Journal on Vehicle Routing Algorithms (2019) 2:1–22
https://doi.org/10.1007/s41604-019-00010-9

REGULAR PAPER

On automatic algorithm configuration of vehicle routing problem solvers

Jussi Rasku[1] · **Nysret Musliu**[2] · **Tommi Kärkkäinen**[1]

Received: 2 February 2018 / Accepted: 30 January 2019 / Published online: 22 February 2019

Abstract
Many of the algorithms for solving vehicle routing problems expose parameters that strongly influence the quality of obtained solutions and the performance of the algorithm. Finding good values for these parameters is a tedious task that requires experimentation and experience. Therefore, methods that automate the process of algorithm configuration have received growing attention. In this paper, we present a comprehensive study to critically evaluate and compare the capabilities and suitability of seven state-of-the-art methods in configuring vehicle routing metaheuristics. The configuration target is the solution quality of eight metaheuristics solving two vehicle routing problem variants. We show that the automatic algorithm configuration methods find good parameters for the vehicle route optimization metaheuristics and clearly improve the solutions obtained over default parameters. Our comparison shows that despite some observable differences in configured performance there is no single configuration method that always outperforms the others. However, largest gains in performance can be made by carefully selecting the right configurator. The findings of this paper may give insights on how to effectively choose and extend automatic parameter configuration methods and how to use them to improve vehicle routing solver performance.

Keywords Vehicle routing problem · Automatic algorithm configuration · Metaheuristics · Meta-optimization

1 Introduction

The vehicle routing problem (VRP) is a practical, relevant, and challenging problem that has been extensively studied by the artificial intelligence (AI) and operations research (OR) communities. One of the main trends in solving VRPs is the shift toward more generic and robust route optimization algorithms [56]. However, optimization models and algorithms are still typically hand-tuned by experts on a case-by-case basis [14, 56]. The need for an expert in this process creates a barrier for the widespread use of the latest scientific advances to solve real-life optimization problems. Therefore, to build more flexible academic and commercial solvers for routing problems, the hand-tuning of the algorithms should be automated. One step toward this goal is to automate the search of the right optimization parameters [14, 31]. This opportunity has been recognized, e.g., by Hutter et al. [30]: "automated algorithm configuration methods ...will play an increasingly prominent role in the development of high-performance algorithms and their applications."

Automatic algorithm configuration [31] (or *parameter tuning* [15]) means off-line modification of an algorithm's parameters. Recently, researchers have proposed several automatic configuration methods, which have proven successful in different domains such as evolutionary computation [55], Boolean satisfiability [1, 30], and mixed-integer programming [25, 34]. In the field of vehicle routing research, Pellegrini and Birattari [48] compared the performance of different metaheuristics with and without automatic algorithm configuration and concluded that, in every instance, the automatically configured version of the solution algorithm yielded better results than the corresponding non-configured one. Furthermore, automatic algorithm

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s41604-019-00010-9) contains supplementary material, which is available to authorized users.

✉ Jussi Rasku
jussi.rasku@jyu.fi

[1] Faculty of Information Technology, University of Jyvaskyla, P.O. Box 35, 40014 Jyväskylä, Finland

[2] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, Institute of Logic and Computation, DBAI, TU Wien, 1040 Vienna, Austria

configuration enabled a fair comparison, which makes it a recommended practice for algorithm developers [16].

Besides our preliminary work presented in [52], there is no comprehensive comparative study on automatic algorithm configuration of vehicle routing solvers. Consequently, this study addresses this knowledge gap by investigating the performance of recent automatic configuration methods in the domain of routing algorithms. In particular, our aim is to answer the following questions:

1. Are existing automatic algorithm configuration methods suitable for configuring routing algorithms?
2. How do these configurators compare, and are there methods that should be preferred when configuring routing algorithms?
3. How does the performance of configurators vary with different metaheuristics, vehicle routing variants, and problem instances?
4. How robust are the methods in configuring routing algorithms?

To address these questions, we compare the performance of seven state-of-the-art automatic algorithm configuration methods on metaheuristics for two different variants of the vehicle routing problem. This extends our previous study [52] by adding new configuration targets, improving experimental setup, and including a thorough analysis of the configuration method performance and resulting parameter configuration values. In our experiments we concentrate on optimizing solution quality instead of algorithm runtime on relatively small problem instance sets. Our results confirm that with these conditions the algorithm performance can be clearly improved by using automatic configuration. Also, while some configuration methods perform better, and are more robust in some algorithm configuration tasks, no single method invariably outperforms all the others.

The paper is structured as follows: Sects. 2 and 3 introduce the vehicle routing and the algorithm configuration problems, and describe the automatic algorithm configuration methods used in this paper. Section 4 contains a literature review on algorithm configuration in routing. Section 5 describes the experimental design used to test the configurators followed by Sect. 6 where the numerical results and analysis are presented and discussed. Finally, Sect. 7 concludes the study and proposes topics for future research.

2 The vehicle routing problem

In the classical vehicle routing problem (VRP) the goal is to find optimal *routes* for *vehicles* leaving from a *depot* to serve a specified number of *customers*. Each customer must be visited exactly once by exactly one vehicle. Each vehicle must leave from the depot and return there after serving all customers on its route. Typical objectives are to minimize the number of vehicles and the total length of the routes. Thus, VRP is a generalization of the well-known travelling salesman problem (TSP).

Multiple extensions and variants of VRP have been proposed in the literature. Many of these add new constraints, such as vehicle capacity, maximal route length, and time windows, or introduce new features, such as stochasticity, split deliveries, or multiple depots. For an introduction to different variants and extensions to VRP, refer to [32]. Problems where several constraints and complex objectives are combined to tackle real-world cases are called *'rich'* VRPs [12].

In this paper, we focus on two variants: the capacitated vehicle routing problem (CVRP) and the vehicle routing problem with stochastic demands (VRPSD). In CVRP, each customer has a demand that needs to be fulfilled and each identical vehicle has a capacity that cannot be exceeded. The objective is to find feasible routes so that the number of vehicles and the total distance of routes are minimized. Also the vehicles in VRPSD have limited capacity, but in this variant the exact demands of the customers are not known until they are served. However, the distributions of the demands are known and should be considered in the optimization of the routes [8].

Algorithms for solving the VRP can be divided into two families: exact and heuristic. The aggregated results from Uchoa et al. [59] suggest that exact algorithms cannot consistently solve CVRP instances with more than two hundred customers, and, therefore, different (meta)heuristics have been proposed to solve larger problems. Examples of such methods include simulated annealing (SA), tabu search (TS), evolutionary algorithms (EA), ant colony optimization (ACO), and iterated local search (ILS). For surveys of the topic, refer to Laporte [35] and Mester and Bräysy [42].

Recently, the trend has been toward developing adaptive and cooperative hybrid algorithms [4, 33, 49, 56], but as Hutter et al. [31], Battiti and Brunato [6], and Sevaux et al. [54] have noted, even these tend to have many parameters that need to be fixed. Therefore, these new approaches further emphasize the need for automatic algorithm configuration.

3 The algorithm configuration problem

Many advanced search algorithms have *free parameters* that can be set by the user. The parameters are usually used to balance the algorithm elements and make trade-offs between diversification, intensification, co-operation, and other aspects [61]. These parameters must be configured in order for the method to perform well, which is a nontrivial task. In fact, Smit and Eiben [55] point out that finding the right

values for the parameters "is a complex optimization task with a nonlinear objective function, interacting variables, multiple local optima, and noise." With stochastic local search (SLS, see [24]) algorithms for VRP, this noise comes from the random problem instance selection and stochasticity of the algorithm that is being configured.

One of the main challenges of automatic algorithm configuration according to Eiben et al. [15] comes from the complex interactions between the parameters. Sometimes the parameters can be configured individually, but the result may be suboptimal, whereas trying all different combinations is often impossible due to the sheer number of possible combinations.

Next, we define the algorithm configuration problem and describe approaches that have been proposed to solve it.

3.1 Introducing the problem

Hutter et al. [30] defines the goal of automatic algorithm configuration to be finding a set of parameter values, *a parameter configuration*, for a given *target algorithm* so that the algorithm achieves the best possible performance, or *utility*, on the given input data set. Formal definitions of the problem are presented by Birattari et al. [9] and by Hutter et al. [30].

Depending on when the algorithm parameters are changed, *automatic algorithm configuration* and *parameter control* can be distinguished from each other [15]. Automatic algorithm configuration is the off-line task of finding good values for the parameters before the actual deployment of the algorithm into production. In contrast, parameter control reactively changes the values of the parameters while the algorithm is running.

Algorithm parameters can be *numerical*, *ordinal*, or *categorical*. Numerical parameters have a value that is an integer or a real number. Ordinal and categorical parameters have a finite set of values that the parameter may take, but categorical parameters cannot be ordered in a meaningful way.

3.2 Automatic algorithm configuration methods

The performance of different *configuration methods* (or *configurators*) has been studied earlier, for example, for mixed-integer programming solvers [26], evolutionary algorithms [45, 55], and SAT solvers [1, 30, 37]. Actually, Kadioglu et al. [34] states that there has been a renaissance in the field of automatic algorithm configuration during the first decade of the 21st century. For a recent review of these methods see Hoos [23]. Eiben and Smit [16] presents a similar survey for the evolutionary algorithm tuning community. In addition to exploring the concepts such as robustness and performance measures, they propose a useful taxonomy for the configuration methods.

Recently, the focus has been in overcoming the challenges posed by heterogeneous and large problem instances. Prime examples of this research are recent studies from Styles and Hoos [57] and Mascia et al. [41], where new techniques for reducing computational effort are proposed. These alone are not always sufficient, as finding good parameter configurations still often requires considerable computational resources. Combining automatic configuration with parallel and cloud computing demonstrates how increased availability of computational resources can allow performing the configuration tasks within reasonable time [18, 28].

In this study, we focus on seven state-of-the-art algorithm configuration methods: CMA-ES [21, 64], GGA [1], Iterated F-Race [3], ParamILS [30], REVAC [46], SMAC [27], and URS [64]. The primary criterion to include a method into this study was previously documented use of the automatic algorithm configuration method on VRP or TSP targets. The secondary criterion was the availability of an implementation, as not all recently introduced automatic configuration methods are publicly available. Short descriptions for each of the selected methods are given below.

CMA-ES is a continuous optimization method that was proposed by Hansen [21]. The method is based on the ideas of self-adaptive evolution strategies. It works by sampling new vectors from a multivariate Gaussian distribution, whose covariance matrix is cumulatively adapted using the search evolution path to form rotationally invariant scatter estimates. CMA-ES is known to be reasonably robust and is therefore suitable for automatic algorithm configuration [55]. We extended CMA-ES with a basic discretization scheme to make it support ordinal and categorical parameters, as they were not supported natively. Recently, Vidal et al. [60] used CMA-ES to configure a hybrid VRP solver with eight numerical parameters.

GGA (Gender-Based Genetic Algorithm) is a robust population-based automatic algorithm configuration method proposed by Ansótegui et al. [1]. The method divides the population into two genders, where the selection pressure is only on the other gender. If the dependencies between the configured parameters are specified, they are taken into account in recombination phase. In addition, GGA uses the aging and death of individuals, and random mutations in the new offspring. The parameters of GGA include truncation percentage X for breeding selection, tree branch inheritance probability B, mutation rate M along with mutation variance S, and maximum age A. GGA also requires the initial population size P and number of generations G to be set. Ansótegui et al. [1] did not report the number of optimized parameters being configured in their experiments, but according to [30] the number of parameters for these targets ranges from 4 to

26 with varying composition of categorical and numerical parameters.

F-Race [9] races a finite set of candidate parameter configurations against each other. The method draws inspiration from Maron and Moore [39] where racing was used to solve a similar problem. At each step of F-Race, candidates are evaluated by running the target algorithm on a single problem instance from the training set. A Friedman test is then used to eliminate those configurations that are significantly worse than the best one. The race is terminated when a maximum number of configurations have been sampled, when the predefined computational budget is used, or when the Friedman test indicates that a dominating best configuration is found.

