Evolving Instances for Maximizing Performance Differences of State-of-the-Art Inexact TSP Solvers

Jakob Bossek(✉) and Heike Trautmann

Department of Informations Systems, University of Münster, Münster, Germany
{bossek,trautmann}@wi.uni-muenster.de

Abstract. Despite the intrinsic hardness of the Traveling Salesperson Problem (TSP) heuristic solvers, e.g., LKH+restart and EAX+restart, are remarkably successful in generating satisfactory or even optimal solutions. However, the reasons for their success are not yet fully understood. Recent approaches take an analytical viewpoint and try to identify instance features, which make an instance hard or easy to solve. We contribute to this area by generating instance sets for couples of TSP algorithms A and B by maximizing/minimizing their performance difference in order to generate instances which are easier to solve for one solver and much harder to solve for the other. This instance set offers the potential to identify key features which allow to distinguish between the problem hardness classes of both algorithms.

Keywords: TSP · Instance hardness · Algorithm selection · Feature selection

1 Introduction

The traveling salesperson problem (TSP) is one of the most famous NP-hard combinatorial optimization problems of highly practical relevance (logistics, circuit boards assembly, etc.). Given a set of N cities and positive distances d_{ij} from city i to city $j, 1 \leq i, j \leq N$ with $i \neq j$, the task is to construct a roundtrip tour of minimal total distance that visits each city exactly once and returns to the origin.

We focus on the *2D Euclidean TSP* which refers to points in the Euclidean plane und thus results in a Euclidean distance matrix. Respective solvers can be distinguished into two classes. For exact algorithms like Concorde [1] optimality of the found solution after algorithm completion can be guaranteed. However, the required runtime might be quite high, especially for large instances. State of the art solvers in inexact TSP solving proved to be able to find solutions of very high quality and simultaneously much faster. Those are therefore of crucial interest, especially for efficient algorithm selection approaches [9].

In [8] we were able to show that per-instance algorithm selection between LKH [6] and the recently introduced evolutionary algorithm EAX [13] together

P. Festa et al. (Eds.): LION 2016, LNCS 10079, pp. 48–59, 2016.
DOI: 10.1007/978-3-319-50349-3_4

with specific restart variants as presented in [8] is very promising w.r.t. to improving the state of the art in TSP solving.

Per-instance algorithm selection makes use of a comprehensive set of instance features from the literature[1] [7,12,15,16]. A crucial aspect is the identification of features which are useful for determining the instance hardness for different solvers. While much progress has been made in this respect, e.g. in [5] where LKH behavior is related to the transition from highly structured, polynomially solvable TSP instances to instances with increasingly random distributions of nodes, this issue is not yet fully understood. This paper specifically aims at improving the analysis of performance differences of the current state of the art inexact TSP solvers LKH+restart and EAX+restart as well as their orignial versions without restart mechanism. More specifically, we use an evolutionary approach to evolve instances on which the solvers exhibit maximum performance difference, i.e. which are easier to solve for one solver and harder for the other. The used evolutionary algorithm, inspired by [16], was introduced in the context of analysing problem hardness of the 2-opt heuristic in [11] and further adapted in [12,14]. However, we now refrain from focussing on a single solver but directly use the performance ratio of two solvers as the fitness function inside the EA.

Section 2 provides details of the TSP solvers and feature sets, while the evolutionary algorithm is presented in Sect. 3. The conducted experimental results are illustrated and discussed in Sect. 4 followed by summarizing remarks and an outlook on future research perspectives in Sect. 5.

