Journal of the Operational Research Society (2013) 64, 1742–1752

www.palgrave-journals.com/jors/

Clustering of search trajectory and its application to parameter tuning

Lindawati*, HC Lau and D Lo

Singapore Management University, Singapore, Singapore

This paper is concerned with automated classification of Combinatorial Optimization Problem instances for instance-specific parameter tuning purpose. We propose the CluPaTra Framework, a generic approach to CLUster instances based on similar PAtterns according to search TRAjectories and apply it on parameter tuning. The key idea is to use the search trajectory as a generic feature for clustering problem instances. The advantage of using search trajectory is that it can be obtained from any local-search based algorithm with small additional computation time. We explore and compare two different search trajectory representations, two sequence alignment techniques (to calculate similarities) as well as two well-known clustering methods. We report experiment results on two classical problems: Travelling Salesman Problem and Quadratic Assignment Problem and industrial case study.

Journal of the Operational Research Society (2013) **64,** 1742–1752. doi:10.1057/jors.2012.167
Published online 13 February 2013

Keywords: generic feature; search trajectory; instance-based automated parameter tuning; sequence alignment; local search algorithm

Introduction

Meta-heuristic algorithms play an important role in solving combinatorial optimization problems (COP) in many practical applications. Even though a meta-heuristic algorithm does not guarantee global optimality, it generally provides good solutions in reasonable time. Previous studies reveal that the performance of a meta-heuristic algorithm is dependent on the instance specific characteristics/features that determine its intrinsic difficulty. Consequently, there has been increasing interest in finding the instances features that have impact on difficulty in terms of performance (Smith-Miles and Lopes, 2012).

Various problem-specific features have been proposed for a wide range of COP in the literature. The most straightforward features are those that are extracted from the problem or instance definition itself, such as the number of variables and constraints, which can be derived to numerous candidate features using computational feature extraction processes (Smith-Miles and Lopes, 2012). Other non-straightforward features may require large-scale experimental studies and highly dependent on the knowledge of a domain expert in a particular problem. Not only does it take tremendous human effort, the features, most of the time, cannot be reused on another problem.

*Correspondence: Lindawati, School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902, Singapore.

On a separate front, there have been approaches that attempted to find problem-independent features using correlation of the objective function and the search space (fitness landscape analysis) (Reeves, 1999; Hoos and Stützle, 2004), such as: fitness distance correlation (FCD) and ruggedness coefficient. Unfortunately, these features can only be measured after an extensive analysis of the landscape which proves to be time consuming and to some extends are impossible for certain instances.

Our work is aimed at finding problem-independent feature within reasonable computation time. In this paper, we propose the CluPaTra framework (CLUstering instances with similar PAtterns according to search TRAjectories) where we introduce the notion of an instance's search trajectory as the problem-independent feature and exploit data-mining techniques to cluster problem instances according to their search trajectories. The advantage of our approach lies in the fact that the search trajectory may be computed from a local search-based algorithm. Hence our approach is problem-independent and may conceptually be applied to any local search-based algorithm.

We implement our proposed framework on instance-specific parameter tuning scheme where we use the framework to cluster training instances and apply an existing one-size-fits-all algorithm (such as CALIBRA or ParamILS) to derive the best parameter configurations for the respective clusters. This cluster-based treatment has been proven effective in solving the parameter tuning problem

(Malitsky and Sellmann, 2009; Kadioglu *et al*, 2010). Our approach is similar to ISAC (Kadioglu *et al*, 2010), but instead of using problem-specific features, we propose a problem-independent feature. It builds on two earlier works: (1) the tight correlation between fitness landscape and search trajectories (Halim *et al*, 2007), and (2) the tight correlation between the fitness landscape and algorithm performance (Reeves, 1999).

This paper extends the vanilla CluPaTra framework recently proposed by the same authors in Lindawati *et al* (2011) by considering different variants for its three major components: search trajectory representation, similarity calculation, and clustering method. We explore these different techniques in seeking to improve the accuracy of clustering. We examine the effects of these different techniques experimentally. Hence, the major contributions (and thus the flow) of this paper are summarized as follows:

- We propose a new problem-independent feature extracted from the instance's search trajectory.
- We present CluPaTra, a novel framework for clustering problem instances using the problem-independent feature, and extend the earlier version by introducing and comparing new different variants for CluPaTra framework components.
- We implement CluPaTra on instance-specific parameter tuning scheme to find good set of parameter for a particular algorithm.

Problem statement and definition

For instance-specific parameter tuning, we refer the algorithm whose performance is being tuned/configured as the *target algorithm* while the one used to tune/configure as the *configurator*. We measure the target algorithm performance based on the quality of their solutions. We define target algorithm performance $\mathcal{H}$ as follows:

Definition 1 (Performance Metric [$\mathcal{H}$]) Let i be a problem instance, and $\mathcal{A}_{\boldsymbol{x}}(i)$ be the objective value of the corresponding solution for instance i obtained by a target algorithm $\mathcal{A}$ when executed under configuration $\boldsymbol{x}$. Let $OPT(i)$ denote the best known values for instance i. $\mathcal{H}_x(i)$ is formulated as: $\mathcal{H}_x(i) = \frac{|OPT(i) - A_{\boldsymbol{x}}(i)|}{OPT(i)}$

For benchmark instances with known global optimum value, we use the known global optimum value as its $OPT(i)$, while for new instances, we use the target algorithm's best previously known solution. The function $\mathcal{H}$ is highly non-linear and very expensive to compute as the parameter and instance space may be extremely large. Using performance metric $\mathcal{H}$, we define the instance-specific parameter tuning problem as follows.

Definition 2 (Instance-Specific Parameter Tuning [ISPT]) Given a set of instances I, a parameter configuration space Θ for a target algorithm $\mathcal{A}$ and a performance metric $\mathcal{H}$, the ISPT problem is to find a parameter configuration $\boldsymbol{x} \in \Theta$ for each $i \in I$ such that $\mathcal{H}_x(i)$ is minimized over Θ.

