Evolving Diverse TSP Instances by Means of Novel and Creative Mutation Operators

Jakob Bossek
bossek@wi.uni-muenster.de
Dept. of Information Systems
University of Münster, Germany

Pascal Kerschke
kerschke@uni-muenster.de
Dept. of Information Systems
University of Münster, Germany

Aneta Neumann
aneta.neumann@adelaide.edu.au
School of Computer Science
The University of Adelaide, Australia

Markus Wagner
markus.wagner@adelaide.edu.au
School of Computer Science
The University of Adelaide, Australia

Frank Neumann
frank.neumann@adelaide.edu.au
School of Computer Science
The University of Adelaide, Australia

Heike Trautmann
trautmann@wi.uni-muenster.de
Dept. of Information Systems
University of Münster, Germany

ABSTRACT

Evolutionary algorithms have successfully been applied to evolve problem instances that exhibit a significant difference in performance for a given algorithm or a pair of algorithms inter alia for the Traveling Salesperson Problem (TSP). Creating a large variety of instances is crucial for successful applications in the blooming field of algorithm selection. In this paper, we introduce new and creative mutation operators for evolving instances of the TSP. We show that adopting those operators in an evolutionary algorithm allows for the generation of benchmark sets with highly desirable properties: (1) novelty by clear visual distinction to established benchmark sets in the field, (2) visual and quantitative diversity in the space of TSP problem characteristics, and (3) significant performance differences with respect to the restart versions of heuristic state-of-the-art TSP solvers EAX and LKH. The important aspect of diversity is addressed and achieved solely by the proposed mutation operators and not enforced by explicit diversity preservation.

CCS CONCEPTS

• **Theory of computation → Generating random combinatorial structures**; *Random search heuristics*;

KEYWORDS

Traveling salesperson problem, optimization, instance features, problem generation, benchmarking

ACM Reference format:
Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. 2019. Evolving Diverse TSP Instances by Means of Novel and Creative Mutation Operators. In *Proceedings of Foundations of Genetic Algorithms XV, Potsdam, Germany, August 27–29, 2019 (FOGA '19),* 14 pages.
https://doi.org/10.1145/3299904.3340307

FOGA '19, August 27–29, 2019, Potsdam, Germany

ACM ISBN 978-1-4503-6254-2/19/08…$15.00
https://doi.org/10.1145/3299904.3340307

1 INTRODUCTION

Over the years it has been shown that there (almost) never exists a single optimization algorithm that performs best across all optimization problems for a specific domain. Instead, there usually exist multiple algorithms with complementary strengths (and weaknesses) that form the state of the art in their respective domain [33]. By means of selecting the best algorithm for the problem at hand (based on a set of informative features characterizing the respective problem) one can exploit the complementarity of the solvers, which in turn enables to leverage the state of the art. Although this selection procedure – commonly denoted *Algorithm Selection Problem* – has been studied for many decades [27], it has attracted more and more interest in recent years [13, 15, 29].

In order to capture and ideally also understand the strengths and weaknesses of different optimization algorithms, it is very important to have a set of diverse benchmark problems. Here, diversity is relevant with respect to a variety of aspects:

- performance space: in order to analyze differences in solver behavior, one needs problems that are hard for one solver, but easy for its contender,
- feature space: features are an essential ingredient for the distinction of solvers; thus, test problems with diverse feature sets facilitate the modeling of accurate algorithm selectors,
- topology: benchmark problems with diverse topologies intuitively are more likely to represent problems that could also occur in real-world applications.

As noted above, understanding a solver's strengths and weaknesses requires test problems that are rather easy for solver A, but hard for solver B (and vice versa). In earlier works, which exemplarily focused on the *Traveling Salesperson Problem* (TSP), researchers used an *Evolutionary Algorithm* (EA) to (successfully) generate TSP instances, which are diverse in the performance space, by maximizing the performance ratio between their two considered solvers A and B [4, 5]. Unfortunately, although the generated instances resulted in diverse solver performances, their topologies and thus also their features still resembled very similar structures.

Meanwhile, other research groups proposed an approach, which incorporates diversity preservation/maximization techniques within the instance generating process. In the context of evolutionary diversity optimization, Gao et al. [8] used a diversity measure that

calculates the contribution of each individual with respect to different features. Here the contribution of an individual (instance) to the diversity with respect to a considered feature depends on the (non-)existence of similar feature values within the population.

In cases in which more than one feature is considered, a weighted sum approach is used to determine the diversity contribution of an individual to the population. This approach has also been applied for evolving diverse sets of images with respect to common image features [1, 24]. However, the weighted sum approach often does not lead to a good diversity distribution if two or more features are used. Therefore, investigations have recently been expanded towards other diversity measures, which are capable of dealing with multiple features. This includes using the discrepancy measure to compute a diverse set of solutions [22], as well as the use of popular multi-objective indicators [23].

The aforementioned EA then evolves a set of instances that maximizes diversity with respect to important features of the underlying problem while simultaneously meeting a performance condition – the ratio of the solvers' performances needs to be above a given threshold.

Although the latter approach successfully preserved diversity in feature space, it strongly impeded the instance generating EA due to the underlying bi-level approach, i.e., iteratively focusing on feature diversity and meeting the performance condition. Moreover, it has so far only been applied to TSP instances consisting of 50 cities. Thus, it is unclear how it will perform on larger instances.

Within this work, we tackle the problem of creating diverse sets of instances – exemplarily shown with a focus on the TSP – by proposing a set of sophisticated, problem-tailored mutation operators that – in contrast to established operators – have a stronger impact on the point coordinates of nodes in the Euclidean plane. We show that a simple iterative, non-evolutionary process is sufficient to produce instances that are quite diverse and look different to well-known and frequently used artificial TSP instances, e.g., Random Uniform Euclidean (RUE) or NETGEN (strongly clustered), and are more similar – from a visual perspective – to, e.g., instances stemming from real-world VLSI applications. In addition, the instances generated with our proposed approach also show a much better spread with respect to many problem features. In summary, our work contributes to all three aforementioned diversity aspects:

- we enable the generation of TSP instances with various topologies and thereby hopefully also create more real-world-like structures,
- we show that these instances also improve the diversity with respect to multiple features, and
- the resulting instances clearly reveal complementary performances of two very prominent TSP solvers.

At last, we want to emphasize that our work so far mainly constitutes a feasibility study. More precisely, we provide a new instance generator and will show that it can be used for efficiently evolving diverse problem instances.

The remainder of this paper is structured as follows. Section 2 provides an overview of previous work on algorithm selection and diversity preservation for the TSP. In Section 3, we introduce an iterative and an evolutionary approach for creating diverse sets of TSP instances. The existing and the novel mutation operators underlying our proposed instance generators are also described therein. An analysis of the instances created by our iterative and evolutionary approaches (with respect to feature as well as performance diversity) is given in Sections 4 and 5, respectively. Section 6 concludes our work and poses perspectives for future work.

2 BACKGROUND

As the methods proposed within this work are consistently illustrated by means of a case study on the TSP, we will first recap the aim of this well-studied research problem. Afterwards, we provide a brief summary of two very promising developments on the TSP from recent years: algorithm selection and diversity maximization.

2.1 Traveling Salesperson Problem

Given a set of nodes and pairwise distances between them, a fundamental $\mathcal{NP}$-hard optimization problem in graph theory is to find a round-trip tour of minimal traveling distance that visits each node exactly once and returns to the starting node. This problem is commonly known as the Traveling Salesperson Problem (TSP) and has countless obvious applications, e.g., in vehicle-routing and manufacturing of integrated circuits, as well as numerous less obvious applications, e.g., in computational biology [9]. If nodes are given by points in the Euclidean plane and distances correspond to pairwise Euclidean distances, the problem is called Euclidean TSP (and remains $\mathcal{NP}$-hard).

Due to the broad interest in the TSP, years of research endeavours resulted in many well-performing algorithms. In the field of exact algorithms, i.e., algorithms that guarantee to find an optimal solution, the sophisticated branch-and-cut based Concorde solver is the state of the art [2]. Due to its worst-case exponential runtime it is not well-suited in situations where a good near-optimal solution is required within a stipulated time limit or the instance size is large. In inexact TSP solving the state of the art is given by two algorithms: LKH [10], a local-search based algorithm which implements the Lin-Kernighan heuristic for k-opt moves, and the evolutionary algorithm EAX [21], which adopts an efficient edge-assembly-crossover operator. It was shown that these two algorithms are highly efficient and solve even large instances in reasonable time (at times within seconds). Respective restart versions of EAX and LKH, which trigger a restart once the original stopping condition is met, are even more successful [7, 14, 16].

Recent studies also investigated the effects of incorporating a generalized partition crossover (GPX) operator within EAX and LKH [17, 28, 31]. However, based on the results shown therein, as well as some preliminary studies that we conducted ourselves, we found that these enhancements are only beneficial for much larger TSP instances (consisting of more than 8 000 cities). We therefore decided to restrict ourselves to the aforementioned restart versions of EAX and LKH. In fact, given the superior performances of the restart versions compared to their vanilla implementations, we will only use the restart versions. And to facilitate readability, we will (from now on) simply refer to them as EAX and LKH throughout this work.

2.2 Algorithm Selection on the TSP

An area which gained considerable attention in recent years is per-instance algorithm selection (AS) [13, 15, 29]. Despite the very strong performances of EAX and LKH across common TSP benchmarks, algorithm selection has nonetheless been able to leverage the performance of the state of the art in inexact TSP solving [13, 14, 16].

