

1-2011

Instance-based parameter tuning via search trajectory similarity clustering

Linda LINDAWATI

Singapore Management University, lindawati.2008@smu.edu.sg

Hoong Chuin LAU


Singapore Management University, hclau@smu.edu.sg

David LO

Singapore Management University, davidlo@smu.edu.sg

DOI: https://doi.org/10.1007/978-3-642-25566-3_10

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Artificial Intelligence and Robotics Commons](#), [Operations Research, Systems Engineering and Industrial Engineering Commons](#), and the [Software Engineering Commons](#)

Citation

LINDAWATI, Linda; LAU, Hoong Chuin; and LO, David. Instance-based parameter tuning via search trajectory similarity clustering. (2011). *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011: Selected Papers*. 6683, 131-145. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/1336

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Instance-Based Parameter Tuning via Search Trajectory Similarity Clustering

Lindawati, Hoong Chuin Lau, and David Lo

School of Information Systems, Singapore Management University, Singapore
lindawati.2008@phdis.smu.edu.sg, {hclau,davidlo}@smu.edu.sg

Abstract. This paper is concerned with automated tuning of parameters in local-search based meta-heuristics. Several generic approaches have been introduced in the literature that returns a "one-size-fits-all" parameter configuration for all instances. This is unsatisfactory since different instances may require the algorithm to use very different parameter configurations in order to find good solutions. There have been approaches that perform instance-based automated tuning, but they are usually problem-specific. In this paper, we propose CluPaTra, a generic (problem-independent) approach to perform parameter tuning, based on CLUstering instances with similar PATterns according to their search TRAjectories. We propose representing a search trajectory as a directed sequence and apply a well-studied sequence alignment technique to cluster instances based on the similarity of their respective search trajectories. We verify our work on the Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP). Experimental results show that CluPaTra offers significant improvement compared to ParamILS (a one-size-fits-all approach). CluPaTra is statistically significantly better compared with clustering using simple problem-specific features; and in comparison with the tuning of QAP instances based on a well-known distance and flow metric classification, we show that they are statistically comparable.

Keywords: instance-based automated tuning parameter, search trajectory, sequence alignment, instance clustering.

1 Introduction

In the last decade there has been a dramatic rise in the design and application of meta-heuristics such as tabu search and simulated annealing to solve combinatorial optimization problems (COP) in many practical applications. The effectiveness of a meta-heuristic algorithm hinges on its parameter configurations. For example, a tabu search will perform differently with different tabu lengths. Previous studies revealed that only 10% of the time is spent on algorithm design and test; while the rest of the development time is spent on fine-tuning the parameter settings [1]. The latter process is either a laborious manual exercise by the algorithm designer, or an automated procedure. The key challenge in

automated tuning is the large parameter configuration space on even a handful of parameters.

Given an algorithm (which we call the target algorithm) to solve a given COP, it has been observed that different problem instances require different parameter configurations in order for the algorithm to find good solutions (e.g. [6,19,24]). An interesting research question is whether there are patterns or rules governing the choice of parameter configurations, and whether such patterns can be learnt.

Several approaches have been proposed to automate the tuning problem, such as the Racing Algorithm by Birratari et al. [3], Decision Tree Classification Approach by Srivastava and Mediratta [22], CALIBRA by Andenso-Daz and Laguna [1], ParamILS by Hutter et al. [12,13] and Randomized Convex Search (RCS) by Lau and Xiao [14]. These are generic approaches which can be used for various COP problems. One common shortcoming of such approaches is that they produce a **one-size-fits-all** configuration for all instances, which may not perform well on large and diverse instances. On the other hand, approaches by Patterson and Kautz [19], Hutter and Hamadi [11], Gagliolo and Schmidhuber [6] and Xu et al. [24] attempted to deal with **instance-based** automatic tuning. However, those approaches are less general in the sense that each of them can only solve a particular problem by making use of problem-specific features. For example, SATzilla constructs per-instance algorithm portfolios for SAT [24]. SATzilla07 uses 48 features, most of which are SAT-specific features. The caveat is that feature selection is itself a very complex problem in general which cannot be done automatically but rather must rely on the knowledge of a domain expert.

Rather than ambitiously attempting instance-based tuning which we believe to be a computationally prohibitive and unachievable task in the near future because of the large parameter configuration space and large number of instances, we turn towards a cluster-based treatment. Our goal extends a preliminary work on features-based tuning proposed in [14] where instances are clustered according to some problem-specific features, but unlike [14], we do not rely on problem-specific features; rather, we propose a generic approach where we make use of the search trajectory patterns as a feature. A search trajectory pattern is defined as the path that the target algorithm follows as it searches from an initial solution to its neighbor iteratively [10]. We then apply a standard clustering algorithm to segment the training set of instances into clusters based on their search trajectory patterns similarity.