Procedure TrainingPhase

Inputs: A : Target algorithm;
I : Training instances;
$\mathbf{x}_{init}$: Initial configuration;
Outputs: *Cluster*: Set of clusters of instances in I;
Method:
1: Let T = set of search trajectories obtained from running A on I using $\mathbf{x}_{init}$;
2: Let S = set of sequences derived from T;
3: For each pair of instances (i, j) in I x I
4: Let $s_1 = S(i)$;
5: Let $s_2 = S(j)$;
6: $Scr\,[s_1,s_2]$ = similarity_score (s_1,s_2);
7: Let *Cluster* = set of clusters obtained by clustering based on Scr;
8: Output *Cluster*;

Figure 1 Training phase.

CluPaTra

The CluPaTra framework is divided into two parts: training and testing phase. In training phase first we execute each training instance and record the solutions visited. Then transform them to a directed-sequence. We calculate the similarity of each sequence using pairwise sequence alignment and perform clustering. In testing phase, we record a testing instance's search trajectory and match it against the clusters to find the most similar cluster. The steps involved in the training and testing phases are shown in Figures 1 and 2.

Search trajectory representation

Search trajectory is defined as a path of solutions discovered by the target algorithm $\mathcal{A}$ as it searches through the neighbourhood search space (Hoos and Stützle, 2004). Search trajectory may vary for each instance, dependent on the number of movements that the target algorithm $\mathcal{A}$ makes. It can be represented as a directed sequence of symbols. In the following, we propose the *exact sequence* and *transition sequence* to transform the search trajectory into directed sequence of symbols.

Exact sequence. In an exact sequence, a symbol on the sequence represents a solution along the trajectory. It encodes two solution attributes: *deviation* and *position type* combined into a symbol with the first two digits being the deviation of the solution quality and the last digit being the position type. The deviation is computed as the deviation of solution quality from OPT (as defined in Definition 1). It represents in a sense a global property of the solution (since it is compared with the global or best known value OPT). The position type represents in a sense a local property of a solution with respect to its

Procedure TestingPhase
Inputs: A : Target algorithm;
i: Arbitrary testing instance;
$Cluster$: Set of clusters (output from training phase);
$\mathbf{x}_{init}$: Initial configuration;
Outputs: B_c : best match cluster;
Method:
1: Let t = a search trajectories obtained from running A on i using $\mathbf{x}_{init}$;
2: Let s = a sequences derived from t;
3: For each cluster $c \in Cluster$
4: Let $Scr\,[c]$ = average similarity from s to all instances in c;
5: Let $B_c = c$, where $Scr\,[c] \geq Scr\,[c']$ for all $c' \neq c$ in $Cluster$;
6: Output B_c;

Figure 2 Testing phase.

Table 1 Position types of solution

Position type label	*Symbol*	*Objective value*		
		Larger	*Equal*	*Smaller*
SLMAX (strict local max)	A	No	No	Yes
LMAX (local max)	X	No	Yes	Yes
LEDGE	L	Yes	Yes	Yes
SLOPE	P	Yes	No	Yes
IPLat (interior plateau)	I	No	Yes	No
LMIN (local min)	M	Yes	Yes	No
SLMIN (strict local min)	S	Yes	No	No

'Yes' = present, 'No' = absent; referring to the presence of neighbours with larger, equal and smaller objective values.

search neighbourhood, and is defined based on the topology of the local search neighbourhood (Hoos and Stützle, 2004). There are seven position types, determined by evaluating the solution objective value with all its local direct neighbours' objective values—whether it is better, worse or equal, as shown in Table 1.

Since not all local search algorithm explore all solution's direct neighbours (as in '*best improvement*' strategy), we explore n additional random direct neighbours (if needed) to determine its position types. This may not be the '*actual*' position types, but it is sufficient to represent the local topology for each solution. The steps to transform the search trajectory into an exact sequence are as follows:

- When running the target algorithm, for each solution, we record its quality and its direct neighbour position. The direct neighbour position is explored based on the target algorithm's neighbourhood structure (ie: *2-opt, 3-opt, LK*, etc). Direct neighbour position is represented as three binary digits with 1 (yes) and 0 (no) for direct neighbour that has same, better and worse objective value respectively.
- For each solution, we calculate its deviation and determine its neighbour position based on Table 1. We then combine those two attributes into a symbol.
- To cater to the fact that some target algorithms may allow cycles and (random) restarts, we add two additional symbols: 'C' and 'J'. 'C' is used when the target algorithm returns to a position that has been discovered previously, while 'J' is used when the local search restarted.

Transition sequence. In contrast to the exact sequence representation, a transition sequence is made up of symbols that represent a transition (or movement) between two neighbouring solutions in the search trajectory. Here we no longer focus on the solution position, but rather we track the movement along the search trajectory in order to detect trajectories that move in parallel but may not be identical (their corresponding positions differ by a constant value). We use transition sequence to capture similarity across different size instances.

In transition sequence, each symbol contains three parts: (I) the absolute difference in deviation between the first and second solutions; (II) the position type of the first solution; and (III) the position type of the second solution. Note that the transition sequence can be derived from the exact sequence. Similar to an exact sequence, a transition sequence may also have two additional symbols: 'C' and 'J'.

Similarity calculation

Having represented trajectories as linear sequences, it is natural to apply pairwise sequence alignment to obtain the similarity score between a pair of trajectories. In the following, we introduce two techniques for pairwise sequence alignment to calculate search trajectory similarities.

Basic sequence alignment. This is the basic sequence alignment in Lindawati *et al* (2011), which applies a standard sequence alignment method to maximize the number of matched symbols between two sequences sequentially. A pair of matched symbols gives a positive score ($+1$), while a gap gives a negative score (-1).