2 Solvers and Features

TSP Solvers. Both LKH, a stochastic local search algorithm based on the Lin-Kernighan procedure [6], and EAX [13], a recently introduced evolutionary algorithm utilizing a specialized new edge assembly crossover procedure, are focussed. LKH is the state of the art in inexact TSP solving since its introduction in 2000. We used the reference implementation LKH 2.0.7 based on the former implementation 1.3 [10]. In [8], we introduced a dynamic restart mechanism as the underlying stochastic search process tends to stagnate too early quite frequently. This version is termed LKH+restart. The first indication that EAX could be competitive to LKH was given in [13], which was confirmed in [8]. In the latter paper, a restart strategy for EAX, EAX+restart, was implemented based on the original internal termination criterion. Once this is met, a restart is conducted and this procedure is repeated until a given accuracy or running time limit is reached.

It could be shown [8] that the respective restart variants outperform the original solver versions. Moreover, EAX+restart emerged as the single best solver over the set of considered representative instances. However, an algorithm selection model based on features that are quite cheaply computable could be learnt that managed to perform even better.

[1] All these feature sets have been recently made availabe in a single R-Package [4].

Features. Established feature sets for characterizing Euclidean TSP instances, i.e. the feature set described in [12] as well as in [7], are used for characterizing both the evolved as well as the baseline (random, TSPLIB) instances. We denote the former as *TSPMeta* and the latter as *UBC* features. The recently introduced additional set of features based on k-nearest neighbours [15] will be focussed in future studies. Both considered sets contain a comprehensive collection of features including e.g. features characterizing the distance structure, identifying possible clusters of nodes, statistics based on angles between cities and its two nearest neighbours as well as minimum spanning tree information. The `R`-package `salesperson` [4] containing all relevant feature sets is used for the feature computation task. Having already the algorithm selection task in mind for further studies, we restricted our feature set to cheaply computable features, i.e. we excluded the local search, branch and cut, and clustering distance features from the UBC feature set (UBC (cheap)) as motivated in [8].

3 On Evolving Instances

A simplified pseudocode is given in Algorithm 1: the initial population is generated by placing the desired number of nodes uniformly at random in the unit square $[0,1]^2$. Subsequently, the next generation is obtained by selecting two parents from the mating pool, applying crossover as well as two mutation strategies in a row, namely uniform and gaussian mutation. Uniform mutation is applied with a very low probability. This operator replaces the node coordinates of selected nodes with new randomly chosen coordinates and thus may be

Algorithm 1. Evolving EA

```
1: function EA(fitnessFun, popSize, instSize, generations, timeLimit)
2:     poolSize = ⌊ popSize / 2 ⌋
3:     for i = 1 → popSize do
4:         population[i] = generateRandomInstance(instSize)          ▷ in [0,1]^2
5:     end for
6:     while stopping condition not met do
7:         for i = 1 → popSize do
8:             fitness[i] = fitnessFun(population[i])
9:         end for
10:        matingPool = createMatingPool                  ▷ 2-tournament-selection
11:        offspring[1] = getBestFromCurrentPopulation                ▷ 1-elitism
12:        for i = 2 → popSize do
13:            Choose p1 and p2 randomly from the mating pool
14:            offspring[i] = applyVariations(p1, p2)
15:            Rescale offsspring to [0,1]^2                          ▷ Algorithm 2
16:            Round to cell grid                                     ▷ Algorithm 3
17:        end for
18:        population = offspring
19:    end while
20: end function
```

termed a global mutator. In contrast, gaussian mutation works locally by adding normally distributed noise to the point coordinates. The two sequential mutation strategies together enable small local as well as global structural changes of the offspring resulting from the crossover operation. Furthermore, a 1-elitist strategy is adopted to ensure survival of the current fittest individual.

A final rescaling of the evolved instances ensures the complete coverage of $[0, 1]^2$ in that the minimum and maximum coordinates are placed on the boundary of the instance space (see Algorithm 2). Therefore the area will be covered quite homogenously and instances become comparable in this regard. Afterwards the instance nodes are rounded to the nearest grid cell center after discretizing the plane using a grid with *cells* sections (see Algorithm 3). This relates to the aim of evolving practically relevant structures (e.g. in the design of circuit boards) and will furthermore affect some features which incorporate the proportion of distinct distances. Note that this strategy conceptually differs from rounding to a predefined number of digits. Figure 1 (taken from [12]) visualizes both rescaling and rounding.