Motivated by earlier works on the tight correlation between fitness landscape and search trajectories [7,8], and the tight correlation between the fitness landscape and algorithm performance [20], our bold conjecture in this paper is that trajectory patterns themselves are correlated with parameter configurations; in other words, we believe that if a parameter configuration works well for a particular instance, then it will also work well for instances with similar fitness landscapes (which can be inferred from the trajectory patterns). Consequently, we train our automated tuning algorithm by first performing clustering on problem instances based on their search trajectories similarity, and then apply existing one-size-fits-all algorithms (such as CALIBRA, ParamILS or RCS) to derive the

best parameter configurations for the respective clusters. Subsequently, given an arbitrary instance, we first map its search trajectory to the closest cluster. The tuned parameter configuration for that cluster is then returned as the parameter configuration for this instance. The result is a fine-grained tuning algorithm that does not produce a one-size-fits-all parameter configuration, but rather instance (or rather cluster)-based parameter configurations. Even though strictly speaking, our approach is cluster-specific rather than instance-specific, it is a big leap from one-size-fits-all schemes. Arguably, our approach, taken to the extreme, can potentially produce instance-based tuning; although we do not know how to scale it well at the moment. The major contributions in this paper are summarized as follows:

- We propose *CluPaTra*, a novel instance-based problem-independent automated parameter tuning approach based on clustering of patterns of instances by their search trajectories.
- A search trajectory can be derived readily from a local-search based algorithm without incurring extra computation (other than the task of storing these solutions as the local search discovers them). Hence our approach can be applied to tune any local search-based target algorithm to solve a given problem.
- We tap into the rich depository of machine learning and data mining, utilizing a clustering method based on two well-studied techniques, sequence alignment and hierarchical clustering. We apply sequence alignment to calculate a similarity score between a pair of instance search trajectories, and hierarchical clustering to form the clusters.

CluPaTra is verified with experiments on two classical COPs - Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP). For TSP, our target algorithm is the classical Iterated Local Search (ILS) algorithm (implemented by [8]), whereas for QAP we use a relatively new hybrid metaheuristic algorithm proposed in [18]. These choices are made on the dual intent to benchmark our approach against best published results (showing that it is capable of producing results compatible to the best-found results), as well as to demonstrate how our approach can yield significant improvement when applied to tune a newly designed algorithm.

2 Preliminaries

In this section, we formally define the Automated Parameter Configuration problem, followed by the concepts of the one-size-fits-all and instance-based configurators.

2.1 Automated Parameter Configuration Problem

Let \mathcal{A} be the target algorithm with n number of parameters to be tuned based on a given set of training instances I . Each parameter x_i can assume a value taken from a (either continuous or discrete) interval $[a_i, b_i]$ in parameter configuration

space Θ . Let the vector $\mathbf{x} = [x_1, x_2, \dots, x_n]$ represent a parameter configuration and \mathcal{H} be a performance metric function that maps \mathbf{x} to a numeric score computed over a set of instances (see details below). The automated parameter configuration problem is thus an optimization problem seeking to find $\mathbf{x} \in \Theta$ that minimizes $\mathcal{H}(\mathbf{x})$.

Notice that unlike standard optimization problems, the function \mathcal{H} is a meta-function on \mathbf{x} is typically highly non-linear and very expensive to compute. Furthermore, as the parameter space may be extremely large (even for discrete values, the size is equal to $(b_1 - a_1)(b_2 - a_2) \cdots (b_n - a_n)$), it is generally impractical to execute a tuning algorithm based on full factorial exploration of good parameter values. As in [13], to avoid confusion between an algorithm whose performance is being optimized and an algorithm used to tune it, we refer to the former as the *target algorithm* and the latter as the *configurator*.

2.2 One-Size-Fits-All Configurator

Since a one-size-fits-all configurator (such as ParamILS) only produces a single parameter configuration for a set of instances I , it calculates the function \mathcal{H} by using a specific statistic (such as mean or standard deviation) measured over the entire set (or distribution) of problem instances. We define the one-size-fits-all configurator as follows.

Definition 1 (One-Size-Fits-All Configurator). *Given a target algorithm \mathcal{A} , a set of training instances I , a set of testing instances I_t , a parameter configuration space Θ and a meta-function \mathcal{H} to measure algorithm \mathcal{A} performance, a one-size-fits-all configurator finds a parameter configuration $\mathbf{x} \in \Theta$ such that \mathcal{H} is minimized over the entire set (or distribution) of I . Subsequently, given a testing instance in I_t , that parameter configuration \mathbf{x} will be used to execute \mathcal{A} .*

2.3 Instance-Based Configurator

In this paper, we are concerned with clustering of problem instances. Hence, using the same notation as the one-size-fits-all configurator, we define the instance-based configurator as follows.

