Local Search and the Traveling Salesman Problem: A Feature-Based Characterization of Problem Hardness

Olaf Mersmann[1], Bernd Bischl[1], Jakob Bossek[1], Heike Trautmann[1], Markus Wagner[2], and Frank Neumann[2]

[1] Statistics Faculty, TU Dortmund University, Germany
{olafm,bischl,bossek,trautmann}@statistik.tu-dortmund.de
[2] School of Computer Science, The University of Adelaide, Australia
{markus.wagner,frank.neumann}@adelaide.edu.au

Abstract. With this paper we contribute to the understanding of the success of 2-opt based local search algorithms for solving the traveling salesman problem (TSP). Although 2-opt is widely used in practice, it is hard to understand its success from a theoretical perspective. We take a statistical approach and examine the features of TSP instances that make the problem either hard or easy to solve. As a measure of problem difficulty for 2-opt we use the approximation ratio that it achieves on a given instance. Our investigations point out important features that make TSP instances hard or easy to be approximated by 2-opt.

Keywords: TSP, 2-opt, Classification, Feature Selection, MARS

1 Introduction

Metaheuristic algorithms such as local search, simulated annealing, evolutionary algorithms, and ant colony optimization have produced good results for a wide range of NP-hard combinatorial optimization problems. One of the most famous NP-hard combinatorial optimization problems is the traveling salesman problem (TSP). Given a set of N cities and positive distances d_{ij} to travel from city i to city j, $1 \leq i, j \leq N$ and $i \neq j$, the task is to compute a tour of minimal traveled distance that visits each city exactly once and returns to the origin.

The perhaps simplest NP-hard subclass of TSP is the Euclidean TSP where the cities are points in the Euclidean plane and the distances are the Euclidean distances between them. We will focus on the Euclidean TSP. It is well known that there is a polynomial time approximation scheme (PTAS) for this problem [2]. However, this algorithm is very complicated and hard to implement.

Many heuristic approaches have been proposed for the TSP. Often local search methods are the preferred methods used in practice. The most successful algorithms rely on the well-known 2-opt operator, which removes two edges from a current tour and connects the resulting two parts by two other edges such that a different tour is obtained [9]. Despite the success of these algorithms for a wide range of TSP instances, it is still hard to understand 2-opt from a theoretical point of view.

Theoretical studies regarding 2-opt have investigated the approximation behavior as well as the time to reach a local optimum. Chandra et al [4] have studied the worst-case approximation ratio that 2-opt achieves for different classes of TSP instances. Furthermore, they investigated the time that a local search algorithm based on 2-opt needs to reach a locally optimal solution. Englert et al. [6] have shown that there are even instances for the Euclidean TSP where a deterministic local search algorithm based on 2-opt would take exponential time to find a local optimal solution. Furthermore, they have shown polynomial bounds on the expected number of steps until 2-opt reaches a local optimum for random Euclidean instances and proved that such a local optimum gives a good approximation for the Euclidean TSP. These results also transfer to simple ant colony optimization algorithms as shown in [12]. Most previously mentioned investigations have in common that they either investigate the worst local optimum and compare it to a global optimal solution or investigate the worst case time that such an algorithm needs to reach a local optimal solution. Although these studies provide interesting insights into the structure of TSP instances they do not provide much insights into what is actually going on in the application of 2-opt based algorithms. In almost all cases the results obtained by 2-opt are much better than the actual worst-case guarantees given in these papers. These motivates the studies carried out in this paper, which aim to get further insights into the search behavior of 2-opt and to characterize hard and easy TSP instances for 2-opt.

We take a statistical meta-learning approach to gain new insights into which properties of a TSP instance make it difficult or easy to solve for 2-opt. Analyzing different features of TSP instances and their correlation we point out how they influence the search behavior of local search algorithms based on 2-opt. To generate hard or easy instances for the TSP we use an evolutionary algorithm approach similar to the one of [21]. However, instead of defining hardness by the number of 2-opt steps to reach a local optimum, we define hardness by the approximation ratio that such an algorithm achieves for a given TSP instance compared to the optimum solution. This is motivated by classical algorithmic studies for the TSP problem in the field of approximation algorithms. Having generated instances that lead to a bad or good approximation ratio, the features of these instances are analyzed and classification rules are derived, which predict the type of an instance (easy, hard) based on its feature levels. In addition, instances of moderate difficulty in between the two extreme classes are generated by transferring hard into easy instances based on convex combinations of both instances, denoted as morphing. Systematic changes of the feature levels along this "path" are identified and used for a feature based prediction of the difficulty of a TSP instance for 2-opt-based local search algorithms.