I/F-Race (Iterated F-Race) is an iterated extension of the F-Race proposed by Balaprakash et al. [3]. In I/F-Race, a relatively small set of new candidates is sampled during each iteration. After each race iteration some or all of the surviving candidates are promoted as elite. Each candidate in the new iteration is sampled from a distribution centered on a randomly selected elite candidate. The standard deviations for this distribution are reduced on each iteration [37]. I/F-Race is parametrized by the number of iterations I, the computation budget for each iteration eb_I, the number of candidates for each iteration N_I, and the stopping condition parameter $N_{\min}$. The additional stopping parameter allows a race iteration to be terminated when only $N_{\min}$ candidates are remaining [9, 37], which will help ensure sufficient exploration in the parameter configuration space [10]. The experiments described by Birattari et al. [10] contained at most 12 configured parameters.

ParamILS [30] uses iterated local search (ILS), which has proven to be a good heuristic for solving a variety of discrete optimization problems [38]. It uses an one-exchange neighborhood (one change to one parameter at a time) to search the space of all possible algorithm parameter value combinations. The ParamILS algorithm starts by sampling R random parameter configurations from which it selects the one performing best on the target algorithm. Then it performs a local search where it moves toward a local optimum. To avoid getting stuck, ParamILS employs random perturbations and restart strategies. The ILS approach allows ParamILS to configure any algorithm, even those with many parameters. However, ParamILS is able to handle only ordinal and categorical parameters and requires discretization of continuous parameters.

REVAC (Relevance Estimation and Value Calibration) by Nannen and Eiben [46] is a population-based estimation-of-distribution algorithm. REVAC starts from an assumption of a uniform distribution over the range of each free parameter. It samples new individuals from the constantly updated parameter distributions and aims, through transformation operations with multi-parent crossover (where N best individuals are selected) and an interval shrinking operation governed by a parameter H, to narrow down on the most promising range of each parameter. After the initial population of size M has been evaluated, only one new individual is sampled at each iteration. After the method has finished, relevance estimates can be used to recognize which parameters are essential to the performance of the target algorithm. Categorical parameters are not supported. EA targets configured by REVAC seem to typically have around six parameters [55].

SMAC [27] is the latest configurator from a series of sequential model-based optimization (SMBO) methods [5, 25, 29]. SMBO is an iterative framework for methods that alternate between fitting a regression model, and using that model to predict performance of new candidates. However, SMAC is the first one to extend this paradigm to general algorithm configuration problems. Thus, while Bartz-Beielstein et al. [5] were one of the first to use these black box continuous optimization methods in algorithm configuration, Hutter et al. [27] further extended the applicability of SMBO by adding support for multiple instances, categorical and conditional parameters, and an option to model the parameter configuration response surface more accurately. More precisely, a random forest with instance features is used to create a surrogate model for the algorithm's performance, which is then used in local search of promising configurations. SMAC and ParamILS were used and tested in scenarios with nearly 80 free parameters by Hutter et al. [28].

URS (Uniform Random Sampling) [64] is used in this study as a reference parameter configurator. During an iteration, a candidate is sampled uniformly from the set of all possible parameter configurations and evaluated on all instances in the training set, while keeping track of the best encountered configuration. The method sets a baseline for the more sophisticated configuration methods presented above.

The features of the seven automatic algorithm configuration methods are summarized in Table 1. The first group of columns from the left shows which target algorithm parameter types are supported by the configurator. The second group shows the algorithmic building blocks that the configurators employ to allocate search efforts effectively. Here, effective allocation is one that concentrates the target algorithm evaluations mostly on the promising parameter configurations. The features also ensure that exploration and exploitation are balanced and the stochasticity of the search and target algorithm properly addressed [64]. Sampling is a strategy that all configurators share, but otherwise these methods

Table 1 Features of the automatic algorithm configuration methods used in this study

Method	Parameter types				Algorithmic concepts				Effort reduction techniques						Number of features
	Categorical	Ordinal	Numerical	Continuous	Sampling	Populations	Statistical model	Local search	Restarts	Capping	Racing	Blocking	Sharpening	Seed configs.	Cond. params.	
CMA-ES			✓	✓	✓		✓		✓					✓		6
GGA	✓	✓	✓	✓	✓	✓						✓	✓	✓	✓	10
I/F-Race	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓	✓	✓	12
ParamILS	✓	✓			✓			✓	✓	✓	✓	✓	✓	✓	✓	11
REVAC			✓	✓	✓	✓	✓									5
SMAC	✓	✓	✓	✓	✓		✓	✓			✓	✓	✓	✓	✓	12
URS	✓	✓	✓	✓	✓											5

use different approaches to solve the automatic algorithm configuration problem.

The third group lists effort reduction techniques that are used to save parameter configuration evaluations by changing how candidate configurations are tested: *Capping* [30] terminates the evaluation as soon as it becomes clear that the candidate configuration cannot produce a good parameter configuration. This is convenient when the objective is to minimize the runtime of a target algorithm, but capping is not applicable to solution quality-based configuration that we are doing in this paper. In *racing* [9] good and bad parameter configurations are recognized early by increasing the number of instances and random seeds to evaluate on each step of the race. This technique is closely related to *blocking* [40], where the parameter configuration candidates are evaluated on the same instances and seeds called a block. These techniques control the noise from the variance in the configuration objective between instances and seeds. *Sharpening* [55] controls the number of available problem instances per iteration, and *seed configurations* allow the use of the default or other user-provided parameter configurations at initialization.

The concepts racing, blocking, and sharpening can be combined like in the *intensify* approach of ParamILS and SMAC [27, 30]. There, the history of evaluations on the best parameter configuration is stored and after a new evaluation is added, new configurations are compared against the history on the same problem instances and seeds. New configurations are rejected or declared as the new best-known configuration early, that is, after there is enough evidence.

Conditional parameters, also known as *parameter hierarchies*, were introduced in ParamILS [31]. They allow the user to specify that algorithm parameters are active only with activation of some other parameter, and, thus, available for automatic configuration. This prevents the configurator from changing parameter values when they have no effect. An in-depth survey of techniques and concepts related to automatic algorithm configuration is given by Hoos [23].

The rightmost column of Table 1 shows the total number of features for each configuration method. Out of the listed methods, CMA-ES, URS and REVAC, rely on a smaller number of features compared to others as they do not use the more sophisticated search effort reduction techniques. We are aware that generic continuous optimization methods such as CMA-ES and URS can be augmented with effort reduction mechanisms. For example, in [64] they were used to identify good parameter configurations with minimal evaluation effort, similarly to racing in I/F-Race. However, CMA-ES has also been used in algorithm configuration without such extensions (see, e.g., [60]), and, additionally, our research was better served with distinctly different approaches to automatic algorithm configuration than variations on the I/F-Race pattern. Those interested in extending

continuous optimization methods such as CMA-ES with effort reduction techniques are referred to [64].

4 Automatic algorithm configuration in routing

In this section, we give a short survey on configuring routing algorithms. Because the number of articles on automatic algorithm configuration of VRP algorithms is relatively small, we also survey the relevant studies for the traveling salesman problem (TSP).

Coy et al. [14] recognized the importance of configuring VRP metaheuristics already in 2001 and proposed a procedure to find a set of good parameter values for a target VRP algorithm. Their procedure is based on a statistical design of experiments that requires expert knowledge on each step of the process. Similar to the more recent automatic algorithm configuration methods, their procedure contains local, inexact steepest descent search on the response surface and uses an average of the locally optimal parameter configurations as the final result. The authors concluded that their method managed to improve the default settings of their VRP algorithms, and that the procedure outperformed random parameter sampling.

Pellegrini [47] used F-Race to configure two heuristic algorithm variants solving a specific rich VRP variant, a VRP with with multiple time windows and heterogeneous fleet. Later, Pellegrini and Birattari [48] showed the benefits of automatically configuring VRP metaheuristics. They configured the IRIDIA VRPSD solvers with F-Race and noted that the configured algorithms were able to clearly outperform the out-of-the-box implementations with default parameters. Becker et al. [7] used racing to configure the parameters of a commercial VRP solver on a heterogeneous training set of 47 real-world routing problem instances.

Balaprakash et al. [3] used automatic algorithm configuration on three different routing variants, including VRPSD, to show the advantages of the iterated F-Race over the standard F-Race. Garrido et al. [17] proposed a hyperheuristic where REVAC was used to choose the low-level heuristics solving CVRPs. More recently, Vidal et al. [60] used CMA-ES to automatically configure his record breaking hybrid genetic algorithm (GA) for multi-depot and periodic vehicle routing problems. By using a meta-GA to configure a hybrid GA, Wink et al. [62] were able to reduce the optimality gap on CVRP benchmark instances from Augerat et al. [2] by 91 % (a 0.55 percentage point improvement) compared to an extensively hand-tuned hybrid GA. They were also able to find a new best-known solution for a 200-customer instance in another problem set by using the same automatic configuration approach.

Even though there are studies of automatic algorithm configuration of routing solvers, we were able to find only three comparative studies on automatic algorithm configuration of TSP solvers. Montero et al. [44] compared F-Race, REVAC, and ParamILS to recognize unused operators in solving the TSP with an evolutionary algorithm. In a second study, Montero et al. [45] focused on comparing the performance of the three previously mentioned configurators. They concluded that all three methods have comparable configuration performance and that they are able to improve the performance of metaheuristics targeting single problem instances. Yuan et al. [64] compared CMA-ES, URS, and three other methods in configuring the ACO algorithm for the TSP, and Styles and Hoos [57] introduced two racing protocols that allow different levels of difficulty of problem instances in training and validation sets. To solve the TSP instances they used an implementation of the Lin–Kernighan algorithm (LKH). They concluded that for various sizes of configuration problems, especially for those with many numerical parameters, CMA-ES appears to be a robust algorithm.

In addition to our workshop paper [52] reporting some preliminary results, we are not aware of comparative studies on automatic algorithm configuration methods configuring vehicle routing solvers. VRP solvers have been configured in many studies, but the lack of comparative experiments with different automatic algorithm configurators makes it hard to determine which method one should use when dealing with different VRP metaheuristics. Also, from the existing literature, it is hard to infer how much the solution quality is expected to improve when a VRP metaheuristic is configured with automatic algorithm configuration.

5 Comparison of methods for configuring VRP solvers

Next, we will describe our computational comparison for the automatic algorithm configuration methods. We will explain the experiments that we carried out, and the costs and benefits of adding a layer of meta-optimization on top of a VRP solver. As noted in the study by Hepdogan et al. [22], the configurator for heuristic algorithms should be fast, efficient, and outperform random parameter value selection. Thus, the additional complexity caused by the automatic algorithm configuration must be empirically justified. We will also present the VRP solvers used as the target algorithms.

5.1 Solvers and benchmark problems

VRPH is a heuristic solver library for the CVRP developed by Groër et al. [20]. The library uses the Clarke-Wright construction heuristic and a selection of well-known local search operators: one-point-move (1*ptm*), two-point-move

Table 2 Free parameters of the VRPH and VRPSD solvers

VRPH	Name	Type	Default	Range	VRPSD	Name	Type	Default	Range
Shared	1*ptm*	B	1	{0, 1}	Shared	p	B	0	{0, 1}
	2*ptm*	B	1	{0, 1}		t	B	0	{0, 1}
	two	B	1	{0, 1}	ACO	m	I	7	[1, 100]
	oro	B	0	{0, 1}		τ	R	0.5	[0.0, 1.0]
	tho	B	0	{0, 1}		ψ	R	0.3	[0.0, 1.0]
	3*ptm*	B	0	{0, 1}		ρ	R	0.1	[0.0, 1.0]
EJ	m	I	10	[0, 45]		q	R	1e7	[10.0, 1e7]
	t	I	1000	[0, 1e4]		α	R	1.0	[0.0, 5.0]
	s	B	0	{0, 1}	EA	p	I	10	[1, 1e3]
RTR	D	I	30	[1, 100]		mr	R	0.5	[0.0, 1.0]
	δ	R	0.01	[0.0, 0.1]		amr	B	0	{0, 1}
	K	I	5	[0, 100]	ILS	x	R	10.0	[0.0, 1e3]
	N	I	4	[0, 75]	SA	μ	R	0.01	[0.0, 0.1]
	P	I	2	[1, 10]		α	R	0.98	[0.0, 1.0]
	p	B	1	{0, 1}		ψ	I	1	[1, 100]
	a	B	1	{0, 1}		ρ	I	20	[1, 100]
	t	I	0	[0, 50]	TS	*ttf*	R	0.8	[0.0, 1.0]
SA	T	R	2.0	[0.0, 10.0]		p_t	R	0.8	[0.0, 1.0]
	n	I	200	[0, 1e3]		p_o	R	0.3	[0.0, 1.0]
	i	I	2	[0, 10]					
	α	R	0.99	[0.8, 1.0]					
	N	I	10	[0, 100]					

The following parameter type key is used: 'B' for Boolean switch (was treated as numerical, or as categorical if the option was available), I for integer values (numerical), R for real values (numerical, continuous)

(2*ptm*), three-point-move (3*ptm*), two-opt (*two*), three-opt (*tho*), and Or-opt (*oro*). These operators can be enabled and disabled using six switches common to all solvers (listed as shared in Table 2). VRPH implements also the cross-exchange operator, but we disabled it because of its tendency to produce infeasible routes.

