Understanding Characteristics of Evolved Instances for State-of-the-Art Inexact TSP Solvers with Maximum Performance Difference

Jakob Bossek(✉) and Heike Trautmann

Information Systems and Statistics Group, University of Münster, Münster, Germany
{bossek,trautmann}@wi.uni-muenster.de

Abstract. State of the Art inexact solvers of the NP-hard Traveling Salesperson Problem (TSP) are known to mostly yield high-quality solutions in reasonable computation times. With the purpose of understanding different levels of instance difficulties, instances for the current State of the Art heuristic TSP solvers LKH+restart and EAX+restart are presented which are evolved using a sophisticated evolutionary algorithm. More specifically, the performance differences of the respective solvers are maximized resulting in instances which are easier to solve for one solver and much more difficult for the other. Focusing on both optimization directions, instance features are identified which characterize both types of instances and increase the understanding of solver performance differences.

Keywords: Transportation · Metaheuristics · Combinatorial optimization · TSP · Instance hardness

1 Introduction

In the Traveling Salesperson Problem (TSP) we aim to find a minimal cost roundtrip tour in an edge-weighted graph, which visits each node exactly once and returns to the starting node. A plethora of algorithmic approaches for this famous NP-hard combinatorial problem was developed in the past decades. Inexact solvers for the TSP are known to produce high-quality solutions in reasonable time compared to exact solvers such as Concorde [1]. Recently, the EAX solver [2] was shown to be competitive to the well-known State of the Art LKH algorithm [3], more specifically respective restart variants LKH+restart and EAX+restart even improve the original versions [4] on the Euclidean TSP. However, there is no single inexact solver which operates best on all possible problem instances regarding solution quality. In this work, we investigate performance differences of the current State of the Art TSP solvers on specifically evolved instances.

Efficient algorithm selection approaches [5] in this field are conducted in a feature- and instance-based fashion. TSP features, e.g. in [6–9][1], are computed on benchmark instances and related to algorithm performance allowing for

[1] These feature sets are available in the `R`-package `salesperson` [10].

G. Adorni et al. (Eds.): AI*IA 2016, LNAI 10037, pp. 3–12, 2016.
DOI: 10.1007/978-3-319-49130-1_1

constructing algorithm selection models for unseen instances based on machine learning techniques.

Understanding which instance characteristics pose a specific level of difficulty onto high-performing TSP solvers is an active research field, see e.g. [4,11]. In this paper, we specifically address LKH+restart compared to EAX+restart as the two current State of the Art TSP solvers with potential for improving their standalone application by means of a portfolio approach [4]. We are specifically interested in instances on which both solvers exhibit maximum performance difference, i.e., which are much harder to solve for one of the solvers, while we focus both directions. Thus, the performance ratio is used as fitness function of a sophisticated evolutionary algorithm for evolving instances which was already used in a similar fashion for single solvers in [7,8,12,13]. Two variants of solver performance are contrasted. The classical mean par10 score is supplemented by focussing on the median solver runtime over a fixed number of runs diminishing the influence of timeouts in individual runs. Moreover, the influence of rounding instances to a grid structure is analysed systematically. Additionally, we contrast characteristics of instances which are much harder or much easier for one solver w.r.t. the other.

Section 2 details the evolutionary algorithm. Experimental results are then presented in Sect. 3. Conclusions and an outlook on future research are given in Sect. 4.

2 EA for Evolving Instances

Algorithm 1 reflects the process of the evolutionary algorithm in terms of pseudocode. The core parameter of the EA is the kind of fitness function used. As the EA aims at generating instances with maximum performance difference of two solvers, we define the fitness function as the performance ratio $P_{(A,B)}(I)$ for a pair of solvers A and B, i.e.

$$P_{(A,B)}(I) = \frac{P_A(I)}{P_B(I)}$$

on a specific instance I, where $P_A(I)$ and $P_B(I)$ are the solver performances of solver A and B on instance I. Solver performance in our scenario is either determined by the standard indicator *penalized average runtime* or by the *penalized median runtime*. The former repeatedly measures the runtime of the solver on an instance until the optimal tour (pre-computed by Concorde) has been found and computes the arithmetic mean subsequently. In case the cutoff time *timeLimit* is reached, ten times the cutoff time is used for further computations as a penalty. However, inside the EA, the actual cutoff time is used ensuring that the probability of removal of such a solution at later algorithm stages is not unreasonably low. The evaluation at the final generation uses the classical par10 score with the full penalty. The median score is much more insensitive to outliers and maximum ratio in medians is much harder to obtain.

