2013

Clustering of Search Trajectory and its Application to Parameter Tuning

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/10.1057/jors.2012.167

Clustering of Search Trajectory and Its Application to Parameter Tuning

Lindawati, Hoong Chuin Lau and David Lo
School of Information Systems, Singapore Management University

[1]

Abstract. This paper is concerned with automated classification of Combinatorial Optimization Problem (COP) 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: Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP) and show that CluPaTra offers encouraging result both in cluster quality and overall performance. We also apply the CluPaTra framework on an industrial case study involving 8 parameters, and compare with the performance of default settings used by the company.

Keywords. generic feature, search trajectory, instance-based automated parameter tuning, sequence alignment, local search algorithm

1 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 [5]. Previous studies reveal that the performance of a meta-heuristic is dependent on the instance specific characteristics/features that determine its intrinsic difficulty [25]. Consequently, there has been increasing interest in finding the instances features that have impact on difficulty in terms of performance to improve the algorithm performance [2, 11, 12, 32, 33, 36, 37, 38, 39, 40, 41].

Various problem specific features have been proposed for a wide range of combinatorial optimization problems in the literature. Some notable features are flow dominance for QAP [11, 37, 38, 40] and population correlation structure and constraint slackness for Knapsack Problem [12, 33]. 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 [36]. 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.

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) [2, 14, 32, 39]. Problem-independent features can be used on different combinatorial optimization problem, such as TSP [32], QAP [2] and Knapsack Problem [39]. Examples of these features are fitness distance correlation (FCD) [32, 14] and ruggedness coefficient [2, 14]. 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, which is defined as the path that a local search algorithm follows as it searches from an initial solution to its neighbour from one iteration to the next, as the problem-independent feature and exploit data-mining techniques to cluster problem instances according to their search trajectories. The rationale of this feature is predicated on the relationship between fitness landscape and search trajectories [8, 9]. We use search trajectory as a proxy for the fitness landscape. 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 [20, 23, 26]. Our approach is similar to ISAC [20], 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 [8, 9], and (2) the tight correlation between the fitness landscape and algorithm performance [32]. Our bold conjecture is that trajectory patterns can provide guidance for setting parameter configurations; more precisely, 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 their trajectory patterns).

This paper extends the vanilla CluPaTra framework recently proposed by the same authors in [24] 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. On the search trajectory representation, we introduce the transition sequence representation, and compare it with the exact sequence representation proposed in [24]. We experiment with two variants of pairwise sequence alignment to calculate search trajectories similarity, and we

[1]This is an extended version of the conference paper that appears in LION5

apply two well-studied clustering methods to cluster the instances. 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.
- We investigate experimental performance in two classical COP problems: the Traveling Salesmen Problem (TSP) and Quadratic Assignment Problem (QAP), as well as an industrial case study.

2 Problem Statement and Definition

In this section, we define the clustering and parameter tuning problem.

2.1 Clustering Problem

We define the clustering problem as follows:

Definition 1 (Clustering Problem [CP]) *Let I be a set of problem instances and W be a set of pairwise similarity score for all instance in I, the* CP *is the partitioning of I into k clusters $\{C_1, C_2, C_3, ..., C_k\}$ such that the cluster quality is maximized.*

The different formulas to compute the pairwise similarity score will be proposed in the subsection 3.3. We measure the *quality* of a clustering by using *extrinsic* method. *Extrinsic* method compares the clusters against the known class labels or *ground-truth* clusters (i.e. the set of clusters which represents the ideal/optimal clustering) [10]. We define the cluster quality as follows:

Let I (resp. I_t) be a set of training (resp. testing) instances, C be the set of clusters generated from the training phase and C_g be the **ground-truth** clusters. Each cluster in $c \in C$ has an associated *home* cluster $c_g \in C_g$ which contains the largest number of instances contained in c (ties broken arbitrarily).

Definition 2 (Training Clusters Quality Score [$\mathcal{Q}_{train}$]) *For each cluster $c \in C$, let $max(c)$ count the number of instances in the cluster that belong to the associated home cluster. $\mathcal{Q}_{train}$ is defined as the sum of $max(c)$ over all $c \in C$ divided by the number of instances in I.*

Definition 3 (Testing Instance Mapping Score [$\mathcal{Q}_{test}$]) *For each instance $i \in I_t$, we say that i is "matched" if it is mapped to a cluster $c \in C$ whose home cluster $c_g \in C_g$ also contains i. $\mathcal{Q}_{test}$ is defined as total number of such matches divided by the number of instances in I_t.*

2.2 Parameter Tuning Problem

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*.