Other solver parameters do not have an effect on the behavior of the local search operators. Use of the library's local search operators is orchestrated by three metaheuristics: Record-to-Record travel (RTR, 6 + 8 free parameters, where the first 6 are the shared parameters between all VRPH metaheuristics and the other 8 parameters are specific to the RTR metaheuristic), Simulated Annealing (SA, 6 + 5), and neighborhood ejection (EJ, 6 + 3). We omit the descriptions of the heuristics, metaheuristics, and solver parameters and refer the reader to Groër et al. [20] and Table 2.

The other set of solvers we used in our experiments was the IRIDIA VRPSD metaheuristics presented by Bianchi et al. [8]. For local search, the IRIDIA VRPSD solvers rely on only one operator, Or-opt. For the metaheuristic, one can choose between ant colony optimization (ACO, 2 + 6 parameters), evolutionary algorithm (EA, 2 + 3), iterated local search (ILS, 2 + 1), simulated annealing (SA, 2 + 4), and tabu search (TS, 2 + 3). The two shared parameters, p and t, are related to determining the local search move cost approximation method. For a thorough explanation of the solver parameters refer to Table 2 and Bianchi et al. [8].

The size of the training set is an important variable when doing automatic algorithm configuration. If the training set is excessively large, evaluating every parameter set on all instances, as it is done in URS and REVAC, becomes infeasible. Even the more sophisticated configuration methods require a significant subset of a large heterogeneous training set to get a reliable estimate on the parameter configuration utility. Conversely, if the training set is small, there is a danger that it is not a representative sample, and even if the resulting parameter configuration can be used to solve the training set effectively it may have been over-tuned and its performance does not generalize [1]. For our configuration tasks, we decided to use a training and validation set size of 14 instances, which is consistent with the experiences of Becker et al. [7] from configuring real-world routing problems.

We acknowledge that the chosen number of instances is atypically small for automatic algorithm configuration tasks. The reasons leading to small number of training instances was threefold: 1. Out of the compared configuration methods, only the advanced ones support sharpening and blocking. To avoid major modifications and extensions to the less

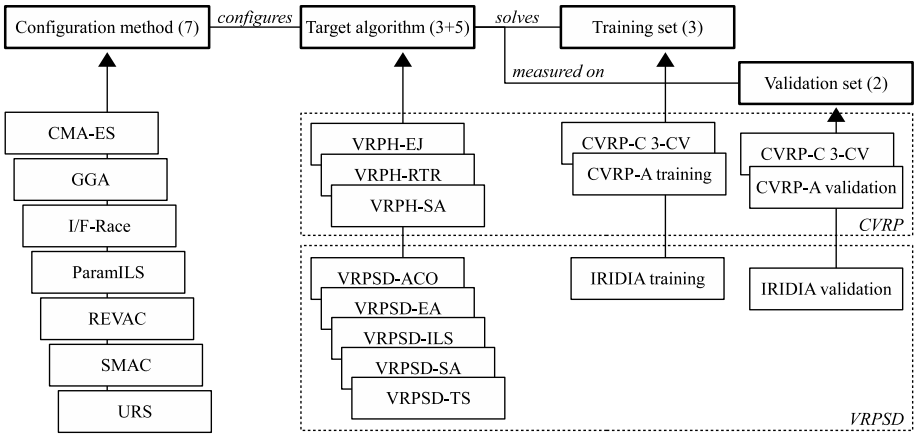


Fig. 1 The experiment setup where the total number of configurators, configuration targets, and VRP problem instance sets are given in the parenthesis

sophisticated configurators, we simply evaluate the entire training set for each parameter configuration candidate. A large training instance set would make this approach infeasible. 2. We wanted to keep the size constant over all targets, and 14 was the size of the smallest problem set used in our experiments. 3. Finally, we would like to point out that a promising practical application of automatic configuration in vehicle routing is the automatic fine tuning of algorithms used in real-world routing [11, 53]. Especially in industry one might not be able to access a large number of specific routing problems because of time and human resource limitations. By using a restricted problem set size we tried to ensure that this study stays relevant to this audience.

For the VRPH solving CVRPs, we used the classic benchmark set CMT with 14 problem instances originating from Christofides et al. [13], which has problems with sizes ranging from 50 to 200 customers. Paired with the three metaheuristics, this creates configuration targets VRPH-EJ-C, VRPH-RTR-C, and VRPH-SA-C. We used a threefold cross-validation with stratified sampling by problem size for this benchmark set because dividing this set into separate training and validation sets would have produced prohibitively small problem sets.

In order to examine the effect different problem sets can have on configuration performance, and how well the performance gains generalize to similar problems, we used the A and B CVRP sets from Augerat et al. [2]. These sets have 27 and 23 instances with sizes from 31 to 79 customers. The problem sizes and demand distributions are similar, but the customers in set A are uniformly distributed and in B clustered. To fix one variable, the size of the training set, we decided to use a subset of the original instance set in our experiments. We used a stratified sampling of 14 instances from set A and set B, to construct disjoint training and validation sets. This forms the next three configuration targets: VRPH-EJ-A, VRPH-RTR-A, and VRPH-SA-A.

Finally, to test the IRIDIA VRPSD solvers, we used training and validation subsets, again with a stratified sampling of 14 instances each, from the IRIDIA problem set of 120 randomly generated instances with 50 to 200 customers [8]. Supporting material[1] for [8] includes the algorithms and description of the problem instances. The configuration targets for VRPSD are: VRPSD-ACO, VRPSD-EA, VRPSD-ILS, VRPSD-SA, and VRPSD-TS.

The experimental setup is illustrated in Fig. 1. To summarize, we selected seven automatic configuration methods, three target algorithms solving the CVRP, and five solving the VRPSD. For each of the eight target algorithms solving a set of VRP benchmarks, the configurators try to find a set of parameters that maximize the quality of the solutions produced. This means there are interchangeable objects in the three levels: a problem set, a solver with the metaheuristic and local search operators, and a configurator that optimizes solver performance. In addition, two of these levels have free parameters: the solver has parameters being configured and the configurator has its own parameters that must be set manually by the experimenter. Furthermore, the selection of the problem instances to the training and validation sets may cause variability in the configuration performance.

5.2 Experimental design

The VRP solvers used in this study were considered to be black boxes from the configurators' point of view. Only the free parameters and their ranges were known prior to starting the configuration task.

When using heuristic algorithms, reaching the optimum in a reasonable time is not guaranteed. Thus, we cannot use the total running time of the target algorithm to compare parameter configuration efficiency, even if it is a more common target for automatic algorithm configuration (see, e.g., [30]). Instead of solver time, we decided to optimize for solution quality and set a 10 CPU second cutoff for all

[1] http://iridia.ulb.ac.be/supp/IridiaSupp2004-001/index.html.

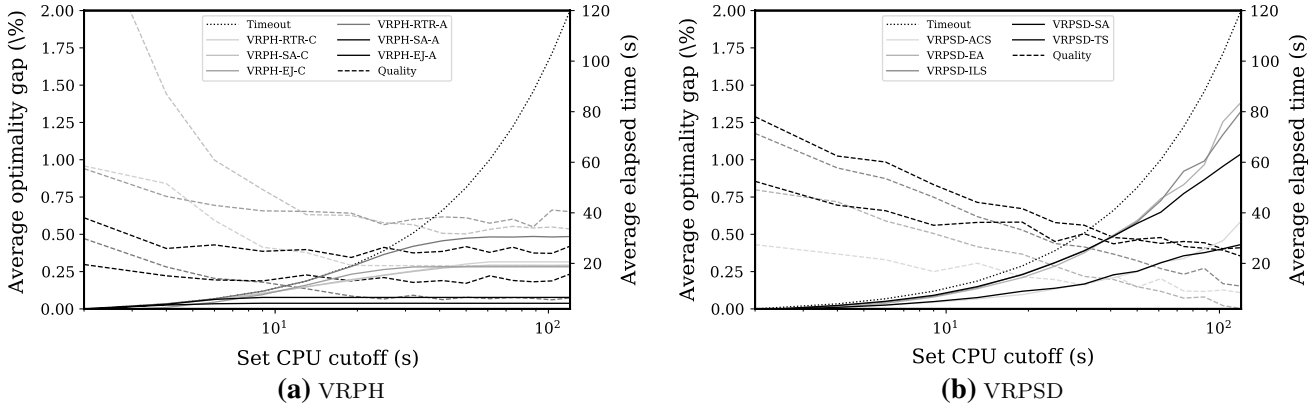


Fig. 2 Effect of a cutoff to the elapsed algorithm runtime and resulting solution quality on an automatically configured solver

invocations of VRP solvers. If an algorithm had not stopped after 10 s, it was terminated and with the quality of the current best solution.

Choosing 10 s as the CPU cutoff was not arbitrary. The experimental design presented here already creates a large number of combinations to test, and the stochasticity in the target algorithms and in the configurators themselves requires multiple configuration trials for statistical reliability. Thus, the algorithm runtime had to be reasonably small. Figure 2 illustrates the effect cutoff has on actual elapsed time of the algorithm and the resulting solution quality. VRPH solvers are able to utilize the more generous computational resources only with two targets and the additional time gives diminishing returns after 10 s. Also, while considering the effect of this decision, please note the scaling of the x-axis. The curves have a negative exponential multiplier, and thus, the quality improvement is logarithmic (not linear) when the CPU time is increased for VRPSD targets. Furthermore, the selected cutoff is in line with [19], where the Augerat et al. [2] instances are solved within 0.3 % of optimal solution on average in 3.5 s. Similarly, Groër [19] reports that the larger CMT [13] instances are solved close to optimality on average in 12.94 s (RTR) or 21.9 s (EJ) on a 2.3 GHz AMD processor.

From Fig. 2 we also see that VRPSD can utilize the more generous computational resources. In their experiments Pellegrini and Birattari [48] used a slightly more generous CPU timeout of 30 s for these targets, although with significantly slower AMD Opteron 244 processor than the Intel Xeon E7 used in this study. Yuan et al. [64] used a 5 s cutoff for ACO solving medium sized TSP instances, which was then configured using, e.g., CMA-ES. These further validate the decision of using a relatively strict cutoff in such configuration scenarios. Furthermore, because our comparison already had many changing variables (configurator, target metaheuristic and its parameters, problem sets and instances, and local search operator selection), we decided to fix the cutoff for all targets.

We acknowledge that the runtime of a routing algorithm to solve large real-world problem with thousands of customers may be measured in hours, especially in cases with complex constraints such as dynamic travel times, compartment compatibilities, or other extensions including separate pickups and deliveries or a heterogeneous fleet [7]. Despite this, modern metaheuristics are usually able to find proper solutions for all but the largest benchmark problems in a few seconds.