The structure of the rest of this paper is as follows. In Section 2, we give an overview about different TSP solvers, features to characterize TSP instances and indicators that reflect the difficulty of an instance for a given solver. Section 3 introduces an evolutionary algorithm for evolving TSP instances that are hard or easy to approximate and carries out a feature based analysis of the hardness

of TSP instances. Finally, we finish with concluding remarks and an outlook on further research perspectives in Section 4.

2 Local Search and The Traveling Salesman Problem

Local search algorithms are frequently used to tackle the TSP problem. They iteratively improve the current solution by searching for a better one in its predefined neighborhood. The algorithm stops when there is no better solution in the given neighborhood or if a certain number of iterations has been reached.

Historically, 2-opt [5] was one of the first successful algorithms to solve larger TSP instances. It is a local search algorithm whose neighbourhood is defined by the removal of two edges from the current tour. The resulting two parts of the tour are reconnected by two other edges to obtain a new solution. Later on, this idea has been extended to 3-opt [14] where three connections in a tour are first deleted, and then the best possible reconnection of the network is taken as a new solution. Lin and Kernighan [13] extended the idea to more complex neighbourhoods by making the number of performed 2-opt and 3-opt steps adaptive. Nowadays, variants of these seminal algorithms represent the state-of-the-art in heuristic TSP optimizers.

Among others, memetic algorithms and subpath ejection chain procedures have shown to be competitive alternatives, with hybrid approaches still being investigated today. In the bio-inspired memetic algorithms for the TSP problem (see [16] for an overview) information about subtours is combined to form new tours via 'crossover operators'. Additionally, tours are modified via 'mutation operators', to introduce new subtours. The idea behind the subpath ejection chain procedures is that in a first step a dislocation is created that requires further change. In subsequent steps, the task is to restore the system. It has been shown that the neighbourhoods investigated by the ejection chain procedures form supersets of those generated by the Lin-Kernighan heuristic [8].

In contrast to the above-mentioned iterative and heuristic algorithms, Concorde [1] is an exact algorithm that has been successfully applied to TSP instances with up to 85,900 vertices. It follows a branch-and-cut scheme [17], embedding the cutting-plane algorithm within a branch-and-bound search. The branching steps create a search tree, with the original problem as the root node. By traversing the tree it is possible to establish that the leafs correspond to a set of subproblems that include every tour for our TSP.

2.1 Characterization of TSP Instances

The theoretical assessment of problem difficulty of a TSP instance at hand a-priori to optimization is usually hard if not impossible. Thus, research has focussed on deriving and extracting problem properties, which characterize and relate to the hardness of TSP instances (e.g. [21, 10, 20]). We refer to these properties as features in the following and provide an overview subsequently. Features

that are based on knowledge of the optimal tour [22, 11] cannot be used to characterize an instance a priori to optimization. They are not relevant in the context of this paper and thus are not discussed in detail.

An intuitive and considered feature is the number of nodes N of the TSP instance [21, 10, 20]). Kanda et al. [10] assume the number of edges to be important as well and introduce a set of features that are based on summary statistics of the edge cost distribution. We will use edge cost or edge weight as a synonym of distance between nodes in the following. The lowest, highest, mean and median edge cost are considered as well as the respective standard deviation and the sum of N edges with lowest edge cost values. Furthermore, the quantity of edges with costs lower than the mean or median edge cost is taken into account. Additional features are the number of modes of the edge cost distribution and related features such as the frequency of the modes and the mean of the modal values.

Smith-Miles et al. [20, 21] list features that assume that the existence and number of node clusters affect the performance of TSP solvers. Derived features are the cluster ratio, i.e. the number of clusters divided by N, and the mean distances to the cluster centers. Uniformity of an instance is further reflected by the minimum, maximum, standard deviation and the coefficient of variation of the normalized nearest-neighbor distances (nnd) of each node. The outlier ratio, i.e. the number of outliers divided by N, and the number of nodes near the edge of the plane are additionally considered. The centroid together with the mean distance from the nodes to the centroid and the bounding box of the nodes reflect the 'spread' of the instance on the plane. The feature list is completed by the fraction of distinct distances, i.e. different distance levels, and the standard deviation of the distance matrix.

Note that in order to allow for a fair comparison of features across instances of different sizes N the features have to be normalized appropriately. This means that all distances and edge costs have to be divided by their total sum. Analogously, all quantities have to be expressed relatively to the corresponding maximum quantity. Ideally, all instances should be normalized to the domain $[0, 1]^2$ to get rid of scaling issues.