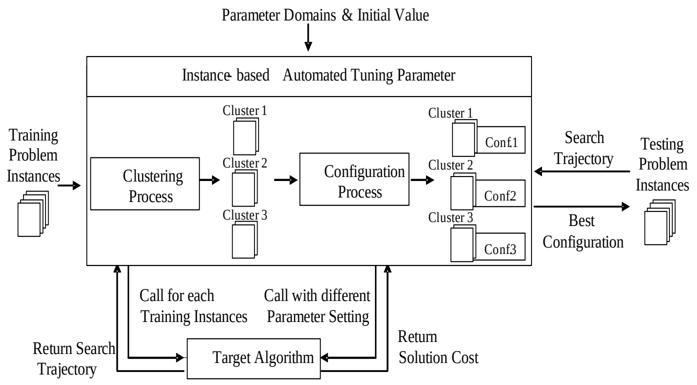

Figure 1: CluPaTra Framework

We measure the target algorithm performance based on the quality of their solutions. We define target algorithm performance $\mathcal{H}$ as follows:

Definition 4 (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* **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 5 (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* **x** *$\in \Theta$ for each $i \in I$ such that $\mathcal{H}_x(i)$ is minimized over Θ.*

3 CluPaTra

In this section, we present the CluPaTra framework and its three major components: search trajectory representation, similarity calculation and clustering method and propose different variants for these components. We then describe the implementation of CluPaTra framework in instance-specific parameter tuning.

3.1 CluPaTra Framework

The CluPaTra framework illustrated in Fig. 1 is divided into two parts: the 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 using hierarchical 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 Fig. 2.