As the utility metric we use an aggregated solution objective function value over the training instance set. Aggregation is a sum over the estimated objective function values for the resulting VRP solutions in the instance set. Thus, the configuration task is to minimize:

$$\underset{\theta}{\text{minimize}} \quad \hat{c} = \sum_{i \in I_t} \hat{a}(i, \theta) \tag{1}$$

Here, $\hat{c}$ is the utility metric estimator, I_t is the training problem set, $\hat{a}$ is an estimator for the utility function for algorithm a a, which in turn solves problem instance $i \in I_t$ guided by parameter configuration θ. Because the metaheuristic algorithm a is stochastic, we also define $\hat{a}$ to be an estimator of the algorithm solution quality. In practice, the estimate is formed through repeated evaluation of the algorithm on the same problem instance and parameter configuration, but with different random seed.

Even if we did not solve rich problems in this study, the number of evaluations that can be allocated into finding a reasonably good parameter configuration remains an important factor. Especially since we would like to keep this study relevant to the operations research practitioners who are solving large-scale real-world vehicle routing instances with complex constraints who could benefit from

automatic algorithm configuration. In their case, the number of solver invocations cannot be too large.

In our experiments each configuration task was given an evaluation budget that defines the number of solver invocations allowed during a configuration task. One call of the routing solver with one parameter configuration and one problem instance is counted as one evaluation. To compare the effect of different budgets, every configurator—solver combination was run with evaluation budgets of 100, 500, and 1000. In addition, VRPSD-ACO was configured with an evaluation budget of 5000 to see the effect of a larger budget.

Whenever the option was available, configurators were set to expect non-deterministic behavior from the target algorithm. Thus, allocating the budget for new parameter configurations and problem instances and controlling the stochasticity through additional evaluations was left for the configurator.

GGA, I/F-Race, ParamILS, and SMAC use effort reduction techniques that can save evaluations, for example, by evaluating only a subset of the training instances on each iteration, whereas CMA-ES, REVAC, and URS evaluated all problem instances in the training set on each iteration. All the tested target parameters could be represented with an integer or real number with a suitable range and an optional discretization step (in ParamILS).

All seven automatic algorithm configuration methods contain a set of parameters related to the basic technique and its actual implementation. The default parameters provided by the original authors were used in the experiments whenever possible. A full listing of the configurator parameters can be found in the online supplementary material for this paper. Also, the target algorithm defaults were provided as a seed configuration for all configurators excluding REVAC and URS, which did not support it.

CMA-ES is claimed to be quasi-parameter-free [21], so we did not change the initial parameters of the Python implementation.[2] For optimization, all continuous parameters were normalized between 0.0 and 1.0 with an initial standard deviation of $\sigma_0 = 0.5$. The restart mechanism of this CMA-ES implementation was not used, because it is not applicable to fixed and relatively small evaluation budgets. Also, instead of relying entirely on the self-adaptive parameters, on tasks with an evaluation budget of 100, we used a population size of 7 to help CMA-ES stay within the specified budget.

For GGA, we used the implementation of [1] with the default values (10, 90, 10, 3, 10) for (X, B, M, A, S). Ansótegui et al. used several different population and generation ratios. We decided to use the $P/G = 2/1$ ratio to avoid extinction of the population. For evaluations with the budget of 5000, we used the $P/G = 4/3$ ratio that Ansótegui et al. [1] used to configure SAT solvers. Using this ratio we set the population size and number of generations carefully on a budget-to-budget basis, because in order to use the evaluation budget effectively, the evolutionary process must converge at the right time. Because GGA did not respect the specified budget for evaluations, setting G and P was the only way to get it to spend approximately the right number of target algorithm evaluations. Also, GGA required each instance to be paired with a fixed random seed.

F-Race is implemented for the statistical software environment R. The Iterated F-Race automatic algorithm configuration method `irace`[3] from López-Ibánez et al. [37] uses it to implement the iterated variant of the racing method. We used defaults, but for an evaluation budget of 100, the parameter eb_I, which governs the computation budget for each iteration step, was set to 60 to make I/F-Race more closely respect the evaluation budget.

ParamILS [30] and SMAC [27] are available online.[4] For ParamILS, we used linear discretization of 10 steps for each of the continuous free parameters. Selecting the most suitable discretization for each parameter can be seen as an additional level of configuration, and therefore, it was omitted from this study. To take the stochastic nature of the target algorithms into consideration, we used the ParamILS built-in FocusedILS approach to limit the time spent on evaluating each parameter configuration [30]. SMAC was used with default parameters. Its ability to use problem instance characteristics to improve the predictive power of the surrogate model for the target algorithm was not used.

For REVAC, we used the implementation from Montero et al. [44, 45]. To allow it to use the evaluation budget effectively, the control parameters M, N, and H were set using the ratios recommended in the literature: $N = M/2$ and $H = N/10$ with a minimum value of 2 for H. For evaluation budgets of 100, 500, and 1000, M was given values 5, 10, and 20, respectively.

Similarly to [64], each configuration task had 10 trials for VRPH-A and VRPSD targets. In threefold cross-validation of the VRPH-C targets, the cross-validation was repeated five times. The folds were different between repetitions but the same between the target algorithms and budgets. This blocking guarantees that the configuration tasks are comparable between methods. After configuring the algorithms, the resulting parameter configurations were evaluated by

[2] Version 0.9.93.4r2658, http://www.lri.fr/~hansen/cmaesintro.html.

[3] Version 0.9, http://iridia.ulb.ac.be/irace/.

[4] Versions 2.3.5 (ParamILS) and 2.0.2 (SMAC), http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/http://www.cs.ubc.ca/labs/beta/Projects/SMAC/.

Table 3 Median automatic configuration results for the VRPH CMT targets with threefold cross-validation

EB	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS
VRPH-EJ-C, defaults: 0.96 (0.12)							
100	*0.99(0.08)*$^{-}$	*0.86(0.09)*$^{+}$	***0.72(0.09)***$^{+}$	*0.93(0.10)*$^{+}$	1.00(0.22)	0.81(0.10)	**0.77(0.12)**
500	*0.83(0.12)*$^{+}$	*0.79(0.10)*$^{-}$	0.71(0.06)	0.76(0.10)	0.68(0.09)	0.70(0.09)	0.73(0.10)
1000	0.78(0.09)	*0.81(0.09)*$^{-}$	0.66(0.06)	0.71(0.09)	0.73(0.09)	0.69(0.09)	0.75(0.09)
VRPH-RTR-C, defaults: 1.42 (0.06)							
100	1.24(0.14)	*0.90(0.14)*$^{-}$	*1.04(0.15)*$^{+}$	1.00(0.11)	1.22(0.25)	0.94(0.06)	**0.81(0.14)**
500	0.82(0.15)	*0.84(0.03)*$^{-}$	0.75(0.05)	0.91(0.17)	1.06(0.14)	0.78(0.12)	0.83(0.10)
1000	0.76(0.08)	*0.85(0.06)*$^{-}$	**0.63(0.09)**	0.67(0.06)	0.74(0.03)	0.79(0.06)	0.78(0.06)
VRPH-SA-C, defaults: 0.80 (0.05)							
100	1.70(0.52)	0.88(0.18)	***0.73(0.04)***$^{+}$	0.89(0.09)	1.68(0.36)	0.77(0.03)	1.39(0.26)
500	1.09(0.21)	*0.89(0.11)*$^{-}$	*0.81(0.08)*$^{-}$	0.84(0.08)	1.29(0.10)	**0.78(0.04)**	1.04(0.18)
1000	1.03(0.16)	*0.89(0.10)*$^{-}$	0.79(0.08)	0.75(0.09)	1.15(0.12)	0.77(0.03)	0.97(0.13)

Results are given as percentage from the aggregated best-known solution (relative optimality gap). Statistically better results of the single best, or pair of best solvers (in cases where no single configurator dominated), are in bold typeface. Evaluation budget (EB) violations of more than 5% are italicized, with $^{+}$ indicating exceeding and $^{-}$ falling short of the budget

running them on all problem instances 10 times and calculating the aggregated objective cost for each repetition.

All configuration tasks were run on a computing server with 64 Intel(R) Xeon(R) CPU E7 2.67 GHz cores and 1 TB of RAM. The server was running the OpenSUSE 12.3 operating system.

6 Numerical results and analysis

The experiment data contain results of 2695 configuration runs.[5] Together with the verification evaluations these took around 250 CPU days to compute. Considering all results, automatic algorithm configuration methods were able to find improved configurations over defaults in 84.1% of the configuration trials. This is a promising result considering that the smallest used budget of 100 evaluations is a very tight restriction for automatic algorithm configuration. Also, the default parameters of the VRPH solvers are expected to be tailored for typical scientific benchmark instances such as those we used. The suitability of the defaults is even more prominent in the case of VRPSD, where the solvers and the benchmarks instances come from the same source.

[5] $(((3 \times 3) \times (3 \times 5)) + ((3 + 5) \times 3 + 1) \times 10) \times 7$ The 3 VRPH-C targets with 3 different budgets were configured using threefold cross-validation repeated 5 times. The 3 VRPH-A targets and 5 VRPSD targets, each with 3 different budgets, plus (1) VRPSD ACS with a budget of 5000, with 10 trials each. All the previous experiments were done for all the 7 automatic configuration methods.

6.1 Performance of the configurators

A median aggregated solution quality and median absolute deviation were calculated for each configuration task. The median was taken over a set of 10 evaluations on validation set for each of the 10 resulting parameter configurations (that is, over 100 aggregated quality values).

The median was used, because we were mostly interested in measuring the typical performance of a configurator. Meanwhile, the median absolute deviation gives an estimate for the robustness of the configurators. The aggregated solution quality for each configuration task is given as a deviation from the sum of best-known solutions for instances in the validation set (relative optimality gap). The VRPSD benchmarks had no recorded best-known solutions, so we used the best observed solution for each problem instance as the best-known solution. Please note that the result data, with a full set of figures and tables with other descriptive statistics, can be found in the online supplementary material.

The results in Tables 3, 4, and 5 are grouped by the target algorithm. The -C and -A suffixes are used to differentiate between the CMT and Augerat et al. [2] benchmarks for the VRPH targets. Each row shows results for a single configuration task consisting of a triplet: target algorithm, evaluation budget, and problem instance set. When comparing the results we note that out of the tested configurators only ParamILS and SMAC strictly, and URS and REVAC closely, respected the evaluation budget. Other methods frequently ignored the input parameter for the evaluation budget and exceeded or fell short of the budget. Results deviating from the given budget by more than 5% are marked with italics. A nonparametric Mann–Whitney U-test ($p < 0.05$) was used with the Bonferroni adjustment

Table 4 Median automatic configuration results for the VRPH Augerat et al. [2] targets on the validation set B

EB	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS
VRPH-EJ-A, defaults: 0.73 (0.03)							
100	0.42(0.06)	*0.43(0.08)*$^{+}$	*0.50(0.14)*$^{+}$	0.44(0.08)	0.42(0.07)	**0.41(0.06)**	0.38(0.03)
500	*0.40(0.06)*$^{+}$	*0.37(0.02)*$^{-}$	0.37(0.04)	0.42(0.09)	0.42(0.06)	0.37(0.02)	0.38(0.04)
1000	*0.38(0.03)*$^{+}$	0.40(0.05)	**0.37(0.04)**	0.37(0.02)	0.42(0.05)	**0.35(0.02)**	0.37(0.03)
VRPH-RTR-A, defaults: 1.40 (0.05)							
100	0.38(0.07)	*0.36(0.17)*$^{+}$	*0.38(0.20)*$^{+}$	0.44(0.08)	0.50(0.16)	0.62(0.21)	0.37(0.22)
500	*0.35(0.06)*$^{-}$	0.34(0.06)	0.42(0.23)	0.45(0.16)	0.32(0.08)	0.66(0.26)	0.37(0.09)
1000	0.34(0.06)	*0.31(0.09)*$^{-}$	0.34(0.11)	0.39(0.22)	0.38(0.11)	0.63(0.27)	0.34(0.06)
VRPH-SA-A, defaults: 0.90 (0.01)							
100	0.90(0.19)	*0.65(0.12)*$^{+}$	***0.61(0.16)***$^{+}$	0.70(0.24)	0.98(0.10)	**0.65(0.12)**	0.73(0.17)
500	*0.62(0.25)*$^{+}$	*0.48(0.17)*$^{-}$	**0.39(0.11)**	0.61(0.12)	0.66(0.10)	**0.38(0.24)**	0.58(0.20)
1000	*0.41(0.22)*$^{+}$	*0.38(0.24)*$^{-}$	0.33(0.20)	0.38(0.22)	0.53(0.17)	0.34(0.15)	0.34(0.23)