Algorithm 2. Rescale Instance

```
1: function RESCALE(instance)
2:     mins ← COLUMN_MINS(instance)
3:     maxs ← COLUMN_MAXS(instance)
4:     δ ← maxs − mins
5:     scaled ← ∅
6:     for city ∈ instance do
7:         scaled ← scaled ∪ {(city − mins)/δ}
8:     end for
9:     return scaled
10: end function
```

Algorithm 3. Round instance

```
1: function ROUND(instance, cells)
2:     gridRnd ← CREATEGRID(resolution = cells)
3:     instRnd ← FLOOR(instance ∗ cells)/cells
4:     for i = 1 to instSize do
5:         instRnd[i, ] ← SetToCellCenter(instRnd, gridRnd)
6:     end for
7:     return instRnd
8: end function
```

Mersmann et al. [11] had chosen the *approximation ratio*, i.e., the arithmetic mean of the tour length computed by the considered stochastic algorithm divided by the length of the optimal tour computed by Concorde as the fitness function to be optimized. The first idea was to adopt this approach with slight modifications, since we aim to generate instances for pairs of algorithms A and B, i.e.,

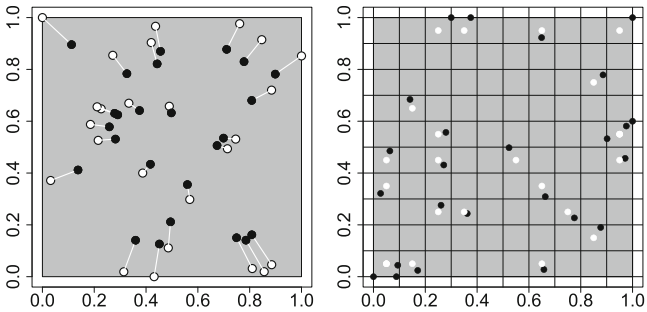

Fig. 1. Rescaling of an instance of size 25 (left). The original instance is reflected by black dots. Rounding of an instance of size 25 to grid cell centers (right). The rounded instance is visualized by white dots.

to focus on the ratio of the arithmetic means of the respective tour lengths [3]. However, we observed LKH and EAX and the restart variants respectively to perform extraordinary well even on large instance sizes up to 1000 in preliminary experiments. Hence, we experienced the approximation ratio to be unrewarding for our aims since we observed the "best" generated instance to have an approximation ratio of approximately 1 in our first series of experiments. In fact, the chosen state of the art solvers turned out to be too powerful for this scenario. Instead, our approach is slighty different. We use the *penalized average runtime* consumed by an algorithm to find the optimal tour (computed by Concorde in advance) as the performance of the algorithm. However, an algorithm reaching the cutoff time is not penalized with 10 times the latter within the EA as it is the standard procedure in par10. The cutoff time itself is used as otherwise the probability that such solutions would be removed at later stages of the EA run would be extremely low. However, for the final instance evaluation, the classsical par10 score is used. For ensuring integer values of the distance matrix inside the EA, the euclidean distance matrix is computed on the original coordinates, multiplied by a scaling factor of 100 and afterwards rounded to the nearest integer.

Let $R_A(I)$ denote the slightly modified penalized average runtime of algorithm A on instance I as explained above. Then the runtime proportion $R_{(A,B)}(I)$ for a pair of algorithms (A, B) is defined as

$$R_{(A,B)}(I) = \frac{R_A(I)}{R_B(I)}.$$

This *runtime ratio* serves as the fitness function in our investigations. We thus moved the focus to the time aspect instead of the solution quality. We minimize $R_{(A,B)}$ in order to generate instances which are easier to solve for algorithm A and harder to solve for algorithm B respectively.