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

Procedure TestingPhase
Inputs:   $A$: Target algorithm;
          $i$: Arbitrary testing instance;
          $C$: Set of clusters (output from training phase);
          $\mathbf{x}_{init}$: Initial configuration;
Outputs:  $BestClust$: best match cluster;
Method:
1: Let $traj$ = a search trajectories obtained from running $A$
     on $i$ using $\mathbf{x}_{init}$;
2: Let $seq$ = a sequences derived from $traj$;
3: For each cluster $c \in C$
4:     Let $Score[c]$ = average similarity from $seq$ to all instances in $c$;
5: Let $BestClust$ = $c$, where $Score[c] \geq Score[c']$ for all $c' \neq c$ in $C$;
6: Output $BestClust$;
```

Figure 2: Training and Testing Phase

3.2 Search Trajectory Representation

Search trajectory is defined as a path of solutions discovered by the target algorithm $\mathcal{A}$ as it searches through the neighborhood search space [14]. 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.

3.2.1 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 the deviation of solution quality from OPT (as defined in Definition 4). 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 the local property of a solution with respect to its search neighborhood, and is defined based on the topology of the local search neighborhood [14]. 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 equal. The 7 positions types are given in Table 1.

Since not all local search algorithm explore all solution's direct neighbors (as in "*best improvement*" strategy), we explore n additional random direct neighbors (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:

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 with larger ('$<$'), equal ('=') and smaller ('$>$') objective values

- When running the target algorithm, for each solution, we record its quality and its direct neighbor position. The direct neighbor position is explored based on the target algorithm's neighborhood structure (i.e.: *2-opt*, *3-opt*, *LK*, etc). Direct neighbor position is represented as 3 binary digits with 1 (yes) and 0 (no) for direct neighbor that has same, better and worse objective value respectively. Each time the target algorithm find direct neighbor that has the same, better or worse objective value, the neighbor position is updated to 1. Generally, this exploration is done by the target algorithm during the local search with small additional computation time. For some target algorithm, we need to run additional neighborhood exploration to find direct neighbor position. We explore few numbers of random direct neighbor and stop the exploration as soon as we find at least one neighbor that has the same, better and worse objective value.

- For each solution, we calculate its deviation and determine its neighbor position based on Table 1. We then combine those two attributes into a symbol.

- We compress the search trajectory sequence by removing the consecutive repetition symbols and represent it by only one 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.

These steps only run once for each instance. An example of the sequence representing the eil51 search trajectory in Fig. 4 is *15L-11L-09L-07L-07P-06P-04S-05L-J-21L-19L*. For each solution in eil51 search trajectory, the quality and neighbor position is recorded and then transform to a sequence (note that the neighbor position is not illustrated in the figure). Notice that after position 8, the target algorithm performs a random restart, hence we add 'J' symbol after position 8.

3.2.2 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 neighboring 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'. These attributes are also generic and can be easily retrieved/computed from any local-search-based algorithm albeit different problems. An example of the transition sequence representing the eil51 search trajectory in Fig. 5 is *4LL-2LL-2LL-0LP-1PP-2PS-1SL-J-2LL* which is derived from the trajectory sequence *15L-11L-09L-07L-07P-06P-04S-05L-J-21L-19L*.

3.3 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.

3.3.1 Basic Sequence Alignment

This is the basic sequence alignment in [24], which applies a standard sequence alignment method to maximize the number of matched symbols between two sequences sequentially, allowing gaps. A pair of matched symbols gives a positive score (+1), while a gap gives a negative score (-1). An example of sequence alignment for the *kroa100* and *bier127* search trajectories from Fig. 4 is illustrated in Table 2.

Since the length of a search trajectory may vary, we implement a local alignment strategy that align only portions of the sequences but not the entire length. One well-known algorithm that performs such sequence alignment is the *Smith-Waterman algorithm* [10] 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. Note that the best alignment may start and end anywhere in the two sequences, so long as it produces the best similarity score. In our paper, we adapt the Smith-Waterman algorithm and use the best similarity score for each pair of sequences. The final similarity score will be normalized by dividing this score by $\frac{1}{2} \times (|Sequence_1| + |Sequence_2|)$.

3.3.2 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).

Table 2: Example of Sequence Alignment from 2 TSP instances