We will use the approximation ratio that an algorithm achieves for a given instance as the optimization accuracy. The approximation ratio is given by the relative error of the tour length resulting from 2-opt compared to the optimal tour length and is a classical measure in the field of approximation algorithms [23]. Based on the approximation ratio that the 2-opt algorithm achieves, we will classify TSP instances either as easy or hard. Afterwards, we will analyze the features of hard and easy instances.

3 Analysis of TSP Problem Difficulty

In this section, we analyze easy and hard instances for the TSP. We start by describing an evolutionary algorithm that we used to generate easy and hard instances. Later on, we characterize these instances by the different features

Algorithm 1 Generate a random TSP instance.

```
function randomInstance(size)
    for i = 1 → size do
        instance[i, 1] ← U(0, 1)        ▷ Uniform random number between 0 and 1
        instance[i, 2] ← U(0, 1)        ▷ Uniform random number between 0 and 1
    end for
    return instance
end function
```

Algorithm 2 EA for evolving problem easy and hard TSP instances

```
function EA(popSize, instSize, generations, time_limit, digits, repetitions, type)
    poolSize ← ⌊popSize/2⌋
    for i = 1 → popSize do
        population[i] ← randomInstance(instSize)
    end for
    for generation = 1 → generations do
        for k = 1 → popSize do
            fitness[k] ← computeFitness(population[k], repetitions)
        end for
        matingPool ← createMatingPool(poolSize, population, fitness)
        nextPopulation[1] ← population[bestOf(fitness)]        ▷ 1-elitism
        for k = 2 → popSize do
            parent1 ← randomElement(population)
            parent2 ← randomElement(population)
            offspring ← uniformMutation(uniformCrossover(parent1, parent2))
            nextPopulation[k] ← round(normalMutation(offspring))
        end for
        population ← nextPopulation
        if over time limit time_limit then
            return population
        end if
    end for
end function
```

that we analyzed and point out which features make a TSP instance difficult to be solved by 2-opt.

3.1 EA-Based Generation of Easy and Hard TSP Instances

As the aim is to identify the features that are crucial for predicting the hardness of instances for the 2-opt heuristic, a representative set of instances is required which consists of a wide range of difficulties. It turned out that the construction of such a set is not an easy task. The generation of instances in a random manner did not provide a sufficient spread with respect to the instance hardness. The same is true for instances contained in the TSPLIB [18] of moderate size, i.e. lower than 1000 nodes, for which, in addition, the number of instances is not high enough to provide an adequate data basis. Higher instance sizes were excluded

Algorithm 3 Compute Fitness

```
function ComputeFitness(instance, repetitions)
    optimalTourLength ← concorde(instance)
    for j ← 1, repetitions do
        twoOptTourLengths[j] ← twoOpt(instance)        ▷ Two Opt Tour length
    end for
    return mean(twoOptTourLengths) / optimalTourLenght
end function
```

Algorithm 4 Mating pool creation

```
function createMatingPool(poolSize, population, fitness)
    for i = 1 → poolSize do
        matingPool[i]
        ← betterOf(randomElement(population), randomElement(population))
    end for
    return matingPool
end function
```

due to the large computational effort required for their analysis, especially the computation of the optimal tours.

Therefore, two sets of instances are constructed in the $[0, 1]$-plane, which focus on reflecting the extreme levels of difficulty. An evolutionary algorithm (EA) is used for this purpose (see Algs. 1 - 4 for a description), which can be parameterized such that its aim is to evolve instances that are either as easy or as hard as possible for a given instance size. The approach is conceptually similar to [21] but focusses on approximation quality rather than on the number of swaps. Since some features depend on equal distances between the cities, we opted to implement a rounding scheme in the mutation step to force all cities to lie on a predefined grid. and consists of a different mutation strategy. Initial studies also showed, that a second mutation strategy was necesarry. "Local mutation" was achieved by adding a small normal pertubation to the location, "global mutation" was performed by replacing each coordinate of the city with a new uniform random value. This later step was performed with a very low probability. All parameters are given at the end of this section.

The fitness function to be optimized is chosen as the approximation quality of 2-opt, estimated by the arithmetic mean of the tour lengths of a fixed number of 2-opt runs, on a given instance divided by the optimal tour length which is calculated using Concorde [1]. In general other summary statistics instead of the arithmetic mean could be used as well such as the maximum or minimum approximation quality achieved. Note that randomness is only induced by varying the initial tour whereas the 2-opt algorithm is deterministic in always choosing the edge replacement resulting in the highest reduction of the current tour length. Depending on the type of instance that is desired, the betterOf and bestOf operators are either chosen to minimze or maximize the approximation quality.

