Self-adaptive Decomposition and Incremental Hyperparameter Tuning Across Multiple Problems

Jialin Liu and Xin Yao†
Shenzhen Key Laboratory of Computational Intelligence
University Key Laboratory of Evolving Intelligent Systems of Guangdong Province
Department of Computer Science and Engineering
Southern University of Science and Technology
Shenzhen 518055, China
Email: liujl@sustech.edu.cn, xiny@sustech.edu.cn

***Abstract*—The Capacitated Arc Routing Problem (CARP) is a NP-hard combinatorial optimisation problem with numerous real-world applications. Several divide-and-conquer approaches, controlled by one or more hyperparameters, have been proposed to tackle large-scale CARPs. The tuning of hyperparameters can be computationally expensive due to the lack of priori knowledge, the size of the configuration space, and the time required for solving a CARP instance. Motivated by this time consuming task, we propose a scalable approach based on self-adaptive hierarchical decomposition (SASAHiD) to scale up existing methods. We take a state-of-the-art decomposition method for large-scale CARPs called SAHiD as an example to carry out experiments on two sets of real-world CARP instances with hundreds to thousands of tasks. The results demonstrate that SASAHiD outperforms SAHiD significantly with fewer hyperparameters, thus the dimension of associated configuration space is reduced. Moreover, we propose an incremental hyperparameter tuning approach across multiple problem instances to learn the hyperparameters of SASAHiD on a set of instances with different sizes. SASAHiD with optimised hyperparameters achieves better or competitive results with the SASAHiD using default hyperparameters when solving problem instances that it has never seen in the training set.**

***Index Terms*—Capacitated arc routing problem, self-adaptive decomposition, incremental hyperparameter tuning, permutation-based optimisation**

I. Introduction

The capacitated arc routing problem (CARP) [1], [2] is a NP-hard combinatorial problem with numerous real-world applications. The CARP aims to allocate efficiently a fleet of vehicles and select the optimal set of routes to deliver a given set of items (e.g., costumers or goods) on time to the given destination with the constraint that the total demand of tasks served on any route does not exceed the vehicle capacity, where each route refers to a subset of arcs or edges in a transport network. The objective is to minimise the total cost while satisfying all the demands or costumers. Different from the well-known vehicle routing problem (VRP) [3], the tasks locate on the edges in the CARP. Examples include the urban waste collection, snow removal and street salting problems [4]. Because of its huge number of real-world applications, several variants of CARPs have been studied, which differ from the characteristic of capacity, demand, and time windows [2], [5].

A large number of associated algorithms have been proposed to solve CARP and its variants, including exact methods and heuristics methods [2]. However, only a few recent work [6]–[9] focus on the large-scale CARP, which is more computationally expensive to evaluate a solution or find an optimal solution, but is more relevant to real-world applications. The above work rely on some decomposition approaches, such as the cooperative coevolution (CC) [6]–[8] and the most recently proposed hierarchical decomposition (HD) [9]. These decomposition approaches usually consist of one or more hyperparameters, e.g., the number of subcomponents and population size in the CC, and the number of clusters in the HD. The hyperparameters are usually set as arbitrarily chosen values or values chosen using grid search (e.g., [9]) or some preliminary testing [6]. Sophisticated algorithm configuration methods are rarely used due to the lack of priori knowledge (e.g., dependencies between parameters), the size of the configuration space, and the time required for solving a large-scale CARP instance or finding a good approximate solution.

In this work, we propose a scalable approach based on self-adaptive hierarchical decomposition (SASAHiD) scheme, which successfully eliminates a hyperparameter compared to the state-of-the-art hierarchical decomposition. In particular, the proposed SASAHiD can be beneficial for permutation-based optimisation problems, such as vehicle routing and scheduling problems. Therefore some real-world large-scale CARP instances, generated using 2 cities' road maps, are considered as test problems in this work. To examine the performance of SASAHiD, we compare it to the SAHiD proposed by [9] to carry out empirical studies, because SAHiD outperformed 3 state-of-the-art algorithms on solving large-scale CARPs with up to 3584 tasks. The results show that our SASAHiD, consisted of fewer parameters, achieves better

†X. Yao (xiny@sustech.edu.cn) is the corresponding author.

This work was supported by National Key R&D Program of China (Grant No. 2017YFC0804003), National Natural Science Foundation of China (Grant No. 61906083, 61976111), Shenzhen Peacock Plan (Grant No. KQTD2016112514355531), the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284, JCYJ20180504165652917) and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

or similar performance than SAHiD on the test problems. Moreover, we propose an incremental hyperparameter tuning approach across multiple problem instances and perform experiments using SMAC [10] as configurator. The hyperparameters are optimised in an incremental manner on the instance set of one city, with which the SASAHiD obtains better or competitive results when solving instances of another city that have never been seen before. To the best of our knowledge, we are the first to propose a self-adaptive hierarchical decomposition scheme and perform incremental hyperparameter tuning approach across multiple problem instances on the proposed SASAHiD, in particular, for solving large-scale CARP instances.

The rest of this paper is structured as follows. We start by introducing the background of this work. We then raise the conducted questions and propose the SASAHiD and the incremental hyperparameter tuning approach across multiple problem instances. Experiments are designed for answering our conducted questions and validate the proposed approaches. Then the experimental results are summarised and discussed. Finally we concludes the work and point out some potential future directions.