Algorithm 1. Evolving EA

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

The initial population of size $popSize$ is randomly generated in $[0,1]^2$ for instances of size $instSize$ and the performance ratio is computed. Distances are scaled by multiplying with a factor of 100 and afterwards rounded to the nearest integer. This step is neccassary since EAX expects integer distances. The EA is then run for a fixed number of *generations* and the evolutionary loop is executed as follows: The mating pool is formed by 2-tournament selection supplemented by the best solution of the current population (1-elitism). Two different mutation operators are applied to each combination of randomly drawn instance pairs of the mating pool. Uniform mutation replacing coordinates of selected nodes with new randomly chosen coordinates is applied with a very low probability possibly followed by gaussian mutation adding normal noise to the selected point coordinates. Therefore, global as well as local changes can come into effect. In the current version the EA does not use any recombination operator. A single EA generation ends after rescaling the instance to completely cover $[0,1]^2$ and, if $rnd = true$, rounding the points to the nearest cell grid. The latter relates to important relevant structures in practice such as the structural design of circuit boards.

3 Experiments

3.1 Experimental Setup

In total 200 TSP instances were evolved. For all four considered optimization directions, i.e. $P_{(LKH,EAX)}$, $P_{(EAX,LKH)}$, $P_{(LKH+restart,EAX+restart)}$ and $P_{(EAX+restart,LKH+restart)}$, in each case 25 instances were generated with activated and deactivated rounding respectively. Based on preliminary experiments and experimental results of [8,12] the EA parameters were set as follows: $timeLimit = 120$, $popSize = 30$, $generations = 5000$, $uniformMutationRate = 0.05$, $normalMutationRate = 0.1$, $normalMutationSD = 0.025$ and $cells = 100$. We used the reference implementation LKH 2.0.7 based on the former implementation 1.3 [14], the original EAX implementation as well as specific restart variants as described in [4]. The solvers were repeatedly evaluated, three times inside the EA due to a limited computational budget but ten times for final evaluations. As described in Sect. 2 either the par10 score or the median score was computed for the final instances.

For comparison and practical validation, performance ratios of the respective solvers on TSPLIB instances[2] of comparable size, i.e. $200 \leq instSize \leq 400$ were computed for both kinds of performance measures. Moreover, 100 random instances in $[0, 1]^2$ were generated while the same rounding strategy of the distance matrix was applied as used inside the EA for the evolved instances. 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.

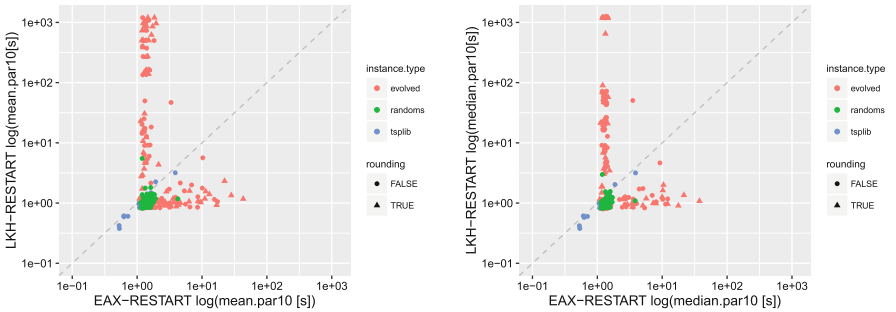

Fig. 1. Average (left) and median (right) par10 scores (log-scale) of LKH+restart and EAX+restart on evolved, random and TSPLIB instances. A specific symbol visualizes whether instances were rounded to a grid structure (rnd) or not (nrnd).

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

3.2 Results

Figures 1 and 2 give an overview about the performance scores of the considered solver pairs, i.e., both for the original as well as the restart variants. Evolved instances are visualized together with random and TSPLIB instances.