Definition 2 (Instance-Based Configurator). *Given a target algorithm \mathcal{A} , a set of training instances I , a set of testing instances I_t , a parameter configuration space Θ and a meta-function \mathcal{H} to measure algorithm \mathcal{A} performance, an instance-based configurator creates a set of clusters C from I and finds a parameter configuration \mathbf{x}_c for each cluster $c \in C$ that minimizes \mathcal{H} for the set of instances in the respective cluster. For a given testing instance in I_t , it will find the most similar cluster $c \in C$ and return the parameter configuration \mathbf{x}_c which will be used to execute \mathcal{A} .*

2.4 Performance Metric

We now define the performance metric function \mathcal{H} , for both the training and testing instances. For training, this value is measured over all training instances, while for testing, ditto test instances.

Definition 3 (Performance Metric). *Let i be a problem instance, and $\mathcal{A}_{\mathbf{x}}(i)$ be the objective value of the corresponding solution obtained by \mathcal{A} when executed under the configuration \mathbf{x} . Let $OPT(i)$ denote either (a) the known global optimal value of i , or (b) where the global optimal value is unknown, the best known value. $\mathcal{H}(x)$ is defined as the mean percentage deviation of $\mathcal{A}_{\mathbf{x}}(i)$ from $OPT(i)$, for all problem instance i in question (training/testing). Obviously, the lower the deviation value the better it is.*

3 Solution Approach

In this section, we present our solution approach *CluPaTra* by first defining the search trajectory similarity and describing *CluPaTra* major components, namely: search trajectory representation, similarity calculation, and the clustering method, followed by the overall steps for the training and testing phases.

3.1 Search Trajectory Similarity

A search trajectory is defined as a path of solutions that the target algorithm \mathcal{A} finds as it searches through the neighborhood search space. Two or more search trajectories are similar if some fragments (several number of consecutive moves) of the path have identical solution’s attributes. An example of solution’s attributes is the deviation of its objective value from global optimum (or best known) value (see section 3.2). The longer the fragments the more similar it is.

As an example, Fig. 1 shows a search trajectory obtained by 10 consecutive moves of the ILS algorithm for three TSP instances, namely: kroa100, bier127 and eil51 with two very different parameter configurations, namely: configuration I and configuration II. Observe that for the same configuration, kroa100 and bier127 have similar search trajectories, while ei151 has a very different trajectory. Observe also that even when different configurations result in different search trajectories for a given instance, the similarity between kroa100 and bier127’s trajectories are preserved. This similarity property is what we need that allows us to perform clustering of instances using an arbitrary parameter configuration.

Since there is a tight correlation between fitness landscape (or commonly known as search space) and search trajectories [7,8], and the tight correlation between the fitness landscape and algorithm performance [20], we assume that instances with similar search trajectories will need the same parameter configuration.

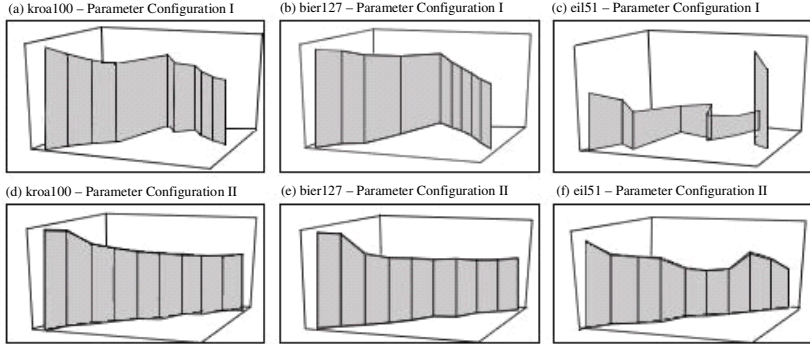


Fig. 1. Search Trajectories of 3 TSP instances kroa100, bier127 and eli51 using two very different parameter configuration (with z-axis as the objective value and x and y axis as the search space)

3.2 Search Trajectory Representation

We present the search trajectory as a directed sequence of symbols, each representing a solution along the trajectory. A symbol encodes a combination of two solution attributes, namely: the position type and its percentage deviation of quality from OPT (as defined in section 2.4).

The position type represents in a sense the local property of a solution with respect to its search neighborhood, and is defined based on the topology of the local neighborhood [10]. There are 7 position types, determined by evaluating the solution objective value with all its local direct neighbors' objective values - whether it is better, worse or the same. The 7 positions types are given in Table 1.