4 Experiments

4.1 Experimental Setup

We generated each 25 instances of size 300 for each solver pairing (EAX, LKH), (LKH, EAX), (EAX_RESTART, LKH_RESTART) and (LKH_RESTART, EAX_RESTART) resulting in an evolved instance set of 100 TSP instances. Inside the evolutionary algorithm, each solver is replicated three times on each instance due to limited computational budget. The final evaluation of the instances, however, is based on 10 replications and the classical par10 score. All internal termination criteria besides cutoff time of two minutes were deactivated in all runs to get reasonable estimates of the performance measure.

Based on preliminary experiments and the experience of [11,12] the EA parameters were set to $popSize = 30$, $generations = 5000$, $uniformMutationRate = 0.001$, $normalMutationRate = 0.1$, $cells = 100$, and the standard deviation of the gaussian mutation operator $normalMutationSD = 0.025$.

As a baseline, 100 random instances were generated by placing the desired number of nodes uniformly at random in the unit square $[0, 1]^2$. The Euclidean distance matrix was computed, multiplied by a scaling factor of 100 and subsequently rounded to the nearest integer. Moreover, Euclidean TSPLIB instances with node sizes between 200 and 400[2] were chosen in order to allow comparisons to practically relevant instances. For all considered algorithms the respective par10 scores were applied for comparison.

All experiments were run on the parallel linux computer cluster PALMA at University of Münster, consisting of 3528 processor cores in total. The utilized compute nodes are 2,6 GHz machines with 2 hexacore Intel Westmere processors, totally 12 cores per node and 2 GB main memory per core.

4.2 Results

Figure 2 visualizes the par10 scores of all instances for both solver pairs (EAX vs. LKH, EAX+restart vs. LKH+restart) in a scatterplot. The instances are marked w.r.t. instance type, i.e. either "random", "evolved" or "tsplib". It becomes obvious that the introduced evolutionary approach very successfully generates instances with high performance differences of the solvers. Even in the restart scenario where solver runtimes are quite homogenously clustered in the center of the plot, the evolved instances can clearly be distinguished and are located far away from the bisecting line. Both optimization directions work well (two clusters of instances) while evolving instances which are easier for EAX+restart is even more successful (upper cluster). Moreover, the TSPLIB instances are much easier to solve compared to the remaining ones in both scenarios. There is only one exception (rd400) which is extremely hard for both solvers.

Figure 3 presents boxplots of the par10 scores distribution for each solver categorized by instance type and confirms the discussed findings. In all cases but the

[2] TSPLIB-Instances: a280, gil262, kroA200, kroB200, lin318, pr226, pr264, pr299, rd400, ts225, tsp225.

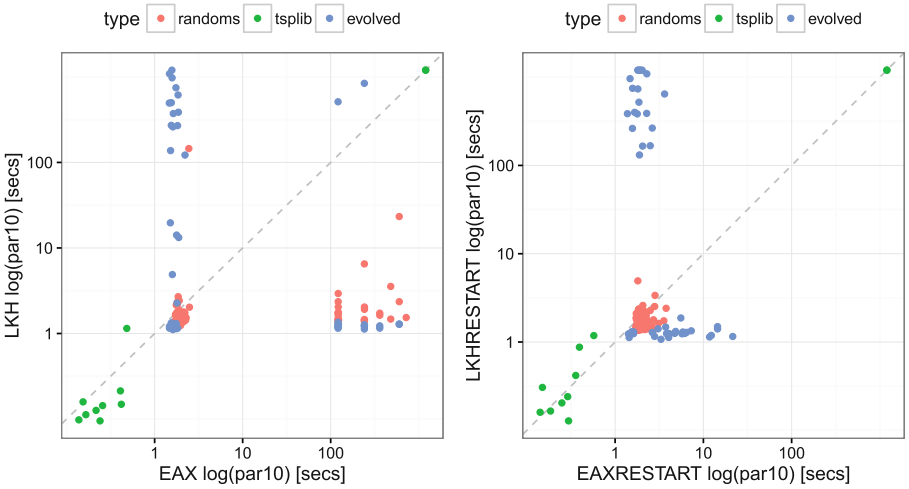

Fig. 2. Par10 scores (log-scale) of EAX vs. LKH (left) resp. EAX+restart vs. LKH+restart (right). Colors help to distinguish between the instance types. (Color figure online)