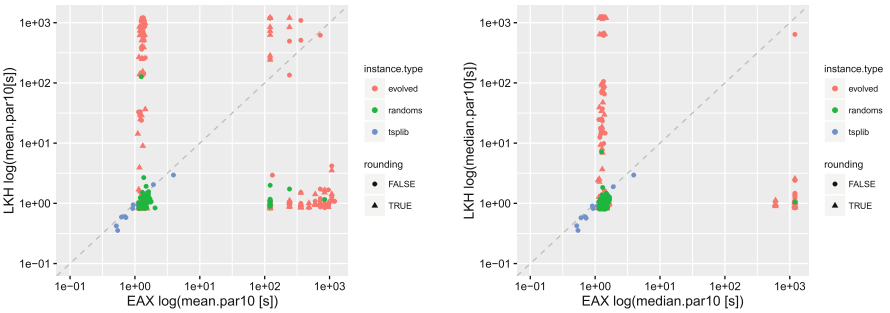

Fig. 2. Average (left) and median (right) par10 scores (log-scale) of LKH and EAX on evolved, random and TSPLIB instances. A specific symbol visualizes whether instances were rounded to a grid structure (rnd) or not (nrnd).

It becomes obvious that in both pairings the presented EA successfully generated instances with much higher performance differences of both solvers than usually present in random as well as TSPLIB instances. Whether the instance was rounded to a grid structure inside the EA does not have a structural influence on the relation of the performance scores. Moreover, we see that generating easier instances for LKH+restart compared to the EAX+restart is a much harder task than in case of considering the opposite direction. Specifically, EAX+restart timeouts did not occur here. On the contrary, for variants without restart, this effect cannot be observed. In addition, in some cases, the EA did not converge in that instances are of similar difficulty for both solvers. This behaviour, however, is due to a part of solver runs resulting in timeouts as reflected by the location of the points in case the median scores are considered (Fig. 2, lower right part). In general, evaluating with median scores diminishes the influence of timeouts. Maximum median scores can only be obtained in case at least fifty percent of solver runs on a specific instance result in a timeout. Therefore, there could be potential of using this kind of performance measure inside the EA. Results are presented further down.

The previous observations are reflected in Fig. 3 as well. Here, boxplots of performance scores on the evolved instances for each solver depending on the optimization direction as well as the rounding activations are given. Supplementary to Figs. 1 and 2 *all* solvers have been evaluated on the respective instances. Not surprisingly, instances specifically generated for the restart variants do not result in such extreme performance differences for the classical variants and the other way round. However, the basic tendency can be observed here as well.

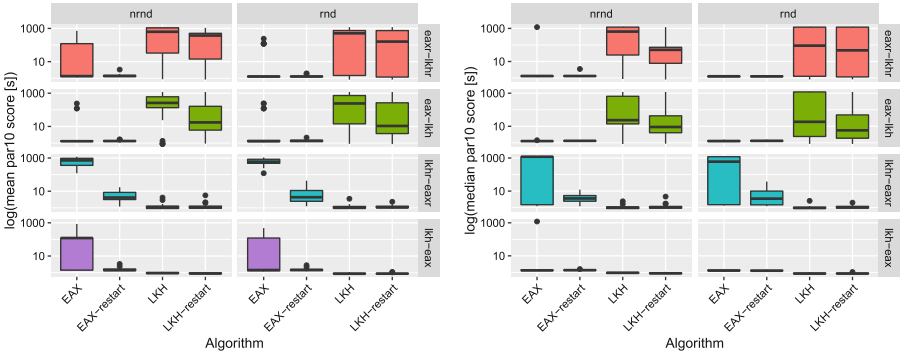

Fig. 3. Mean (left) and median par10 scores (right) of the four solvers depending on rounding (rnd) and type of optimization (log-scale).

Evaluating with median scores shows that especially the pairing (LKH,EAX), i.e. generating easier instances for LKH, does not show the desired performance effects. Figure 4 explicitly provides boxplots of the performance ratios $P_{(A,B)}$ and thus summarizes all effects previously listed, in particular the huge differences in performance ratios compared to random and TSPLIB instances.

Understanding Characteristics of Extreme Instances. Next we try to understand which structural TSP characteristics, termed *TSP instance features*, of the evolved instances are suitable to distinguish between easy and hard to solve instances respectively. Those features could be used in algorithm selection scenarios to select the best solver out of a portfolio of solvers. A classification approach was used in order to separate the respective groups of the most extreme instances by means of well-known established TSP features as introduced in [6,8]. The `R`-package `salesperson` [10] was used for this purpose. However, we did not consider expensive features such as local search, clustering distance based features as well as branch and cut techniques in order to avoid much computational overhead, especially regarding the possible influence on future algorithm selection models as e.g. in [4].