We use a 1-elitism strategy such that only the individual with the current best fitness value survives and will be contained in the next population. The population is completed by iteratively choosing two parents from the mating pool, applying uniform crossover, uniform and normal mutation and adding the offspring to the population. This procedure is repeated until the population size is reached. Two sequential mutation strategies enable small local as well as global structural changes of the offspring resulting from the crossover operation.

In the experiments 100 instances each for the two instance classes (easy, hard) with a fixed instance size of 100 are generated. The remaining parameters are set as follows: $popSize = 30$, $generations = 1500$, $time\_limit = 22h$, $uniformMutationRate = 0.001$, $normalMutationRate = 0.01$, $digits = 2$, and the standard deviation of the normal distribution used in the $normalMutation$ step equals $normalMutationSd = 0.025$. The parameter levels were chosen based on initial experiments. However, a matter of future research will be a systematic tuning of the EA parameters in order to check if the results can be significantly improved. The number of 2-opt repetitions for calculating the approximation quality is set to 500.

3.2 Characterization of the Generated Instances

The average approximation qualities and respective standard deviations of the evolved easy and hard instances are (1.032 ± 0.0041) and (1.177 ± 0.0044), i.e. for the easy instances the average tour length of the 2-opt is about three percent higher than the optimal tour. The corresponding value for the hard instances is 18 percent, which results in a sufficiently high performance discrepancy between the two evolved sets.

In Figure 1 three EA generated instances of both classes are shown together with the corresponding optimal tours computed by Concorde. The main visual observations can be summarized as follows:

- The distances of the cities on the optimal tour appear to be more uniform for the hard instances than it is the case for the easy ones. This is supported by Figure 2 that shows boxplots of the standard deviations of the edge weights on the optimal tour. There we see that respective standard deviations of the easy instances are roughly twice as high than for the hard instances.
- The optimal tours of the hard instances are more similar to a "U-shape" whereas the optimal tours of the easy instances rather match an "X-shape".
- It seems that the easy instances consist of many small clusters of cities whereas this is not the case for the hard instances up to the same extent.

3.3 Feature-Based Prediction of TSP Problem Hardness

A decision tree [3] is used to differentiate between the two instance classes. This leads to the following classification rule, which is based on the coefficient of variation of the nearest neighbor distances (CVND) and the highest edge cost value (HEC):

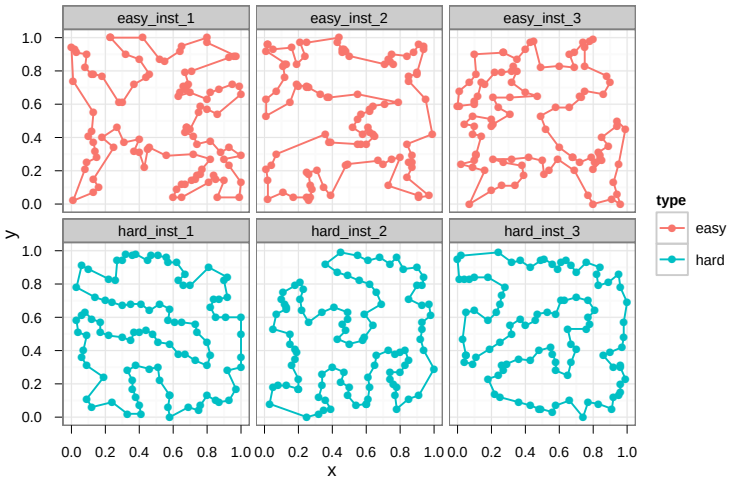

Fig. 1. Examples of the evolved instances of both types (easy, hard) including the optimal tours computed by Concorde.

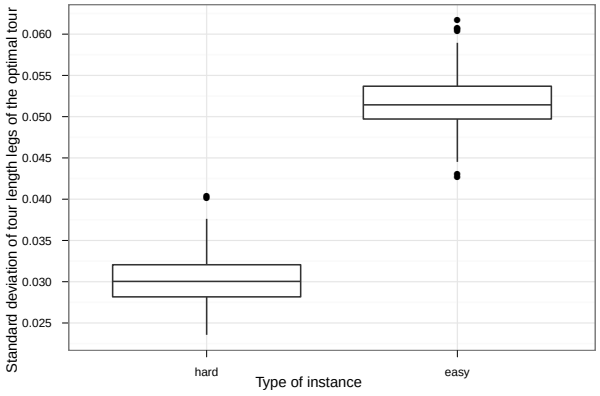

Fig. 2. Boxplots of the standard deviations of the tour length legs of the optimal tour, both for the evolved easy and hard instances.

```
coefficient_of_variation_of_nnds>=0.5167739 → easy
coefficient_of_variation_of_nnds< 0.5167739
  ↪ highest_edge_cost>=0.000485 → easy
  ↪ highest_edge_cost< 0.000485 → hard
```