The deviation of solution quality measures in a sense the global property of the solution (since it is compared with the global value OPT). If the global optimum value is unknown, we use the best known value. Granted however that

Table 1. Position Types of Solution

Position Type Label	Symbol	<	=	>
SLMIN (strict local min)	S	+	-	-
LMIN (local min)	M	+	+	-
IPLat (interior plateau)	I	-	+	-
SLOPE	P	+	-	+
LEDGE	L	+	+	+
LMAX (local max)	X	-	+	+
SLMAX (strict local max)	A	-	-	+

'+' = present, '-' = absent; referring to the presence of neighbors with larger ('<'), equal ('=') and smaller ('>') objective values

the best known value is not the same as the global optimal value, it provides a reasonably good upper bound (for a minimization problem); and since our aim is to find similar patterns of the transition from one solution to the next, and not to measure the actual absolute performance of the algorithm, the best known value suffices in providing a good proxy to the global optimal value for our purpose of representing the trajectory.

These two attributes are combined into a symbol with the first two digits being the deviation of the solution quality and the last digit being the position type. Note that the attributes are generic in the sense that they can be easily retrieved/computed from any local-search-based algorithm albeit different problems. Being mindful that some target algorithms may have cycles and (random) restarts, we intentionally add two additional symbols: 'CYCLE' and 'JUMP'; 'CYCLE' is used when the target algorithm returns to a position that has been found previously, while 'JUMP' is used when the local search is restarted.

In order to obtain the search trajectory for a given problem instance, we naturally need to execute the target algorithm with a certain parameter configuration and record all the solutions visited. We refer to this configuration as the *initial sequence configuration*.

An example of the sequence representing the `eil51` search trajectory in Fig. 1 is `15L-11L-09L-07L-07P-06P-04S-05L-J-21L-19L`. Notice that after position 8, the target algorithm performs a random restart, hence we add 'JUMP' symbol after position 8.

3.3 Similarity Calculation

Having represented trajectories by linear sequences, it is natural to use pairwise sequence alignment to determine the similarity between a pair of trajectories. In pairwise sequence alignment [9], the symbols of one sequence will be matched with those of the other sequence while respecting the sequential order in the two sequences. It can also allow gaps to occur if symbols do not match. There are two kinds of alignment strategies: local and global. In local alignment, only portions of the sequences are aligned, whereas global alignment aligns over the entire length of the sequences. Because search trajectory sequences have varying lengths, we find local alignment best fits our need.

To measure the similarity score between two search trajectory sequences, a metric based on the best alignment is used. The matched symbol contributes a positive score (+1), while a gap contributes a negative score (-1). The sum of the scores is taken as the maximal similarity score of the two sequences. We may find situations as follows: (a) a search trajectory sequence is a subsequence of another one thus having a very high similarity score or (b) longer sequences get higher similarity score. To avoid these situations, the final similarity score will be divided by $\frac{1}{2} \times (|Sequence_1| + |Sequence_2|)$.

Our sequence alignment is implemented using standard dynamic programming [9], with a complexity of $O(n^2)$. As an example, the sequence alignment for the `kroa100` and `bier127` search trajectories from Fig. 1 is illustrated in Table 2.

Table 2. Example of Sequence Alignment from 2 TSP instances search trajectory, kroa100 and bier127

kroa100	19L	19P	18P	17P	16P	15P	14P	13P	11P	10P
bier127		19P	18P	17P		15P		13P	11P	10P 09P 08P
score		+1	+1	+1	-1	+1	-1	+1	+1	+1

To cluster instances (see the subsection below), we need to compute similarity scores for all possible pairs of training instances. Hence, the total time complexity for sequence alignment is $O(m^2 \times n^2)$, where n is the maximum sequence length of the sequences and m is the number of instances in the training set.

3.4 Clustering Method

Here, our goal is to group similar instances according to their search trajectory similarity. A typical clustering algorithm requires a distance measure between data points. For the distance measure we use $\frac{1}{\text{similarity score}}$. After such a measure is known, a standard clustering algorithm could be employed. For our purpose, we adopt the well-known hierarchical clustering approach AGNES (AGglomerative NESTing) to cluster the instances [9]. AGNES works by placing each instance initially in a cluster of its own. It then iteratively merges two closest clusters (i.e., a pair of clusters with the smallest distance) resulting in lesser number of clusters of larger sizes. The process is repeated until all nodes belong to the same cluster unless a termination condition applies. Examples of termination conditions are minimal number of cluster is reached or the maximal inter-cluster distance goes below a certain value. The complexity of AGNES is $O(n^2)$ with n being the number of instances.