The usual setup of TSP-related AS studies can be summarized as follows: given a predefined cutoff time (of usually one hour), one executes a portfolio of (inexact) TSP solvers on a collection of (ideally diverse) TSP instances with known optimal tour lengths – which usually have been verified upfront by Concorde. Each algorithm runs at most until the given cutoff time has expired and terminates either successfully – if it has found a tour of optimal length before the cutoff time was reached – or unsuccessfully. The time needed by the solver to meet one of the two termination criteria, and the accompanying runstatus (successful vs. unsuccessful) of the solver are then aggregated (across all instances) within a scalarized performance indicator such as PAR10, PQR10 or (more recently) the multi-objective Hypervolume indicator [3, 6, 12].

In addition to measuring the solver performances on the TSP sets, one also needs to grasp characteristics of the problem instances by means of ideally quickly computable, yet very informative numeric features [11, 13, 20, 26, 30]. In case of the TSP, features usually aggregate information based on an instance's distance matrix, minimum spanning tree, nearest neighbor graph, convex hull, or distance between clusters of the cities.

Per-instance algorithm selection models then use machine learning to find a mapping – usually a classification model – from the features to the best TSP solver for the corresponding instance. However, the most recent studies also revealed a weakness within the current setup: EAX and LKH achieved very strong and often very similar results across the entire benchmark. This indicates that the currently used benchmark problems are likely too easy for both solvers and – more importantly – their topologies are not sufficiently diverse too expose strengths and weaknesses of the two solvers.

2.3 Diversity

Motivated by earlier work of Ulrich and Thiele [32], who considered diversity of high quality solutions in the search space, the concept of evolutionary diversity optimization has been used in [8] to evolve TSP instances based on given problem features. In particular, this approach evolves instances that are hard or easy to be solved by a given algorithm. In [8], diversity is measured according to a weighted distribution with respect to differences in feature values.

Recently, the notion of *star discrepancy* [25] has been used in the context of evolutionary diversity optimizaton [22]. The star discrepancy measures the diversity of a set $X := \{X_1, \ldots, X_n\}$ with respect to all axis-parallel boxes $\left[\vec{0}, \vec{b}\right]$, $\vec{b} \in [0,1]^d$. Here d is the number of features that are considered and we assume that feature values are normalized to $[0, 1]$ to carry out the diversity measurement. Formally, the star discrepancy of X is given by

$$D(X) := \sup \left\{ \mathcal{VOL}([\vec{0}, \vec{b}]) - \frac{\left|X \cap [\vec{0}, \vec{b}]\right|}{n} \;\middle|\; \vec{b} \in [0,1]^d \right\},$$

Algorithm 1 TSPGEN-ITER (n, M, iters)

1: x = set of n points placed uniformly at random in $[0, 1]^2$
2: **for** $i = 1 \rightarrow$ iters **do**
3: Choose a mutation operator $m \in M$ uniformly at random.
4: $x = m(x)$
5: **return** x

where $\mathcal{VOL}([\vec{0}, \vec{b}])$ is the volume of the d-dimensional hyperbox given by $\left[\vec{0}, \vec{b}\right]$. The goal is to compute a set X of n elements that has minimal discrepancy.

3 ON RANDOM INSTANCE GENERATION

The instance generation approach that we employ in the following is a stochastic one. In a nutshell, n points are placed uniformly at random in the Euclidean plane, more precisely $[0, 1]^2$, and moved around by iterative application of stochastic mutation operators until some stopping criterion is met. We consider two instance generation algorithms: (1) a simple sequential generation process and (2) an evolutionary approach. Next, we first describe the algorithms before we discuss their underlying mutation operators.

Prior to this, let us briefly introduce some terminology. Given an instance size $n \in \mathbb{N}$ we denote by $\vec{\pi}_1, \ldots, \vec{\pi}_n \in [0, 1]^2$ the n node/city/point coordinates.[1] We identify a TSP instance with a set of points $\Pi = \{\vec{\pi}_1, \ldots, \vec{\pi}_n\}$. The instance generation process is a stochastic mapping that produces such a point set given an instance size and a bunch of mutation operators M along with their respective parameters. Likewise, each mutation operator $m \in M$ is a mapping $m : \Pi_n \rightarrow \Pi_n$ where Π_n is the set of all possible point clouds of n points in $[0, 1]^2$. Further we denote by $\vec{0} \in \mathbb{R}^2$ the two-dimensional zero-vector (or origin).

3.1 Algorithms

Our iterative instance generation process initializes a Random Uniform Euclidean (RUE) instance in a first step and iteratively applies a randomly chosen mutation operator until a predefined number of iterations is reached (see Alg. 1).

Besides the simple iterative generation process we consider an EA for targeted generation of instances (see Alg. 2), which are easy for one solver A and hard for another solver B with respect to the ratio of PAR10-scores similar to approaches presented by [4, 5, 19]. More precisely, the algorithm maintains a population P of $|P| = \mu$ instances initialized with RUE instances. In the evolutionary loop each individual $x \in P$ is selected uniformly at random and likewise a mutation operator $m \in M$ is chosen. After mutation, the fitness of the new individual $y = m(x)$ is determined and survival selection following a $(\mu + 1)$-strategy assures that the individual with the poorest fitness is dropped. It should be noted that in contrast to work conducted in the field of tailored instance generation for the TSP our algorithm is simplified: (1) it relies on mutation as the exclusive variation operator (in contrast to, e.g., [20]), and (2) it does not include any built-in diversity preservation mechanism (in contrast to, e.g., [8]). However, based on our observations from

[1] The process operates inside the $[0, 1]^2$ bounding-box for convenience, but this does not pose a restriction since generated instances can easily be scaled, shifted or rotated arbitrarily by standard matrix operations.

Algorithm 2 TSPGEN-EA (n, μ, M)

1: Generate a population P of size μ; each individual is a set of n points placed uniformly at random in $[0,1]^2$
2: **while** termination criterion not satisfied **do**
3: Choose $x \in P$ at random
4: Choose a mutation operator $m \in M$ uniformly at random.
5: $y = m(x)$
6: $P = P \cup \{y\}$
7: $P = P \setminus \arg\max_{x \in P} f(x)$
8: **return** $\arg\min_{x \in P} f(x)$

exemplary runs of both algorithms we are confident that this does not constitute a disadvantage. We will support these assumptions with empirical evidence in Sections 4 and 5. Implementations of algorithms and mutation operators are wrapped in the software package `tspgen`.[2]

The fitness of an individual $x \in P$ is the ratio of PAR10-scores for two solvers A and B on x and we minimize this ratio. Hence, the algorithm prefers individuals which are easy for algorithm A and (more) difficult for algorithm B. For clarity: the PAR10-score of an algorithm A is the average running time of A on the instance at hand. However, the running time of unsuccessful runs is replaced with $f \cdot T$, where T is the predefined cutoff time and f is a penalty factor for failed runs, which is usually set to $f = 10$ [3]. This requires a single run of the exact Concorde solver to calculate the true optimal tour length in order to determine the runstatus.

3.2 Mutation Operators

Next, we introduce the considered mutation operators in detail. The reader is recommended to take a look at Fig. 1 while studying this section as it presents a visual support for the detailed formal explanations.

At this point it should be noted that some mutation operators may move points outside the boundaries $[0,1]^2$. These boundary-violating points are then simply replaced uniformly at random within the bounding-box.

3.2.1 Existing Simple Mutation Operators. To the best of our knowledge, Mersmann et al. [20] were the first authors who approached targeted instance generation of Euclidean TSP instances by means of evolutionary algorithms. Their EA adopted the following two straightforward mutation operators.

Normal Mutation. A random subset of nodes $Q \subseteq \Pi$ is selected at random (each node independently with probability $p_m \in (0,1)$). Next, each selected point $\vec{\pi} \in Q$ is subject to Gaussian perturbation, i.e.,

$$\vec{\pi}' = \vec{\pi} + \mathcal{N}\left(\vec{0}, \Sigma\right) \text{ with } \Sigma = \text{diag}(\sigma, \sigma) = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix}$$

with a predefined standard deviation $\sigma > 0$.

Uniform Mutation. Another very simple mutation operator that, e.g., has been used in [20]. Each point $\vec{\pi} \in \Pi$ is replaced with probability $p_m \in (0,1)$ by $\vec{\pi}' = (\pi_1', \pi_2')^T$, where $\pi_i' \sim \mathcal{U}[0,1]$.

[2] GitHub code-repository: https://github.com/jakobbossek/tspgen

3.2.2 New Sophisticated Mutation Operators. Both aforementioned mutation operators, normal and uniform mutation, affect the instance under construction only marginally; even if a lot of nodes are subject to mutation (as performed in Fig. 1 for visual clarity) the mutant is – at least purely visually – indistinguishable to its parent. This holds true after multiple iterations and the effect is even intensified with growing instance size. These observations were the starting point and the motivation for the creative design of new, more sophisticated and problem tailored mutation operators. The ideas are based on observations of real-world instances, above all instances from applications in Very Large Scale Integration (VLSI), where we are often confronted with dense groups of nodes, areas with very few or even no nodes at all and nodes arranged along a line or in grid structures.