The ten-fold cross-validated error rate is very low, i.e. equals 3.02% so that an almost perfect classification of instances into the two classes based on only

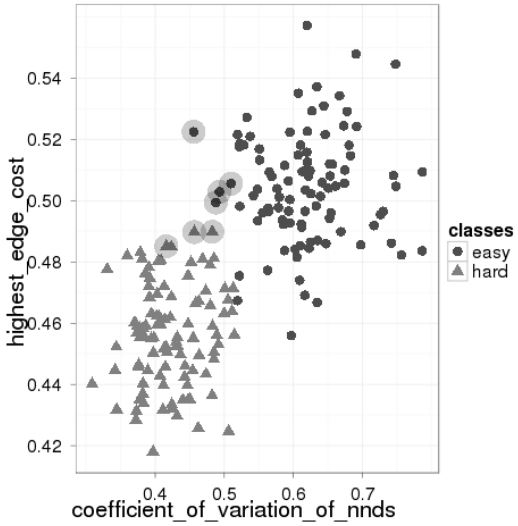

Fig. 3. Scatterplot of the features CVND and HEC for all evolved instances. Feature classes predicted by the decision tree are marked by different colors. Incorrectly classified instances during cross-validation are labeled by a grey circle.

two features is possible. Basically, the classification relies on the single feature CVND, which is shown in Figure 3. The two-dimensional feature space of the 200 instances is visualized using the features selected by the decision tree and labels the instances based on the presented classification rule. Incorrectly classified instances are marked by a grey circle.

It can be seen that the key feature for classifying the evolved instances into the two classes is CVND, which is perfectly in line with the exploratory analysis in Section 3.2. As the nearest neighbor of a node is the most likely candidate to be chosen in the course of the construction of the optimal tour, the CVND is highly correlated with the standard deviation of the distances on the optimal tour. In addition, the interpretation of the subrule regarding the feature HEC allows the same interpretation such that with increasing HEC value the less likely a uniform distribution of the edge weights becomes.

Classification rules generated for classifying easy and hard instances w.r.t. the Chained Lin-Kernighan (CLK) and Lin-Kernighan with Cluster Compensation (LKCC) algorithms in [21] also incorporate the feature CVND but the rule points into the opposite direction for CLK. Low CVND values characterize instances that are easy to solve for CLK. Although in [21] the approximation quality is measured by the number of swaps rather than by the resulting tour length relative to the optimal one, this is an interesting observation. In contrast, the results for LKCC are similar to the 2-opt rule, i.e. instances are classified as easy for high CVND values in combination with a low number of distinct distances.

3.4 Morphing Hard into Easy Instances

We are now in the position to separate easy and hard instances with the classification rule presented in Section 3.3. In this section, instances in between, i.e. of moderate difficulty, are considered as well. Starting from the result in [6] that a hard TSP instance can be transformed into an easy one by slight variation of the node locations, we studied the 'transition' of hard to easy instances by morphing a single hard instance I_H into an easy instance I_E by a convex combination of the points of both instances, which generates an instance I_{new} in between the original ones based on random point matching, i.e.

$$I_{new} = \alpha \cdot I_H + (1 - \alpha) \cdot I_E \quad \text{with} \quad \alpha \in [0, 1].$$

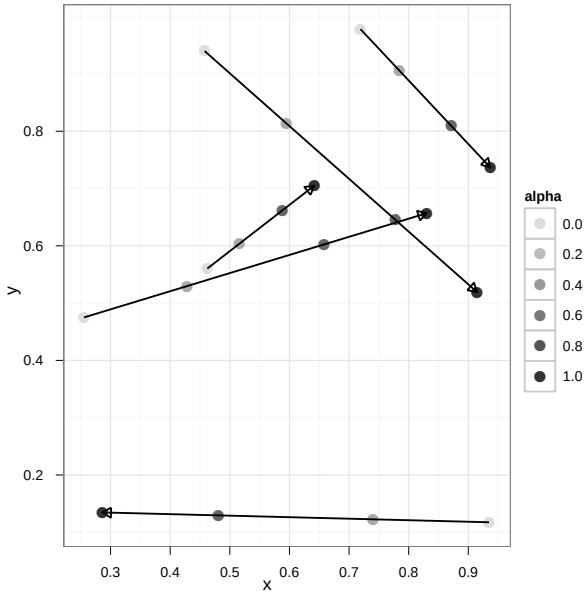

Fig. 4. Example: Morphing of one instance into a different instance for different α-levels of the convex combination. Nodes of the same color belong to the same instance.