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

Both sequence alignment techniques are implemented using standard dynamic programming [10], 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.

3.4 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: AGNES [22] and k-medoids clustering [21].

3.4.1 AGNES

AGNES or AGglomerative NESting is a well-studied hierarchical clustering approach in data mining and machine learning [22]. It works by creating clusters for each individual instance and then merging two closest clusters (i.e., 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 (e.g. a prescribed number of clusters is reached).

To determine the minimal number of clusters to be used, we apply the L method [34]. 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 determines the number of clusters by fitting the evaluation graph into two lines that most closely fit the curveand choosing the intersection point between those two lines as the optimum number of clusters. The intersect point is the point of maximum curvature of this graph which has minimum average distance (calculated using root mean square error) for both the left and right side of the intersect point. It is calculated using the following formula:

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

where: $RMSE(L)$ is root mean squared error in the left side of c; n_L is number of points in the left side of c; $RMSE(R)$ is root mean squared error in the right side of c; and n_R is number of points in the right side of c

This method only requires AGNES algorithm to be run once, since all the clusters generated by AGNES can be recorded in one run. And

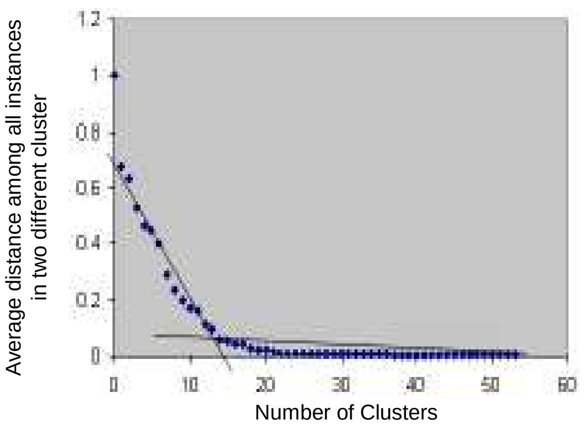

Figure 3: L-Method Illustration

since we want to produce a compact set of clusters, we limit the number of clusters to less then 10. Thus, the x-axis only show 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.

3.4.2 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 [21], which in this paper, is the average distance among all instances in two different cluster. It is a variant of k-means method but it selects real data points as centers (medoids or exemplars) instead of imaginary points. Here we implement the simplest k-medoids approach, which is Partitioning Around Medoids (PAM) [30].

To obtain k-medoids clusters, PAM begins with an arbitrary selection of k instances as medoids and assign all non-medoids instances to the closest medoids. Then in each step, a swap between a medoids and a non-medoid is made as long as such swap would result in an improvement of cluster evaluation value. PAM stops when the swap no longer improve the cluster evaluation value. The complexity of PAM 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 (i.e., the parameter k).

3.5 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.

4 Experiment Design

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

Traveling Salesmen Problem (TSP)

Table 3: 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]

The target algorithm to solve TSP is a well-known Iterated Local Search (ILS) algorithm as implemented in [9] with 4 discrete parameters to be tuned as describe in Table 3(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 [29]). 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 3(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 iteration 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 4. Note that the "Standard" instantiation is the one proposed in [24].

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 other wise. The length may be 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 [20]. 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

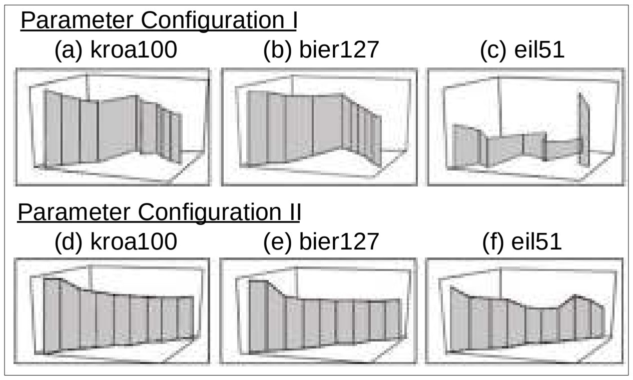

Figure 4: Search Trajectories of 3 TSP instances *kroa100*, *bier127* and *eli51* using two random parameter configuration

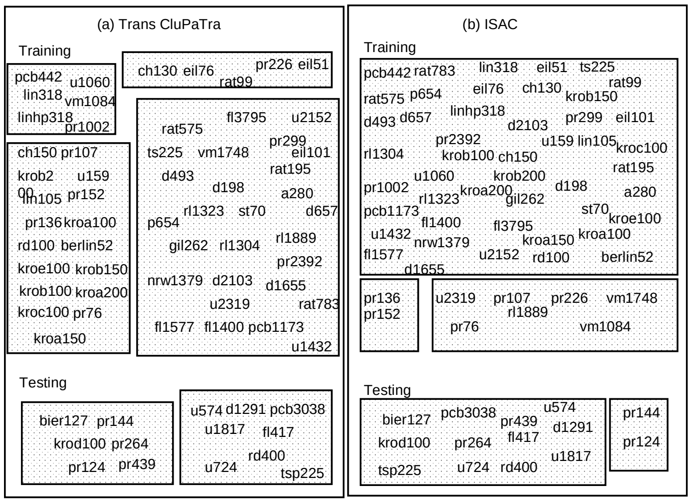
(a) Trans CluPaTra
Training
pcb442 u1060 lin318 vm1084 linhp318 pr1002
ch130 eil76 rat99 pr226 eil51
ch150 pr107 krob200 u159 lin105 pr152 pr136 kroa100 rd100 berlin52 kroe100 krob150 krob100 kroa200 kroc100 pr76 kroa150
rat575 fl3795 u2152 ts225 vm1748 pr299 eil101 rat195 d493 d198 a280 rl1323 st70 d657 p654 gil262 rl1304 rl1889 pr2392 nrw1379 d2103 d1655 u2319 rat783 fl1577 fl1400 pcb1173 u1432
Testing
bier127 pr144 krod100 pr264 pr124 pr439
u574 d1291 pcb3038 u1817 fl417 rd400 u724 tsp225
(b) ISAC
Training
pcb442 rat783 lin318 eil51 ts225 rat99 rat575 p654 eil76 ch130 krob150 d493 d657 linhp318 d2103 pr299 eil101 pr2392 u159 lin105 kroc100 rl1304 krob100 ch150 rat195 u1060 krob200 pr1002 d198 a280 kroa200 rl1323 gil262 pcb1173 st70 kroe100 fl1400 fl3795 u1432 kroa100 nrw1379 kroa150 fl1577 u2152 rd100 berlin52 d1655
pr136 pr152
u2319 pr107 pr226 vm1748 rl1889 pr76 vm1084
Testing
bier127 pcb3038 u574 pr439 d1291 krod100 pr264 fl417 u1817 tsp225 u724 rd400
pr144 pr124

Figure 6: TSP Cluster Result Comparison. (a) Trans CluPaTra; and (b) ISAC.

Table 4: 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.

TSP [35] and flow dominance and sparsity of flow matrix for QAP [37].

Tuning Setup