Since the length of a search trajectory may vary, we implement a local alignment strategy that aligns only portions of the sequences but not the entire length. In our paper, we adapt the *Smith-Waterman algorithm* (Han and Kamber, 2006) that works by comparing all possible alignments regardless of their lengths, start and end positions. It then chooses the best alignment as the alignment that maximizes the similarity score, which is the sum of the scores for matched symbols and gaps in the alignment. The final similarity score will be normalized by dividing this score by $\frac{1}{2} \times (|Sequence_1| + |Sequence_2|)$.

Robust sequence alignment. The difference between robust sequence alignment and basic sequence alignment is the matching rule. In the latter, two symbols are a match if and only if the two symbols are exactly identical, while in robust sequence alignment, we consider partial matching.

This relaxed similarity calculation allows us to capture search trajectory's similarity more robustly. Under robust sequence alignment, a match occurs if one of the following conditions is satisfied: (I) The two symbols are identical, and (II) The *position type* of the symbols is the same and the absolute difference in the *deviation* attribute of the two symbols is less than a certain threshold (for simplicity, we set this threshold value to 1 in our experiment).

Both sequence alignment techniques are implemented using standard dynamic programming (Han and Kamber, 2006), with a complexity of $O(n^2)$. To cluster instances (see 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 m is the number of instances in the training set and n is the maximum sequence length of the sequences.

Clustering

After having computed the similarity scores, we derive the distance scores by taking the reciprocal of the corresponding similarity scores, upon which instances are then clustered. We apply two well-known clustering approaches (Han and Kamber, 2006): AGNES and k-medoids clustering.

AGNES. AGNES or AGglomerative NESting is a well-studied hierarchical clustering approach in data mining and machine learning (Han and Kamber, 2006). It works by creating clusters for each individual instance and then merging two closest clusters (ie, a pair of clusters with the smallest distance) resulting in fewer number of clusters of larger sizes until all instances belong to the same cluster or a termination condition is reached (eg a prescribed number of clusters is reached).

To determine the minimal number of clusters to be used, we apply the L method (Salvador and Chan, 2004). The L method works by using the 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, which in this paper is the average distance among all instances in two different clusters. L method fits the curve in the evaluation graph into two lines and chooses the intersection point between these two lines as the optimal number of clusters. This method only requires AGNES algorithm to be run once, since all the clusters generated 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 less than 10. Thus, the x-axis only shows the number of clusters from 1 to 10. The overall complexity of AGNES with L method is $O(n^2)$ with n being the number of instances.

k-medoids. k-medoids is a partition-based clustering method that repeatedly breaks the data set up into k groups as an attempt to improve clusters' evaluation function (Han and Kamber, 2006), which in this paper is the average distance among all instances in two different clusters. It is a variant of k-means method but it selects real data points as centres (medoids or exemplars) instead of imaginary points. The complexity of k-medoids is $O(k(n-k)^2)$ with k being the number of clusters and n being the number of instances. We need to manually specify the number of clusters (ie, the parameter k).

Instance-specific parameter tuning

Using the cluster from CluPaTra, we apply existing one-size-fits-all algorithms (such as CALIBRA, ParamILS or GGA) 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.

Experiment design

Here we briefly explain our experimental design for two classical problems, Travelling Salesmen Problem (TSP) and Quadratic Assignment Problem (QAP).

Travelling salesmen problem (TSP)

The target algorithm to solve TSP is a well-known Iterated Local Search (ILS) algorithm as implemented in Halim *et al* (2007) with four discrete parameters to be tuned as described in Table 2(I). For all instances, we set the maximum number of iteration to 1000. We applied our target algorithm to 70 benchmark instances extracted from TSPLib. For best known values, we used the optimum/best values 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.

Quadratic assignment problem (QAP)

The target algorithm to solve QAP is the hybrid Simulated Annealing and Tabu Search (SA-TS) algorithm (presented in Ng *et al* (2008). It uses the Greedy Randomized Adaptive Search Procedure (GRASP) to obtain an initial solution, and then using a combined Simulated Annealing (SA) and Tabu Search (TS) algorithm to improve the solution. There are four parameters, discrete and continuous, to be tuned as described in Table 2(II). For continuous parameter, we discretize it to 20 possible values by simple enumeration from minimum to maximum value. For all instances, we set the maximum number of iterations

Table 2 Parameters for target algorithm

Parameter	*Description*	*Range*
I. ILS on TSP		
Pert	number of perturbations being done	[1,10]
n_improve	max non-improving moves	[1,10]
choice	perturbation strategy where: 3 = 3-opt change and 4 = double-bridge move	[3,4]
acp	acceptance criteria strategy where: 0 = accept only improving moves and 1 = accept all moves	[0,1]
II. SA-TS on QAP		
Temp	Initial temperature of SA	[100,5000]
Alpha	Cooling factor	[0.1,0.9]
Length	Length of tabu list	[1,10]
Pct	Percentage of non-improving iterations	[0.01,0.1]
III. Industrial case study		
MaxS	max successes number	[100, 1000]
maxTries	max tries	[100, 1000]
maxComp	max solutions generated	[1000, 50 000]
maxReject	max consecutive rejections	[100, 1000]
maxChG	max change in a variable value	[100, 1000]
maxTriesG	max tries to generate a feasible solution	[100, 1000]
coolFactor	factor to reduce the temp	[0.5, 1]
oStrictness	the strictness of the oracle function	[0, 100]

to 500. We used 50 benchmark instances from QAPLib, and randomly selected 40 instances for training and 10 for testing. The problem size (number of facilities) varied from 20 to 150. For best known values, we used the optimum/best values from QAPLib.