Since the learning is unsupervised, we need to determine the number of clusters to be used. For this purpose, we apply the L method from [21] which makes use of an evaluation graph where the x -axis is the number of clusters and the y -axis is the value of the evaluation function at x clusters. The evaluation function can be any evaluation metric based on distance, similarity, error or quality. In this paper, we use the average distance among all instances in two different clusters. It determine the number of clusters by finding the point that has minimum root mean square error for both the left and right side. It is calculated using the following formula:

$$c^* = \min \left[\frac{RMSE(L)}{n_L} + \frac{RMSE(R)}{n_R} \right] \quad (1)$$

where:

Notation	Definition
$RMSE(L)$	root mean squared error of points in the left side of c
n_L	number of points in the left side of c
$RMSE(R)$	root mean squared error of points in the right side of c
n_R	number of points in the right side of c

This method only requires AGNES algorithm to be run once, since all the clusters created by AGNES can be recorded in one run. And since we want to produce a compact set of clusters, we limit the number of clusters to be less than 10. Thus, for the x -axis, we only use the number of clusters from 1 to 10.

3.5 Training and Testing Phases

The steps involved in the training and testing phases are shown in Fig. 2 (which are quite self-explanatory, and details are skipped in the interest of space).

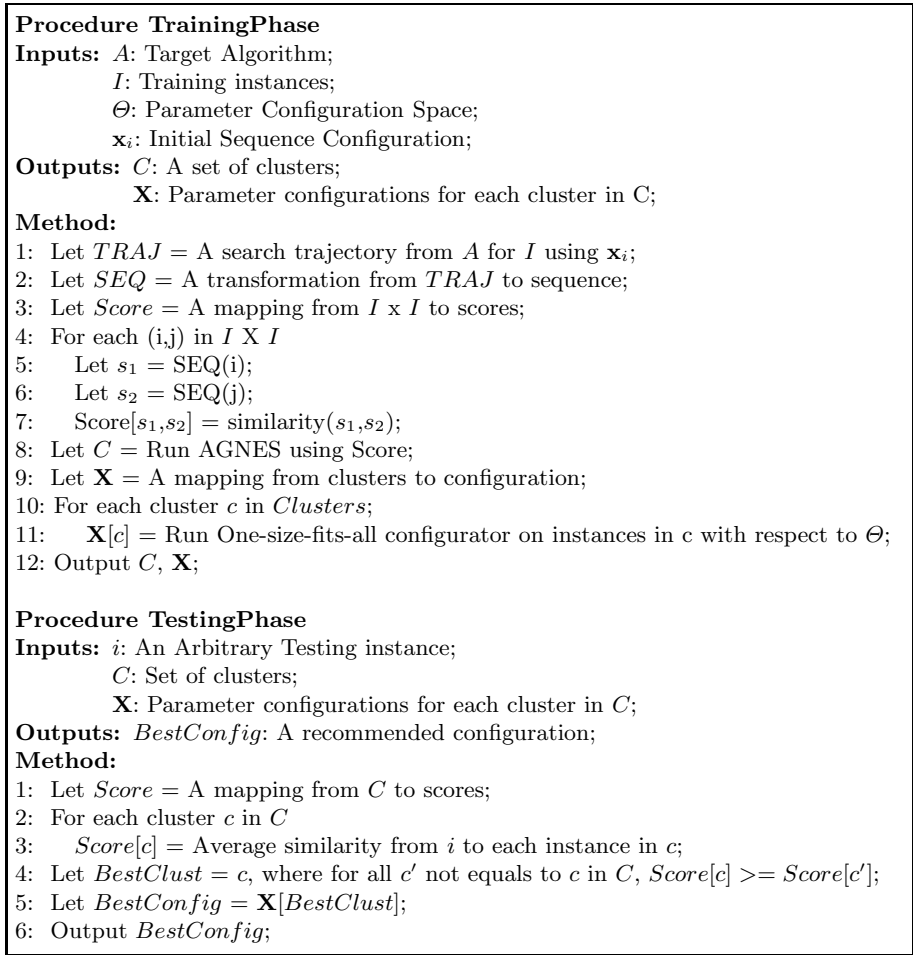


Fig. 2. Training and Testing Phase

4 Experimental Design

In this section, we provide information on the experiments presented in the following section. First, we present our experiment settings. Second, we present our validity and statistical significant measurement. And finally, we describe the low-level details of our experimental setup.

4.1 Experiment Settings

Here we briefly explain the target algorithm, one-size-fits-all configurator, benchmark instances and initial sequence configuration.

Target Algorithm. We used two different target algorithms respectively for solving two different problems. The first algorithm is a variant of a well-known Iterated Local Search (ILS) algorithm [15] for solving the classical TSP, as implemented in [8]. It has 5 discrete-value parameters to be tuned. The second algorithm is a new hybrid Simulated Annealing and Tabu Search (SA-TS) algorithm for solving QAP (presented in [18]). It has 4 parameters; some are discrete while the others are continuous.