An example of morphing is shown in Figure 4. The morphing strategy is applied to all possible combinations of single hard and easy instances of the two evolved instance sets using 51 levels of α ($\alpha \in \{0, 0.02, 0.04, ..., 1\}$). Each generated instance is characterized by the levels of the features discussed in Section 2.1. Thus, the changes of the feature levels with increasing α can be studied which is of interest as it should lead to an understanding of the influence of the different features on the approximation quality.

Figure 5 shows the approximation quality for the instances of all 10 000 morphing sequences for the various α levels in the bottom subfigure. Starting from a hard instance on the left side of the plot ($\alpha = 0$) the findings of [6] are confirmed. The approximation quality of 2-opt quickly increases, i.e. the value of *approx* decreases, with slight increases of α. Interestingly, roughly for $\approx 0.1 < \alpha < 0.9$ the approximation quality is quite stable, whereas it nonlinearly increases for $\alpha >= 0.9$. Additionally, the feature levels of the generated instances are visualized.

On the one hand features exist that do not show any systematic relationship with the approximation quality, e.g. all features related to the modes, lowest_edge_cost or cities_near_edge_ratio_one_percent. Other features exhibit a convex or concave shape obtaining similar values for the extreme α-levels and minimum resp. maximum value at $\alpha \approx 0.5$, e.g. mean_distance_to_cluster_centroids_0.1, cities_near_edge_ratio_five_percent or edges_lower_than_average_cost. The ratio of distinct_distances is only low for $\alpha \in \{0, 1\}$ and rather constant on a much higher level in between. A systematic nonlinear relationship can be detected for the features on which the classification rule is based, i.e. the CVND and HEC as well as for the mean_of_normalized_nnds and sum_of_lowest_edge_values.

In order to get a more accurate picture of the relationship between the approximation quality and the features a Multivariate Adaptive Regression Splines (MARS) [7] model is constructed in order to directly predict the expected approximation quality of 2-opt on a given instance based on the candidate features. Only a subset of the data is considered for the analysis (all morphed instances with $\alpha \in \{0, 0.2, ..., 1\}$), in order to adequately balance the data w.r.t. the various levels of 2-opt approximation quality. Had all 51 α-levels been used, the moderately difficult instances would have been massively overrepresented compared to the easy and hard instances.

We used a MARS model with interaction effects up to the second degree. Although this model class consists of an internal variable selection strategy a forward selection process together with a threefold cross-validation is applied in order to systematically focus on model validation and minimizing the root mean squared error (RMSE) of the prediction. Starting from an empty model, successively the feature that maximally reduces the RMSE is added to the existing model until the RMSE improvement falls below the threshold $t = 0.0005$. The results of the modeling procedure are shown in Table 1. The final root mean squared error is 0.0113.

Feature list	**RMSE**
empty model	0.0484
+ coefficient of variation of nnds (CVND)	0.0246
+ distinct distances (DD)	0.0163
+ highest edge cost (HEC)	0.0119
+ sum of lowest edge values (SLEV)	0.0113

Table 1. Results of the MARS model.

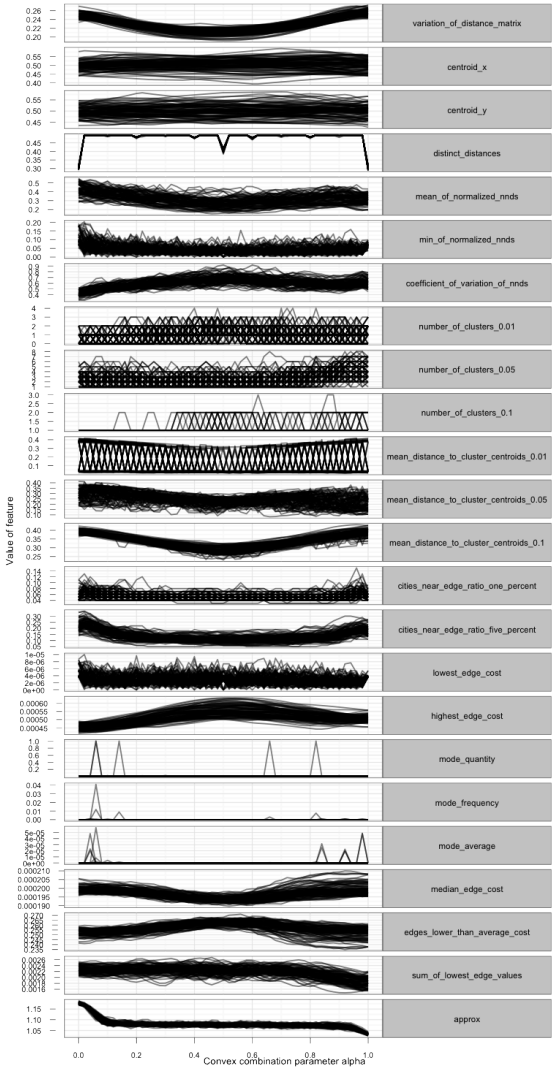

Fig. 5. Approximation quality and feature values for different α levels of all conducted morphing experiments. The annotations "_0.01","_0.05" and "_0.1" identify different levels of the reachability distance as a parameter of GDBSCAN [19] used for clustering.