Experiment setting and setup

We construct four instantiations of CluPaTra resulting from two search trajectory representations (exact and transition) and two similarity calculation techniques (basic and robust) using AGNES as its clustering method. The terminology used subsequently is given in Table 3. Note that the 'Standard' instantiation is the one proposed in Lindawati *et al* (2011).

To record the search trajectory, we run the target algorithm against all instances using a random configuration and record all the moves of the target algorithm, unless stated otherwise. The length may vary. We did not set any parameter for CluPaTra since it does not have parameter.

We compared our experiment results with the ISAC method, a similar clustering-approach that uses problem-specific features, that we implemented based on Kadioglu *et al* (2010). Since ISAC requires problem-specific features, we selected the standard deviation of the city distances, the variance of the normalized nearest neighbour distances and the coefficient of variation of the normalized nearest neighbour distances for TSP (Smith-Miles and Lopes, 2012) and flow dominance and sparsity of flow matrix for QAP (Stützle and Fernandes, 2004).

Table 3 CluPaTra instantiations for performance comparison

Instantiation	*Search trajectory representation*	*Similarity calculation*
Standard	Exact sequence	Basic Seq. Align.
Trans	Transition sequence	Basic Seq. Align.
Robust	Exact sequence	Robust Seq. Align.
Trans-Robust	Transition sequence	Robust Seq. Align.

Tuning setup

We chose to use Hutter *et al* (2009) as the one-size-fits-all configurator. For each cluster (or training set), we randomly sorted the instances, ran ParamILS 5 times and took the average performance. To ensure unbiased evaluation, we then run 5-fold cross-validation (Han and Kamber, 2006) over those instances and measured the average performance over all folds. To do 5-fold cross validation, we randomly divided the instances into five random groups and used four groups as training instances and one group as testing instances. We repeat the process five times and take the average. We did a non-parametric Wilcoxon signed-rank test to compare CluPaTra's overall performance with that of ParamILS. We considered p-value below 0.05 to be statistically significant (confidence level 5%).

All experiments were performed on a 1.7GHz Pentium-4 machine running Windows XP. We measured runtime as the CPU time needed by this machine. As an input to the configurator, we set a cut-off time of 10 s per run for the TSP target algorithm and 100 s for the QAP target algorithm. For each CluPaTra cluster, we allowed each configuration process to execute the target algorithm for a maximum of 2 CPU hours and to call the target algorithm for a maximum of $25 \times n$ times, where n is the number of instances in the cluster. To ensure fair comparison, we set the time budget for ISAC and ParamILS to be equal to the average total time needed to run a full process of CluPaTra instantiations. This time budget is the stopping condition for ISAC and ParamILS.

Verification of similarity preservation

Prior to presenting experimental results, we provide a scientific argument for our approach. In the following, we

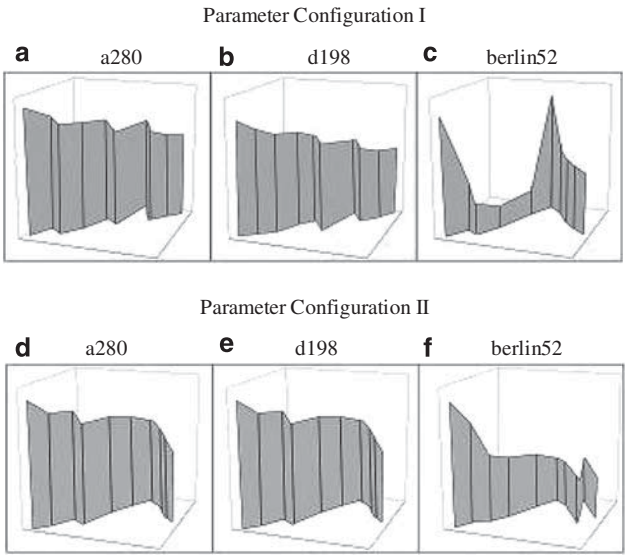

Figure 3 Search trajectories of 3 TSP instances *a280*, *d198* and *berlin52* using two random parameter configuration.

will justify our claim that the *similarity* of search trajectories between instances is preserved across configurations, by providing a series of experimental observations.

First, we provide a visual intuition for similarity preservation across different parameter configurations. Figure 3 shows the trajectories obtained by 10 consecutive moves of an Iterated Local Search (ILS) algorithm for three TSP instances, namely *a280*, *d198* and *berlin52* when the algorithm is run on two random parameter configurations, namely configuration I and configuration II. The *xy* plane represents the search space while *z* axis represents the objective value. To layout the moves into a 2-dimensional *xy* plane, we calculate the distance between two solutions (eg, number of different cities in TSP) and apply '*the spring model*' (Halim *et al*, 2007). '*The spring model*' provides a heuristic for good layout where the Euclidean distance between two solutions in the *xy* plane is roughly proportional to their Hamming distance. In this example, we observe that for both configurations, *a280* and *d198* exhibit very similar topology ((a) and (b), (d) and (e)), while *berlin52* has a different topology compared with the similarity of *a280* and *d198*.

Next, we provide a statistical verification of the notion similarity preservation on the trajectories produced by the TSP and QAP target algorithms used in our experiments. For this purpose, we verify on random pairs of instances across different parameter configurations. We do the following: first, we randomly select two source instances (namely, benchmark instances *a280, berlin52* for TSP and *chr20a, sko100b* for QAP); we next select randomly 10 other destination TSP (resp. QAP) instances. We randomly generate five parameter configurations for each of the target algorithms, and generate the trajectory for each instance. To simplify the experiment, we take the first 300 solutions obtained from the target algorithm as the search trajectory samples and calculate its similarity score.