Explosion Mutation. This operator is meant to tear holes into the point cloud. This is achieved by simulating a random explosion. First, we select the *center of explosion* $\vec{c} \in [0,1]^2$ and the *explosion radius* or *explosion strength* $r \in \mathcal{U}[r_{\min}, r_{\max}]$ at random. All points $\vec{\pi} \in \Pi$ within the radius r of the center $\vec{c}$, i.e., points with a Euclidean distance of $d(\vec{\pi}, \vec{c}) \leq r$, are affected by the explosion. These points are moved away from the center of explosion as follows:

$$\vec{\pi}' = \vec{c} + (r + s_i) \cdot \frac{\vec{c} - \vec{\pi}}{||\vec{c} - \vec{\pi}||},$$

where $s_i \sim \text{Exp}(\lambda = 1/10)$ is a random value sampled from an exponential distribution defining the impact of the explosion (outside of the explosion radius r) on each of the individual points. The formula ensures, that all points inside the explosion radius are pushed out of the explosion area and – depending on the magnitude of s_i – ride the detonation wave even a little further. Note that due to the random location and strength of the explosion, points may be moved outside the feasible space $[0,1]^2$. Here the repairing mechanism described earlier takes care of these points.

Implosion Mutation. This operator is kind of the inverse to the explosion mutation and hence its setup is basically identical: sample a *center of implosion* $\vec{c} \in [0,1]^2$ (internally termed the *black hole*) and an *implosion radius* $r \in \mathcal{U}[r_{\min}, r_{\max}]$ and shift all points located within the implosion region, i.e., points $\vec{\pi} \in \Pi$ with a distance $d(\vec{\pi}, \vec{c}) \leq r$, closer towards the implosion center, i.e.,

$$\vec{\pi}' = \vec{\pi} + (\vec{c} - \vec{\pi}) \cdot \min\{|\tau|, r\}.$$

Here, $(\vec{c} - \vec{\pi})$ is the direction vector between implosion center $\vec{c}$ and point $\vec{\pi}$, and $\tau \sim \mathcal{N}(0,1)$ is a random number which stems from a standard normal distribution.

Cluster Mutation. This operator introduces dense clusters of points by performing the following steps. First, we sample a *cluster center* $\vec{c} \in \mathcal{U}[0,1]^2$ at random. Next, a set of points $Q \subseteq \Pi$ are selected at random and replaced by an identical number of points, which are randomly drawn from an uncorrelated bivariate normal distribution $\mathcal{N}(\vec{c}, \text{diag}(\sigma, \sigma))$. Here $\sigma \sim \mathcal{U}[0.001, 0.3]$ determines the compactness of the generated cluster.

Rotation Mutation. A subset of points $Q \subseteq \Pi$ is sampled uniformly at random. Let $\Pi_Q \sim (|Q|, 2)$ denote the $|Q|$ times two matrix of point coordinates of the selected points. The mutation

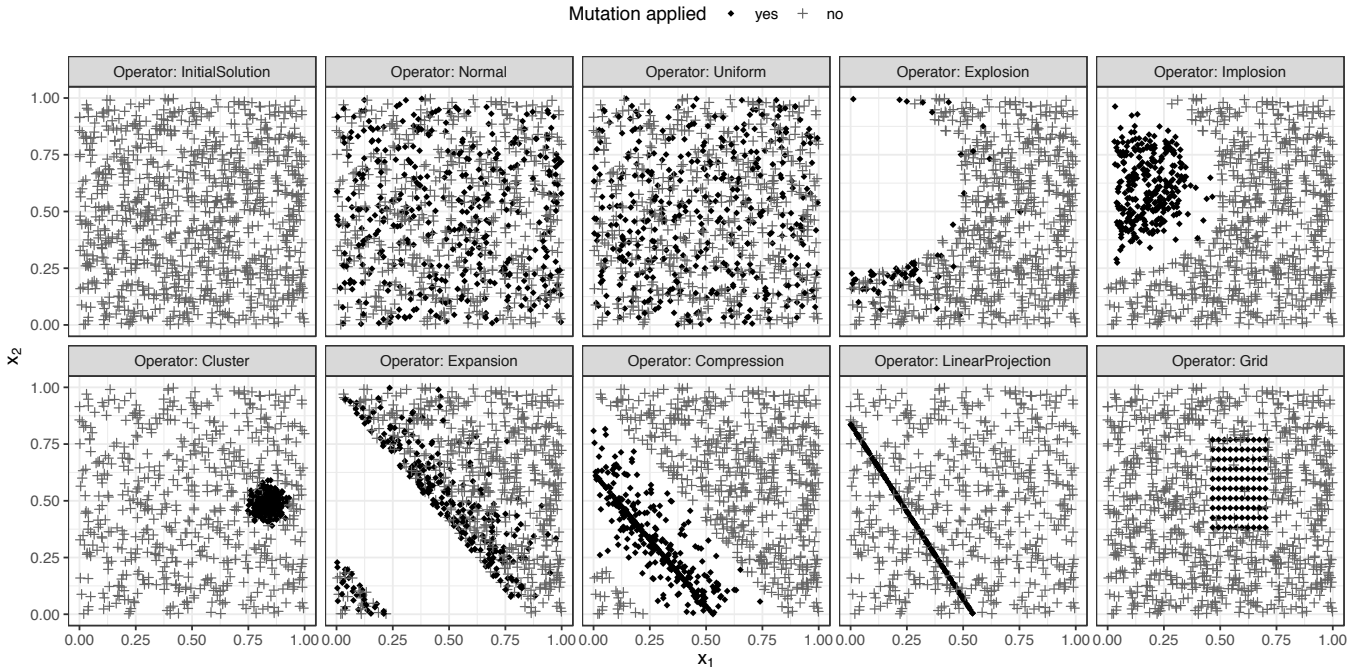

Figure 1: Visualization of considered mutation operators. The leftmost plot in the top row shows the initial solution used for each mutation. Points $Q \subseteq \Pi$ affected by mutation are shown as filled black diamonds while the remaining (untouched) points $\Pi \setminus Q$ are shown as gray crosses. Note that for demonstration purposes points which are moved beyond the boundaries $[0,1]^2$ by mutation (e.g., by explosion) are not "repaired" for clarity and hence not visible in the above plots since occasionally the repairing mechanism obscures the impact of mutation.

operator performs a rotation, i.e.,

$$\Pi'_Q = \Pi_Q \cdot R \text{ with } R = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

where $\alpha \in \mathcal{U}[0, 2\pi]$ is a randomly selected rotation angle. As the rotation is anchored in the origin of the Euclidean plane, the rotated set of points is afterwards shifted to a uniformly at random selected point within $[0,1]^2$.

Linear Projection Mutation. A subset of points $Q \subseteq \Pi$ is chosen at random. Next, a random linear function $f(x) = \beta_0 + \beta_1 \cdot x$ is sampled. Here, β_0 stems from a $\mathcal{U}[0,1]$ distribution and β_1 is sampled from $\mathcal{U}[-3,0]$ if $\beta_0 < 0.5$ and from $\mathcal{U}[0,3]$ otherwise. This distinction of cases ensures that the linear function runs inside the bounding box $[0,1]^2$ with high probability. In a final step the operator sets $\pi'_1 = \pi_1$ and $\pi'_2 = f(\pi_1) = (\beta_0 + \beta_1 \cdot \pi_1)$ for each $\vec{\pi} = (\pi_1, \pi_2)^T \in Q$, i.e., the mutants will be projected to the linear function. Finally, with probability $p_{\text{jit}} \in [0,1]$ Gaussian noise with a mean of zero and a standard deviation of σ_{jit} is added to the second coordinate π'_2 for each mutated point independently.

Expansion Mutation. This operator combines the ideas of the explosion and linear projection mutation. In fact, one could imagine placing a tube around a linear function and all points within that tube will be (orthogonally) pushed out of that region. As this operator also acts on linear functions, its first steps are identical to the previous operator: randomly sample a subset of potential mutants $Q \subseteq \Pi$ and a linear function $f(x) = \beta_0 + \beta_1 \cdot x$ as described before. In order to push the points orthogonally away from the linear function, we first compute the orthogonal projection

$$\vec{q} = \vec{b} + \left((\vec{\pi} - \vec{b})^T \cdot \vec{u}\right) \cdot \frac{\vec{u}}{||\vec{u}||}$$

for each of the mutants $\vec{\pi} \in Q$ to the linear function $f(x)$. Here $\vec{b} = (0, \beta_0)^T$ is a reference point on and $\vec{u} = (1, \beta_1)^T$ the direction vector of the linear function. The mutants are then pushed away from their orthogonal projections – and hence also from the linear function – as follows:

$$\vec{\pi}' = \vec{q} + (w + s_i) \cdot \frac{\vec{\pi} - \vec{q}}{||\vec{\pi} - \vec{q}||},$$

where $w \in \mathcal{U}[w_{\min}, w_{\max}]$ is the (uniformly at random drawn) width of the tube surrounding the linear function and $s_i \sim \text{Exp}(\lambda = 1/10)$ is a random value providing a small additional impact of the explosion on the affected individuals.

Compression Mutation. Complementing the expansion mutation, this operator squeezes a set of randomly selected points $Q \subseteq \Pi$ from within a tube (surrounding a linear function) towards the tube's central axis. The mechanics of this mutation operator are completely identical to the expansion mutator except for the final transformation. That is, the compressed points are computed as

$$\vec{\pi}' = \vec{q} - (\vec{\pi} - \vec{q}) \cdot \min\{|\tau|, 1\},$$

where $(\vec{\pi} - \vec{q})$ is the direction vector from point $\vec{\pi}$ to its orthogonal projection $\vec{q}$ and $\tau \in \mathcal{N}(0,1)$ is a standard normal distributed value defining the compression strength.