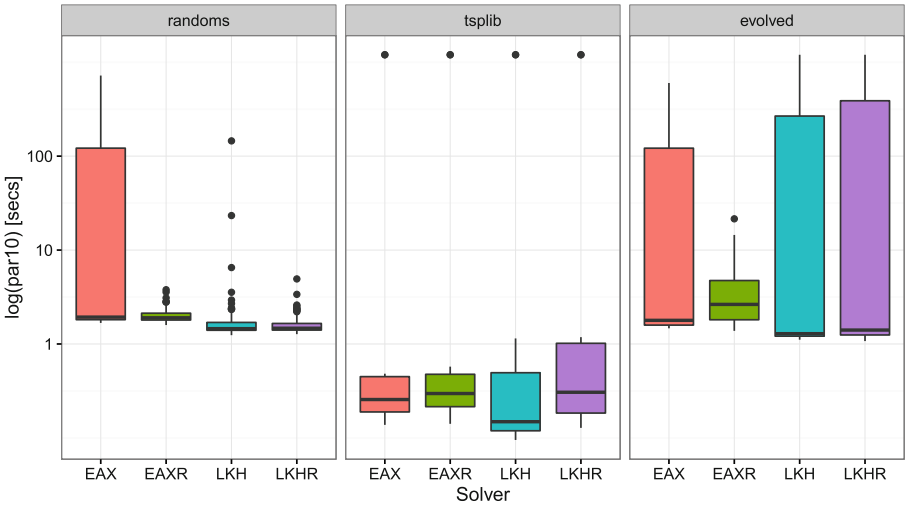

Fig. 3. Boxplots of the par10 scores (log-scale) for each solver categorized by instance type. (Color figure online)

original EAX the runtimes of the evolved instances are substantially higher compared to the random ones. Specifically, the variability of the runtimes increases reflected by the upper quartile, i.e. the upper border of the boxes. Frequently, even the runtime of Concorde was exceeded so that using an inexact solver was of no merit in retrospect. In general, as problem hardness tends to increase for

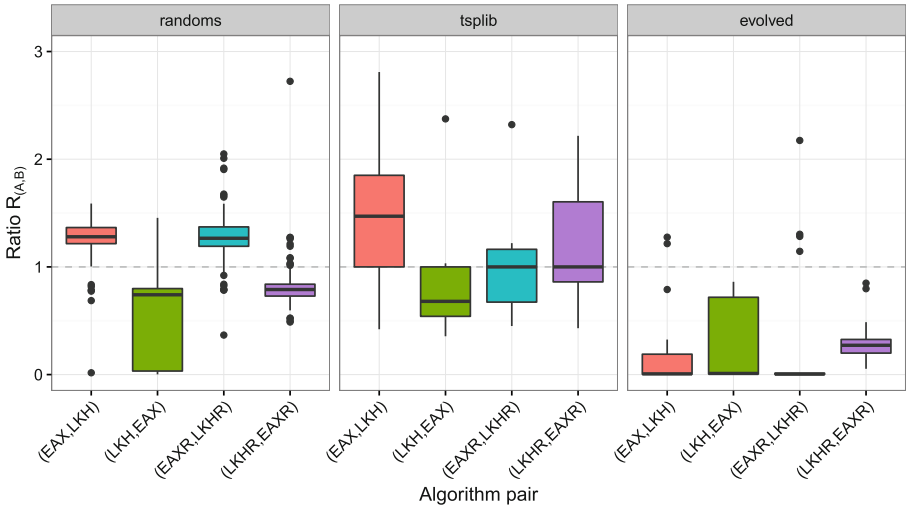

Fig. 4. Ratios of par10 scores for each solver pair, i.e. $R_{(A,B)}$. A value smaller than one means that the runtime of the first solver is smaller than the respective one of the second. This ratio is the fitness function to be minimized inside the evolutionary algorithm for evolving the respective instances. (Color figure online)

both solvers, local search in general becomes harder for the evolved instances. However, the extent varies by solver which results in low par10 score ratios. Par10 scores on the TSPLIB instances differ significantly from the remaining ones, the whole distribution is located far more to the left. Here, you can see the single hard outlier instance as well.