Table 5 Median automatic configuration results for the VRPSD IRIDIA targets on the validation set

EB	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS
VRPSD-ACO, defaults: 0.63 (0.04)							
100	**0.39(0.07)**	*0.43(0.07)*$^{+}$	*0.41(0.04)*$^{+}$	0.39(0.04)	0.43(0.06)	**0.37(0.02)**	0.39(0.04)
500	***0.31(0.05)***$^{+}$	0.38(0.03)	0.36(0.05)	0.36(0.04)	0.41(0.05)	**0.30(0.08)**	0.35(0.05)
1000	***0.28(0.06)***$^{+}$	*0.37(0.03)*$^{-}$	0.37(0.03)	0.33(0.07)	0.37(0.03)	**0.27(0.07)**	0.35(0.06)
5000	0.30(0.09)	0.32(0.06)	0.27(0.06)	0.26(0.06)	0.40(0.02)	**0.16(0.06)**	0.31(0.05)
VRPSD-EA, defaults: 0.77 (0.03)							
100	0.72(0.10)	*0.68(0.08)*$^{+}$	*0.57(0.04)*$^{+}$	0.59(0.07)	0.67(0.06)	**0.53(0.05)**	0.58(0.05)
500	*0.62(0.06)*$^{+}$	*0.57(0.07)*$^{-}$	**0.48(0.04)**	0.57(0.06)	0.58(0.04)	0.51(0.04)	0.49(0.05)
1000	0.56(0.07)	*0.56(0.06)*$^{-}$	0.48(0.04)	0.55(0.06)	0.57(0.04)	0.49(0.04)	0.49(0.05)
VRPSD-ILS, defaults: 0.78 (0.04)							
100	0.71(0.06)	*0.75(0.06)*$^{+}$	*0.74(0.07)*$^{+}$	0.76(0.03)	0.78(0.05)	0.72(0.03)	*0.78(0.03)*$^{+}$
500	*0.73(0.03)*$^{+}$	*0.72(0.08)*$^{-}$	0.76(0.05)	0.76(0.03)	0.77(0.13)	0.71(0.07)	0.78(0.03)
1000	*0.73(0.03)*$^{+}$	***0.71(0.04)***$^{-}$	0.74(0.08)	0.76(0.03)	0.77(0.13)	0.77(0.03)	0.78(0.03)
VRPSD-SA, defaults: 0.79 (0.04)							
100	0.83(0.05)	***0.77(0.07)***$^{+}$	*0.88(0.06)*$^{+}$	0.88(0.08)	1.18(0.23)	0.86(0.05)	*0.87(0.06)*$^{+}$
500	0.84(0.03)	0.78(0.06)	0.87(0.06)	0.85(0.06)	0.88(0.12)	0.88(0.04)	0.86(0.06)
1000	0.84(0.03)	*0.77(0.06)*$^{-}$	0.82(0.03)	0.85(0.06)	0.88(0.11)	0.85(0.02)	0.86(0.06)
VRPSD-TS, defaults: 1.86 (0.13)							
100	**0.75(0.08)**	*1.80(0.05)*$^{+}$	*1.75(0.07)*$^{+}$	**0.72(0.14)**	1.77(0.07)	1.73(0.10)	1.78(0.04)
500	***0.60(0.11)***$^{+}$	*1.74(0.09)*$^{+}$	1.74(0.11)	**0.61(0.12)**	1.73(0.09)	1.74(0.07)	1.70(0.04)
1000	**0.59(0.10)**	*1.75(0.09)*$^{-}$	1.80(0.08)	**0.59(0.10)**	1.73(0.09)	1.83(0.10)	1.70(0.04)

to test whether the differences to defaults and other configurators were statistically significant. Whenever a single dominating method for an algorithm target was not found, existence of a dominating pair of methods was checked. Statistically significantly better automatic algorithm configuration methods (or pairs) for each configuration task are marked in bold typeface.

If we consider only the best configuration found for each configuration task, and average over all targets, automatic algorithm configuration was able to reduce the optimality gap by 0.72. This is a 69.7% improvement compared to using default parameters. According to our results, this is the improvement that can be expected when a suitable configurator is used. On average, the optimality gap was reduced by 0.27 (a 25.2% improvement over defaults). The greatest single improvement was seen on VRPSD-TS, where ParamILS was able to reduce the optimality gap on by 1.51, allowing an 81.2% improvement over defaults.

Before focusing on the differences between configuration methods, we compare the performance of more sophisticated methods against the reference configurator that was

Table 6 Configurator performance on the metaheuristics, which are split into three difficulty classes (D_c, 1 being easiest and 3 hardest)

Target	$\#P_{(B/I/R)}$	d.s.	D_c	CMA-ES	GGA	I/F-Race	ParamILS	REVAC	SMAC	URS	Rank on defaults	EB's		
												100	500	1000
VRPH-EJ-C	9 (7/2/0)	3	3			1				1	2	**1**	**1**	**2**
VRPH-RTR-C	14 (9/4/1)	3	3			1				1	3	3	**2**	**1**
VRPH-SA-C	11 (6/3/2)	1	1			1			1		**1**	**2**	**3**	**3**
VRPH-EJ-A	9 (7/2/0)	3	3			1			2	1	**1**	**2**	**2**	**3**
VRPH-RTR-A	14 (9/4/1)	3	3								3	**1**	**1**	**1**
VRPH-SA-A	11 (6/3/2)	2	2			2			2		2	**3**	**3**	**2**
VRPSD-ACO	8 (2/1/5)	3	1	3					4		**1**	**1**	**1**	**1**
VRPSD-EA	5 (3/1/1)	3	3			1			1		2	**2**	**2**	**2**
VRPSD-ILS	3 (2/0/1)	3	1		1						4	3	4	4
VRPSD-SA	6 (2/2/2)	1	1		1						3	5	5	5
VRPSD-TS	5 (2/0/3)	2	2	3			3				5	4	**3**	**3**
	Total wins			6	2	7	3	0	**10**	3				

Based on the results, the suitability of the default parameters (d.s.) is estimated. Here 1 stands for good default parameters. Middle columns keep score for the statistically significantly best configurators for each target. The $\#P$ column indicates the number of parameters for each metaheuristic, and $\#P_B$, $\#P_I$, and $\#P_R$ their division into **B**oolean, **I**nteger, and **R**eal valued parameters. Rightmost columns show the ranking between VRP solvers with the default and the automatically configured parameters. Also, the table illustrates how the ranking changes when the solvers are configured. Results with bold ranks are of better or equal utility in comparison to the best solver for that instance set with default parameters

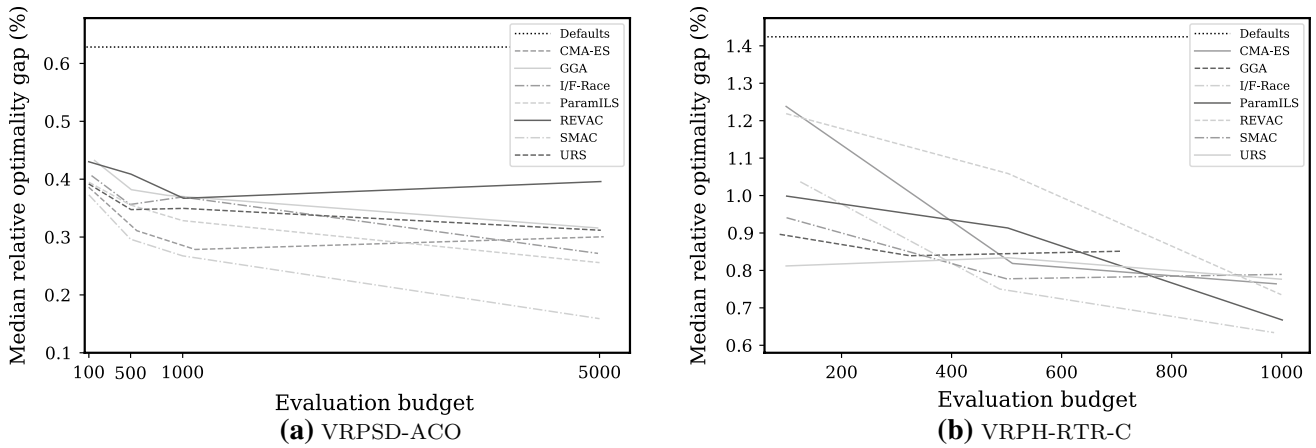
the uniform random sampling (URS). Contrary to expectations, the configurators are able to produce statistically significantly better results over URS only in 35.8% of the pairwise Mann–Whitney U tests ($p < 0.05$). In contrast, the observed performance was worse than URS in 35.3% of the pairwise comparisons. However, as can be seen from the main result tables (Tables 3, 4, and 5), the results in contrast to URS are not evenly distributed. Additionally, the two Augerat et al. [2] instance sets were included to see how well the performance gains of automatic algorithm configuration generalize to similar problems. Therefore, it was expected a see that random strategy (URS) works well. According to Coy et al. [14], such behavior can be caused by a large heterogeneity among the problem instances in a problem set, but it seems this applies also to heterogeneity between training and validation sets. Also I/F-Race, and to some extent SMAC, show a good generalization ability from a problem set to another on these targets.

Another noteworthy observation is that REVAC seems to struggle with all algorithm targets and it is able to beat URS only in 6.1% of the pairwise parameter configuration comparisons. As a whole, our results indicate that performance of REVAC on routing targets is worse than that of SMAC and I/F-Race. This is in contrast to results from Montero et al. [44], where they reported only small differences between F-Race, ParamILS, and REVAC in automatically configuring an EA for the TSP. If we leave out the Augerat et al. [2] targets and REVAC from the pairwise comparisons against URS, configurators are better than URS in 50.4% and worse in 26.4% of the tests. In addition, as the evaluation budget is increased, the advantages of more sophisticated configuration methods become more apparent (see, e.g., ACO in Table 5).

Table 6 shows that SMAC, I/F-Race, and CMA-ES are the methods that most frequently tend to find good parameter configurations for the VRP metaheuristics in this study. However, please note that CMA-ES, I/F-Race and GGA have the tendency to exceed the specified evaluation budget. Of the statistically significant results, only I/F-Race for the VRPH-SA-C and VRPH-SA-A targets with a budget of 100 exceeded the given budget by more than 15% (by 25 % to be exact) and this may give them some unfounded advantage. Still, considering the competitive performance of I/F-Race on those targets with budgets of 500 and 1000 this should not induce significant bias into our analysis.

I/F-Race, together with SMAC, and in some cases URS, seem to be the configuration methods to choose when faced with a highly limited computational budget. These methods are able to quickly produce relatively high-quality parameter configurations. However, no single method clearly dominates the others. The summary of winning configurators in Table 6 illustrates that different automatic algorithm configuration methods are successful with different targets, although if a method manages to find good parameter configurations for a target with a specific evaluation budget, it seems to be able do this with other budgets as well.