II. Background

A. Capacitated Arc Routing Problem

We consider the undirected version of capacitated arc routing problem (CARP) [1], [3], which is defined as follows. Let $G = (V, E)$ be a connected undirected graph where $V = \{0, \ldots, n\}$ is the set of vertices, the vertex v_0 is the depot and the others correspond to n customers. E refers to the edge set. Each edge $e \in E$ has a cost $c(e) > 0$, which could be a distance or time. If there is a task on this edge, a positive demand $d(e)$ is associated to e, otherwise, $d(e) = 0$. The task set T is the ensemble of edges with $d(e) > 0$. The objective of CARPs is to allocate efficiently a fleet of vehicles, which start and end at the identical depot v_0, and select the optimal set of routes to serve tasks while the total demand of tasks served on any route does not exceed a given vehicle capacity Q.

Because of its huge number of real-world applications, several variants of CARPs have been studied [2], [5]. However, the studied CARP instances usually consist of a small number of tasks, until very recently, [9] used two sets of real-world CARP instances with up to 3584 tasks. In this work, we test our approaches on the sets of large-scale CARP instances used by [9][1], which are the sets with highest task number in the literature.

1) Benchmark Sets.: Two benchmark sets, namely Hefei and Beijing, have been generated from the map of the Hefei city and the central area of Beijing city in China by [9]. Readers are referred to the Section V.A and Table 1 of [9] for more about the sets. Highlights are given as follows. Hefei set consists of 850 vertices and 1212 edges (main roads in the map), while Beijing set has 2820 vertices and 3584 edges. 10 instances have been sampled from Hefei set (Beijing set) by selecting uniformly at random a certain number of edges as tasks, denoted as Hefei-i (Beijing-i), respectively, where $i \in \{1, \ldots, 10\}$. The index i indicates the percentage of edges to be selected as tasks with a factor of 10. Thus, Hefei-1 consists of $|T_1| = 121$ tasks and Hefei-10 consists of $|T_{10}| = 1212$ tasks. The vehicle capacity is 9000 and 25000 for Hefei and Beijing instances, respectively. The minimal number of vehicles needed to serve all the tasks varies from 7 to 69 [9].

B. Decomposition Approaches for Scaling Up

A large number of algorithms have been proposed to solve CARPs and its variants, including the exact methods (branch-and-bound, branch-and-cut, and branch-and-pricing algorithms), heuristics and meta-heuristics, such as cellular Genetic Algorithm [11]. Most search methods suffer the scalability problem. Divide-and-conquer is a natural idea to scale up existing search methods, while the potential dependencies among variables for nonseparable problems are often ignored. For this reason, [12] proposed a group-based problem decomposition approach, EACC, which is widely used to tackle large-scale CARPs [6]–[8]. Cooperative-coevolution-based methods divide a problem to groups of subproblems in a linear way. When the dimension of the problem scales up, either the number of groups or the number of subproblems in a group will increase significantly. An alternative is using a hierarchical structure. Tang et al. [9] proposed a scalable approach based on hierarchical decomposition, called SAHiD. SAHiD outperformed 3 state-of-the-art algorithms, Variable Neighbourhood Search [13], Tabu Search Algorithm 1 [14] and RDG-MAENS [7] on solving large-scale CARPs [9]. For its outstanding performance, we compare SASAHiD to SAHiD in this work.

C. SAHiD

The SAHiD aims to find a good permutation of tasks efficiently. Given a CARP instance, it hierarchically decomposes the tasks into subgroups of tasks and solves the induced subproblems recursively [9]. Each of the nodes at the bottom of the hierarchy represents a subgroup of ordered tasks, corresponded to a permutation of tasks, which is further considered as a virtual task. Each node on a nonbottom layer groups the virtual tasks of its children nodes and obtains a new virtual task of a larger size. The only root node (top layer) contains exactly one virtual task which is a permutation of all the tasks. A tree hierarchy of virtual tasks is built recursively and the number of tasks (or virtual tasks on nonbottom layers) decreases exponentially from the bottom to the top layer. To group virtual tasks, a well-known clustering technique, K-means algorithm [15], is applied, considering the virtual task with minimal average deadheading cost to other virtual tasks as the centroid of a cluster. The clustering results of K-means algorithm highly depend on the number of clusters (i.e., groups), K. Tang et al. [9] set K_l as an integer generated uniformly at random between 1 and $\lceil \beta K_{l-1} \rceil$, where β is

[1]Instances of the two sets are available at http://staff.ustc.edu.cn/~ketang/codes/LSCARPset.zip, provided by the authors of [9].

a hyperparameter $\in (0,1)$ and K_l denotes the number of virtual tasks to be grouped at layer l. The bottom layer is indexed by 1 and K_1 equals to $|T|$, the number of actual tasks of the given CARP instance. After generating a permutation of tasks or virtual tasks, a solution for this permutation can be obtained by splitting the permutation into feasible routes using the Ulusoy's splitting procedure [9]. Then a local search algorithm is used to further improve the solution.

1) Configuration of Hyperparameters: SAHiD has 5 hyperparameters: (i) $\beta \in (0,1]$, the shrink parameter introduced by the K-means algorithm; (ii) $\alpha \in (0,1)$, the probability of partitioning a route introduced by the Ulusoy's splitting procedure; (iii) $\theta \in (0,1)$, the acceptance threshold; (iv) σ a number of non-improved iterations to accept a worse solution, introduced by the Local Search; and (v) p, the number of routes to be selected to form an unordered list of tasks to be modified, introduced by the merge-split operator [16]. We would like to refer the audiences to [9] for more details about the parameters. [9] tested the performance of SAHiD with uniformly sampled values for α and β on 4 CARP instances with different numbers of tasks. σ and p were set as $10,000$ and 2. We consider α, β and θ in this work as σ depends on the number of possible iterations given a budget, and increasing p will lead to an increment of runtime.