We chose to use ParamILS [19] 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 [10] over those instances and measured the average performance over all folds. To do 5-fold cross validation, we randomly divided the instances into 5 random groups and used 4 groups as training instances and 1 group as testing instances. We repeat the process 5 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 configuratior, we set a cutoff time of 10 seconds per run for the TSP target algorithm and 100 seconds for the QAP target algorithm. For each CluPaTra cluster, we 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 25 x 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.

5 Verification of Similarity Preservation

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

Recall that CluPaTra departs from existing approaches in parameter tuning in that it does not rely on problem-specific features; rather, it makes use of search trajectory patterns as a generic feature. As mentioned earlier, the rationale of using this feature is predicated on the relationship between fitness landscape and search trajectories [9], and the tight correlation between the fitness landscape and algorithm performance [32]. Since generating the entire fitness landscape for each instance is time consuming and generally impractical, we propose to use the search trajectory as a proxy for the fitness landscape. Granted that different parameter configurations may produce (very) different search trajectories for a given instance, our claim is that the *similarity* of search trajectories between instances is preserved across configurations.

In the following, we will justify this claim by providing a series of experimental observations.

First, we provide a visual intuition for similarity preservation across different parameter configurations. Fig. 4 shows the trajectories obtained by 10 consecutive moves of an Iterated Local Search (ILS) algorithm for three TSP instances, namely *kroa100*, *bier127* and *eil51* 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 represent the objective value. To layout the moves into a 2-dimensional xy plane, we calculate the distance between two solutions (e.g., number of different cities in TSP) and apply *"the spring model"* [8]. *"The spring model"* provides a heuristic for good layout where the Euclidean distance between 2 solutions in the xy plane is roughly proportional to their Hamming distance. In this example, we observe that for both configurations, *kroa100* and *bier127* exhibit very similar topology ((a) and (b), (d) and (e)), while *ei151* has a different topology compared to the similarity of *kroa100* and *bier127*.

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 (ILS and SA-TS, see section following). For this purpose, we verify on random pairs of instances

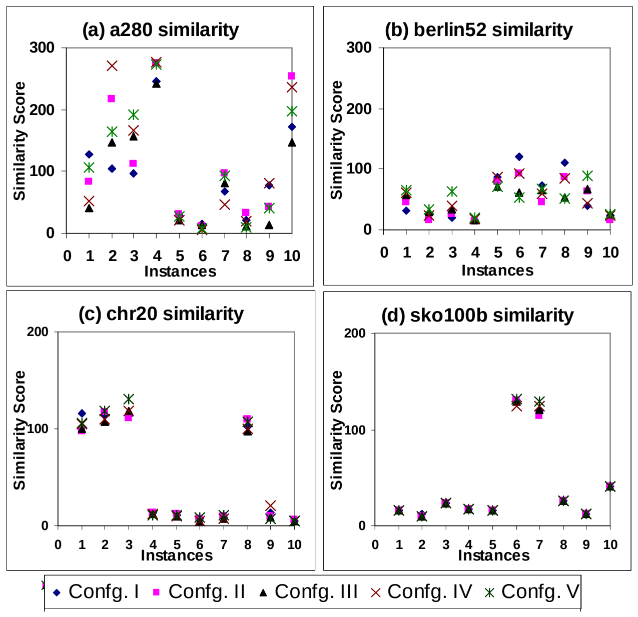

Figure 5: Search Trajectory Similarity Score between TSP (*a280, berlin51*) and QAP (*chr20a, sko100b*) instance and 10 other random instances using 5 Different Random Parameter Configurations.

Table 5: 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;

across different parameter configurations. We do the following: first, we randomly select 2 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 5 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 Fig. 5. 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 and mean of similarity values for the different parameter configurations are given in Table 5. We also calculate the coefficient of variance (CV) which predicted the likelihood difference in each similarity values [28]. The lower the CV, the higher the likelihood that there was no difference in similarity values. For most pairs, the coefficient of variance (CV) value is low (especially for QAP pairs), which means that the similarity score 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 2 instances is preserved across configurations. In other words, two instances whose trajectories are similar under one configuration will also likely be similar under another configuration. This similarity preservation property allows us

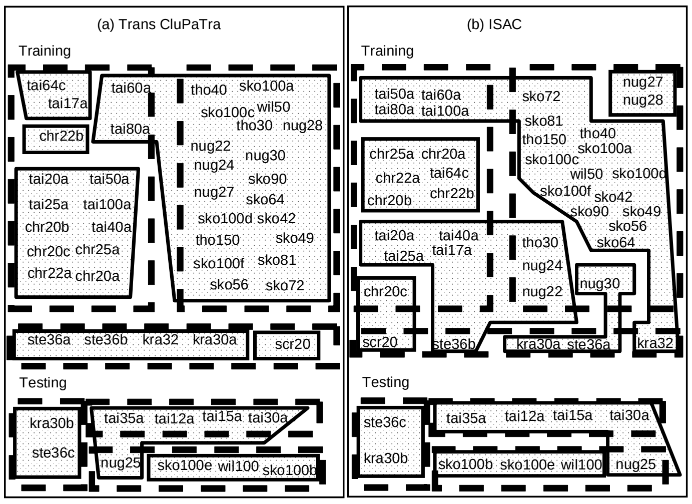
(a) Trans CluPaTra
(b) ISAC
Training
Testing
tai64c tai17a
chr22b
tai60a tai80a
tho40 sko100a sko100c wil50 tho30 nug28 nug22 nug30 nug24 sko90 nug27 sko64 sko100d sko42 tho150 sko49 sko100f sko81 sko56 sko72
tai20a tai50a tai25a tai100a chr20b tai40a chr20c chr25a chr22a chr20a
ste36a ste36b kra32 kra30a
scr20
kra30b ste36c
tai35a tai12a tai15a tai30a
nug25
sko100e wil100 sko100b
tai50a tai60a tai80a tai100a
sko72
nug27 nug28
sko81 tho150 tho40 sko100c sko100a wil50 sko100d sko100f sko42 sko90 sko49 sko56 sko64
chr25a chr20a chr22a tai64c chr20b chr22b
tai20a tai40a tai25a tai17a
tho30 nug24 nug22
nug30
chr20c
scr20 ste36b kra30a ste36a kra32
ste36c kra30b
tai35a tai12a tai15a tai30a
sko100b sko100e wil100 nug25

Figure 7: QAP Cluster Result Comparison. (a) Trans CluPaTra; and (b) ISAC.

to perform clustering of instances using an arbitrary parameter configuration.

6 Clusters Result

In this section, we report our cluster results and time performance for Traveling Salesmen Problem (TSP) and Quadratic Assignment Problem (QAP).

Table 6: Empirical Result

	TSP		QAP	
Technique	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

6.1 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 Fig. 6 for TSP and Fig. 7 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 the non-existence of the *ground-truth* classification for TSP benchmark instances, we cannot compute the cluster qualities ($\mathcal{Q}_{train}$ and $\mathcal{Q}_{test}$) directly instead it is inferred from the performance of the target algorithm which is describe in later section.