Regarding the parameter types and composition, our results support the observation made by Yuan et al. [64] that CMA-ES is suitable for configuration tasks with a high number of continuous parameters. We also note that I/F-Race



(a) VRPSD-ACO **(b)** VRPH-RTR-C

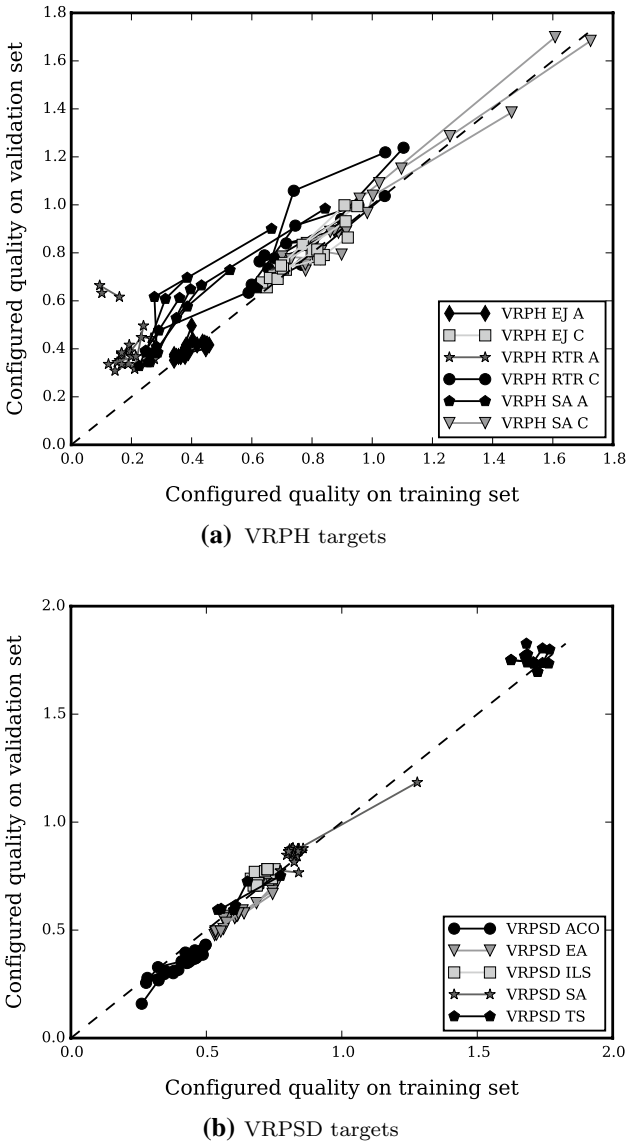
Fig. 3 Comparison of different configurators on two selected targets

seems to perform well in tasks that contain many Boolean parameters.

Random sampling (URS) works surprisingly well on VRPH-EJ, VRPH-RTR, and VRPSD-EA. The ruggedness of the configuration target fitness landscape probably interferes with the exploitation schemes of the more advanced automatic configuration methods. URS is, by definition, very explorative and is therefore capable of effectively exploring large areas of the parameter configuration search space. Hutter et al. [27] utilizes this in another configurator called ROAR, which can be described roughly as URS with configuration effort reduction techniques. However, as we can see from the result of configuring VRPSD-TS, sampling is not a strategy without disadvantages.

SMAC dominates in configuring VRPSD-ACO with a budget of 5000 evaluations (Fig. 3a). We also observe a possible case of over-tuning in the results of REVAC and CMA-ES. The effect is smaller with CMA-ES so there is a possibility that CMA-ES cannot effectively use the larger budget and prematurely converges to a local optimum.

Overall, despite the relatively small training set size, there is reasonably little over-tuning as can be seen from Fig. 4. In a case of over-tuning the figure would show good performance on training set, but poor performance on validation set. That is, the data point would be clearly above the dashed line designating unequal performance between the sets. The largest over-tuning effect is seen on the left side of the figure where SMAC automatically configures VRPH-RTR-A (Fig. 4a). In fact, VRPH-RTR-A and VRPH-SA-A targets show relatively large difference in training and validation set solution quality. This is due to the inherent differences of the training and validation sets with the Augerat et al. [2] targets. This is not surprising as this benchmark set was included to test how well the performance gains of automatic algorithm configuration transfer to solving similar problem



(a) VRPH targets

(b) VRPSD targets

Fig. 4 Solution quality on validation versus training set

instances. Furthermore, if the data is examined per configuration method, none of the configurators shows a clear tendency to over-tune.

To examine robustness, we study the median absolute deviations (MAD) in Tables 3, 4, and 5. Out of the tested configuration methods, SMAC is the most robust. That is, is able to consistently produce good parameter configurations. It closely followed by I/F-Race as they both have the lowest average MAD and still provide good automatic configuration performance. However, the differences over all experiments are small, and even SMAC fails to always produce good parameter configurations for configuring some targets such as VRPH-RTR-A and VRPSD-TS where in turn CMA-ES excels.

6.2 Configuration target difficulty

By comparing the configuration performance of URS against other methods in Tables 3, 4, and 5, we recognize three difficulty classes in the tested VRP algorithm targets (see Table 6). The first class consists of targets VRPH-SA-C, VRPSD-ACO, VRPSD-ILS, and VRPSD-SA, which seem to have relatively smooth parameter configuration landscapes where sophisticated intensification and search techniques work well. Pellegrini and Birattari [48] reported similar results that showed that ACO, ILS, and SA are metaheuristics that respond favorably to automatic configuring and that F-Race outperforms random sampling on these targets. Note that the default parameters for the targets VRPH-SA-C and VRPSD-SA seem to be already very good because only 25.6% of the parameter configurations produced by the configurators show improved performance over them. GGA seems to be the best method to automatically configure VRPSD-SA, although it, likewise, struggles to find better configurations than the defaults. For the other targets in this class, 95.3% of the produced configurations are better than the defaults. As we can see from Fig. 3, the results also seem to be getting better as we increase the evaluation budget.

The second class of automatic algorithm configuration problems contains VRPSD-TS and VRPH-SA-A. Configuration performance on these targets shows large variation. For VRPSD-TS, only CMA-ES and ParamILS are able to find parameter configurations that clearly outperform the defaults, whereas other methods are able to only slightly improve the solution quality. VRPH-SA-A shows similar behavior with high variance. For this algorithm, all of the configurators, except REVAC, were repeatedly able to produce a parameter configuration that allowed solving all of the 14 instances in the Augerat et al. [2] validation set to optimality. One such configuration is given later in Table 7.

In the third difficulty class, we have the targets VRPH-EJ, VRPH-RTR, and VRPSD-EA. Based on our experiments, these seem to be hard to configure effectively and even the more sophisticated automatic algorithm configuration methods struggle to challenge the uniform random sampling on these targets. As can be seen from Table 6 these targets share the feature of having many binary parameters. If we examine the boxplot of Fig. 5, the multimodal nature of these configuration targets can be seen as clustering of outliers around a local optimum of the configuration search space. However, even for these targets, the configurators were able to improve the solution quality over the default parameter configuration with a success rate of 93.8%. Additionally, improvements were often found even with an evaluation budget as small as 100.

Our experiments clearly indicate that the nature of the configured target or, more specifically, the solver algorithms and the problem instance set, has great impact to the configurability, configuration method selection, and generic performance of the solver. In Table 6, the solver performance is compared among the metaheuristics solving the same problem set. In solving the CVRP, we can see that VRPH-RTR clearly benefits from using automatic algorithm configuration. For the CMT instances, VRP-SA-C produces the best-quality solutions with default parameters, but after configuration has been performed, it is beaten by VRPH-RTR-C and VRPH-EJ-C. With Augerat et al. [2] instances on small configuration budgets VRPH-EJ-A and VRPH-RTR-A are performance-wise very similar. With an evaluation budget of 1000, GGA is able to find very good parameters for VRPH-RTR-A, which outperforms the other two solvers on the Augerat instance set. Note that the large median absolute deviation in VRPH-SA-A results indicates that there is a lot of variation between the configured parameter configurations or their evaluations, which means that this good performance is inconsistent. The reason behind this may be in the optimal cooling schedule band for SA is known to be narrow [43]. For the IRIDIA instances, automatic configuration changes the ranking between the solvers only slightly. VRPSD-ACO is the winner in solving given VRPSD instances with VRPSD-EA being a close second, not surprisingly given the state-of-the-art performance of the evolutionary approach in the literature [50, 60].

6.3 Automatically configured parameters

The parameter configurations with the best median solution quality can be found in Table 7. However, it is likely that the parameter values are highly instance and solver implementation specific, which limits our ability to make general recommendations. Also, please remember that a 10 s cutoff was used in our experiments, and this should be considered when generalizing the parameter values. Still, the best found parameter configurations offer a basis for our discussion on algorithm nature and solution space structure in the

Table 7 The best median solution quality Q for a single parameter configuration

			Local search operators	Optimized parameters							
	Q_C	Q_A	1*pm*/2*pm*/*two*/*oro*/*tho*/3*pm*	m	t	s					
VRPH EJ default	0.96	0.73	1 / 1 / 1 / 0 / 0 / 0	10	1000	0					
VRPH EJ C (SMAC)	0.54		1 / 1 / 1 / 0 / 1 / 1	17	7465	1					
VRPH EJ A (SMAC)		0.34	1 / 1 / 1 / 0 / 0 / 1	19	5316	1					
	Q_C	Q_A	1*pm*/2*pm*/*two*/*oro*/*tho*/3*pm*	D	δ	K	N	P	p/a	t	
VRPH RTR default	1.42	1.40	1 / 1 / 1 / 0 / 0 / 0	30	0.01	5	4	1	1/1	0	
VRPH RTR C (GGA)	0.40		1 / 1 / 1 / 1 / 0 / 1	18	0.01	51	11	4	0/0	39	
VRPH RTR A (GGA)		0.01	1 / 1 / 1 / 1 / 0 / 0	98	0.04	43	30	6	1/0	6	
	Q_C	Q_A	1*pm*/2*pm*/*two*/*oro*/*tho*/3*pm*	T	n	i	α	N			
VRPH SA default	0.80	0.90	1 / 1 / 1 / 0 / 0 / 0	2.00	200	2	0.99	10			
VRPH SA C (GGA)	0.63		1 / 1 / 1 / 0 / 0 / 0	2.00	200	5	0.99	10			
VRPH SA A (SMAC)		0.01	1 / 1 / 1 / 1 / 1 / 1	8.79	498	5	0.99	23			
			Obj.f. est.	Optimized parameters							
	Q	p	t	m	au	ψ	ρ	q	α		
VRPSD ACO default	0.63	0	0	7	0.50	0.30	0.10	1.0e7	1.00		
VRPSD ACO (GGA)	0.15	0	0	1	0.53	0.85	0.41	4.2e6	3.12		
	Q	p	t	p	mr	amr					
VRPSD EA default	0.77	0	0	0	0.20	0					
VRPSD EA (GGA)	0.42	1	1	1	0.63	1					
	Q	p	t	x							
VRPSD ILS default	0.78	0	0	10.00							
VRPSD ILS (SMAC)	0.70	0	0	29.80							
	Q	p	t	μ	α	ψ	ρ				
VRPSD SA default	0.79	0	0	0.01	0.98	1	20				
VRPSD SA (GGA)	0.77	0	0	0.08	0.18	1	20				
	Q	p	t	ttf	p_t	p_o					
VRPSD TS default	1.86	0	0	0.80	0.80	0.30					
VRPSD TS (CMA-ES)	0.51	1	0	1.00	1.00	1.00					

conclusions. Table 7 also allows comparison of parameter values and resulting solution quality between the default configuration and the configured one. Out of the compared configurators SMAC and GGA seem to be most successful in finding very good parameter configurations. If this evidence is combined with observations from boxplots such as the one presented in Fig. 5, the overall impression is that SMAC has more consistent performance, while GGA is occasionally able find better configurations.