The main and interaction effects of the model are visualized in Figure 6. Analogously to the classification rule the CVND is a key feature in predicting the approximation quality. From the plots, it is obvious that high values of the approximation ratio only occur for very low CVND values. However, the main effect in this case does not reflect the classification rule generated before. The

HEC is only part of an interaction with the remaining features. The problem hardness tends to be higher for low HEC values combined with high DD values and low CVND values. In addition, problem hardness decreases with lower SLEV values. The selection of the DD feature seems to be somewhat arbitrary in that the slope of the effect line could just as well have been negative from visual analysis. Summarizing, the interpretation of the model results is not straightforward but nevertheless a quite accurate prediction of 2-opt approximation quality on the considered instances is achieved.

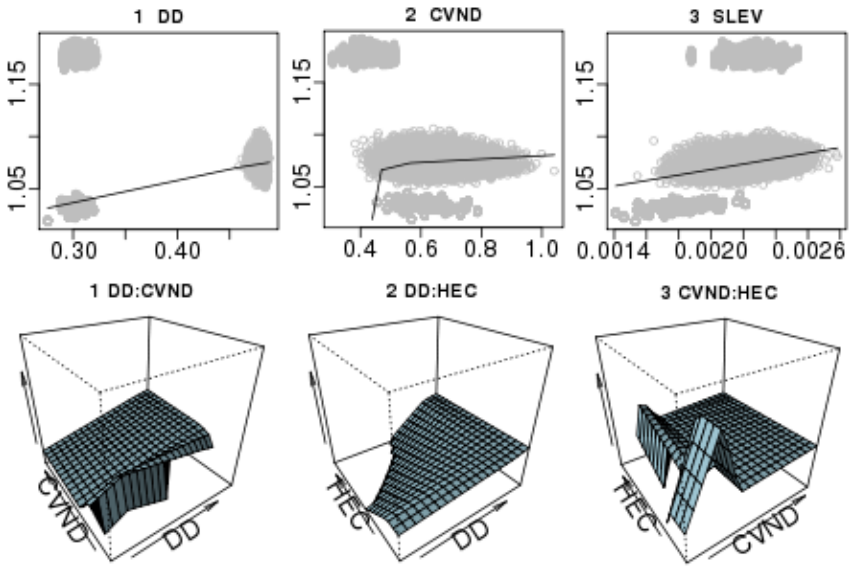

Fig. 6. Effect plots resulting from the MARS model. Main effects are plotted (top) together with the feature interactions (bottom). Coefficient of variation of nnds (CVND), highest edge cost (HEC), distinct distances (DD) and sum of lowest edge values (SLEV) were selected. In all plots the last axis reflects the approximation quality of 2-opt. The plots are generated by setting the non-displayed variables to their median values.

4 Summary and Outlook

In this paper we investigated concepts to predict TSP problem hardness for 2-opt based local search strategies on the basis of experimental features that characterize the properties of a TSP instance. A crucial aspect was the generation of a representative instance set as a basis for the analysis. This turned out to be far from straightforward. Therefore it was only possible to generate very hard and very easy instances using sophisticated (evolutionary) strategies. Summarizing, we managed to generate classes of easy and hard instances for which

we are able to predict the correct instance class based on the corresponding feature levels with only marginal errors. The coefficient of variation of nearest neighbor distances was identified as the key feature for differentiating between hard and easy instances, and the results are supported by exploratory analysis of the evolved instances and the respective optimal tours. However, it should be noted that most probably not the whole space of possible hard instances is covered by using our evolutionary method, i.e. probably only a subset of possible characteristics or feature combinations that make a problem hard for 2-opt can be identified by the applied methodology.

Instances of moderate difficulty were constructed by morphing hard into easy instances where the effects of the transition on the corresponding feature levels could be studied. A MARS model was successfully applied to predict the approximation quality of 2-opt based on the features of an adequate subset of the generated instances with very high accuracy.

The analysis offers promising perspectives for further research. A generalization of the results to other instance sizes should be addressed as well as a systematic comparison to other local and global search as well as hybrid solvers with respect to the influence of the feature levels of an instance on the performance of the respective algorithms. However, it has to be kept in mind that the computational effort intensely increases with increasing instance size as the optimum solution, e.g. computable via Concorde, is required to calculate the approximation quality of 2-opt.