D. Automatic Algorithm Configuration

A non-parameter-free algorithm or policy has one or more hyperparameters with continuous, discrete (including boolean) or categorical values, which have significant or minor impact on the algorithm performance. Algorithm configuration aims to search for the optimal values for hyperparameters with which the algorithm achieves optimal performance on a specific instance. Understanding the trend and landscape of the configurations will be helpful to study the parameters' sensibilities and exploit how the parameters work (together) on an algorithm. However, in real-world applications, it is not always possible to model the search space and the landscape is usually unknown. Moreover, an evaluation of a configuration could be very expensive due to the problem or the nature of the algorithm. The algorithm configuration is usually a complex black-box or grey-box optimisation problem without or with little priori knowledge. Different approaches have been proposed for automated algorithm configuration, and differ in if offline or online; if parallel or sequential; and if explicit models (also called response surfaces) are used or not. Examples of popular algorithm configuration approaches are GGA [17], F-Race [18], ParamILS [19], and SMAC [10]. [20] reviewed the advanced state-of-the-art in predicting the algorithm performance for hard combinatorial problems. [21] compared two racing algorithms, F-race and DW-Race, on configuring a stochastic local search (SLS) method for solving a university course timetabling problem [22]. Solving large-scale CARPs is more computationally expensive. In this work, we set our configurator as the SMAC [10], a state-of-the-art automatic algorithm configuration method which builds iteratively explicit regression models based on random forests. SMAC is selected as it has been shown to have outstanding performance for optimising parameters both on single and multiples problem instances [10].

III. Self-Adaptive Hierarchical Decomposition

The SAHiD has shown its strength on two new large CARP benchmark sets, mainly thanks to the HD scheme it has used. We propose a scalable approach based on self-adaptive hierarchical decomposition (SASAHiD) scheme to achieve fast problem decomposition yet with fewer hyperparameters, thus smaller configuration space.

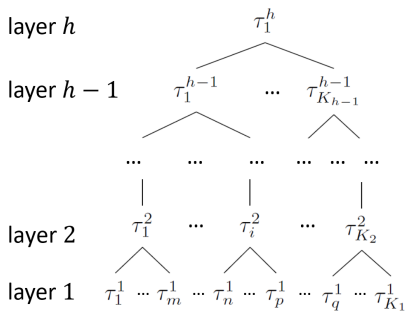

Fig. 1. Hierarchical decomposition. K_i, the number of clusters at layer i, is an integer randomly generated between 1 and a self-adaptive upper bound, as in (3), which leads to a stochastic number of layers h at each run of SASAHiD.

The hierarchical structure is illustrated in Figure 1. SAHiD builds recursively a tree hierarchy of virtual tasks, at each layer of which K-means algorithm [15] is applied to group the (virtual) tasks and create new virtual tasks (i.e., groups or clusters). For instance, at layer 3 in Figure 1, K_3 virtual tasks, $\{\tau_1^3, \ldots, \tau_{K_3}^3\}$, are created based on the K_2 virtual tasks at layer 2. SAHiD sets the cluster number K_i at layer i as an integer randomly selected between in a random range, formalised as follows:

$$\forall i \geq 2,\ K_i^{max} = \lceil RandomFloat(0,\beta) \times K_{i-1} \rceil, \quad (1)$$

$$K_i = RandomInteger(1, K_i^{max}), \quad (2)$$

where $\beta \in (0,1)$ is a hyperparameter. The lowest layer $i = 1$ has $K_1 = |T|$ leaf nodes, where T is the set of tasks. β guaranties a fast decreased upper bound for the number of intermediate nodes.

However, two random value generations have been involved in the design of cluster number, shown in (1) and (2), which can be redundant. Additionally, when using very low value for β, K decreases to 1 quickly given a number of tasks. We propose a self-adaptive upper bound as shown in (3) for K_i which also decreases exponentially from the bottom to the top layer, and gets rid of the hyperparameter β.

$$\forall i \geq 2,\ K_i = RandomInt(1, \lceil \sqrt{K_{i-1}} \rceil),\ K_1 = |T|. \quad (3)$$

The number of hyperparameters is reduced. At the same time, replacing a random number selection in a random range by a random number selection in a deterministic range, will reduce the randomness in the algorithms. We are not certain if the double-stochastic process in the non-self-adaptive hierarchical

decomposition is beneficial or not, and how the reduced randomness will affect the performance. To answer this question, we compare the performance of SASAHiD and SAHiD on the two benchmark sets that have been described previously.

A. Experimental Setting

Tang et al. [9] have tested the performance of SAHiD with uniformly sampled values for α and β within $(0, 1)$ on 4 CARP instances with different numbers of edges. It has been observed that for instances with higher number of edges, smaller α value (e.g., 0.1) leads to better solutions, while β has minor impact when α is small. So the results reported in [9] were all obtained with $\alpha = 0.1$ and $\beta = 0.1$ which were found to achieve best results on the 2 largest instances. We want to verify if SASAHiD using same α will achieve similar performance.

We set $\alpha = 0.1$, $\theta = 0.1$, $\delta = 10000$ and $p = 2$ as in [9] (reported in its Tables III-V). Both SAHiD and SASAHiD are given $30mins$ as budget. 25 experimental trials each have been performed, thus 50 optimisation trials in total for each of the problem instances. As the optimal solutions for CARP instances are unknown, we use the lowest cost found among the 50 trials of each CARP instance as a reference and refer to it as $\tilde{cost}^*$. We define a *simple regret* as the difference of the cost of best-so-far solution to the best value $\tilde{cost}^*$ obtained at the end of search, $SR = cost(current\ best) - \tilde{cost}^*$. As for each of the problem instances, the magnitude of the cost is different, we use a normalised regret $NSR = \frac{cost(current\ best) - \tilde{cost}^*}{\tilde{cost}^*}$. All the experiments presented in this paper are run on the same computing platform Intel Xeon E5-2630 processor with 2.2GHz.