The actual par10 score ratios are presented by boxplots in Fig. 4. Values smaller than one reflect the superiority of the first solver from the pair (SolverA, SolverB), i.e. Solver A has a smaller par10 score than Solver B. As this ratio forms the objective function of the evolutionary algorithm, which has to be minimized, we expect these values to be substantially lower than one. Apart from very few outliers this is confirmed by the respective boxplots. Furthermore, significant differences are obvious compared to the random instances, even for the (LKH, EAX) pair where LKH already exhibits much lower runtimes than EAX on the random instances.

Representative instances are plotted in the Euclidean plane in Fig. 5. For each solver pair the four smallest instances regarding the par10 score ratio are displayed. Unfortunately, structural difference are very particular and cannot be clearly detected visually. Therefore, we additionally made use of machine learning techniques. A classification approach was conducted for each solver pairing with the aim of predicting the instance type (random, evolved) based on the TSP feature set comprising *TSPMeta* and UBC (cheap). To derive the most important features a nested feature selection with 10-fold crossvalidation was performed based on a simple classification tree. Deterministic forward and backward search

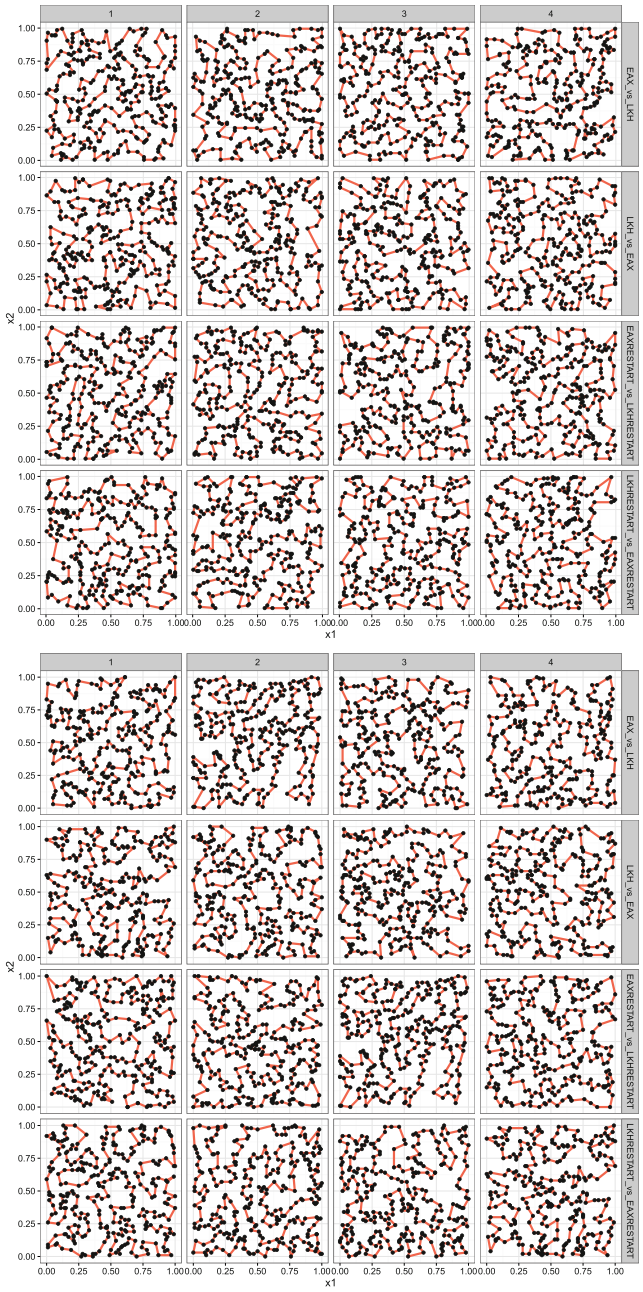

Fig. 5. Four lowest ranked instances regarding the par10 score ratio for all scenarios. Evolved instances are visualized on top, the random instances below.

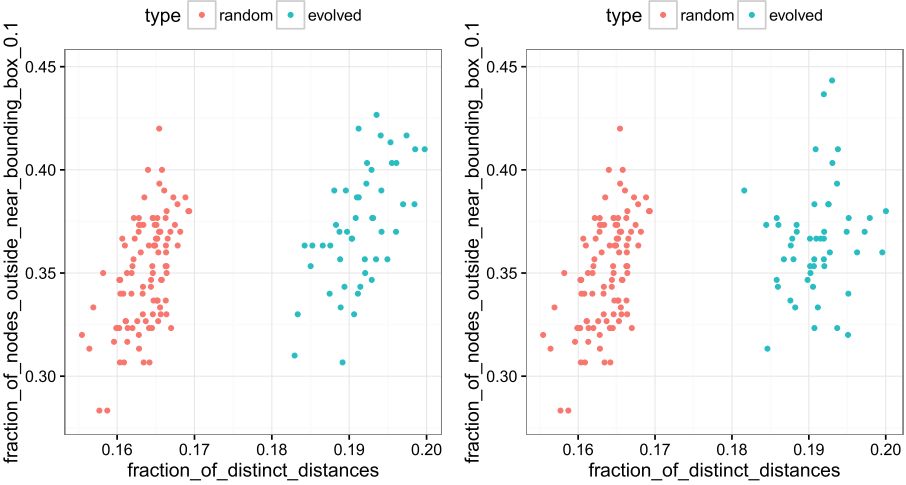

Fig. 6. Scatterplots of the two most important features selected during the feature selection step for predicting the instance type (random, evolved) based on instance features for the original versions (left) and the restart versions (right). (Color figure online)