Axis Projection Mutation. This is a special case of linear projection mutation, where the linear function is axis-parallel with probability 1. An axis $j \in \{1,2\}$ and a value $c \sim \mathcal{U}[0,1]$ are sampled uniformly at random. Note that $j = 1$ corresponds to the x-axis and $j = 2$ to the y-axis. Next, for a random subset $Q \subseteq \Pi$ we set $\pi_j = c$ for all $\vec{\pi} = (\pi_1, \pi_2)^T \in Q$, i.e., we replace the x- or y-values (depending on the sampled choice of a) for all points in $\vec{\pi} \in Q$ by c. Afterwards, with probability p_{jit} the mutated points are subject to normal perturbation, i.e., $\vec{\pi}' = \vec{\pi} + \mathcal{N}(\vec{0}, \text{diag}(\sigma, \sigma))$ with $\sigma > 0$.

Grid Mutation. The width w and height h of a box are drawn uniformly at random from the $\mathcal{U}(b_{\min}, b_{\max})$-distribution. Here, $b_{\min}$ and $b_{\max}$ are parameters of the mutation operator which determine the possible range of box-width and -height, respectively, for sampling. Next, the box is placed at a random location in the Euclidean subspace $[0,1]^2$. Upon placement, all points $\vec{\pi} \in Q \subseteq \Pi$ located inside the box are subject to mutation. Q is replaced with a regular quadratic grid of points with dimension $d = \lfloor \sqrt{|Q|} \rfloor$. Note that the alignment of the points within the grid is quadratic, but this does not imply that the horizontal and vertical distances between neighbors are identical. Further, if $d^2 < |Q|$, a subset $Q_{\text{sub}} \subset Q$ is sampled uniformly at random (and without replacement), whose points are then aligned in the aforementioned quadratic grid structure. All the remaining points of $Q \setminus Q_{\text{sub}}$ remain untouched. Afterwards, the entire grid is rotated with probability p_{rot}, which is another parameter of the mutation operator. The rotation is performed by a randomly selected angle $\alpha \in \mathcal{U}[0, \pi/2]$ and it is anchored in the grid's center. At last, the mutated points will be normally perturbed, i.e., $\vec{\pi}' = \vec{\pi} + \mathcal{N}(\vec{0}, \text{diag}(\sigma, \sigma))$, with probability p_{jit} and $\sigma > 0$.

4 ANALYSIS OF GENERATED INSTANCES

In this section we experimentally evaluate the suitability of the sophisticated mutation operators to generate a diverse set of instances. Here, we focus on the iterative generation process.

First, in order to illustrate the working principle of the iterative generation process, Alg. 1 was run for 250 iterations. Exemplary results are shown in Fig. 2, where M either is comprised of only simple operators (normal or uniform mutation) or is the group of introduced sophisticated operators. It becomes obvious that the sophisticated operators lead to instances of varying topologies and structures at all stages of the generation process (bottom row of Fig. 2), whereas usage of the simple operators (top row) does not substantially alter the random distribution of points.

The differences between the instance topologies are also reflected in feature space as well, as illustrated in Fig. 3. The feature space is exemplarily spanned by the two features *hull ratio* and *number of strong connected components in the nearest neighbor graph (nng)* which proved to be very informative in previous studies [20, 26]. The iterative instance generation process is tracked in feature space, i.e., each colored dot represents the topology of a generated instance at a specific iteration. The previously described instances in Fig. 2 are specifically labelled reflecting the $50^{th}, 100^{th}, \ldots, 250^{th}$ iteration. A huge increase in feature space diversity, especially in terms of the *nng* feature, is clearly visible in case the sophisticated mutation operators are used for generating the instances.

4.1 Experimental Setup

For a systematic investigation of the mutation operators, we generated a benchmark set – denoted TSPGEN in the following – comprising of 600 TSP instances in total, which corresponds to 150 instances per instance size $n \in \{500, 1\,000, 1\,500, 2\,000\}$. Each instance is the result of 1 000 iterations of Alg. 1. For comparison we consider each 600 instances of three artificial instance sets stemming from a recent study on per-instance algorithm selection for the TSP [14]: Random Uniform Instances (RUE), instances with distinct clusters structures (NETGEN) and instances which originate from "morphing" pairs of RUE and NETGEN instances of equal size (called MORPHED in the following).[3] Fig. 5 plots embeddings of (randomly chosen) exemplary instances from the considered sets (RUE, NETGEN, MORPHED in the first row) and five TSPGEN instances of sizes 500, 1 000 and 2 000 (second to fourth row).

From a purely visual inspection the human eye can detect obvious differences between the established benchmarks and the new benchmark set. Recalling the effects of the different mutation operators one can easily recognize the resulting structures and combinations of them in the TSPGEN instances. Moreover, in particular the linear/axis projection and grid mutation operators – inspired by observations on real-world TSP instances stemming from applications in Very Large Scale Integration (VLSI) – lead to much stronger (visual) similarities to VLSI instances (see top-right plot in Fig. 5 for an example of a real-world VLSI instance[4]). We are confident that a restriction to a subset of the new operators and tweaking their parameters may result in instances that are even more similar to VLSI. Albeit this is a promising research question, we consider it out of scope for this work.

4.2 Impact on Feature Space

As expected, the new instance set TSPGEN achieved a much higher diversity in feature space than the already existing TSP sets RUE, NETGEN or MORPHED. This is confirmed by Fig. 4, which shows exemplary embeddings of all considered instances in the two-dimensional feature space for a selection of informative TSP features. Moreover, our findings are not only based on visual comparisons, but can also be quantified by means of discrepancy values (listed in Tab. 1). Noticeably, for the majority of considered feature pairs $P1$ to $P4$ and instance sizes the discrepancy values for TSPGEN are consistently the lowest in comparison with the other three TSP sets with a single exception. This observations holds true for the majority of features across the considered feature sets.

4.3 Comparison of Algorithm Performance

In addition to the diversity in instance space – more precisely in topology and feature space – solver performances on the newly generated instances are of crucial interest. For the purpose of performance comparison we rely on the well-known PAR10-score [3] and thus first determine the optimal tour length per instance by

[3] Morphing is designed to create instances "in-between" two instances of equal size by building convex combinations of points [18, 20].
[4] Source: http://www.math.uwaterloo.ca/tsp/vlsi/

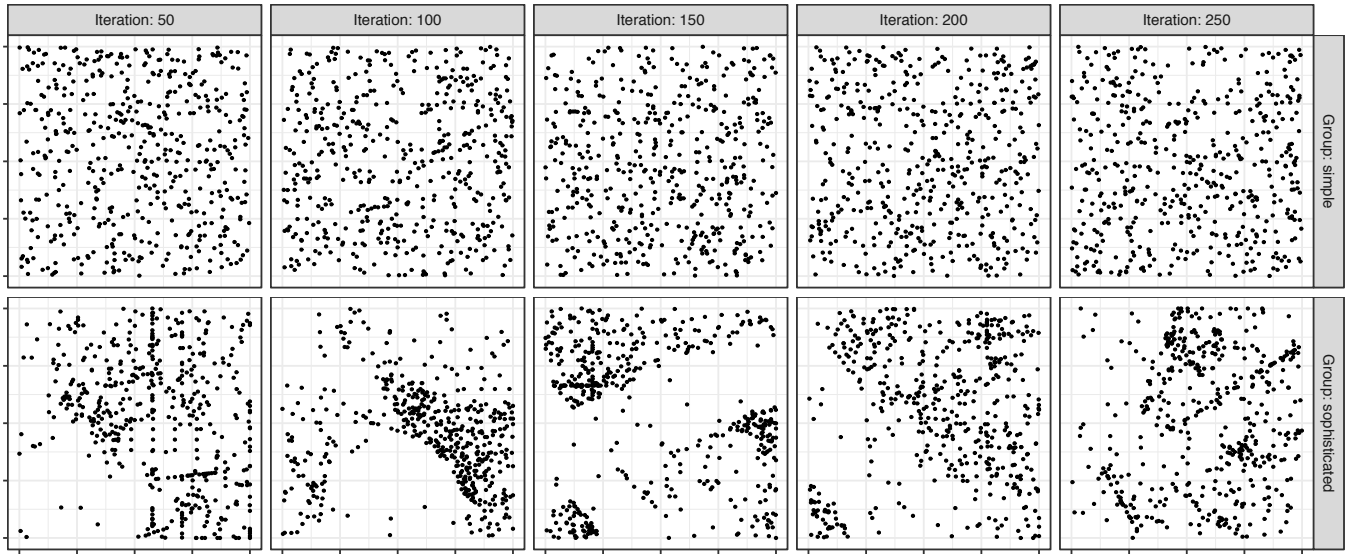

Figure 2: TSP instances at iterations 50, 100, 150, 200 and 250 generated during an exemplary generation process with only simple mutation operators (uniform and random mutation; top row) and the set of introduced sophisticated operators (bottom row). See Fig. 3 for a visualization of some exemplary feature vectors.

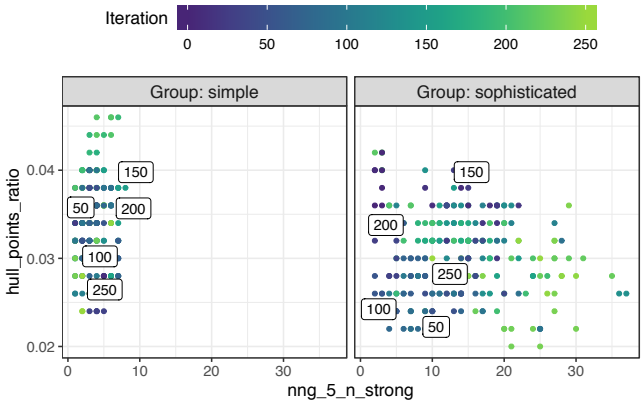

Figure 3: Visualization of the iterative instance generation process ($n = 500$ nodes) in the two-dimensional feature subspace spanned by *ratio of nodes on the convex hull* and *number of strong connected components in the (five) nearest neighbor graph*. Each point represents the feature vector of the instance at iteration $i \in \{1, \ldots, 250\}$. In the left plot only simple mutation operators (uniform and normal) were used, while the instances in the right plot are based on the proposed (sophisticated) mutation operators. See Fig. 2 for exemplary instances corresponding to the labeled feature vectors.