B. Results

For a closer comparison, we calculate the gap between the average NSRs of final recommendations after $30mins$ by SAHiD and SASAHiD as $\triangle_{NSR} = \frac{1}{n}\sum_1^n NSR_{SASAHiD} - \frac{1}{n}\sum_1^n NSR_{SAHiD}$, where n refers to the number of independent trials. Table I summarises the $\triangle_{NSR}$ and the average NSRs of initial solutions that SAHiD and SASAHiD have used. The NSR for an initial solution indicates how far the initial value is to the optimal value. Additionally, the NSR of the best solution found among 25 trials of SASAHiD is shown on the last column. "0.00" means that the $\tilde{cost}^*$ is found by SASAHiD, otherwise, found by SAHiD. According to Table I, the final recommended solutions reduce the cost for around 10% or more on average compared to the initial solutions using either SAHD or HD, additionally, the solution quality converges after a small amount of time. Within a fixed optimisation budget ($30mins$), the average of gap between the final recommendations of SASAHiD and SAHiD over 10 instances is minor (0.1% and 0.21%). Among 25 trials each, SASAHiD finds better solutions 11 times out of 20 problem instances, and achieves competitive results on the other instances. The proposed SASAHiD significantly outperforms the original SAHiD in terms of solution quality and number of hyperparameters, further reduces the size of configuration space. We expect the self-adaptive hierarchical decomposition method to help solving other permutation-based optimisation problems more efficiently.

C. Comparison with Various SAHiD

For comparison reason, we further compare SASAHiD with $\alpha = 0.1$ to SAHiD with $(\alpha, \beta) \in \{0.1, 0.3, 0.5, 0.7, 0.9\}^2$ on Hefei-1 (121 tasks), Hefei-10 (1212 tasks), Beijing-1 (358 tasks), and Beijing-10 (3584 tasks). These 4 instances and candidate values for α are selected for facilitating the comparison, as they are also presented in [9]. Results show that SASAHiD significantly outperforms SAHiD on large-scale problems. Due to the limited length, we only present the results of Beijing-1, and Beijing-10 with $\alpha = 0.1$, 0.5 and 0.9 in Figure 2. SAHiD with some hyperparameter settings finds better solution than SASAHiD with $\alpha = 0.1$, however, its hyperparameter search space is one dimension higher than SASAHiD. SASAHiD with $\alpha = 0.1$ performs significantly better than SAHiD with various configurations on large-scale CARPs.

IV. Incremental Hyperparameter Tuning across Multiple Problem Instances

It is unclear if the performance of SASAHiD can be further improved by optimising hyperparameters. As SASAHiD is designed for large-scale problems (e.g., a CARP instance with thousands of tasks or more), it may take hours to achieve a satisfactory solution. Thus, an evaluation of SASAHiD with certain hyperparameter values is time-consuming and it's not trivial to find good setting manually. It is nature to use some hyperparameter tuning methods and surrogate models to predict if the performance is favourable. In particular, we are interested in answering the following questions:

1) Are all the hyperparameters necessary or do they affect significantly the performance of SASAHiD?
2) Can the performance of SASAHiD be further enhanced by tuning the hyperparameters automatically?
3) Will the SASAHiD using the "optimal" hyperparameter setting tuned on a problem instance I_1 (or a set of problem instances) achieve a satisfactory solution for an unseen instance I_2 (or an unseen set of instances)?

An extension to the question 3) is "Will SASAHiD with the hyperparameter setting p_1, tuned on instance I_1, be further improved by tuning on instance I_2? And will SASAHiD with the resulted setting p_2 still perform well on instance I_1?"

To answer these questions, we propose an incremental hyperparameter tuning across multiple problem instances approach. The main procedure is as follows. A model-based algorithm configurator $\mathcal{T}$ is used to tune the hyperparameter(s) of an algorithm or solver $\mathcal{S}$. At the beginning, the model is initialised without any priori knowledge or memory. The automatic hyperparameter tuning process starts by one call or successive calls to the solver $\mathcal{S}$ on an instance I_1. Periodically, $\mathcal{T}$ decides to generate and evaluate hyperparameters by one call or successive calls to the solver $\mathcal{S}$ on another instance I_2, while the previously learnt model is used as the initial model at this stage. Thus, all the statistics obtained when training

TABLE I

COMPARISON OF SASAHID AND SAHID. A NEGATIVE $\triangle_{NSR}$ REFERS TO A LOWER NSR OF SASAHID, THUS BETTER PERFORMANCE THAN SAHID. THE 2^{nd} AND 3^{rd} COLUMNS ILLUSTRATE THE AVERAGE NSRS OF THE RANDOMLY INITIALISED SOLUTIONS. THE LAST COLUMN REFERS TO THE AVERAGE NSR OF FINAL RECOMMENDATIONS OF SASAHID. "0.00" MEANS THAT AT LEAST ONE OUT OF ITS 25 TRIALS FINDS A SOLUTION BETTER THAN ALL SOLUTIONS FOUND BY SAHID. ALL THE FIGURES ARE SHOWN IN PERCENT. TO FACILITATE THE COMPARISON, WE FOLLOW THE SAME METHODOLOGY AS IN [9]. BOLD (UNDERLINED) RESULTS INDICATE THAT SASAHID IS BETTER (WORSE, RESPECTIVELY) THAN SAHID BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. "*w-d-l*" INDICATES THE WIN-DRAW-LOSE OF SASAHID VERSUS SAHID.