For each source-destination pair and each configuration, we compute their similarity score (based on the Standard instantiation). The results are presented in Figure 4. Observe that most pairs of instances maintain their similarity across different parameter configurations as shown by the small scatter of similarity values in each column (with the exception of several instances in the a280 case). The deviation, mean and coefficient of variance (CV) of similarity values for the different parameter configurations are given in Table 4. For most pairs, the CV value is low (especially for QAP pairs), which means that the similarity scores across different parameter configurations do not differ substantially from one another.

Based on the above observations, we argue that even though a given instance may have different search trajectories under different configurations, the *similarity* between two instances is preserved across configurations. This similarity preservation property allows us to perform clustering of instances using an arbitrary parameter configuration.

Empirical result

In this section, we report our cluster results and tuning result.

Clustering analyses

We compare the clusters generated by CluPaTra and ISAC and take the *ground-truth* classification (if exists) as the benchmark. Average number of clusters from 5-folds for CluPaTra (standard, trans, robust, trans-robust) and ISAC for TSP are 3, 6, 6, 6, and 5.8 respectively; while for QAP are 4.2, 6, 6, 6, and 5.8 respectively. The example of cluster generated by the Trans CluPaTra instantiation and ISAC is reported in Figure 5 for TSP and Figure 6 for QAP.

For TSP, we observe that Trans CluPaTra method is able to capture the similarity of instances with differing sizes which may have different search trajectory symbols but have similar transitions along the search trajectories. Because of the non-existence of the *ground-truth* classification for TSP benchmark instances, we cannot compute the cluster qualities directly, instead it is inferred from the performance of the target algorithm, which is described in later section.

For QAP, we use the existing well-studied classification based on the distance and flow metrics (Taillard, 1995) as the *ground-truth* classification. Notice that the clustering by Trans CluPaTra is almost the same as the *ground-truth* classification. Furthermore, Trans CluPaTra constructs

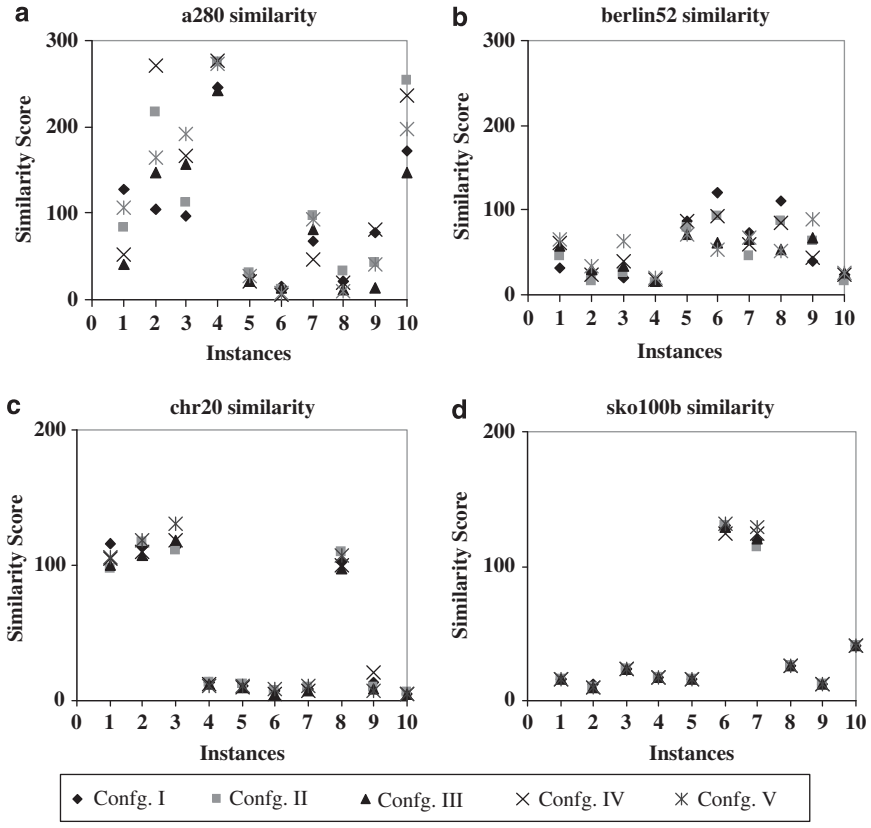

Figure 4 Search trajectory similarity score between TSP (*a280*, *berlin51*) and QAP (*chr20a*, *sko100b*) instance and 10 other random instances using 5 different random parameter configurations.

Table 4 Similarity score of instance pairs

Instances	σ	μ	c_v	σ	μ	c_v
I. TSP		a280			berlin52	
ch150	32.70	**82.20**	0.40	12.42	52.20	0.24
d1655	57.47	**181.20**	0.32	6.02	25.60	0.00
d657	35.31	**144.60**	0.24	15.54	36.20	0.43
fl3795	14.81	**262.00**	0.06	2.24	16.40	0.14
kroa150	4.12	25.80	0.16	6.73	**78.80**	0.09
krob100	3.58	11.00	0.33	24.33	**84.20**	0.29
lin105	18.18	**77.20**	0.24	9.35	62.40	0.15
pr152	7.78	18.80	0.41	22.38	**77.40**	0.29
rd100	25.32	50.80	0.50	17.85	**60.40**	0.30
ts225	39.55	**201.60**	0.20	3.88	22.40	0.17
II. QAP		chr20a			sko100b	
chr22a	6.49	**104.80**	0.06	0.00	16.00	0.00
chr22b	4.13	**113.40**	0.04	1.20	10.60	0.11
lipa50b	6.83	**118.40**	0.06	0.00	24.00	0.00
nug28	0.75	12.20	0.06	0.00	**18.00**	0.00
nug30	0.75	10.80	0.07	0.00	**16.00**	0.00
sko100e	1.60	6.80	0.24	2.87	**129.40**	0.02
sko90	1.60	8.80	0.18	5.04	**121.20**	0.04
ste36a	4.71	**103.20**	0.05	0.00	26.00	0.00
tai30a	4.71	12.20	0.39	0.00	**13.00**	0.00
wil100	0.40	5.20	0.08	0.00	**41.00**	0.00

σ = standard deviation; μ = mean; c_v = coefficient of variation.
Boldface indicates best similarity score mean.

better clusters compared with the Standard CluPaTra and ISAC with respect to cluster quality metric that compare the clusters with *ground-truth* cluster as shown in Table 5(I). We observe that the cluster quality score for Trans CluPaTra is the highest compared with other CluPaTra instantiation and ISAC.