In case of the restart variants, the ten most extreme instances w.r.t. performance ratios on mean par10 scores are selected, i.e. the ten best EA results of both optimization directions for the solver pairings. A random forest was used to distinguish between both instance sets combined with a feature selection approach based on leave-one-out cross-validation and an outer resampling resulting in a median misclassification error of zero and a respective mean misclassification error of 0.3 regarding all folds. Thus, we are able to separate both instance sets in a satisfying way including an indication which features are of crucial importance here. Figure 5 shows the respective feature importance plot created by means of the `R`-package `flacco` [15]. Red dots reflect features which are used in at least 70 % of the folds, orange labeled features at least in 30 % and black ones at least once. In this regard the median distance of the minimum spanning tree is

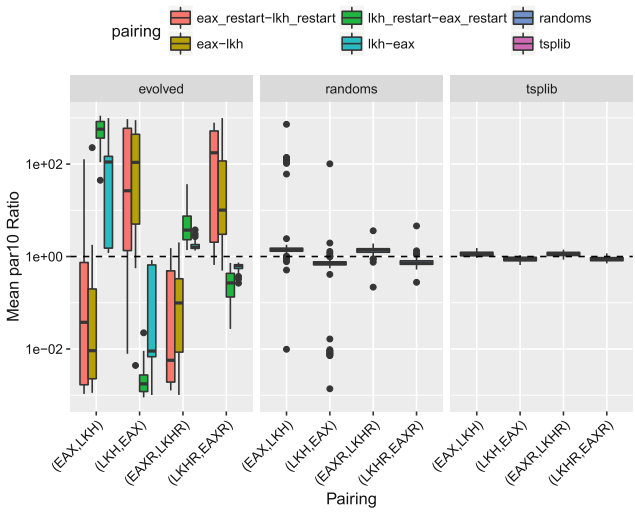

Fig. 4. Performance ratios based on mean par10 scores of the four considered algorithms on all considered instance sets.

identified as the crucial feature separating both instance classes. The results conicide with the results of [8] where the mean distance of the minimum spanning tree was identified as a separating feature between easy and hard instances for TSP approximation algorithms. This result is promising with respect to future work in the algorithm selection context: The computation of minimum spanning tree characteristics is an computationally cheap task and we strive for cheap feature, since wasting a lot of runtime for the feature computation before actually solving the TSP itself is senseless.

The same analysis was conducted for the original solver variants. However, as evolved instances are much denser in the lower right and upper left corner in Fig. 2 than in the restart case, we only selected the respective five most extreme instances. In this case different features play a key role in explaining solver performance differences including nearest-neighbor based features as visualized in Fig. 5. Again, the median misclassification error vanishes while the mean misclassification error is 0.2, i.e. only two out of the ten instances are misclassified.

Median Scores as EA Internal Performance Measures. In order to investigate possible potential of using the median par10 score inside the EA for performance evaluation, we conducted a smaller experiment focussed on the restart variants with inactive rounding as these solvers in our view are most interesting. This lead to fifty evolved instances, i.e. 25 for each optimization direction.

Figure 6 gives an overview about the resulting median par10 scores on the newly evolved instances together with the random instance and TSPLIB results. We see that the EA is not successful in improving the performance ratio of both

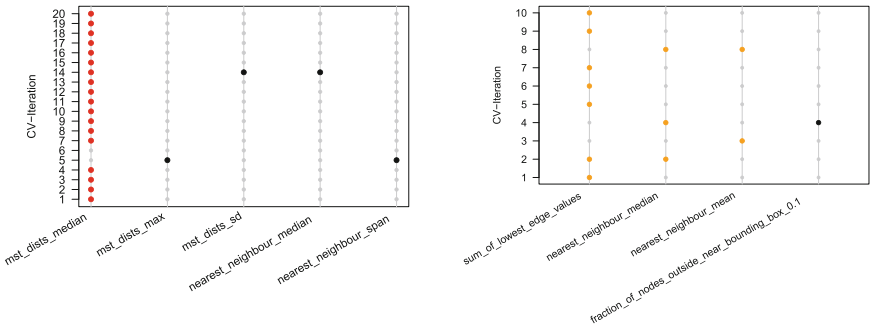

Fig. 5. Variable Importance Plot of Random Forest distinguishing the (left): ten most extreme instances w.r.t. performance ratio for the restart variants, (right): five most extreme instances w.r.t. performance ratio for the original algorithm variants.