For QAP, we use the existing well-studied classification based on the distance and flow metrics [38] as the *ground-truth* classification. [38] divided the instances 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 limitation of the target algorithm (which is unable to solve groups (4) and (5) problems), we only use instances from groups (1), (2) and (3). The clusters from CluPaTra and ISAC are shown in solid boxes while the *ground-truth* classification (for QAP only) are shown in dashed boxes. Notice that the clustering by Trans CluPaTra is almost the same as the *ground-truth* classification. Furthermore, Trans CluPaTra constructs better clusters compared to the Standard CluPaTra and ISAC with respect to cluster quality metric ($\mathcal{Q}_{train}$ and $\mathcal{Q}_{test}$) as defined in Definition 2 and 3 as shown in Table 6(I). We observe that the cluster quality score for Trans CluPaTra is the highest compared to other CluPaTra instantiation and ISAC.

6.2 Time Performance

Two most time-consuming procedures in the training phase are those of calculating the similarity of trajectories and running the one-size-fits-all configurator (for each cluster). Evidently, different similarity calculation techniques require different computational budget for calculating the similarity. The Robust sequence alignment technique take almost four times longer than the basic sequence alignment. This happens because it requires more computation time to find partial-match symbols. For TSP, the average time needed to calculate the similarity of trajectories for Standard, Trans, Robust and Trans-Robust are 10, 12, 38 and 42 minutes respectively. For QAP, the average time needed to calculate the similarity of trajectories for Standard, Trans, Robust and Trans-Robust are 8, 9, 32 and 35 minutes respectively.

The other procedures in the training phase are relatively fast. Even with different techniques, all of them require less than 1 minute to complete. The average total time (in hour) needed to run the overall process in training phase for each fold is shown in Table 6(II).

7 Parameter Tuning Result

In this section, we report our parameter tuning result for Traveling Salesmen Problem (TSP) and Quadratic Assignment Problem (QAP).

7.1 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 4. Table 6(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 Fig. 7 using parameter configurations from each cluster and show the result in Table 7. From the table we observe that each testing instance, except for tai35a, has the best performance using parameter configuration from the most similar cluster.

Table 7: Testing Instances Performance using Different Cluster's Parameter Configuration

		Parameter Configuration for each Cluster					
Instance	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

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 8: Parameters for SA on Industrial Case Study

Parameter	Description	Range
maxSuccess	max successes number	[100, 1000]
maxTries	max tries	[100, 1000]
maxComp	max solutions generated	[1000, 50000]
maxConsReject	max consecutive rejections	[100, 1000]
maxChangeG	max change in a variable value	[100, 1000]
maxTriesG	max tries to generate a feasible solution	[100, 1000]
coolingFactor	factor to reduce the temp	[0.5, 1]
oracleStrictness	the strictness of the oracle function	[0, 100]

7.2 Comparison on Clustering Method

Next, we analyze 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 6(IV) shows that AGNES performs slightly better than k-medoids even though it is not statistically significant.

7.3 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 fulfill specific target services levels. The target algorithm is a *Simulated Annealing* (SA) algorithm [6] which has 8 parameters that used to control SA behavior as described in Table 8.

We apply CluPaTra to 50 sample instances, with 25 randomly picked instances as training instances and the remaining 25 as testing instances. We set cutoff times of 500 seconds 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 x n times, where n is the number of instances in the cluster. In Table 9, we present the average of percentage deviation value (Definition 4). 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.

Table 9: Industrial Case Study Result

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

+ = statistically significant against Default Configuration
* = statistically significant against ParamILS Configuration

8 Discussion and Future Direction

As shown from the experimental results, the CluPaTra framework yields a significant improvement in performance compared with the pure one-size-fits-all configurator ParamILS, under different instantiations. We also observe that Trans, Robust and Trans-Robust instantiations of CluPaTra perform significantly superior to the standard CluPaTra instantiation. Those three instantiations yield results that are statistically equivalent to one another but the latter two require higher computational budget compared to Trans. Based on this result, we confirm the result in [24] that dividing the instances into clusters using CluPaTra, especially using the Trans. Configuration, before running one-size-fits-all configurator provides better parameter configuration for each instance and significantly improves the performance.