Time performance

The most time-consuming procedure in the training phase is similarity calculation. Different similarity calculation techniques require different computational budget. The Robust sequence alignment technique takes almost four times longer than the basic sequence alignment. This happens because it requires more computation time to find partial-match symbols. The average total time (in hour) needed to run the overall process in training phase for each fold is shown in Table 5(II).

Tuning performance comparison

We evaluate the effectiveness of four CluPaTra instantiations against the vanilla one-size-fits-all configurator (ParamILS) and ISAC. We measure the performance by using the performance metric described in Definition 1.

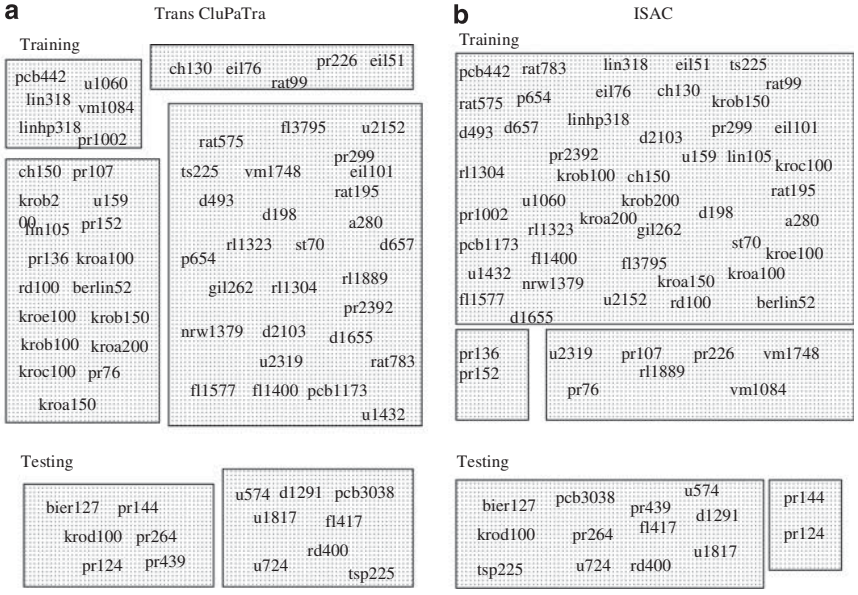

Figure 5 TSP cluster result comparison. (a) Trans CluPaTra; and (b) ISAC.

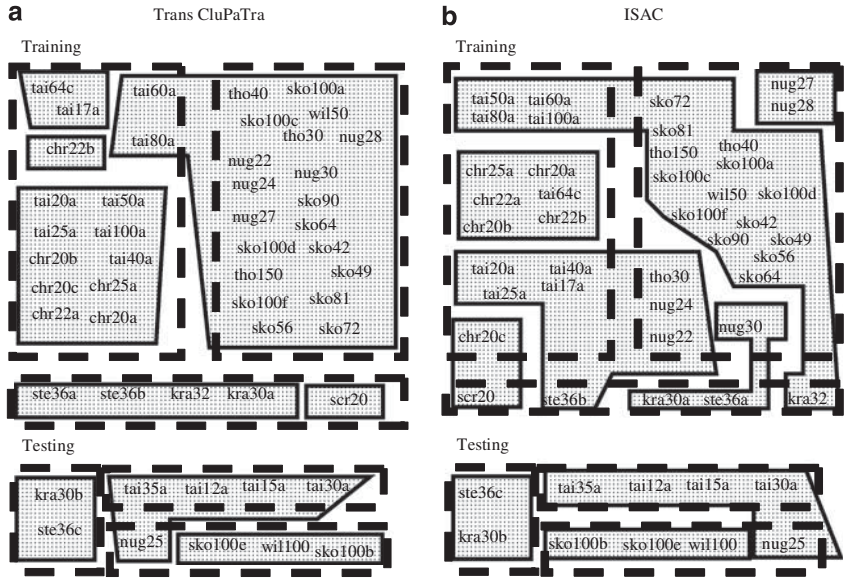

Figure 6 QAP cluster result comparison. (a) Trans CluPaTra; and (b) ISAC.

Table 5(III) shows the results of the average comparison. Comparing the four CluPaTra instantiations, we observe that the performance of Trans, Robust and Trans-Robust is significantly superior to the Standard instantiation.

To verify the CluPaTra effectiveness in providing best configuration for each testing instance, we run the target algorithm for all QAP testing instance in Figure 6 using parameter configurations from each cluster and show the result in Table 6. From the table we observe that each testing instance, except for tai35a, has the best performance using parameter configuration from the most similar cluster.

Comparison on clustering method

Next, we analyse the effects of different clustering methods by applying AGNES and *k*-medoids clustering methods on the Trans instantiation. We set k to be equal to the AGNES cluster number. Table 5(IV) shows that AGNES performs slightly better than *k*-medoids even though it is not statistically significant.