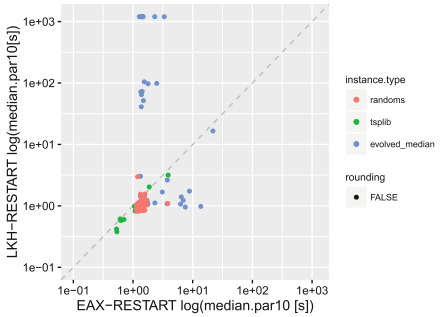

Fig. 6. Median par10 scores (log-scale) of LKH+restart and EAX+restart on evolved (median score fitness), random and TSPLIB instances.

solvers even further compared to the resulting median evaluation on the instances originally generated inside the EA using mean par10 scores. The same is true for comparing the mean par10 scores on both scenarios (see Fig. 7). However, slight improvements are visible in case easier instances for LKH+restart are evolved. Most probably the median alone does not provide enough differentiation between varying solver results over the repetitions.

However, in our view an adequate combination of mean and median scores inside the EA fitness function is promising in order to get deeper insights into solver variance on the considered instances. We will investigate this issue in further studies together with increasing the number of solver repetitions along the evolutionary loop.

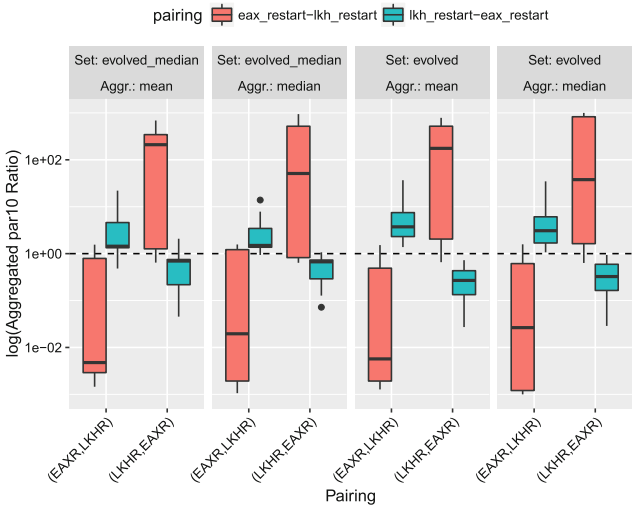

Fig. 7. Comparison of mean and median par10 ratios (log-scale) of instance sets evolved for LKH+restart and EAX+restart either by using the mean par10 or the median score inside the EA as fitness function.

4 Conclusions

This work focusses on the two current State of the Art inexact TSP solvers LKH and EAX together with their respective restart variants. In order to increase understanding of performance differences of both solvers, a sophisticated evolutionary algorithm was used to evolve instances which lead to maximum performance difference of both solvers on the specific instances. Both directions are analyzed, i.e. we generated instances which are easier for solver A but much harder for solver B as well as the opposite case. In this regard we observed substantial differences in solver performance ratios compared to random or TSPLIB instances on the evolved instances. By feature-based analysis of the most extreme instances in terms of performance ratio crucial features are identified for both solver pairings which are indicated to have an influence on solver-specific problem difficulty. Moreover, we contrasted the classical mean par10 score with a respective median version to even increase the challenge of evolving instances with high solver performance differences.

Future studies will focus on generalizing the results to higher instance sizes and on designing a more sophisticated fitness function inside the EA to even increase solver performance differences on the evolved instances.

Acknowledgments. The authors acknowledge support by the European Research Center for 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. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS J. Comput. **25**, 346–363 (2013)
3. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math. Program. Comput. **1**, 119–163 (2009)
4. 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
5. Kotthoff, L.: Algorithm selection for combinatorial search problems: a survey. AI Mag. **35**, 48–60 (2014)
6. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: methods & evaluation. Artif. Intell. **206**, 79–111 (2014)
7. Smith-Miles, K., van Hemert, J.: Discovering the suitability of optimisation algorithms by learning from evolved instances. Ann. Math. Artif. Intell. **61**, 87–104 (2011)
8. 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. **69**, 1–32 (2013)
9. Pihera, J., Musliu, N.: Application of machine learning to algorithm selection for TSP. In: Fogel, D., et al. (eds.) Proceedings of the IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE press (2014)
10. 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)
11. 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)
12. 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)
13. 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) (accepted)
14. 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)
15. Kerschke, P., Dagefoerde, J.: flacco: Feature-Based Landscape Analysis of Continuous and Constraint Optimization Problems. R package version 1.1 (2015)