served as the search strategy guiding the feature selection. Additionally an outer 10-fold crossvalidation was used in order to preserve that the selected features depend on the underlying fold. All computations were conducted with the R package mlr [2].

It turns out that the fraction of distinct distances is sufficient to separate random and evolved instances (see the scatterplots in Fig. 6) for both solver pairings $\{EAX, LKH\}$ and $\{EAX\_RESTART, LKH\_RESTART\}$. Misclassification errors vanish for both pairings. Thus, local search gets harder for higher fractions of distinct distances which was already hinted at in other studies. The observed characteristic of the evolved instances might be linked to the rounding strategy to grid cell centers applied to each individual in the evolutionary algorithm after mutation. However, the considered solvers face different levels of difficulties not solely depending on this fraction. The kind of classification probably will differ for instances evolved based on deactivated rounding to grid cell centers. We are going to investigate this in future work.

5 Conclusion

In this paper, we used an evolutionary approach to evolve TSP instances with maximal perfomance difference of LKH+restart vs. EAX+restart and of the pair of the respective original variants (without restart). For instances of size 300 a substantial decrease of solver performance ratios compared to the behaviour on random and TSPLIB instances could be obtained. This especially holds for the

restart variants which are the state of the art inexact TSP solvers while it turned out that it was more effective to generate easier instances for EAX+restart together with much harder instances for LKH+restart than the opposite case. However, one has to be aware that a small performance ratio $R_A(I)/R_B(I)$ does not necessarily mean, that I is easy to solve for A. It may be hard for both, but easier for A.