One-Size-Fits-All Configurator. In order to derive meaningful experimental comparison, we deliberately chose to use ParamILS [13] as our configurator. ParamILS is itself an iterated local search algorithm used for tuning discrete parameters. Since ParamILS works only with discrete parameters, we first discretize the values of the parameters if the target algorithm has parameters that assume continuous values.

Benchmark Instances. For TSP, we applied our target algorithm to 70 benchmark instances extracted from TSPLib. Fifty six random instances were used as training instances and the remaining 14 instances as testing instances. The problem size (the number of cities) varies from 51 to 3038. For QAP, we used 50 benchmark instances from QAPLib, and randomly picked 40 instances for training and 10 for testing. The problem size (number of facilities) varied from 20 to 150.

Initial Sequence Configuration. The initial sequence configuration is a random configuration from the configuration space Θ .

4.2 Validity and Statistical Significant Measurement

To ensure unbiased evaluation, we used a 5-fold cross-validation [9]. The overall result is recorded to be the average performance over all iterations. We also performed a statistical test to compare the significance of our result. We performed a t-test [17]; we consider p-value below 0.05 to be statistically significant (confidence level 5%).

4.3 Experimental Setup

All experiments were performed on a 1.7GHz Pentium-4 machine running Windows XP. We measured runtime as the CPU time on this machine. As an input to the one-size-fits-all configurator, we fairly set cutoff times of 10 seconds per run for TSP target algorithm and 100 seconds for QAP target algorithm and allowed each configuration process to execute the target algorithm for a maximum of two CPU hours and to call the target algorithm for a maximum of 10 x n times, where n is the number of instances in the cluster.

5 Empirical Evaluation

In this section, we present our experiment results on the effectiveness of *CluPaTra*. First, we compare *CluPaTra* against one-size-fits-all configurator. Then, to analyze the effectiveness of our generic feature, we compare it with simple specific feature. In addition to that, we also compare *CluPaTra* against an existing classification of QAP instances based on distance and flow metrics [23]. Next we analyze the effect of different initial sequence configurations to our result. We also present the computational time of *CluPaTra*. Finally, a brief discussion regarding the experiment is presented. For the entire experiment, we measure the performance by using the performance metric described in Definition 3.

5.1 Performance Comparison

We evaluated the effectiveness of *CluPaTra* against the vanilla one-size-fits-all configurator (ParamILS). In Fig. 3a, we show the performance achieved by the two approaches for two target algorithms. This result is an average from each of the 5-fold results. The average improvement using *CluPaTra* is 7.78% for TSP training instances, 12.31% for TSP testing instances, 14% for QAP training instances and 21.78% for QAP testing instances. *CluPaTra* performed better and the difference was statistically significant.

5.2 Comparison on Feature Selection

To evaluate the effectiveness of the generic feature (i.e. search trajectory) used by *CluPaTra*, we compared *CluPaTra* with a simple problem-specific feature clustering for TSP and QAP, and a known instance classification for QAP.

First, we compared *CluPaTra* with simple specific feature clustering (SpecFeat). For the specific feature cluster, we used the number of cities (for TSP) [14] and the number of facilities (for QAP). Besides using different features, steps in training and testing phase for both approaches are the same. In Fig. 3b, we present the average performance achieved using 5-fold cross-validation by the two approaches for two target algorithm. *CluPaTra* always perform better and the differences are statistically significant.

Next, we compared *CluPaTra* against an existing well-studied classification of QAP instances based on the distance and flow metrics, due to [23]. We refer

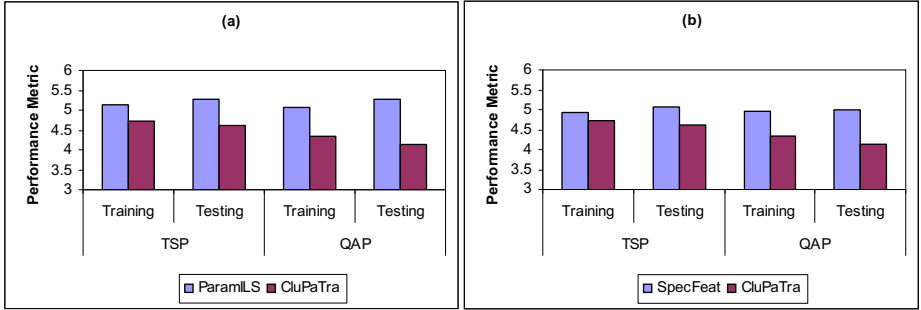


Fig. 3. Performance Comparison (a) *CluPaTra* and ParamILS, (b) *CluPaTra* and Specific Feature

to this as the Natural Cluster (Natural). (We conducted this comparison only for QAP since the classification of QAP benchmark instances is already well-studied and well characterized). Under this classification, QAP instances are divided into 5 groups: (1) random and uniform distances and flows, (2) random flows on grids, (3) real-life problems, (4) characteristics of real-life problems and (5) non-uniform, random problems. Due to the target algorithm limitation (it does not solve groups (4) and (5) problems), we can only provide results on groups (1), (2) and (3). The average performance of 5 folds is shown in Table 3. It shows that *CluPaTra* performs slightly better but the results are not statistically significant. Nonetheless, we can claim that the performance of *CluPaTra* is not inferior to tuning based on the natural classification.