The extension of the feature set is another relevant topic that could be studied. For example, in the context of benchmarking algorithms on continuous black-box optimization problems the extraction of problem properties that might influence algorithm performance is an important and current focus of research, denoted as exploratory landscape analysis (ELA, [15]). Although the TSP search space is not continuous, e.g. the ELA feature that tries to capture the number of modes of an empirical density function, could be transferred to the problem at hand. Furthermore, possible advantages of sophisticated point matching strategies during the morphing of hard into easy instances can be investigated. Finally, it is open how representative the generated instances are for real-world TSP instances. It is therefore very desirable to create a much larger pool of small to medium sized, real-world, TSP instances for comparison experiments.

Acknowledgements: This work was partly supported by the Collaborative Research Center SFB 823, the Graduate School of Energy Efficient Production and Logistics and the Research Training Group “Statistical Modelling” of the German Research Foundation.

References

1. Applegate, D., Cook, W.J., Dash, S., Rohe, A.: Solution of a min-max vehicle routing problem. INFORMS Journal on Computing 14(2), 132–143 (2002)
2. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. J. ACM 45(5), 753–782 (1998)
3. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth, Belmont, CA (1984)

4. Chandra, B., Karloff, H.J., Tovey, C.A.: New results on the old k-Opt algorithm for the traveling salesman problem. SIAM J. Comput. 28(6), 1998–2029 (1999)
5. Croes, G.A.: A method for solving traveling-salesman problems. Operations Research 6(6), pp. 791–812 (1958)
6. Englert, M., Röglin, H., Vöcking, B.: Worst case and probabilistic analysis of the 2-opt algorithm for the tsp: extended abstract. In: Bansal, N., Pruhs, K., Stein, C. (eds.) SODA. pp. 1295–1304. SIAM (2007)
7. Friedman, J.H.: Multivariate adaptive regression splines. Annals of Statistics 19(1), 1–67 (1991)
8. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65(1-3), 223–253 (1996)
9. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study in local optimization. In: Aarts, E.H.L., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization. Wiley (1997)
10. Kanda, J., Carvalho, A., Hruschka, E., Soares, C.: Selection of algorithms to solve traveling salesman problems using meta-learning. Hybrid Intelligent Systems 8, 117–128 (2011)
11. Kilby, P., Slaney, J., Walsh, T.: The backbone of the travelling salesperson. In: Proc, of the 19th international Joint Conference on Artificial intelligence. pp. 175–180. IJCAI'05, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
12. Kötzing, T., Neumann, F., Röglin, H., Witt, C.: Theoretical properties of two ACO approaches for the traveling salesman problem. In: Proc. of ANTS 2010. LNCS, vol. 6234, pp. 324–335 (2010), extended journal version appears in Swarm Intelligence
13. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. Operations Research 21, 498–516 (1973)
14. Lin, S.: Computer solutions of the travelling salesman problem. Bell Systems Technical Journal 44(10), 2245–2269 (1965)
15. Mersmann, O., Bischl, B., Trautmann, H., Preuss, M., Weihs, C., Rudolph, G.: Exploratory landscape analysis. In: Proc. of the 13th annual conference on Genetic and evolutionary computation. pp. 829–836. GECCO '11, ACM, New York, NY, USA (2011)
16. Merz, P., Freisleben, B.: Memetic algorithms for the traveling salesman problem. Complex Systems 13(4), 297–345 (2001)
17. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAMR 33(1), 60–100 (1991)
18. Reinelt, G.: Tsplib - a traveling salesman problem library. ORSA Journal on Computing 3(4), 376–384 (1991)
19. Sander, J., Ester, M., Kriegel, H., Xu, X.: Density-based clustering in spatial databases: The algorithm gdbscan and its applications. Data Mining and Knowledge Discovery 2(2), 169–194 (1998)
20. Smith-Miles, K., van Hemert, J.: Discovering the suitability of optimisation algorithms by learning from evolved instances. Annals of Mathematics and Artificial Intelligence p. forthcoming (2011)
21. Smith-Miles, K., van Hemert, J.I., Lim, X.Y.: Understanding tsp difficulty by learning from evolved instances. In: Blum, C., Battiti, R. (eds.) LION. Lecture Notes in Computer Science, vol. 6073, pp. 266–280. Springer (2010)
22. Stadler, P.F., Schnabl, W.: The Landscape of the Traveling Salesman Problem. Physics Letters A 161, 337–344 (1992)
23. Vazirani, V.V.: Approximation algorithms. Springer (2001)