Comparing random and evolved instances, it turned out that the number resp. the fraction of distinct distances is a central factor for separating both instance sets, i.e. local search in general gets harder given this situation. However, this feature is not suited for distinguishing between the performance of the solvers within the evolved instance sets. The next step will therefore consist of predicting the optimization direction for each solver pairing $\{A, B\}$ (easy for A or easy for B), i.e. a detailed analysis which features resp. feature combinations allow for identifying the kind of solver which performs worse than its competitor within the evolved set will be conducted. Of course, the fraction of distinct distances alone does not provide sufficient information to separate here as in this respect the set is quite homogenous. Preliminary studies indicate that features based on relating the node locations to the centroid of all nodes might play a role here.

In future work we will moreover work on adapting the EA to specifically focus on diversity of evolved instances to generate distinct structures as well as on assessing the influence of the internal rescaling and rounding steps. Moreover, larger instances will be addressed in a systematic way.

Acknowledgements. The authors acknowledge support from the European Center of Information Systems (ERCIS).

References

1. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton (2007)
2. Bischl, B., Lang, M., Richter, J., Bossek, J., Judt, L., Kuehn, T., Studerus, E., Kotthoff, L.: mlr: Machine Learning in R. R package version 2.6 (2015). http://CRAN.R-project.org/package=mlr
3. Bossek, J.: Feature-basierte performance-analyse von Algorithmen für das travelling-salesperson-problem. Bachelor thesis, Technical University of Dortmund (2012)
4. Bossek, J.: salesperson: Computation of Instance Feature Sets and R Interface to the State-of-the-Art Solvers for the Traveling Salesperson Problem. R package version 1.0 (2015). https://github.com/wwu-wi/salesperson/
5. Fischer, T., Stützle, T., Hoos, H.H., Merz, P.: An analysis of the hardness of TSP instances for two high-performance algorithms. In: Proceedings of the 6th Metaheuristics International Conference, Vienna, Austria, pp. 361–367 (2005)
6. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math Program. Comput. **1**(2–3), 119–163 (2009)
7. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: methods & evaluation. Artif. Intell. **206**, 79–111 (2014)

8. Kotthoff, L., Kerschke, P., Hoos, H., Trautmann, H.: Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In: Dhaenens, C., Jourdan, L., Marmion, M.-E. (eds.) LION 2015. LNCS, vol. 8994, pp. 202–217. Springer, Heidelberg (2015). doi:10.1007/978-3-319-19084-6_18
9. Kotthoff, L.: Algorithm selection for combinatorial search problems: a survey. AI Mag. **35**(3), 48–60 (2014)
10. Lacoste, J.D., Hoos, H.H., Stützle, T.: On the empirical time complexity of state-of-the-art inexact TSP solvers, optimization Letters, to appear
11. Mersmann, O., Bischl, B., Bossek, J., Trautmann, H., Markus, W., Neumann, F.: Local search and the traveling salesman problem: a feature-based characterization of problem hardness. In: Hamadi, Y., Schoenauer, M. (eds.) LION 6. LNCS, vol. 7219, pp. 115–129. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34413-8_9
12. Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., Neumann, F.: A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. Ann. Math. Artif. Intell. 1–32 (2013). http://dx.doi.org/10.1007/s10472-013-9341-2
13. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS J. Comput. **25**(2), 346–363 (2013)
14. Nallaperuma, S., Wagner, M., Neumann, F., Bischl, B., Mersmann, O., Trautmann, H.: A feature-based comparison of local search and the christofides algorithm for the travelling salesperson problem. In: Foundations of Genetic Algorithms (FOGA) (2013)
15. Pihera, J., Musliu, N.: Application of machine learning to algorithm selection for TSP. In: Fogel, D., et al. (ed.) Proceedings of the IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE Press (2014)
16. Smith-Miles, K., van Hemert, J.: Discovering the suitability of optimisation algorithms by learning from evolved instances. Ann. Math. Artif. Intell. **61**(2), 87–104 (2011)