Instance	Initial regret (%) SAHiD	Initial regret (%) SASAHiD	$\triangle_{NSR}$ (%)	Best regret (%)
Hefei-1	14.54	13.14	**-1.06**	0
Hefei-2	14.51	14.38	0.44	0.41
Hefei-3	13.70	15.13	-0.17	0.07
Hefei-4	14.07	14.61	0.03	0
Hefei-5	14.40	13.85	-0.02	0.04
Hefei-6	13.93	13.40	0.02	0
Hefei-7	12.69	13.60	-0.10	0.34
Hefei-8	13.10	13.98	-0.18	0
Hefei-9	12.45	12.07	-0.12	0
Hefei-10	11.54	11.46	0.01	0
Number of "w-d-l"			*1-8-1*	
Average	13.49	13.56	0.01	0.09

Instance	Initial regret (%) SAHiD	Initial regret (%) SASAHiD	$\triangle_{NSR}$ (%)	Best regret (%)
Beijing-1	11.24	10.11	-0.48	0
Beijing-2	10.72	11.63	**-1.07**	0
Beijing-3	11.59	13.07	-0.21	0
Beijing-4	11.07	10.25	**-0.83**	0
Beijing-5	11.48	12.53	-0.41	0
Beijing-6	10.38	10.52	-0.20	0.51
Beijing-7	10.50	11.01	0.11	0.18
Beijing-8	9.42	9.98	0.02	0.08
Beijing-9	9.94	10.95	0.08	0.12
Beijing-10	9.63	10.41	0.21	0.22
Number of "w-d-l"			*2-8-0*	
Average	10.60	11.05	0.21	0.11

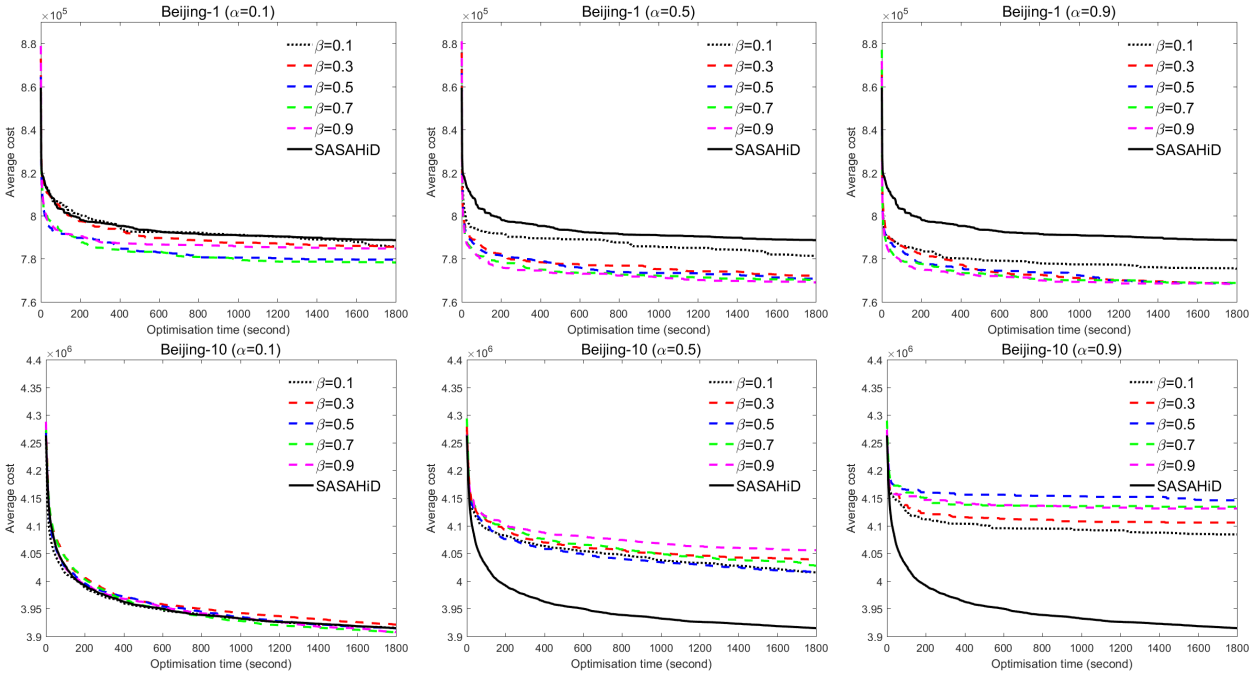

Fig. 2. Comparison of SASAHiD with $\alpha = 0.1$ to SAHiD with various values of α and β on instances Beijing-1 (top) and Beijing-10 (bottom). To facilitate the comparison, an identical range for *y-axis* is used in all the 3 figures of each problem instance (i.e., each row).

on instance I_1 are kept and re-used when starting to train on instance I_2. The model is updated in an incremental way.

We let SMAC train the model on Hefei set and validate the recommended configurations on Beijing set. These two sets have different structures and the Beijing instances are 3 times bigger than the Hefei ones. The split is not performed within single data sets but between different data sets so as to study the transfer ability.

A. Model Training using SMAC

We use the Python implementation of SMAC (https://github.com/automl/SMAC3) to tune the two parameters of SASAHiD, α and θ. As the precision is less important in both cases, we take values of α (and θ) uniformly distributed between 0.01 and 0.99 (0.05 and 0.2, respectively) with a gap of 0.01. Thus, a discrete search space of size 1881 is formed, As discussed previously, the parameter settings can be compared after running a small amount of time. We evaluate the configuration by the lowest cost obtained by SASAHiD after optimising during $30s$. This is also the amount of time that [9] have used for comparison in terms of runtime. Additionally, the performance ranking of SAHiD instances with different parameter settings is usually stable after 30 seconds, there is no use comparing the very last recommendations. We can

compare the parameter settings at an earlier stage for saving budget and stopping unlucky runs earlier. This idea has been previously used by [23], [24] when comparing different solvers and same solvers using different parameter settings.