Table 5 Empirical result

Technique	*TSP*		*QAP*	
	Training	*Testing*	*Training*	*Testing*
I. Clustering analyses				
Standard	–	–	0.68	0.70
Trans	–	–	**0.85**	**0.90**
Robust	–	–	0.78	0.70
Trans-Robust	–	–	0.7	0.80
ISAC	–	–	0.80	0.80
II. Total computation time				
Standard	5.58	**0.04**	**8.20**	**0.01**
Trans	**5.46**	0.05	8.23	**0.01**
Robust	6.02	0.07	9.01	0.03
Trans-Robust	6.05	0.07	9.05	0.03
III. Performance comparison^{+}				
ParamILS	2.671	2.022	2.212	2.273
	(0.29)	(0.22)	(0.15)	(0.25)
ground-truth	–	–	1.928*	2.094*
			(0.15)	(0.21)
Standard	2.222*	1.929*	1.991*	2.195*
	(0.24)	(0.26)	(0.19)	(0.20)
Trans	**2.011***	**1.715***	**1.878***	**2.081***
	(0.23)	(0.27)	(0.17)	(0.24)
Robust	2.102*	1.812*	1.889*	2.101*
	(0.21)	(0.27)	(0.18)	(0.28)
Trans-Robust	2.056*	1.927*	1.901*	2.185*
	(0.27)	(0.23)	(0.16)	(0.21)
ISAC	2.020	1.884	1.982	2.153
	(0.25)	(0.21)	(0.19)	(0.21)
IV. Different clustering				
AGNES	**2.012**	2.053	**1.718**	1.771
k-medoids	**1.879**	1.898	**2.083**	2.161

$+$ = mean (coefficient of variation).
* = statistically significant against ParamILS.
Boldface indicates best cluster quality, computation time and performance using performance metric in Definition 1.

Table 6 Testing instances performance using different cluster's parameter configuration

Instance	*Cluster*	*Parameter configuration for each cluster*					
		C#1	*C#2*	*C#3*	*C#4*	*C#5*	*C#6*
nug25	1	**0.48**	0.64	0.69	0.58	0.58	0.58
tai12a	1	**0**	**0**	**0**	**0**	**0**	2.80
tai15a	1	**0.19**	0.76	0.52	1.22	1.72	2.66
tai30a	1	**1.86**	2.81	2.20	2.57	3.03	2.65
tai35a	1	1.49	**1.38**	3.37	3.75	3.047	3.95
kra30b	2	**0.07**	**0.07**	0.97	**0.07**	1.88	1.18
ste36c	2	1.91	**1.71**	5.08	8.95	7.84	7.82
sko100b	3	0.69	1.22	**0.53**	1.16	1.31	1.29
sko100e	3	1.18	1.18	**1.10**	1.30	1.34	1.21
wil100	3	0.65	0.69	**0.63**	0.81	0.96	0.93

Boldface indicates best performance using performance metric in Definition 1.
Parameter configuration for:
C#1: Temp = 4000, Alpha = 0.9, Length = 7, Pct = 0.08.
C#2: Temp = 2000, Alpha = 0.5, Length = 7, Pct = 0.09.
C#3: Temp = 3000, Alpha = 0.3, Length = 10, Pct = 0.1.
C#4: Temp = 4000, Alpha = 0.3, Length = 10, Pct = 0.07.
C#5: Temp = 100, Alpha = 0.3, Length = 10, Pct = 0.03.
C#6: Temp = 5000, Alpha = 0.1, Length = 1, Pct = 0.08.

Table 7 Industrial case study result

	Default	*ParamILS*	*CluPaTra*
Training	2.574	2.676	**0.226**$^{+}$*
Testing	2.391	1.521^{+}	**0.154**$^{+}$*

Boldface indicates best performance using performance metric in Definition 1.
$+$ = statistically significant against Default Configuration.
* = statistically significant against ParamILS Configuration.

Industrial case study

In this section, we report results on an industrial case study. We apply CluPaTra to tune an algorithm for an aircraft spares inventory optimization problem of a large commercial aircraft maker based in Europe. The objective of this algorithm is to determine the optimal inventory allocation strategy that can fulfil specific target services levels. The target algorithm is a *Simulated Annealing* (SA) algorithm (Gunawan *et al*, 2011), which has eight parameters that used to control SA behaviour as described in Table 2(III).

We apply CluPaTra to 50 sample instances, with 25 randomly picked instances as training instances and the remaining 25 as testing instances. We set cut-off times of 500 s per run and allowed each configuration process to execute the target algorithm for a maximum of 48 CPU hours and to call the target algorithm for a maximum of $25 \times n$ times, where n is the number of instances in the cluster. In Table 7, we present the average of percentage deviation value (Definition 1). We compare the result of the three approaches with the best known values used by our industry partner. The result shows that the CluPaTra results are superior to the default and ParamILS results.

Discussion and future direction

To represent the search trajectory, we need the best known/optimum solution value (OPT) for each instance. We use either (a) the known global optimal value, or (b) when the global optimal value is unknown, the best known value. From the experiment result, we observe that CluPaTra method using either known global optimal value or best known value is able to generate good clusters and hence improve the overall performance. We will further explore this issue in future work.

In dealing with complex optimization problem for an industrial case study, we show that CluPaTra can provide better parameter configurations than the default manually tuned parameters. It illustrates the practical impact of our proposed approach on tuning local search algorithms.

As future direction, we would like to extend CluPaTra for other purposes such as algorithm selection or hyperheuristic.

Related work

A wide variety of generic strategies for automated parameter configuration have been explored in the literature. These approaches focus on finding the best parameter configuration for the entire set (or distribution) of problem instances by using the average quality or other statistical measures. We term these approaches as *one-size-fits-all* automated tuning. There are broadly two schemes: model-based and model-free approaches. Recent approaches in model-free approaches are *F-Race* (Birattari *et al*, 2002), *ParamILS* (Hutter *et al*, 2009), and *GGA* (Ansótegui *et al*, 2009); while in model-based ones are *CALIBRA* (Adenso-Díaz and Laguna, 2006), and *SMAC* (Hutter *et al*, 2011). A good review of three recent automated parameter tuning methods can be found in Hoos (2012).

