium on Advances in Intelligent Data Analysis (IDA), Springer, Lecture Notes in Computer Science, vol 9385, pp 298–309

Rizzoli AE, Montemanni R, Lucibello E, Gambardella LM (2007) Ant colony optimization for real-world vehicle routing problems. Swarm Intelligence 1(2):135–151

Smith-Miles K (2008) Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys 41(1)

Smith-Miles K, Baatar D, Wreford B, Lewis R (2014) Towards objective measures of algorithm performance across instance space. Computers & OR 45:12–24

Stützle T, Hoos HH (2000) MAX-MIN ant system. Journal of Future Generation Computer Systems 16:889–914

Vilalta R, Drissi Y (2002) A perspective view and survey of meta-learning. Artif Intell Rev 18(2):77–95

Wagner M (2016) Stealing Items More Efficiently with Ants: A Swarm Intelligence Approach to the Travelling Thief Problem, Springer, Cham, pp 273–281

Weise T, Zapf M, Chiong R, Nebro AJ (2009) Why is optimization difficult? In: Chiong R (ed) Nature-Inspired Algorithms for Optimisation, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 1–50

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

Xu L, Hutter F, Hoos H, Leyton-Brown K (2011) Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In: RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion at the International Joint Conference on Artificial Intelligence (IJCAI)

Xu L, Hutter F, Hoos H, Leyton-Brown K (2012) Evaluating component solver contributions to portfolio-based algorithm selectors. In: Cimatti A, Sebastiani R (eds) Proceedings of the Fifteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'12), Springer-Verlag, Lecture Notes in Computer Science, vol 7317, pp 228–241

Yafrani ME, Chand S, Neumann A, Wagner M (2017) A Case Study of Multi-objectiveness in Multi-component Problems. URL `http://cs.adelaide.edu.au/~optlog/research/combinatorial.php`