B. Tuning on Hefei Benchmark

We let SMAC optimise the hyperparameters using 100 SASAHiD runs on the problem instance Hefei-1 and save the obtained model. Then SMAC continues searching in the same hyperparameter space and updates the restored model with outputs of 100 SASAHiD calls on Hefei-2 using same budget, and so on till Hefei-10, thus 10 batches of experiments are run and a total of $1,000$ SASAHiD calls are used as optimisation budget. The total evaluation number is much lower than the size of search space. As the costs of solutions for different problem instances locate in different range, we need to normalise the output of SASAHiD. However, neither the bounds nor the optimal cost is known. Therefore, before tuning using SMAC, we run firstly SASAHiD with default parameters on each instance for $30s$, the outputs of which are then used as the denominator for normalisation. Then, 20 SMAC optimisation trails are run using different initial hyperparameters α_0 and θ_0 as shown in Table II. The recommendations are reported in Table II as well. During one run, the intermediate recommended settings after each batch varies because of the different instances that the model is training on. Among 20 trials, the recommended settings differ due to the different initial points and the stochasticity in SASAHiD and SMAC. Surprisely, after having trained on 10 instances in an incremental way, the setting $(\alpha, \theta) = (0.39, 0.20)$ is recommended 10 times among 20 runs. After rounding, the following three parameter settings are summarised, $(0.40, 0.20)$, $(0.40, 0.10)$ and $(0.75, 0.20)$, which have been recommended 10, 4 and 3 times, respectively, out of 20 hyperparameter optimisation runs.

C. Validation on Beijing Benchmark

The three recommended parameter settings are validated on the 10 instances of Beijing city, given a real time budget of $30mins$. As the hyperparameters have been optimised on 10 instances of Hefei sequentially, we also test the settings on the Hefei instances with a budget of $30mins$ to verify if the recommended settings still guaranty a stable performance on Hefei-1, that has been learnt the most previously. The lowest cost, averaged cost and the standard deviation over 25 independent trials are illustrated in Table III. To reduce the randomness, all the settings have been used the identical set of 25 distinct random seeds. The results obtained by SASAHiD with $\alpha = 0.1$ and $\theta = 0.1$ are also presented for reference, denoted as "Baseline" in Table III. Results show that SASAHiD with setting $(0.4, 0.2)$ performs better or as good as SASAHiD with $(0.1, 0.1)$ on most of the Beijing instances. On both sets, the recommended settings enhance SASAHiD when solving the first 5 to 7 instances (of smaller or medium size). This is probably due to the order of instances that have been used for training. As the evaluations of SASAHiD have been performed on instances with indices 1 to 10, the model is initially built using the data of evaluations on smaller instances and is biased to better solving such instances. The order of instances may have a significant impact. Nevertheless, Beijing-5 has much more tasks than the CARP instances considered in other work. For example, EGL-G benchmark set [14] consists of up to 375 edges and 375 task, while Beijing-5 has 3584 edges and 1792 tasks. It is also notable that the recommended α value for SASAHiD (0.4) is much higher than the recommended one for SAHiD (0.1). Intuitionally, this could be a way to make up the loss of stochasticity due to the use of a deterministic upper bound of cluster number as in (3) instead of using a random one, determined by (1). Note that higher value of α leads to a higher possibility of partitioning a route in the Ulusoy's splitting procedure.

V. Conclusions and Future Work

The scalability is a difficult problem for searching methods in combinatorial optimisation. Some decomposition methods, such as cooperative co-evolution and hierarchical decomposition, have been proposed in the literature for tackling large-scale problems. In this work, we propose a scalable approach based on self-adaptive hierarchical decomposition scheme with only two hyperparameters, called SASAHiD. We compare SASAHiD to a non-self-adaptive hierarchical decomposition method, SAHiD [9], on two real-world benchmark sets of large-scale CARP instances. The proposed SASAHiD, with a hyperparameter space one dimension lower than the one of SAHiD, achieves better or similar performance to SAHiD using various configurations. The SAHiD has been proved to outperform 3 state-of-the-art algorithms, Variable Neighbourhood Search [13], Tabu Search Algorithm 1 [14] and RDG-MAENS [7] on solving the same benchmark problems [9]. Thus, our SASAHiD naturally outperforms the 3 state-of-the-art algorithms on the benchmark problems.

Moreover, we propose an incremental hyperparameter tuning across multiple problem instances approach and use SMAC [10] to tune the hyperparameters automatically. The hyperparameters of SASAHiD are optimised in an incremental manner on the instance set of one city, with which the SASAHiD obtains better or competitive results when solving instances of another city that have never been seen before. To the best of our knowledge, we are the first to propose a self-adaptive hierarchical decomposition scheme and use an incremental hyperparameter tuning approach to perform automatic hyperparameter tuning for solving large-scale CARPs.

We see several directions for future research. Evaluations of (SA)SAHiD on large-scale problems are computationally expensive. It would be interesting to apply some surrogate-assisted approaches for very expensive problems [25]. A recently proposed approach called Per Instance Algorithm Configuration (PIAC) [26] is worth investigating to make use of the instance features. When scaling to computationally expensive instances, [27] selected configurations based on their performance on a set of intermediate instances that are harder than the training set, but easier than the testing set. For future work, we would like to apply these methods to (SA)SAHiD

TABLE II

RECOMMENDED CONFIGURATION BY SMAC AFTER LEARNING SEQUENTIALLY ON HEFEI INSTANCES. FOR EXAMPLE, THE 1^{st} ROW OF (HEADED "HEFEI-1") REFERS TO THE CONFIGURATIONS RECOMMENDED AFTER LEARNING ON HEFEI-1, THUS AFTER 100 EVALUATIONS; WHILE THE 2^{nd} ROW OF (HEADED "HEFEI-2") REFERS TO THE CONFIGURATIONS RECOMMENDED AFTER LEARNING ON HEFEI-1 AND HEFEI-2, THUS AFTER 200 EVALUATIONS.