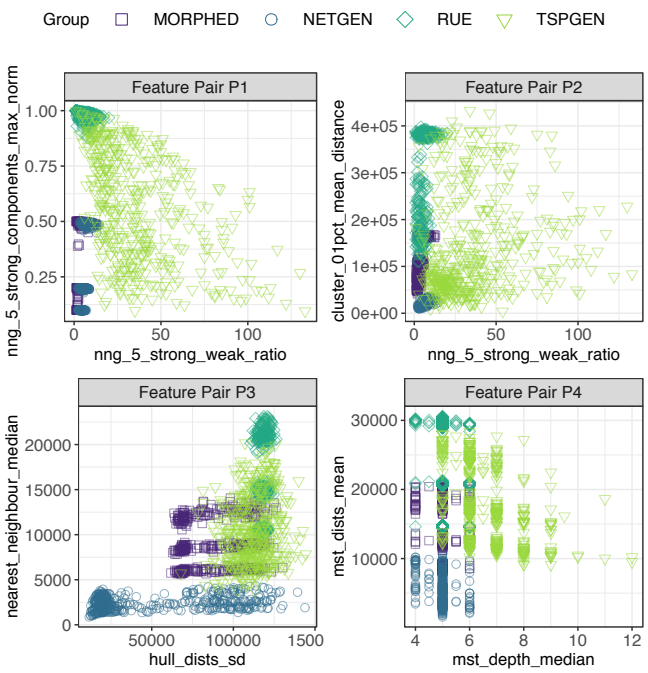

Figure 4: Two-dimensional embeddings of all instances in the feature space shown for four exemplary feature pairs. The shape and color of an instance indicate the corresponding TSP set.

a single run of Concorde. Then, EAX and LKH are both run 10 times on each instance until they either found an optimal tour or reached the cutoff time of one hour (= 3 600 seconds).

Fig. 6 shows a scatterplot in which EAX and LKH are compared in terms of PAR10. Noticeably, neither the established nor the newly generated TSP sets pose severe challenges for EAX as none of the PAR10 scores comes close to the cutoff time of one hour. Contrarily, several of such incidences can be observed for LKH. However, as reflected by Fig. 7 none of the differences in solver behaviour

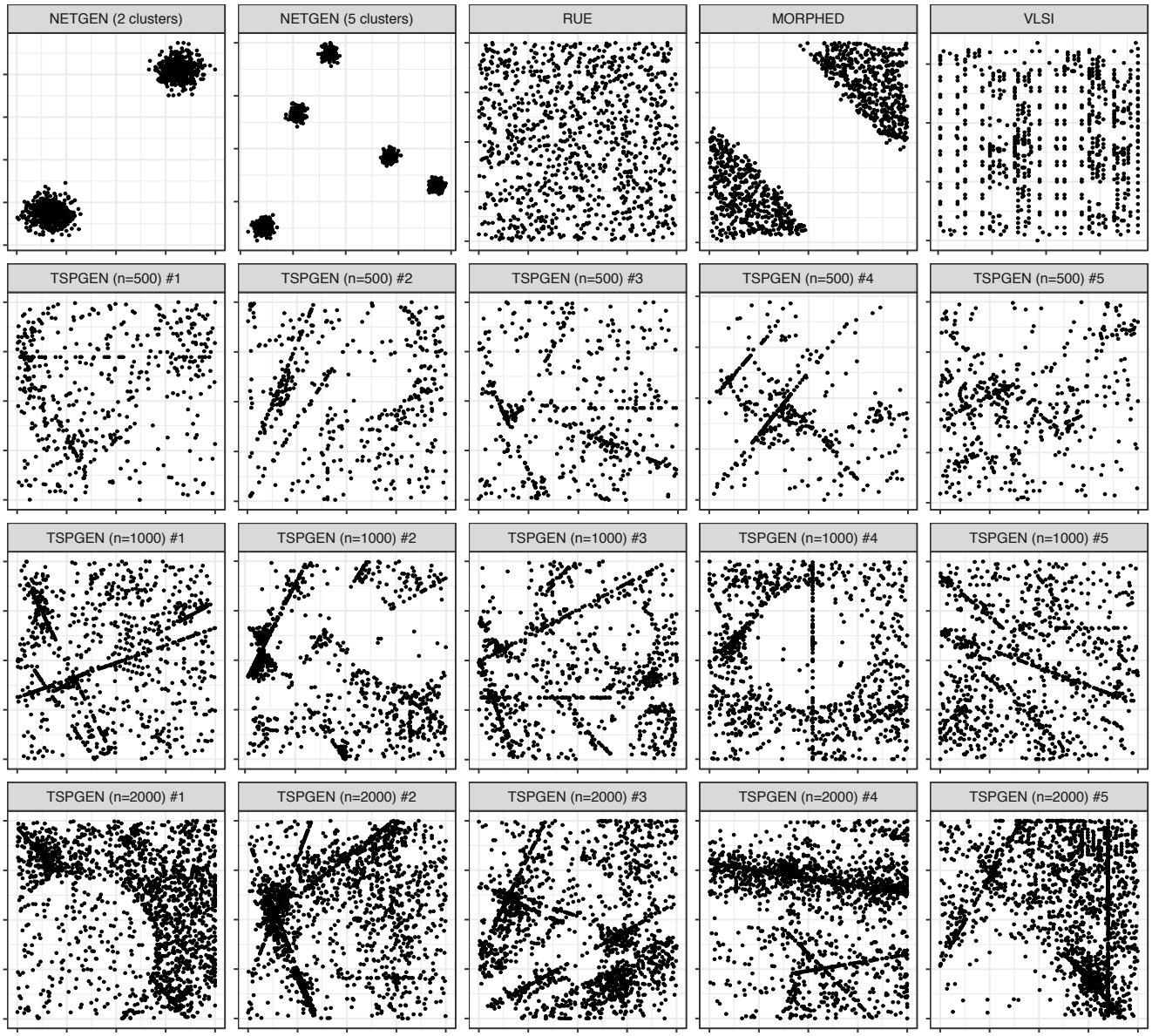

Figure 5: Exemplary TSP instances. The top row depicts randomly drawn examples for RUE, NETGEN, MORPHED and VLSI (from left to right), while the remaining rows show examples of TSPGEN instances with $n = 500$, $n = 1\,000$ and $n = 2\,000$ (second to fourth row), respectively.

appears to be substantial – independent of the considered TSP set or instance size – as all boxplots are rather narrow and most of them are located close to the dashed horizontal line indicating equal solver performances.

Therefore, diversity in instance space itself does not necessarily impact solver performance and specifically does not automatically allow for differentiating between contrasting behavior of EAX and LKH. Instead, the instance generation process rather has to be tailored to the latter requirement, i.e., diverse and structured instances with contrasting solver performances resulting in low or high solver ratios, respectively, are desired. Section 5 successfully addresses this issue by introducing an evolutionary algorithm for this purpose which relies on the sophisticated mutation operators.

5 EVOLVING INSTANCES

As the results of Section 4 have shown, our proposed mutation operators already enable the creation of instances with different topologies and diverse feature spaces. However, in order to learn more about the strengths and weaknesses of the TSP solvers, the instance generator has to be modified such that the produced instances are also diverse in performance space. For this purpose, we

Table 1: Discrepancy values per TSP set (NETGEN, MORPHED, RUE and TSPGEN) based on four different feature pairs. The best values per feature pair and instance size are highlighted by a grey background and shown in bold face.

Feature Pair	n	TSP Set NETGEN	MORPHED	RUE	TSPGEN
P1	500	0.963701	0.984723	0.938320	**0.891051**
	1000	0.965516	0.979235	0.944619	**0.876623**
	2000	0.965124	0.976613	0.942641	**0.799466**
	all	0.938831	0.963547	0.912327	**0.777744**
P2	500	0.999999	0.999996	0.999981	**0.999954**
	1000	0.999997	0.999992	0.999968	**0.999796**
	2000	0.999991	0.999987	0.999954	**0.999647**
	all	0.991710	0.999987	0.999953	**0.743355**
P3	500	0.868605	**0.543763**	0.778247	0.544127
	1000	0.919629	0.704024	0.726990	**0.553577**
	2000	0.947947	0.795681	0.736480	**0.656737**
	all	0.869564	0.550744	0.718269	**0.538920**
P4	500	0.925268	0.845305	0.878857	**0.634727**
	1000	0.957370	0.899190	0.837975	**0.644285**
	2000	0.977442	0.951574	0.890660	**0.721881**
	all	0.926310	0.845305	0.781557	**0.518082**

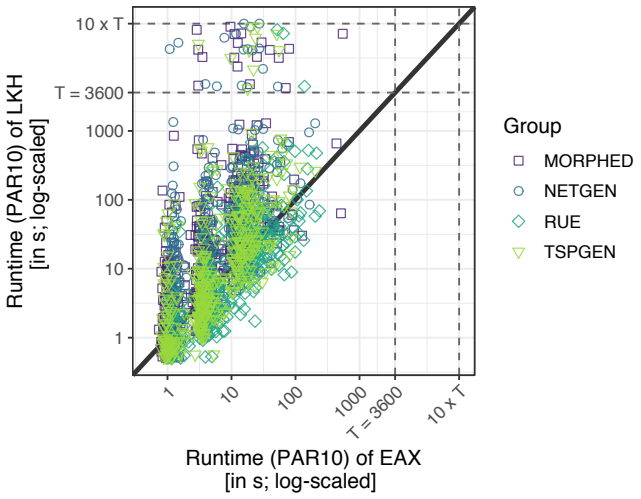

Figure 6: Scatterplots of mean running times for EAX and LKH measured by means of PAR10. The dashed lines indicate the cutoff time ($T = 3\,600s$) used for terminating unsuccessful runs of EAX and LKH, as well as the penalty score for unsuccessful runs (ten hours).

abandon the iterative approach from Alg. 1 and instead make use of the EA described in Alg. 2, which is tailored to the creation of instances that are hard for solver A and easy for solver B (or vice versa).