Table 3. Comparison between *CluPaTra* and Natural Clustering

	<i>CluPaTra</i>	Natural	Difference(%)	p-value
Training	4.36	4.54	3.96	0.84108
Testing	4.13	4.14	0.24	0.61976

5.3 Sensitivity Analysis on Different Initial Sequence Configurations

Being mindful that our results may be biased depending on the initial sequence configuration used (to find the trajectories), we consider here two different parameter configurations - a configuration derived from running ParamILS versus a random configuration. Table 4 shows no statistically significant difference between these two initial sequence configurations.

5.4 Computational Results

The two most time-consuming processes in the training phase are those of calculating the similarity of trajectories and running the one-size-fits-all configurator

Table 4. Comparison between Different Initial Sequence Configurations

		Random	ParamILS	Different(%)	p-value
TSP	Training	4.74	4.66	1.69	0.94486
	Testing	4.63	4.72	(1.95)	0.40181
QAP	Training	4.36	3.97	8.94	0.1499
	Testing	4.13	4.5	(8.96)	0.54516

(ParamILS) for each cluster. For the 56 TSP instances with a maximal sequence length of 1560, the time taken for similarity computation is approximately 3 minutes; and for 40 QAP instances with a maximal sequence length of 520, the time taken is approximately 1 minute. For all clusters in one fold, the time needed to run ParamILS was approximately 44 minutes for TSP and 1 hour and 45 minutes for QAP. The total time needed to run the training phase for each fold is hence approximately 48 minutes on TSP and 1 hour and 47 minutes on QAP. While the time needed to run ParamILS alone for each fold is approximately 45 minutes on TSP and 1 hour and 40 minutes on QAP.

For the testing phase, we need to find the best cluster to fit the testing instances. For TSP instances, it took approximately 1.5 minutes in total; while for QAP instances, it took approximately 42 seconds.

5.5 Discussion

As shown from the results, compared to the vanilla one-size-fits-all configuration ParamILS, *CluPaTra* gives a significant improvement in performance (with respect to the performance metric we defined) with a small additional computation time. The additional computation time is needed to cluster the instances (approximately 6.66% for TSP and 7% for QAP from ParamILS run time). Based on this observation, we claim that dividing the instances into cluster using *CluPaTra* before running one-size-fits-all configurator provides a better parameter configuration for each instance and significantly improves the performance with minor additional computational time.

The effectiveness of using the search trajectory as the generic feature is evaluated by comparing it with problem-specific features. For the simple specific feature tried (number of cities for TSP and number of facilities for QAP), our approach is significantly better. Furthermore for QAP, we benchmarked against the natural clustering proposed in [23]. *CluPaTra* is statistically equivalent with the existing natural clustering approach. This shows that search trajectory can be used as a generic feature to cluster the instances without deep prior knowledge of the problem structure. We also evaluated the effect of different initial sequence configurations. Even though different initial sequence configurations may create different search trajectories, the effect of different initial sequence configurations is not significant.

6 Conclusion and Future Works

In this paper, we presented *CluPaTra*, a computationally efficient approach for generic instance-based configurator via clustering of patterns according to the instance search trajectories. We verified our approach on TSP and QAP and observed a significant improvement compared to a vanilla one-size-fits-all approach.

We see two limitations of our proposed approach. First, in terms of scope, our approach can only be applied to target algorithms which are local-search-based, since our approach uses search trajectory as feature. Second, there is an inherent computational bottleneck introduced by the method used for sequence alignment whose worst-case time complexity is $O(m^2 \times n^2)$ (where m is the number of instances in the training set and n is the maximum length of the sequences). As future work, one may investigate the effects of exploiting a less computationally intensive sequence alignment algorithm such as [5] or limit the length of the sequences.

There are also a number of challenges that remain to be explored. On the feature selection method, we proposed a single generic feature, search trajectory. It will be interesting to see if the accuracy can be improved if we combine several fitness landscape features, such as fitness distance correlation, run time distribution and density of local optima. On the metric and clustering method, we use only one metric (sequence alignment) and one clustering method (agglomerative clustering). It may also be interesting to learn how different possible metrics and how different clustering methods can influence the performance. And on a separate front, our approach is to learn to set parameter values based on training instances. This contrasts and complements the volume of works which seek to adaptively adjust the parameter configuration dynamically during search (such as the works of reactive search by Battiti (e.g. [2]) and many others). In adaptive scenario, the parameter values are modified in response to the search algorithm's behavior during its execution. It will be interesting to see if synergies can be exploited to create better instance-based configurators.