Analysis of the configured parameter configurations reveals that in VRPSD-TS, where we observe strikingly different performance between two groups of configurators, the good utility is achieved when at least one of the values for the parameters ttf, p_t, and p_o is at the minimum or maximum. This can also be seen from Table 7. Statistically, it is improbable for a uniform sampling to produce exactly the parameter endpoint value of an interval. Therefore, methods that uniformly sample from within the given range are unable to find these good parameter configurations for VRPSD-TS, whereas configurators that use a robust statistical model or local search are well-suited to the task. The effect was not considered by Balaprakash et al. [3] when they introduced the iterative sampling extension to F-Race, and, to our knowledge, this effect has not previously been

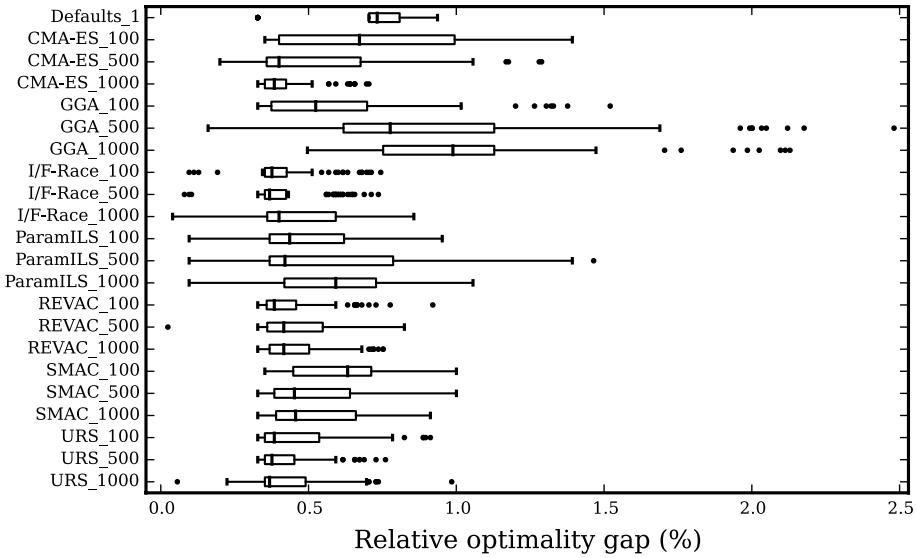


Fig. 5 The distribution of the parameter configuration utility for VRPH-EJ-A target

reported in automatic algorithm configuration literature. The original F-Race would probably find the values of these good parameter configurations, given parameter range end points are chosen as design points in the factorial design. However, VRPSD-TS is a special target in this regard, and F-Race based on factorial design candidate configuration generation would probably show far worse automatic configuration performance on different kind of targets. This is especially true in our configuration scenarios because a full factorial design requires a rather large configuration budget. Also, note that Pellegrini and Birattari [48] did not use the iterative variant of F-Race, and, thus, this behavior did not manifest in their results.

If we now turn to the resulting parameter configurations of the VRPH targets, we can examine how the probability of a local search heuristic to be selected changes with the metaheuristic and the instance set (see Fig. 6). Out of the tested targets, VRPH-SA-C seems to somewhat differ from the rest in its composition of local search operators. With this algorithm, configurations that avoid the more computationally intensive Or-opt, three-opt, and three-point-move operations yield higher utility (routes with a lower cost). Also, in VRPH-SA-C the selection of the operator plays a major role in the resulting solution quality as the local search operator composition of the top 10% parameter configurations differs clearly from the worst 90%. A similar effect can be observed in VRPH-EJ-A, where the use of two-opt operators is preferred over other operations.

It seems that definite connections exist among the composition of local search operators, performance of a metaheuristic, and the instances to be solved. Automatic algorithm configuration makes it possible to find suitable local search operator composition to optimize the performance of a routing solver. This verifies the observation made by Garrido et al. [17] that careful selection of local search operators for a set of instances is a relatively stable way of improving the overall performance of a solver. However, in this study we refrain from examining the differences in local search operator selection between the configurators further.

7 Conclusions and future research

In this paper, we have presented a comprehensive empirical evaluation of seven well-known automatic algorithm configuration methods in the task of configuring eight metaheuristic algorithms solving two vehicle routing problem (VRP) variants. The tested configurators were CMA-ES, GGA, I/F-Race, ParamILS, REVAC, SMAC, and URS. The VRPH library, which is used to solve capacitated vehicle routing problems, offers three solvers with EJ, SA, and RTR metaheuristics. The IRIDIA solvers for the VRPSD uses ACS, EA, ILS, SA, and TS metaheuristics. The solvers had from 3 to 14 free parameters. Each configurator was given a task to find a parameter configuration producing high quality solutions for each algorithm used to solve a relatively small benchmark set of VRP instances. Runtime of the solvers was limited to 10 s.

The results show that, in general, the configuration methods were able to find parameter configurations that produced better solutions than the solver default, even when restricted to as little as 100 solver invocations. This is consistent with previous research where it has been shown repeatedly that automatic algorithm configuration can remarkably improve the performance of stochastic search algorithms over the default parameters. Despite this prior assumption, the low computational cost of achieving performance improvement can be considered surprising. Using just a plain random uniform sampling strategy with a highly limited computational budget would often produce a clearly better performing

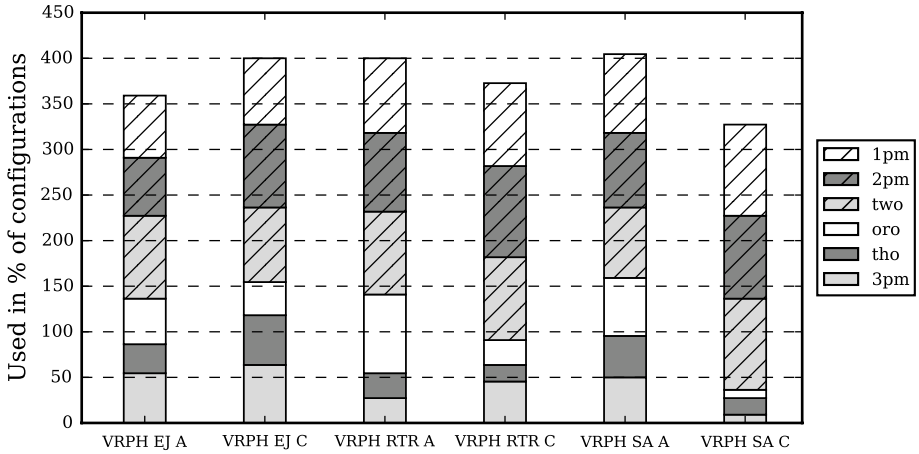


Fig. 6 Local search operator composition of the 10% best VRPH parameter configurations for each VRPH target

parameter configuration than the defaults. Occasionally, random sampling even beat the more advanced configurators by a clear margin as sophistication does not dominate in cases where the solution space has little structure to exploit.

To answer the question of configuration method suitability, our analysis suggests that there is no single best automatic algorithm configuration method for the tested VRP metaheuristics. However, the statistically significant evidence in this study verified that CMA-ES is a good choice when dealing with targets that have many continuous parameters, and that I/F-Race is well-suited for algorithm configuration targets that have many on-off switches for enabling and disabling solver features. Our experimentation also revealed that GGA and REVAC require a lot of trial-and-error and expertise to find parameter values that enable them to use the evaluation budget effectively. This creates an additional level of parameters to tweak on top of the original problem, which makes it hard to effectively apply these configuration methods.

We argue that robustness, and being parameter-free, are desirable properties for an automatic configuration method. Based on our survey, out of the tested configurators CMA-ES, I/F-Race, ParamILS, SMAC, and URS fulfill these requirements. If good performance and robustness is required, and a relatively generous evaluation budget is available, we would recommend SMAC and I/F-Race. Based on our experiments they are both capable of reliably producing good quality parameter configurations. Also GGA is in some situations a competitive choice, but in our experiments it was not as robust as SMAC and I/F-Race. While we could not give a definite recommendation on which single configurator one should use to configure VRP metaheuristics, the results together with the provided survey should help VRP researchers and practitioners to select a method that is probably a good fit. Also, confirming that these results apply with other metaheuristics, time limits, and problem instance sizes will warrant additional computational experiments of configuring VRP algorithms.

The way a target algorithm responded to configuration efforts varied between configurators, evaluation budgets, and even between problem instance sets. We acknowledge that the time limits do affect the results of the comparison of metaheuristics or configuration methods. In our experiments, we varied the evaluation budget and tested the configurator performance on three problem instance sets, but used the same time limit in all experiments. Considering this, the results of our experiments conclude that there does not exist a single, best configuration method for different algorithms. However, we were able to distinguish differing configurator behavior with the different evaluation budget constraints. The results suggest that SMAC and I/F-Race would be most appropriate configurators for larger instances with longer execution times. Our study also showed that in our set of configuration problems some configurators are more robust than others.

Our recommended strategy to address the inconclusive nature of the results is to have several state-of-the-art automatic algorithm configuration methods at the user's disposal. Experimenting with different configurators helps one to see when a good fit is found, as the solver usually responds quickly to automatic configuration attempts even with a small evaluation budget. Furthermore, our findings have important implications for future practice. Contributed evidence to the usefulness of automatic algorithm configuration of VRP metaheuristics strongly suggests that routing algorithm developers should start using an automatic algorithm configuration method in their experiments. This is important in particular when making algorithm performance comparisons, as configuring the parameters of a set of algorithms allows one to avoid confirmation bias, that is, the performance of the algorithm is not determined by the suitability of its default parameters or the amount of manual fine-tuning it receives.

Regarding generalization of the results, we see from Table 1 that the included configurators address a large set of different features and aspects of automatic algorithm

configuration. The same holds true for the various features represented in benchmark problems and their solvers as depicted in Table 2: we hypothesize that the experimental results hold true also for different mixtures of the same solution method constituents. The use of these approaches is common in designing heuristics for combinatorial optimization problems [54].

Additional and extensive comparison between these configurators with different experiment parameters, e.g., with larger, more difficult, or 'rich' [12] problem instances or with a longer metaheuristic CPU runtime, would be required to reliably estimate how well our observations generalize. The recent advances in the field of automatic algorithm configuration addressing the issues with long running algorithms are relevant here [18, 28, 41, 57]. The experiments could also be extended with configuration targets that have more binary and categorical parameters.

A typical use for a routing solver is to solve sets of slightly different problem instances repeatedly. Automatic configuration in such a scenario can be considered as modeling the interactions of the triplet: instance, parameter configuration, and solution quality. Further work is required to establish the feasibility of utilizing these previously discovered interactions in future solving tasks. This research avenue is also recognized, e.g., in [61]. The reasonable next step could be to explore the feature extraction of VRP instances, solutions, and routes, and then investigate the suitability of instance-specific algorithm configuration methods. These methods use instance features to make utility predictions for the parameter configuration candidates. SMAC can be used as instance-specific method, but there are other methods, such as ISAC from Kadioglu et al. [34]. Also, of particular interest from practical operations research and vehicle routing viewpoint would be extending our investigations to algorithm selection. Especially applicability and implications of using algorithm selectors, such as Hydra [63] or AutoFolio from Lindauer et al. [36], should be explored.

It is also important to acknowledge the drawbacks of automatic algorithm configuration. The configuration methods rarely provides useful information on why a certain parameter configuration was selected. Here, domain knowledge and understanding of the target algorithm is required to understand the general implications of the resulting parameter configuration. This is especially important in academic research, where understanding why an optimization strategy works is of paramount importance. Therefore, we see implementing in features like parameter sensitivity analysis and visualization of the parameter configuration search space as important development and research aims of configuration method community.

Regarding validity of the study, we would like discuss four things: configuration budget, CPU cutoff, problem set size, configuration objective, and generalization of the results to other VRP variants. The possible limitations of the study are due to the extensive computational requirements required with each additional variability dimension introduced to the comparison. Also, the research questions and the specifics in solving vehicle routing problems made it possible, or in some cases necessary, to fix some aspects of the experimental setup.

In this study an evaluation budget was used to limit the computational resources available during automatic algorithm configuration. However, some configuration methods failed to adhere to this budget. To control the effect this issue might have on validity, we have addressed deviations from the budget in our analysis.

We also acknowledge that future comparisons should study the effect of a more generous configuration budget on configuring VRP metaheuristics. In our study we tested a single target with a budget of 5000 evaluations, which does not allow analyzing the variation between configuration targets in the scenario of a larger budget.

The large number of experiments, and a decision to keep as many of the variables constant as possible in the experimental setup lead us to limit the CPU time of the solvers to 10 s. We experimentally verified that the VRPH solver performance stabilizes by the 10 s mark. However, it is likely that this creates a bias to prefer parameter configurations specifying a more explorative search strategy, especially on larger instances of the CMT problem set and when solving VRPSD instances. Similarly. the choice to use a problem set size of 14 was done to keep it constant over all three configuration target groups, and to include automatic algorithm configuration methods that lack training set subset evaluation mechanisms. We acknowledge that when using a small training set, there is a danger of overfitting. However, in our experiments only few selected configuration targets show weak signs of such behavior, and these do not affect our overall results and recommendations.