5.1 Experimental Setup

Our goal is to evolve instances that are easy for solver A and hard for its contender B by means of the EA described in Section 3, and

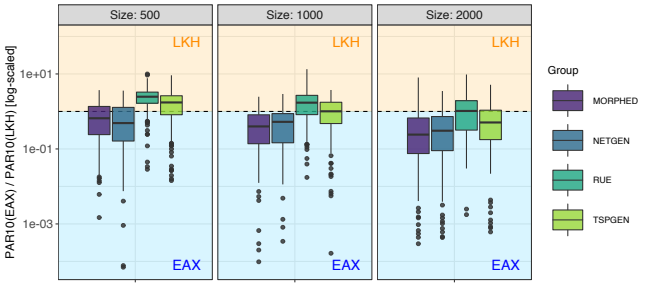

Figure 7: Boxplots of the log-scaled mean PAR10-ratios split by instance size and TSP set. The dashed horizontal line marks the ratio of 1, where both solvers exhibit a similar performance. Ratios below that line (blue background) indicate an advantage for EAX, whereas ratios above the line (yellow background) are advantageous for LKH.

to follow a similar setup as used in [4, 5]. We generate 50 instances for each pair (A, B) with $A, B \in \{\text{EAX}, \text{LKH}\}$ and each collection of mutation operators (simple vs. sophisticated). In total, this process produced 200 instances: 50 instances that are supposedly easy for EAX and have been generated exclusively based on the simple mutation operators, another 50 that are easy for EAX but are based on the sophisticated operators, etc. Note that in this series of experiments we (for now) restricted ourselves to instances of size 500 due to computational reasons.[5]

The rest of the setup is as follows: population size $\mu = 5$, cutoff time of five minutes (= 300 seconds) for each algorithm run of A and B with five independent runs per algorithm and instance. The mutation operators are drawn uniformly at random and the mutation parameters are set as follows: probability of mutation $p_m = 0.1$, probability of rotation $p_{\text{rot}} = 0.5$, probability to add jitter to mutated points $p_{\text{jit}} = 0.5$, standard deviation for jitter $\sigma_{\text{jit}} = 0.05$ and $r_{\min} = b_{\min} = 0.1$, as well as $r_{\max} = b_{\max} = 0.4$ for the minimal and maximal radius/distance (e.g., for explosion mutation). Note that each node is mutated independently with probability p_m and therefore in expectation 10% of the nodes are moved. We find this quite high number adequate for our work because there is no crossover at all. Further, we want to mention that p_m is not relevant for mutation operators that are based on distances (e.g., implosion or explosion). Here, the fraction of mutated points may be even higher if a large explosion/implosion takes place.

The instance generating EA was given 48 hours of walltime for each job as the single stopping condition.

5.2 Analysis of Evolved Instances

As our experiments showed, it is indeed possible to generate instances that are easy for EAX and hard for LKH (and vice versa). Fig. 8 shows boxplots of the PAR10-ratios for the four TSP sets generated by our tailored EA. In particular the two instance sets that were supposed to be favorable for EAX clearly managed to

[5] Our fitness function minimizes the PAR10-ratio of A and B. In order to compute the PAR10-scores for A and B, respectively, the optimal tour per instance has to be known upfront – and hence needs to be computed with Concorde. As the latter has exponential worst-case running time, we minimize the risk of very long running times by keeping the instance size low.

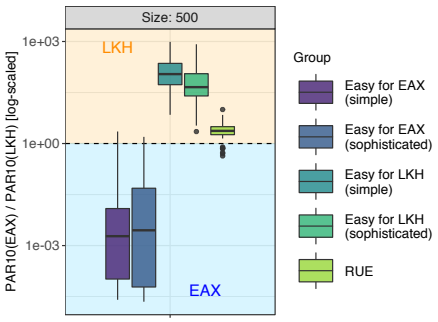

Figure 8: Boxplots of the log-scaled mean PAR10-ratios (between EAX and LKH) on the four evolved instance sets, as well as on the RUE instance set as a baseline.

Table 2: Summary of three commonly used average performance measures: running time of successful runs (RTS), failure ratio (FR) and the PAR10 score. The performances are listed per solver (EAX and LKH) and across all evolved instance groups, as well as for RUE (as a baseline).

TSP Set	Mutation	RTS*		FR†		PAR10	
		EAX	LKH	EAX	LKH	EAX	LKH
RUE	-	1.26	0.74	0.00	0.00	1.26	0.74
Easy for EAX	simple	1.34	912.78	0.00	0.20	1.34	7 608.11
	sophistic.	0.97	830.80	0.00	0.22	0.97	8 230.61
Easy for LKH	simple	117.97	0.74	0.00	0.00	117.97	0.74
	sophistic.	67.90	0.88	0.00	0.00	67.90	0.88

* RTS: Running time of successful runs, † FR: Failure ratio

draw the boxplots into their beneficial (light blue) area. However, possibly even more remarkable are the two boxplots depicting the PAR10-ratios of the LKH-friendly instances. Given that EAX consistently showed a very strong performance and usually neither comes close to the cutoff time (of one hour) nor performs much worse than LKH – as indicated by the 'blank space' in the bottom right of Fig. 6 – observing a shift of the PAR10-ratios (towards magnitudes of 10^2) in favor of LKH is a very remarkable result.

The effectiveness of the evolutionary instance generator is also clearly visible in Fig. 9, which depicts the PAR10-scores of the two solvers for each of the evolved instances. Noticeably, both mutation operator sets manage to produce instances that can be solved by EAX within roughly a second but are much harder for LKH (depicted by purple squares) and frequently even result in timeouts. Similarly, one can see that the EA is also capable of producing instances that are solved by LKH in less than a second, whereas EAX requires up to three minutes (blue circles). And while these values themselves might not sound impressive at first sight, one should recall that EAX showed very strong performances throughout our experiments and thus finding instances that pose at least some challenge for EAX should be considered a success.

These findings are also in line with the performance values given in Tab. 2, according to which it is much harder to evolve instances that are hard for EAX than for LKH. While *all* runs of EAX

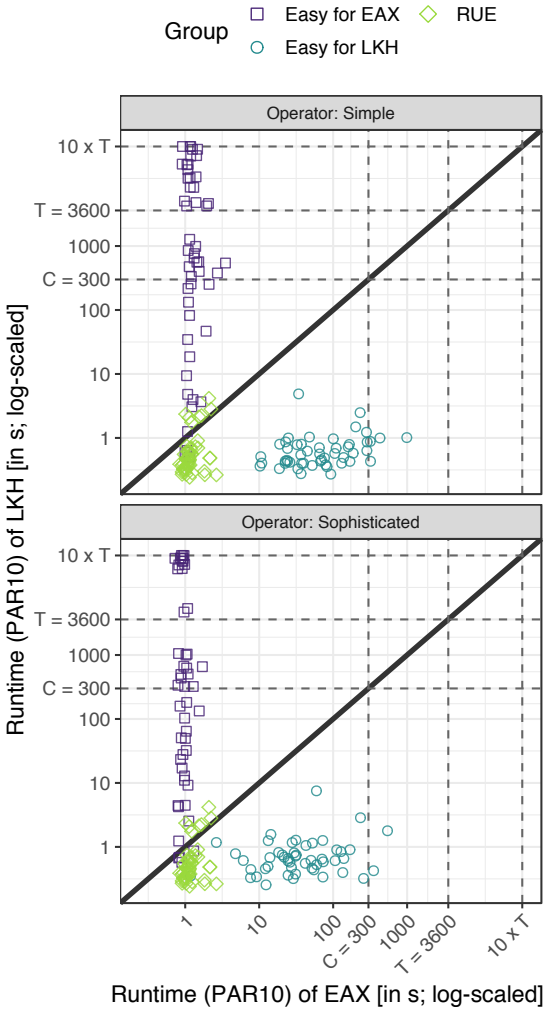

Figure 9: Scatterplots of log-scaled PAR10 performance values for EAX and LKH on the evolved instance sets. The dashed lines indicate the cutoff times – for the evolution process ($C = 300s$) and the actual benchmark study ($T = 3600s$) – as well as the penalty score for the unsuccessful runs.

across *all* TSP sets terminated successfully within the given time, roughly 20% of the LKH runs on the EAX-friendly instances failed to find an optimal tour within the given time budget. Moreover, we observed that both solvers managed to solve instances, which have been generated using the sophisticated mutation operators, faster than their corresponding counterparts, which are based on the simple mutation operators. This might indicate that both solvers can handle structured instances better than instances with rather random topologies (such as RUE). From a practical point of view, these would be great news – if they can be confirmed in future studies – as real-world problems usually possess more structure than the nonetheless frequently studied RUE instances.