References

1. Adenso-Diaz, B., Laguna, M.: Fine-Tuning of Algorithms Using Fractional Experimental Design and Local Search. *Operations Research* 54(1), 99–114 (2006)
2. Battiti, R., Brunato, M., Campigotto, P.: Learning While Optimizing an Unknown Fitness Surface. In: Maniezzo, V., Battiti, R., Watson, J.-P. (eds.) *LION 2007 II*. LNCS, vol. 5313, pp. 25–40. Springer, Heidelberg (2008)
3. Birattari, M., Stuzle, T., Paquete, L., Varrentrapp, K.: A Racing Algorithm for Configuring Metaheuristics. In: *Genetic and Evolutionary Computation Conference*, pp. 11–18. Morgan Kaufmann, San Francisco (2002)
4. Coy, S.P., Golden, B.L., Runger, G.C., Wasil, E.A.: Using Experimental Design to Find Effective Parameter Setting for Heuristics. *Journal of Heuristic* 7(1), 77–97 (2001)
5. Edgar, R.C.: MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* 35(5), 1792–1797 (2004)

6. Gagliolo, M., Schmidhuber, J.: Dynamic Algorithm Portfolio. In: Amato, C., Bernstein, D., Zilberstein, S. (eds.) Ninth International Symposium on Artificial Intelligence and Mathematics (2006)
7. Halim, S., Yap, R., Lau, H.C.: Viz: A Visual Analysis Suite for Explaining Local Search Behavior. In: 19th Annual ACM Symposium on User Interface Software and Technology, pp. 57–66. ACM, New York (2006)
8. Halim, S., Yap, R., Lau, H.C.: An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 332–347. Springer, Heidelberg (2007)
9. Han, J., Kamber, M.: Data Mining: Concept and Techniques, 2nd edn. Morgan Kaufman, San Francisco (2006)
10. Hoos, H.H., Stutzle, T.: Stochastic Local Search: Foundation and Application, 1st edn. Morgan Kaufman, San Francisco (2004)
11. Hutter, F., Hamadi, Y.: Parameter Adjustment Based on Performance Prediction: Towards an Instance-Aware Problem Solver. Technical Report, Microsoft Research (2005)
12. Hutter, F., Hoos, H.H., Stutzle, T.: Automatic Algorithm Configuration based on Local Search. In: 22nd National Conference on Artificial Intelligence, pp. 1152–1157. AAAI Press, Menlo Park (2007)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stutzle, T.: ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36, 267–306 (2009)
14. Lau, H.C., Xiao, F.: Enhancing the Speed and Accuracy of Automated Parameter Tuning in Heuristic Design. In: 8th Metaheuristics International Conference (2009)
15. Lourenco, H.R., Martin, O.C., Stutzle, T.: Iterated Local Search. In: Glover, F., Kochenberger, G.A. (eds.) *Handbook of Metaheuristics*. International Series in Operations Research & Management Science, vol. 57, pp. 320–353. Springer, Heidelberg (2003)
16. Merz, P., Freisleben, B.: Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem. *IEEE Transactions on Evolutionary Computation* 4, 337–351 (2000)
17. Montgomery, D.C., Runger, G.C.: *Applied Statistics and Probability for Engineers*, 2nd edn. John Wiley & Son, Chichester (1999)
18. Ng, K.M., Gunawan, A., Poh, K.L.: A hybrid algorithm for the quadratic assignment problem. In: International Conf. on Scientific Computing (2008)
19. Patterson, D.J., Lautz, H.: Auto-WalkSAT: A Self-Tuning Implementation of WalkSAT. *Electronic Notes in Discrete Mathematics* 9, 360–368 (2001)
20. Reeves, C.R.: Landscapes, operators and heuristic search. *Annals of Operations Research* 86(1), 473–490 (1999)
21. Salvador, S., Chan, P.: Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms. In: 16th IEEE International Conference on Tools with Artificial Intelligence, pp. 576–584 (2004)
22. Srivastava, B., Mediratta, A.: Domain-dependent parameter selection of search-based algorithms compatible with user performance criteria. In: 20th National Conference on Artificial Intelligence, pp. 1386–1391. AAAI Press, Pennsylvania (2005)
23. Taillard, E.D.: Comparison of Iterative Searches for The Quadratic Assignment Problem. *Location Science* 3(2), 87–105 (1995)
24. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based Algorithm Selection for SAT. *Journal of Artificial Intelligence Research* 32, 565–606 (2008)