Instance	$\theta_0 = 0.05$										$\theta_0 = 0.10$									
	$\alpha_0 = 0.01$		$\alpha_0 = 0.25$		$\alpha_0 = 0.50$		$\alpha_0 = 0.75$		$\alpha_0 = 0.99$		$\alpha_0 = 0.01$		$\alpha_0 = 0.25$		$\alpha_0 = 0.50$		$\alpha_0 = 0.75$		$\alpha_0 = 0.99$	
	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ
Hefei-1	0.90	0.08	0.97	0.09	0.89	0.09	0.94	0.09	0.85	0.20	0.86	0.19	0.50	0.18	0.85	0.17	0.96	0.18	0.91	0.12
Hefei-2	0.90	0.08	0.97	0.09	0.91	0.10	0.94	0.09	0.85	0.20	0.86	0.19	0.50	0.18	0.85	0.17	0.75	0.19	0.95	0.11
Hefei-3	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20
Hefei-4	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20
Hefei-5	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20
Hefei-6	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20
Hefei-7	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20
Hefei-8	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20
Hefei-9	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20
Hefei-10	0.37	0.09	0.76	0.19	0.39	0.20	0.39	0.20	0.75	0.18	0.39	0.20	0.39	0.09	0.39	0.20	0.49	0.05	0.39	0.20

Instance	$\theta_0 = 0.15$										$\theta_0 = 0.20$									
	$\alpha_0 = 0.01$		$\alpha_0 = 0.25$		$\alpha_0 = 0.50$		$\alpha_0 = 0.75$		$\alpha_0 = 0.99$		$\alpha_0 = 0.01$		$\alpha_0 = 0.25$		$\alpha_0 = 0.50$		$\alpha_0 = 0.75$		$\alpha_0 = 0.99$	
	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ	α	θ
Hefei-1	0.91	0.12	0.91	0.12	0.75	0.10	0.74	0.20	0.86	0.06	0.94	0.09	0.88	0.08	0.96	0.07	0.95	0.05	0.76	0.15
Hefei-2	0.91	0.12	0.91	0.12	0.92	0.19	0.83	0.15	0.86	0.06	0.99	0.15	0.88	0.08	0.96	0.07	0.97	0.07	0.76	0.15
Hefei-3	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20
Hefei-4	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20
Hefei-5	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20
Hefei-6	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20
Hefei-7	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20
Hefei-8	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20
Hefei-9	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20
Hefei-10	0.39	0.20	0.75	0.19	0.26	0.20	0.36	0.09	0.40	0.10	0.37	0.20	0.39	0.20	0.88	0.05	0.39	0.20	0.39	0.20

on solving large-scale CARPs. Moreover, we have trained the model on problem instances of smallest to largest size, which is a simple but naive instance selection policy. Results in Table III imply that the order of instances used for training may play an essential role. As a future work, we are interested in exploring selection policies in our incremental hyperparameter tuning approach for deciding the next instance to train from a set of instances.

ACKNOWLEDGMENT

The authors would like to thank Prof. Ke Tang for providing the source code of SAHiD and data sets.

REFERENCES

[1] B. L. Golden and R. T. Wong, "Capacitated arc routing problems," *Networks*, vol. 11, no. 3, pp. 305–315, 1981.

[2] S. Wøhlk, *A Decade of Capacitated Arc Routing*. Boston, MA: Springer US, 2008, pp. 29–48. [Online]. Available: https://doi.org/10.1007/978-0-387-77778-8_2

[3] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[4] H. Handa, L. Chapman, and X. Yao, *Robust Salting Route Optimization Using Evolutionary Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 497–517. [Online]. Available: https://doi.org/10.1007/978-3-540-49774-5_22

[5] M. Fadzli, N. Najwa, and M. Luis, "Capacitated arc routing problem and its extensions in waste collection," in *International Conference on Mathematics, Engineering and Industrial Applications 2014 (ICoMEIA 2014)*, M. Ramli, A. Junoh, N. Roslan, M. Masnan, and M. Kharuddin, Eds., vol. 1660. American Institute of Physics, 2015.

[6] Y. Mei, X. Li, and X. Yao, "Decomposing large-scale capacitated arc routing problems using a random route grouping method," in *Evolutionary Computation, 2013 IEEE Congress on*. IEEE, 2013, pp. 1013–1020.

[7] ——, "Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 435–449, 2014.

[8] ——, "Variable neighborhood decomposition for large scale capacitated arc routing problem," in *Evolutionary Computation, 2014 IEEE Congress on*. IEEE, 2014, pp. 1313–1320.

[9] K. Tang, J. Wang, X. Li, and X. Yao, "A scalable approach to capacitated arc routing problems based on hierarchical decomposition," *IEEE transactions on cybernetics*, vol. 47, no. 11, pp. 3928–3940, 2017.

[10] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 507–523.

[11] E. Alba and B. Dorronsoro, "Computing nine new best-so-far solutions for capacitated vrp with a cellular genetic algorithm," *Information Processing Letters*, vol. 98, no. 6, pp. 225–230, 2006.

[12] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985–2999, 2008.

[13] M. Polacek, K. F. Doerner, R. F. Hartl, and V. Maniezzo, "A variable neighborhood search for the capacitated arc routing problem with intermediate facilities," *Journal of Heuristics*, vol. 14, no. 5, pp. 405–423, 2008.

[14] J. Brandão and R. Eglese, "A deterministic tabu search algorithm for the capacitated arc routing problem," *Computers & Operations Research*, vol. 35, no. 4, pp. 1112–1126, 2008.