Both figures that have been discussed so far within this section, i.e., Fig. 6 and 8, also display results for RUE instances to allow for a

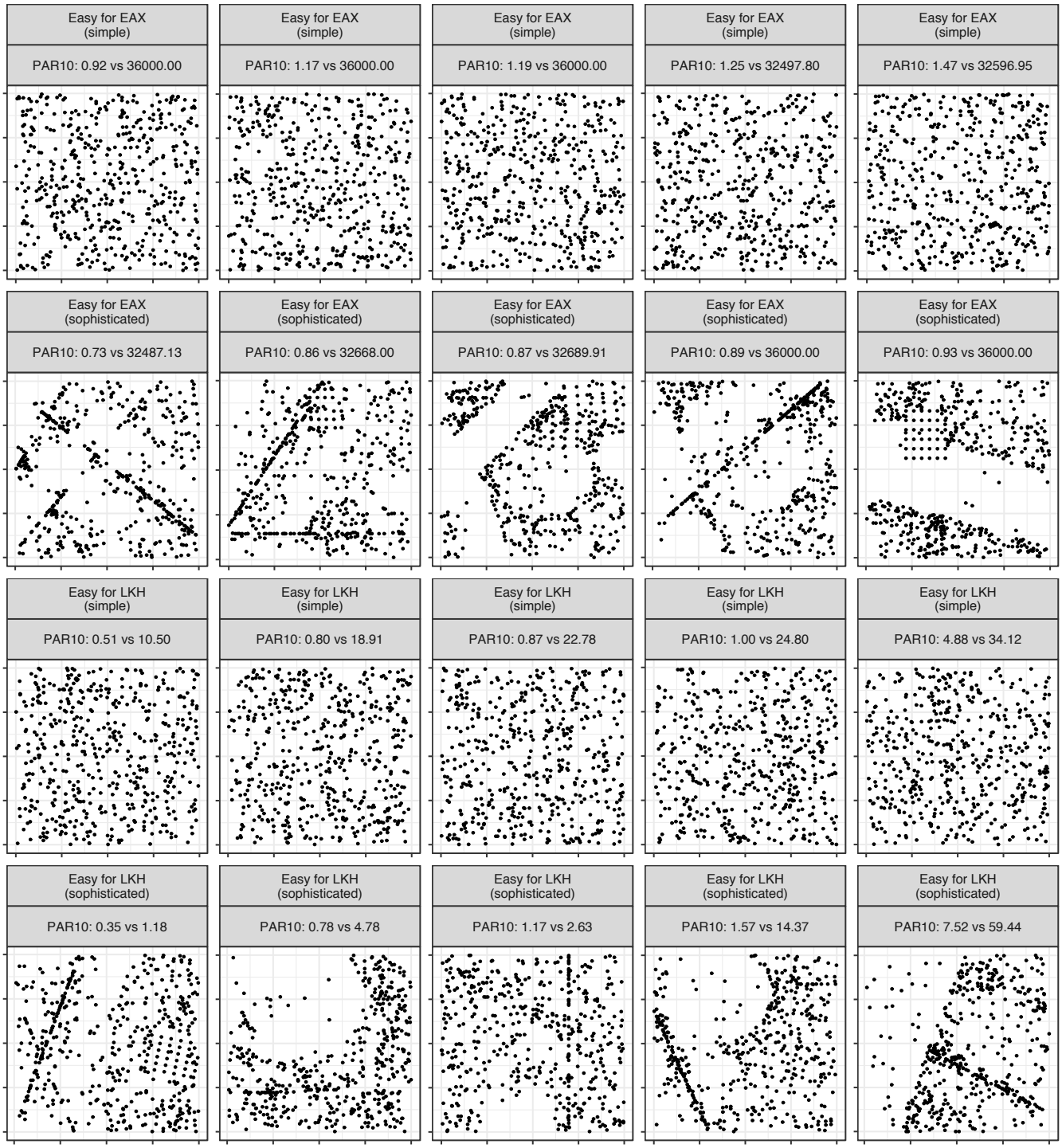

Figure 10: The five most extreme evolved instances based on the smallest PAR10-ratios (from left to right). The instances are shown for each direction of optimization (easy for EAX or LKH, respectively) and set of used mutation operators (simple vs. sophisticated).

baseline comparison.[6] Obviously, this instance set does not pose any challenge to either one of the two solvers as both algorithms solve the majority of RUE instances (with 500 nodes) within a second. Therefore, although this set is frequently used within TSP studies, it appears to be too easy to deliver useful insights into the solvers' strengths and weaknesses – unlike our evolved and solver-specific tailored TSP instances.

Recalling the setup of the evolutionary generation process, we noticed another striking result within the solvers' performances. Throughout the evolutionary process, our instance generator assessed an instance's difficulty based on five independent runs with a maximum time of 300*s* each, whereas the final results are based on ten independent runs with a maximum of 3 600*s* each. However, as one can see in Fig. 9, instances that are challenging for LKH given a five minute cutoff time ($C = 300$), frequently also resulted in failed runs when given a time budget of one hour.[7] Therefore, we can conclude that instances that are not solved to optimality within a few minutes are likely to remain unsolved within an hour.

We further depict the five most extreme instances (regarding the PAR10-ratio) for each of the four evolved instance sets in Fig. 10. As already discussed within Section 4, instances that were generated with the simple mutation operators (first and third row) are visually indistinguishable (for humans) – neither from each other nor from RUE instances. However, instances generated using the sophisticated mutation operators might be useful to detect patterns that make a problem hard (or easy) for a solver. For instance, the cities within TSP instances that are easy for EAX (second row) tend to be more often aligned along a linear line than for the LKH-friendly instances (fourth row). In case of the EAX-friendly instances, we also observed a tendency towards larger gaps between the cities, which have to be bridged by the solvers when trying to find the optimal tour.

Concerning the diversity in feature space, a further pattern can be found. The evolved TSPGEN instances result, similar to the TSPGEN instances from Fig. 4, in a wider spread across the feature space of feature pairs P1 to P4 than the established RUE instances (see Fig. 11). However, by looking at the corresponding four scatterplots in more detail, one can observe that especially the instances evolved by means of the sophisticated mutation operators (depicted as blue circles and green triangles) cause the spread. Please note that these results are achieved even though the EA has no sophisticated diversity maximization sub-routine at its disposal. These findings are also supported by Tab. 3, which lists the discrepancy values of the four evolved instance sets and the RUE instances across the four feature pairs P1 to P4. Here, discrepancy values of the TSP sets based on the sophisticated mutation operators are systematically smaller than the ones of their simpler counterparts.

6 CONCLUSION

This work addressed the problem of target-oriented generation of diverse benchmark instances for the Euclidean Traveling Salesperson Problem (TSP). For this, we introduced a set of sophisticated stochastic mutation operators that very effectively rearrange points

[6]RUE instances were used for the initial population of the evolving EA and are also commonly used within the majority of TSP studies.

[7]Note that we used the restart version of LKH which triggers a restart once the internal stopping criteria of vanilla LKH are met.

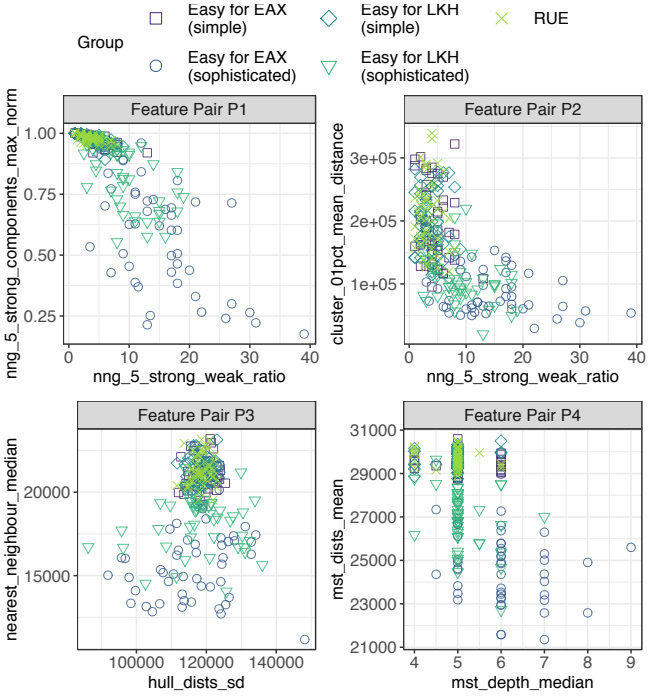

Figure 11: Two-dimensional embeddings of all evolved instances in the feature space exemplarily shown for the four feature pairs illustrated in Fig. 4. The evolved instances are complemented by a random subset of RUE instances for a baseline comparison. The shape and color of an instance indicate its corresponding group membership.

Table 3: Discrepancy values computed for the four exemplary feature pairs from Fig. 4 and across five TSP sets: four sets created with our proposed iterative instance generator (see Alg. 1), and a set of RUE instances for comparison.

Feature	Easy for EAX		Easy for LKH		RUE
Pair	simple	sophisticated	simple	sophisticated	
P1	0.852740	**0.400787**	0.837660	**0.581755**	0.886720
P2	0.812654	**0.650487**	0.835355	**0.712188**	0.664041
P3	0.634240	**0.498067**	0.648000	**0.422840**	0.648950
P4	0.715040	**0.545934**	0.759549	**0.657589**	0.737224

in the Euclidean plane. Our experiments revealed that instances generated by a simple iterative algorithm, which incorporates these stochastic operators, possess very multifaceted topologies and moreover are much more diverse in the feature space – with respect to visual perception and the discrepancy measure – than instances evolved with established operators.