VRP is a challenging, well-known, and well-studied combinatorial optimization problem that generalizes several other problems. Therefore, it can serve as an interesting benchmark for evaluating the robustness of automated algorithm configuration methods and tools. In this study we focused only on CVRP and VRPSD, but there are many other variants with different constraints, objectives and features. Also, even different problem instances of a single variant can have differing characteristics (see, e.g., [51]). Thus, we would like to see automatic algorithm configuration method comparisons on the VRPTW (VRP with time windows), PDP (pickup and delivery problem), large-scale CVRP, and rich VRP benchmarks, which could show different facets of configuring VRP solvers and perhaps provide further support for our results.

Acknowledgements Open access funding provided by University of Jyväskylä (JYU). The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development for Nyset Musliu is gratefully acknowledged.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I. (ed.) Principles and Practice of Constraint Programming - CP'09. Lecture Notes in Computer Science, vol. 5732, pp. 142–157. Springer, Berlin (2009)
2. Augerat, P., Belenguer, J., Benavent, E., Corberán, A., Naddef, D., Rinaldi, G.: Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report 949-M, Universite Joseph Fourier, Grenoble, France (1995)
3. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-Race algorithm: sampling design and iterative refinement. Technical Report TR/IRIDIA/2007-011, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2007)
4. Barbucha, D.: Experimental study of the population parameters settings in cooperative multi-agent system solving instances of the VRP. In: Nguyen, N.T. (ed.) Transactions on Computational Collective Intelligence IX. Lecture Notes in Computer Science, vol. 7770, pp. 1–28. Springer, Berlin (2013)
5. Bartz-Beielstein, T., Lasarczyk, C., Preuß, M.: Sequential parameter optimization. In: IEEE Congress on Evolutionary Computation—CEC'05, vol. 1, pp. 773–780 (2005)
6. Battiti, R., Brunato, M.: Reactive search optimization: learning while optimizing. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, 2nd edn, pp. 543–571. Springer, New York (2010)
7. Becker, S., Gottlieb, J., Stützle, T.: Applications of racing algorithms: an industrial perspective. In: Proceedings of the 7th International Conference on Artificial Evolution—EA'05, pp. 271–283. Springer, Berlin (2006)
8. Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O., Schiavinotto, T.: Hybrid metaheuristics for the vehicle routing problem with stochastic demands. J. Math. Model. Algorithms **5**(1), 91–110 (2005)
9. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference—GECCO'02, pp. 11–18. Morgan Kaufmann, San Francisco, CA (2002)
10. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: an overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer, Berlin (2010)
11. Bräysy, O., Hasle, G.: Vehicle Routing: Problems, Methods, and Applications, chap. 12 Software tools and emerging technologies for vehicle routing and intermodal transportation, pp. 351–380. In: [58] (2014)
12. Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., Juan, A.A.: Rich vehicle routing problem: survey. ACM Comput. Surv. (CSUR) **47**(2), 32 (2015)
13. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. Revue Française d'Informatique et de Recherche Opérationnelle **10**(2), 55–70 (1976)
14. Coy, S.P., Golden, B.L., Runger, G.C., Wasil, E.A.: Using experimental design to find effective parameter settings for heuristics. J. Heuristics **7**, 77–97 (2001)
15. Eiben, A., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evolut. Comput. **3**(2), 124–141 (1999)
16. Eiben, A., Smit, S.: Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol. Comput. **1**(1), 19–31 (2011)
17. Garrido, P., Castro, C., Monfroy, E.: Towards a flexible and adaptable hyperheuristic approach for VRPs. In: Arabnia, H.R., de la Fuente, D., Olivas, J.A. (eds.) Proceedings of the 2009 International Conference on Artificial Intelligence—ICAI'09, pp. 311–317. CSREA Press, USA (2009)
18. Geschwender, D., Hutter, F., Kotthoff, L., Malitsky, Y., Hoos, H.H., Leyton-Brown, K.: Algorithm configuration in the cloud: a feasibility study. In: Pardalos, M.P., Resende, G.M., Vogiatzis, C., Walteros, L.J. (eds.) Learning and Intelligent Optimization: 8th International Conference, Lion 8, Gainesville, FL, USA, February 16–21, 2014. Revised Selected Papers, pp. 41–46. Springer (2014)
19. Groër, C.: Parallel and serial algorithms for vehicle routing problems. Dissertation, University of Maryland (2008)
20. Groër, C., Golden, B., Wasil, E.: A library of local search heuristics for the vehicle routing problem. Math. Program. Comput. **2**(2), 79–101 (2010)
21. Hansen, N.: The CMA evolution strategy: a comparing review. In: Lozano, J., Larranaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, pp. 75–102. Springer, Berlin (2006)
22. Hepdogan, S., Moraga, R., DePuy, G., Whitehouse, G.: Nonparametric comparison of two dynamic parameter setting methods in a meta-heuristic approach. J. Syst. Cybern. Inf. **5**(5), 46–52 (2008)
23. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 37–71. Springer, Berlin (2012)
24. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Elsevier, Amsterdam (2004)
25. Hutter, F., Bartz-Beielstein, T., Hoos, H., Leyton-Brown, K., Murphy, K.: Sequential model-based parameter optimization: an experimental investigation of automated and interactive approaches. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 363–414. Springer, Berlin (2010a)
26. Hutter, F., Hoos, H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Lodi, A., Milano, M., Toth, P. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Lecture Notes in Computer Science, vol. 6140, pp. 186–202. Springer, Berlin (2010b)
27. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer, Berlin (2011)

28. Hutter, F., Hoos, H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 7219, pp. 55–70. Springer, Berlin (2012)
29. Hutter, F., Hoos, H., Leyton-Brown, K., Murphy, K.: Time-bounded sequential parameter optimization. In: Blum, C., Battiti, R. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6073, pp. 281–298. Springer, Berlin Heidelberg (2010c)
30. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. J. Artif. Intell. Res. (JAIR) **36**, 267–306 (2009)
31. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, pp. 1152–1157. AAAI Press, Menlo Park, CA (2007)
32. Irnich, S., Toth, P., Vigo, D.: Vehicle Routing: Problems, Methods, and Applications, chap. 1 the family of vehicle routing problems, pp. 1–33. In: [58] (2014)
33. Jourdan, L., Basseur, M., Talbi, E.G.: Hybridizing exact methods and metaheuristics: a taxonomy. Eur. J. Oper. Res. **199**(3), 620–629 (2009)
34. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC—instance-specific algorithm configuration. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) 19th European Conference on Artificial Intelligence—ECAI 2010, Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 751–756. IOS Press, Amsterdam, Netherlands (2010)
35. Laporte, G.: What you should know about the vehicle routing problem. Nav. Res. Log. **54**(8), 811–819 (2007)
36. Lindauer, M., Hoos, H., Hutter, F., Schaub, T.: Autofolio: An automatically configured algorithm selector. J. Artif. Intell. Res. **53**, 745–778 (2015)
37. López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles (2011)
38. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, Chap. 12, vol. 146, 2nd edn, pp. 363–397. Springer, New York (2010)
39. Maron, O., Moore, A.W.: Hoeffding races: Accelerating model selection search for classification and function approximation. In: Cowan, J., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing Systems, vol. 6, pp. 59–66. Morgan Kaufmann, San Francisco (1994)
40. Maron, O., Moore, A.W.: The racing algorithm: model selection for lazy learners. Artif. Intell. Rev. **11**(1–5), 193–225 (1997)
41. Mascia, F., Birattari, M., Stützle, T.: Tuning algorithms for tackling large instances: an experimental protocol. In: Nicosia, G., Pardalos, P. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 7997, pp. 410–422. Springer, Berlin (2013)
42. Mester, D., Bräysy, O.: Active-guided evolution strategies for large-scale capacitated vehicle routing problems. Comput. Oper. Res. **34**(10), 2964–2975 (2007)
43. Miki, M., Hiroyasu, T., Jitta, T.: Adaptive simulated annealing for maximum temperature. In: 2003 IEEE International Conference on Systems, Man and Cybernetics—SMC 2003, vol. 1, pp. 20–25 (2003)
44. Montero, E., Riff, M., Neveu, B.: New requirements for off-line parameter calibration algorithms. In: 2010 IEEE Congress on Evolutionary Computation—CEC'10, pp. 1–8 (2010)
45. Montero, E., Riff, M.C., Neveu, B.: An evaluation of off-line calibration techniques for evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference—GECCO'10, pp. 299–300. ACM, New York (2010)
46. Nannen, V., Eiben, A.E.: Efficient relevance estimation and value calibration of evolutionary algorithm parameters. In: 2007 IEEE Congress on Evolutionary Computation—CEC'07, pp. 103–110 (2007)
47. Pellegrini, P.: Application of two nearest neighbor approaches to a rich vehicle routing problem. Technical Report TR/IRIDIA/2005-015, IRIDIA, Université Libre de Bruxelles (2005)
48. Pellegrini, P., Birattari, M.: Implementation effort and performance. In: Stützle, T., Birattari, M., Hoos, H.H. (eds.) Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics. Lecture Notes in Computer Science, vol. 4638, pp. 31–45. Springer, Berlin (2007)
49. Penna, P.H.V., Subramanian, A., Ochi, L.S., Vidal, T., Prins, C.: A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. Ann. Oper. Res. **273**(1–2), 5–74 (2017)
50. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Comput. Oper. Res. **31**(12), 1985–2002 (2004)
51. Rasku, J., Kärkkäinen, T., Musliu, N.: Feature extractors for describing vehicle routing problem instances. In: SCOR, OASICS, vol. 50, pp. 7:1–7:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
52. Rasku, J., Musliu, N., Kärkkäinen, T.: Automating the parameter selection in VRP: an off-line parameter tuning tool comparison. In: Fitzgibbon, W., Kuznetsov, A.Y., Neittaanmäki, P., Pironneau, O. (eds.) Modeling, Simulation and Optimization for Science and Technology, Proceedings of Optimization and PDEs with Applications Workshop, June 18–19, 2012, University of Jyväskylä, Finland, Computational Methods in Applied Sciences, vol. 34, pp. 191–209. Springer (2014)
53. Rasku, J., Puranen, T., Kalmbach, A., Kärkkäinen, T.: Automatic customization framework for efficient vehicle routing system deployment. In: Diez, P., Neittaanmäki, P., Periaux, J., Tuovinen, T., Bräysy, O. (eds.) Computational Methods and Models for Transport: New Challenges for the Greening of Transport Systems, pp. 105–120. Springer, New York (2018)
54. Sevaux, M., Sörensen, K., Pillay, N.: Adaptive and multilevel metaheuristics. In: Martí, R., Panos, P., Resende, M. (eds.) Handbook of Heuristics, pp. 1–19. Springer, Cham (2018)
55. Smit, S.K., Eiben, A.E.: Comparing parameter tuning methods for evolutionary algorithms. In: 2009 IEEE Congress on Evolutionary Computation—CEC'09, pp. 399–406 (2009)
56. Sörensen, K., Sevaux, M., Schittekat, P.: Multiple neighbourhood search in commercial VRP packages: evolving towards self-adaptive methods. Stud. Comp. Intell. **136**, 239–253 (2008)
57. Styles, J., Hoos, H.: Using racing to automatically configure algorithms for scaling performance. In: Nicosia, G., Pardalos, P. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 7997, pp. 382–388. Springer, Berlin (2013)
58. Toth, P., Vigo, D.: Vehicle Routing: Problems, Methods, and Applications. SIAM, Philadelphia (2014)
59. Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A.: New benchmark instances for the capacitated vehicle routing problem. Eur. J. Oper. Res. **257**(3), 845–858 (2017)
60. Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W.: A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. **60**(3), 611–624 (2012)
61. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur. J. Oper. Res. **231**(1), 1–21 (2013)

62. Wink, S., Back, T., Emmerich, M.: A meta-genetic algorithm for solving the capacitated vehicle routing problem. In: IEEE Congress on Evolutionary Computation—CEC'12, pp. 1–8 (2012)
63. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: Automatically configuring algorithms for portfolio-based selection. In: Twenty-Fourth Conference of the Association for the Advancement of Artificial Intelligence (AAAI-10), pp. 210–216 (2010)
64. Yuan, Z., de Oca, M.A.M., Birattari, M., Stützle, T.: Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. Swarm Intell. **6**(1), 49–75 (2012)