On a separate front, there have been approaches that attempted to select the best parameter configuration on a per-instance basis. Approaches like McAllester *et al* (1997), Patterson and Kautz (2001), Hutter *et al* (2006) use regression model to construct a model based on the instance's features that will determine the configurator's strategy, while others like Kadioglu *et al* (2010) use clustering to divide the instances and find a good parameter configuration on the cluster created.

Conclusion

Given the high complexity of the tuning problem that demands instance-based accuracy, we have proposed a solution framework that is a relatively intuitive, computationally efficient and generic *vis-à-vis* existing approaches (which are mostly problem-specific). As ongoing work, we are exploring ways to address the 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 the 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).

References

Adenso-Díaz B and Laguna M (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research* **54**(1): 99–114.

Ansótegui C, Sellmann M and Tierney K (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent IP (ed). *Principles and Practice of Constraint Programming – CP 2009*. Proceedings of the 15th International Conference. Lisbon, Portugal, 20–24 September, Springer-Verlag: Berlin, Heidelberg, pp 142–157.

Birattari M, Stützle T, Paquete L and Varrentrapp K (2002). A racing algorithm for configuring metaheuristics. In: Langdon WB *et al* (eds), *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, 9–13 July, Morgan Kaufmann: San Francisco, CA, pp 11–18.

Gunawan A, Lau HC and Wong E (2011). Real-world parameter tuning using factorial design with parameter decomposition. In: *MIC 2011: The IX Metaheuristics International Conference*. Udine, Italy, 25–28 July.

Halim S, Yap Y and Lau HC (2007). An integrated white + black box approach for designing and tuning stochastic local search. In: Bessiere C (ed). *Principles and Practice of Constraint Programming – CP 2007*. Proceedings of the 13th International Conference. Providence, RI, 23–27 September, Springer-Verlag: Berlin, Heidelberg, pp 332–347.

Han J and Kamber M (2006). *Data Mining: Concept and Techniques*. 2nd edn, Morgan Kaufman: San Francisco, CA.

Hoos HH (2012). Automated algorithm configuration and parameter tuning. In: Hamadi Y, Monfroy E, and Saubion F (eds). *Autonomous Search*. Springer: New York, pp 37–72.

Hoos HH and Stützle T (2004). *Stochastic Local Search: Foundation and Application*. Morgan Kaufman: San Francisco, CA.

Hutter F, Hamadi Y, Hoos HH and Leyton-Brown K (2006). Performance prediction and automated tuning of randomized and parametric algorithms. In: Benhamou F (ed). *Principles and Practice of Constraint Programming – CP 2006*. Proceedings of CP 2006, the 12th International Conference, Nantes, France, 25–29 September, Springer-Verlag: Berlin, Heidelberg, pp 213–228.

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

Hutter F, Hoos HH and Leyton-Brown K (2011). Sequential model based optimization for general algorithm configuration. In: Coello Coello CA (ed). *Learning and Intelligent Optimization*. 5th International Conference, LION 5, Selected Papers. Rome, Italy, 17–21 January, Springer-Verlag: Berlin, Heidelberg, pp 507–523.

Kadioglu S, Malitsky Y, Sellmann M and Tierney K (2010). ISAC: Instance-specific algorithm configuration. In: Coelho H, Studer R and Wooldridge M (eds). *ECAI 2010*: *19th European Conference on Artificial Intelligence*, 16–20 August, Lisbon, Portugal, Proceedings, IOS Press: Amsterdam, The Netherlands, pp 751–756.

Lindawati L, Lau HC and Lo D (2011). Instance-based parameter tuning via search trajectory similarity clustering. In: Coello Coello CA (ed). *Learning and Intelligent Optimization: 5nd International Conference*, LION 5, Rome, Italy, 17–21 January, Selected Papers, Springer-Verlag: Berlin, Heidelberg, pp 131–145.

Malitsky Y and Sellmann M (2009). Stochastic offline programming. In: *The 21st IEEE International Conference on Tools with Artificials Intelligence*, Proceedings, Newark, New Jersey, 2–5 November, IEEE Computer Society: California, USA, pp 784–791.

McAllester D, Selman B and Kautz H (1997). Evidence for invariants in local search. In: Reame NK (ed). *14th National Conference on Artifical Intelligence*. AAAI Press, California, USA, pp 321–326.

Ng KM, Gunawan A and Poh KL (2008). A hybrid algorithm for the quadratic assignment problem. In: Arabnia HR (ed). *Proceedings of the 2008 International Conference on Scientific Computing*, CSC 2008: At WORLDCOMP'08, 14–17 July, Las Vegas, NV, CSREA Press: Athens, GA, USA, pp 14–17.

Patterson DJ and Kautz H (2001). Auto-Walksat: A self-tuning implementation of Walksat. *Electronic Notes in Discrete Mathematics* **9**: 360–368.

Reeves CR (1999). Landscapes, operators and heuristic search. *Annals of Operations Research* **86**: 473–490.

Salvador S and Chan P (2004). Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In: *The 16th IEEE International Conference on Tools with Artificials Intelligence*, Proceedings, 15–17 November, Boca Raton, FL, IEEE Computer Society: California, USA, pp 576–584.

Smith-Miles K and Lopes L (2012). Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research* **39**(5): 875–889.

Stützle T and Fernandes S (2004). New benchmark instances for the qap and the experimental analysis of algorithms. In: Gottlieb J and Raidl GR (eds). *Evolutionary Computation in Combinatorial Optimization: 4th European Conference,* EvoCOP 2004, Coimbra, Portugal, 5–7 April, Proceedings, Springer-Verlag: Berlin, Heidelberg, pp 199–209.

Taillard ÉD (1995). Comparison of iterative searches for the quadratic assignment problem. *Location Science* **3**(2): 87–105.

Received March 2011;
accepted November 2012 after three revisions