[15] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[16] K. Tang, Y. Mei, and X. Yao, "Memetic algorithm with extended neighborhood search for capacitated arc routing problems," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1151–1166, 2009.

[17] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2009, pp. 142–157.

TABLE III

VALIDATION RESULTS GIVEN $30mins$ AS OPTIMISATION BUDGET. TO FACILITATE THE COMPARISON, WE FOLLOW THE SAME METHODOLOGY AS IN [9]. BOLD (UNDERLINED) RESULTS INDICATE THAT THE TUNED SASAHID WITH CORRESPONDING SETTING IS BETTER (WORSE) THAN THE BASELINE (I.E., SASAHID WITH $(0.1, 0.1)$) BASED ON WILCOXON RANK-SUM TEST WITH THE LEVEL OF SIGNIFICANCE 0.05. "W-D-L" INDICATED THE WIN-DRAW-LOSE OF THE SASAHID WITH TUNED PARAMETER SETTING VERSUS THE BASELINE. THE LOWEST AVERAGE COST IS MARKED WITH "*" .

Re-test on 10 Hefei instances using $30mins$ each trial

Id	$\|T\|$	Baseline			$(0.40, 0.20)$			$(0.40, 0.10)$			$(0.75, 0.20)$		
		Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
1	121	246877	248719	1228	246401	**247222**	425	246753	**247349**	344	246611	**247060**	273
2	242	437991	441503	2069	435977	**438287**	1379	436966	**438572**	874	435452	**437876**	965
3	364	585247	590264	2281	583967	**587341**	1676	581889	**587100**	1916	585299	**588396**	1559
4	485	752406	759806	3068	749454	**756703**	2899	749454	**757083**	2888	755645	763964	3636
5	606	963432	975360	4688	968203	976117	4113	968203	975982	3858	981240	991007	5659
6	727	1098660	1109319	5584	1100100	1112229	7506	1100280	1112280	7470	1132290	1145915	7171
7	848	1306340	1314498	5480	1314000	1330085	8186	1314000	1330141	8168	1365470	1382972	7086
8	970	1478710	1486581	6155	1496430	1514146	9955	1496430	1514122	9947	1553130	1567486	8211
9	1091	1647760	1663170	7503	1685830	1712361	12503	1685830	1712361	12503	1739000	1763253	10422
10	1212	1803250	1814804	6118	1864300	1880489	8132	1864300	1880489	8132	1917310	1928001	5883
						4-2-4			4-2-4			3-0-7	

Validation on 10 Beijing instances using $30mins$ each trial

Id	$\|T\|$	Baseline			$(0.40, 0.20)$			$(0.40, 0.10)$			$(0.75, 0.20)$		
		Best	Average	Std	Best	Average	Std	Best	Average	Std	Best	Average	Std
1	358	774940	784300	5501	767420	**772886**	3702	764538	**769370**	2780	764853	**767502**	1996
2	717	1168070	1183056	7114	1147000	**1167685**	7121	1144050	**1165256**	6819	1154030	**1163978**	5495
3	1075	1586410	1612158	10717	1580850	**1593422**	7492	1571880	**1589013**	7698	1578930	**1589864**	7773
4	1433	1924780	1940109	11111	1908420	**1922964**	6937	1905140	**1916427**	7058	1901180	**1920891**	9223
5	1792	2277420	2308528	15153	2271500	**2300198**	17511	2270860	**2291656**	13386	2284010	2308899	17037
6	2151	2664810	2701230	16289	2663620	2702558	20291	2663420	2693417	17087	2689170	2730445	28749
7	2509	2975670	3003097	13241	2998660	3025902	20043	2983980	3014393	22490	3031640	3079966	24386
8	2868	3247490	3276867	18945	3260250	3301394	19223	3248900	3291093	19749	3306710	3374332	31713
9	3226	3579810	3622531	21690	3607040	3656403	24421	3594160	3642613	23516	3698200	3752642	25491
10	3584	3881420	3916252	17038	3934020	3985646	25706	3925070	3979544	24873	4015910	4081212	27968
						5-1-4			5-2-3			4-1-5	

[18] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle, "F-race and iterated f-race: An overview," in *Experimental methods for the analysis of optimization algorithms*. Springer, 2010, pp. 311–336.

[19] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, pp. 267–306, 2009.

[20] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[21] S. Becker, J. Gottlieb, and T. Stützle, "Applications of racing algorithms: An industrial perspective," in *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer, 2005, pp. 271–283.

[22] S. Abdullah, E. K. Burke, and B. McCollum, "A hybrid evolutionary approach to the university course timetabling problem," in *2007 IEEE Congress on Evolutionary Computation*. IEEE, 2007, pp. 1764–1768.

[23] M.-L. Cauwet, J. Liu, and O. Teytaud, "Algorithm portfolios for noisy optimization: Compare solvers early," in *International Conference on Learning and Intelligent Optimization*. Springer, 2014, pp. 1–15.

[24] M.-L. Cauwet, J. Liu, B. Rozière, and O. Teytaud, "Algorithm portfolios for noisy optimization," *Annals of Mathematics and Artificial Intelligence*, vol. 76, no. 1-2, pp. 143–172, 2016.

[25] H. Tong, C. Huang, J. Liu, and X. Yao, "Voronoi-based efficient surrogate-assisted evolutionary algorithm for very expensive problems," in *2019 IEEE Congress on Evolutionary Computation*. IEEE, June 2019, pp. 1996–2003.

[26] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, "Per instance algorithm configuration of cma-es with limited budget," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 681–688.

[27] J. Styles, H. H. Hoos, and M. Müller, "Automatically configuring algorithms for scaling performance," in *Learning and Intelligent Optimization*. Springer, 2012, pp. 205–219.