The effect of different clustering methods is also evaluated by two well-studied clustering approaches, AGNES and k-medoids. Our result shows that there is no significant difference with these two clustering methods, this may indicate that the underlying clustering method does not have a substantial effect on CluPaTra. We will explore this issue further in future work.

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. For all TSP instances and several QAP instances, we use the known global optimal value from TSPLib and QAPLib respectively while for other QAP instances, we use best known value from QAPLib. For industrial case study instances, we use best known solution used by our industrial partner. 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 [24], we also show that CluPaTra is statistically equivalent with the existing and well-known QAP instances classification [38]. We expand those results by comparing CluPaTra with ISAC, an instance-specific algorithm configuration which use problem specific features, and show that CluPaTra is significantly better then ISAC. In future work, we would like to explore this further for ISAC using more complex specific-features.

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. In a world where new local search algorithms are being designed for solving large complex optimization problems, the power of our approach rests in its ability to produce effective and instance-specific parameter settings automatically in a computationally efficient manner, rather than to rely on the tedious and mostly ad hoc manual tuning. As future direction, wee would like to extend CluPaTra for other purposes such as algorithm selection or hyperheuristic.

9 Related Work

9.1 Instance Features

Recently, there has been increasing interest in finding the features of instances that can provide an insight on what make the problem hard and its relationship with algorithm performance. We divide the features into two, namely: problem-specific and problem-independent features. Various problem-specific features have been explored in the literature, most of these are extracted based on extensive study on the problem knowledge domain. Example of problem-specific features are flow dominance for QAP [11, 37, 38, 40] and population correlation structure and constraint slackness for Knapsack Problem [12, 33]. A comprehensive study on problem-specific features of six combinatorial optimization problem is reviewed in [36].

On problem-independent features, one successful approach is to characterize the instances based on its search space (fitness landscape analysis) [2, 14, 32, 39]. A fitness landscape is a defined by a set of solutions ω, a fitness function ϕ and a neighborhood relation N_k over the set of ω [14, 36]. It can be imagined as mountainous regions where each points represent the possible solution in ω with the fitness function ϕ as its altitude and the neighborhood relation N_k as its distance [7]. Examples of features extracted from fitness landscape are fitness distance correlation (FCD) [32, 14] and ruggedness coefficient [2, 14]. A detail review on fitness landscape and its features can be found in [14].

9.2 Automated Parameter Tuning

A wide variety of strategies for automated parameter configuration have been explored in the literature. Some of these were proposed for a specific target algorithm than a generic algorithm. 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* [4], *ParamILS* [19], *RCS* [23] and *GGA* [3]; while in model-based ones are *CALIBRA* [1], *SPO+* [18] and *SMAC* [17]. A good review of three recent automated parameter tuning methods can be found in [13].

All of the above studies focus on finding the best algorithm configuration for an entire set (or distribution) of problem instances. Such approaches may not be suitable for large and diverse set of problem instances. There have been approaches that attempted to select the best parameter configuration on a per-instance basis [15, 16, 20, 27, 31]. Approaches like [15, 16, 31] use regression model to construct a model based on the instance's features that will determine the configurator's strategy, while other like [20, 24] use clustering to divide the instances and find a good parameter configuration on the cluster created. Unfortunately, these approaches tend to be problem-specific because they make use of problem-specific features. As an example, ISAC [20] uses problem specific features to identify the characteristic of instances of a particular problem; for instance, it uses 24 specific features for Set Covering from [26].

10 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-viz 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

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

[2] E. Angel and V. Zissimopoulos. On the hardness of the quadratic assignment problem with metaheuristics. *Journal of Heuristics*, 8(4):399–414, 2002.

[3] C. Ansótegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *15th international Conference on Principles and Practice of Constraint Programming*, pages 142–157, 2009.

[4] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Genetic and Evolutionary Computation Conference*, pages 11–18, 2002.

[5] C. Blum and A. Roli. Metaheuristcs in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[6] A. Gunawan, H. C. Lau, and E. Wong. Real-world parameter tuning using factorial design with parameter decomposition. In *MIC 2011: The IX Metaheuristics International Conference*, 2011.

[7] S. Halim. *An Integrated White+Black Box Approach for Designing and Tuning Stochastic Local Search Algorithms*. Ph.D. thesis, National University of Singapore, Singapore, 2009.

[8] S. Halim, Y. Yap, and H.C. Lau. Viz: A visual analysis suite for explaining local search behavior. In *19th ACM symposium on User Interface Software and Technology*, pages 57–66, 2006.

[9] S. Halim, Y. Yap, and H.C. Lau. An integrated white+black box approach for designing and tuning stochastic local search. In *LNCS: 13th International Conference on Principles and Practice of Constraint Programming*, pages 332–347, 2007.

[10] J. Han and M. Kamber. *Data Mining: Concept and Techniques, 2nd Edition*. Morgan Kaufman, San Francisco, 2006.

[11] W. Herroelen and A. Van Gils. On the use of flow dominance in complexity measures for facility layout problems. *International Journal of Production Research*, 23:97–108, 1985.

[12] R.R. Hill and C.H. Reilly. The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance. *Management Science*, 46(2):302–317, 2000.

[13] H.H. Hoos. Automated algorithm configuration and parameter tuning. In Youssef Hamadi, Eric Monfroy, and Frdric Saubion, editors, *Autonomous Search*, pages 37–72. Springer, 2012.

[14] H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundation and Application*. Morgan Kaufman, San Francisco, 2004.

[15] F. Hutter and Y. Hamadi. Parameter adjustment based on performance prediction: Towards an instance-aware problem solver. In *Technical Report*. Microsoft Research, 2005.

[16] F. Hutter, Y. Hamadi, H.H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *LNCS: Principles and Practice of Constraint Programming 2006*, pages 213–228, 2006.

[17] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LNCS: 5nd Learning and Intelligent OptimizatioN Conference*, 2011.

[18] F. Hutter, H.H. Hoos, K. Leyton-Brown, and K. Murphy. Time-bounded sequential parameter optimization. In *LNCS: 4nd Learning and Intelligent OptimizatioN Conference*, 2010.

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

[20] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. Isac-instance-specific algorithm configuration. In *19th European Conference on Artificial Intelligence*, 2010.

[21] J. Kaufman and P. J. Rousseeuw. Clustering by means of medoids. *Statistical Data Analysis Based on the L Norm*.

[22] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1990.

[23] H.C. Lau and F. Xiao. Enhancing the speed and accuracy of automated parameter tuning in heuristic design. In *8th Metaheuristics International Conference*, 2009.

[24] Lindawati, H.C. Lau, and D. Lo. Instance-based parameter tuning via search trajectory similarity clustering. In *LNCS: 5nd Learning and Intelligent OptimizatioN Conference*, 2011.

[25] W. Macready and D. Wolpert. What makes an optimization problem hard. *Complexity*, 5:40–46, 1996.

[26] Y. Malitsky and M. Sellmann. Stochastic offline programming. In *IEEE International Conference on Tools with Artificial Intelligence*, 2009.

[27] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *14th National Conference on Artifical Intelligence*, page 321326. AAAI Press, 1997.

[28] D.C. Montgomery and G.C. Runger. *Applied Statistics and Probability for Engineers 2nd Edition*. John Wiley & Son, 1999.

[29] K.M. Ng, A. Gunawan, and K.L. Poh. A hybrid algorithm for the quadratic assignment problem. In *International Conf. on Scientific Computing*, pages 14–17, 2008.

[30] R.T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *20th International Conference on Very Large Databases*, pages 144–155, 1994.

[31] D.J. Patterson and H. Kautz. Auto-walksat: A self-tuning implementation of walksat. *Electronic Notes in Discrete Mathematics*, 9:360–368, 2001.

[32] C.R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations Research*, 86(1):473–490, 1999.

[33] C.H. Reilly. Synthetic optimization problem generation: Show us the correlations! *INFORMS Journal on Computing*, 21:458–467, 2009.

[34] S. Salvador and P. Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 576–584, 2004.

[35] K. Smith-Miles, J. Hemert, and X.Y. Lim. Understanding tsp difficulty by learning from evolved instances. In *LNCS: 4th Learning and Intelligent OptimizatioN Conference*, pages 266–280, 2010.

[36] K. Smith-Miles and L. Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computer and Operations Research*, 39:875–889, 2012.

[37] T. Stützle and S. Fernandes. New benchmark instances for the qap and the experimental analysis of algorithms. In *LNCS: Evolutionary Computation In Combinatorial Optimization*, 2004.

[38] É.D. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3(2):87–105, 1995.

[39] J. Tavares, FB. Pereira, and E. Costa. Multidimensional knapsack problem: a fitness landscape analysis. *IEEE Trans Syst Man Cybern B Cybern*, 38(3):604–616, 2008.

[40] T.E. Vollmann and E.S. Buffa. The facilities layout problem in perspective. *Management Science*, 12(10):450–468, 1966.

[41] L. Xu, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.