Motivated by the promising results, we plugged these operators into a simple evolutionary algorithm to evolve instances that minimize the PAR10-ratio of two state-of-the-art solvers in inexact TSP-solving (the restarts versions of EAX and LKH). The resulting instance sets (1) expose a large difference in algorithm performance (easy for one solver and hard for its contender), and (2) cover a

broader spectrum of important instance characteristics in comparison to the baseline operators commonly used in the literature so far. We stress that the latter property is highly desirable and achieved without a complicated diversity preservation mechanisms build into the instance generating evolutionary algorithm. Both aspects clearly pave the way for further interesting studies on algorithm selection for the TSP by adopting the newly developed methods for benchmark generation purposes.

Moreover, we see much potential in the mutation operators themselves. Adjusting the parameters of the mutation operators, considering only subsets of operators and adopting a bi-level optimization approach, in which we optimize both PAR10-ratio and diversity, may lead to even more diversity in feature space.

As a concluding remark, this work can only be seen as a foundation for exciting future work in the field of algorithm selection.

ACKNOWLEDGMENTS

Jakob Bossek, Pascal Kerschke and Heike Trautmann acknowledge support from the *European Research Center for Information Systems*[8] *(ERCIS)*. Jakob Bossek and Pascal Kerschke also acknowledge funding from COST Action CA15140[9]: *Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice* (ImAppNIO). Furthermore, this work has been supported by the Australian Research Council through grant DP190103894 and by the South Australian Government through the Research Consortium "Unlocking Complex Resources through Lean Processing".

REFERENCES

[1] Bradley Alexander, James Kortman, and Aneta Neumann. 2017. Evolution of artistic image variants through feature based diversity optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*, Peter A. N. Bosman (Ed.). ACM, 171–178. https://doi.org/10.1145/3071178.3071342

[2] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. 2007. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, USA.

[3] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Thomas Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. 2016. ASlib: A Benchmark Library for Algorithm Selection. *Artificial Intelligence (AIJ)* 237 (2016), 41 – 58. https://doi.org/10.1016/j.artint.2016.04.003

[4] Jakob Bossek and Heike Trautmann. 2016. Evolving Instances for Maximizing Performance Differences of State-of-the-Art Inexact TSP Solvers. In *Proceedings of the 10th International Conference on Learning and Intelligent Optimization (LION)*. Springer, 48 – 59. https://doi.org/10.1007/978-3-319-50349-3_4

[5] Jakob Bossek and Heike Trautmann. 2016. Understanding Characteristics of Evolved Instances for State-of-the-Art Inexact TSP Solvers with Maximum Performance Difference. In *Advances in Artificial Intelligence (AI*IA)*. Springer, 3 – 12. https://doi.org/10.1007/978-3-319-49130-1_1

[6] Jakob Bossek and Heike Trautmann. 2019. Multi-Objective Performance Measurement: Alternatives to PAR10 and Expected Running Time. In *Proceedings of the 12th International Conference on Learning and Intelligent Optimization (LION) (Lecture Notes in Computer Science)*, Vol. 11353. Springer International Publishing, Kalamata, Greece, 215–219.

[7] Jérémie Dubois-Lacoste, Holger H. Hoos, and Thomas Stützle. 2015. On the Empirical Scaling Behaviour of State-of-the-art Local Search Algorithms for the Euclidean TSP. In *Proceedings of the 17th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 377 – 384. https://doi.org/10.1145/2739480.2754747

[8] Wanru Gao, Samadhi Nallaperuma, and Frank Neumann. 2016. Feature-Based Diversity Optimization for Problem Instance Classification. In *Proceedings of the 14th International Conference on Parallel Problem Solving from Nature (PPSN XIV) (Lecture Notes in Computer Science (LNCS))*. Springer, 869–879. https://doi.org/10.1007/978-3-319-45823-6_81

[8] https://www.ercis.org/

[9] https://www.cost.eu/actions/CA15140/

[9] Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA.

[10] Keld Helsgaun. 2009. General k-opt submoves for the Lin-Kernighan TSP heuristic. *Mathematical Programming Computation* 1, 2-3 (2009), 119 – 163. https://doi.org/10.1007/s12532-009-0004-6

[11] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm Runtime Prediction: Methods & Evaluation. *Artificial Intelligence (AIJ)* 206 (2014), 79 – 111. http://www.sciencedirect.com/science/article/pii/S0004370213001082

[12] Pascal Kerschke, Jakob Bossek, and Heike Trautmann. 2018. Parameterization of State-of-the-Art Performance Indicators: A Robustness Study Based on Inexact TSP Solvers. In *Proceedings of the 20th Annual Conference on Genetic and Evolutionary Computation (GECCO) Companion*. ACM, 1737 – 1744. https://doi.org/10.1145/3205651.3208233

[13] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2019. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary Computation (ECJ)* 27, 1 (2019), 3 – 45. https://doi.org/10.1162/evco_a_00242

[14] Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger H. Hoos, and Heike Trautmann. 2018. Leveraging TSP Solver Complementarity through Machine Learning. *Evolutionary Computation (ECJ)* 26, 4 (2018), 597 – 620. https://doi.org/10.1162/evco_a_00215

[15] Lars Kotthoff. 2016. Algorithm Selection for Combinatorial Search Problems: A Survey. *Data Mining and Constraint Programming* 10101 (2016), 149 – 190. https://doi.org/10.1007/978-3-319-50137-6_7

[16] Lars Kotthoff, Pascal Kerschke, Holger H. Hoos, and Heike Trautmann. 2015. Improving the State of the Art in Inexact TSP Solving Using Per-Instance Algorithm Selection. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization (LION)*. Springer, 202 – 217. https://doi.org/10.1007/978-3-319-19084-6_18

[17] Paul McMenemy, Nadarajen Veerapen, Jason Adair, and Gabriela Ochoa. 2019. Rigorous Performance Analysis of State-of-the-Art TSP Heuristic Solvers. In *Evolutionary Computation in Combinatorial Optimization (Lecture Notes in Computer Science (LNCS))*, Arnaud Liefooghe and Luís Paquete (Eds.), Vol. 11452. Springer, 99 – 114. https://doi.org/10.1007/978-3-030-16711-0_7

[18] Stephan Meisel, Christian Grimme, Jakob Bossek, Martin Wölck, Günter Rudolph, and Heike Trautmann. 2015. Evaluation of a Multi-Objective EA on Benchmark Instances for Dynamic Routing of a Vehicle. In *Proceedings of the 17th Genetic and Evolutionary Computation Conference (GECCO)*. ACM, New York, NY, USA, 425–432. https://doi.org/10.1145/2739480.2754705

[19] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. 2011. Exploratory Landscape Analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 829 – 836. https://doi.org/10.1145/2001576.2001690

[20] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Markus Wagner, Jakob Bossek, and Frank Neumann. 2013. A Novel Feature-Based Approach to Characterize Algorithm Performance for the Traveling Salesperson Problem. *Annals of Mathematics and Artificial Intelligence* 69 (October 2013), 151 – 182. Issue 2. https://doi.org/10.1007/s10472-013-9341-2

[21] Yuichi Nagata and Shigenobu Kobayashi. 2013. A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem. *INFORMS Journal on Computing* 25, 2 (2013), 346 – 363.

[22] Aneta Neumann, Wanru Gao, Carola Doerr, Frank Neumann, and Markus Wagner. 2018. Discrepancy-Based Evolutionary Diversity Optimization. In *Proceedings of the 20th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 991 – 998. https://doi.org/10.1145/3205455.3205532

[23] Aneta Neumann, Wanru Gao, Markus Wagner, and Frank Neumann. 2018. Evolutionary Diversity Optimization Using Multi-Objective Indicators. *CoRR* abs/1811.06804 (2018), 1 – 22. arXiv:1811.06804 http://arxiv.org/abs/1811.06804 Conference version appears at GECCO 2019.

[24] Aneta Neumann, Christo Pyromallis, and Bradley Alexander. 2018. Evolution of Images with Diversity and Constraints Using a Generative Adversarial Network. In *Neural Information Processing - 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part VI (Lecture Notes in Computer Science)*, Long Cheng, Andrew Chi-Sing Leung, and Seiichi Ozawa (Eds.), Vol. 11306. Springer, 452–465. https://doi.org/10.1007/978-3-030-04224-0_39

[25] Harald Niederreiter. 1972. Discrepancy and convex programming. *Annali di matematica pura ed applicata. Series 4* 93, 1 (1972), 89–97.

[26] Josef Pihera and Nysret Musliu. 2014. Application of Machine Learning to Algorithm Selection for TSP. In *Proceedings of the IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 47–54. https://doi.org/10.1109/ICTAI.2014.18

[27] John Rischard Rice. 1976. The Algorithm Selection Problem. *Advances in Computers* 15 (1976), 65 – 118. https://doi.org/10.1016/S0065-2458(08)60520-3

[28] Danilo Sanches, L. Darrell Whitley, and Renato Tinós. 2017. Building a Better Heuristic for the Traveling Salesman Problem: Combining Edge Assembly Crossover and Partition Crossover. In *Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 329 – 336. https://doi.org/10.1145/3071178.3071305

[29] Kate Amanda Smith-Miles. 2009. Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection. *ACM Computing Surveys (CSUR)* 41 (January 2009), 1 – 25. https://doi.org/10.1145/1456650.1456656

[30] Kate Amanda Smith-Miles and Jano Ilja van Hemert. 2011. Discovering the Suitability of Optimisation Algorithms by Learning from Evolved Instances. *Annals of Mathematics and Artificial Intelligence* 61, 2 (2011), 87 – 104. https://doi.org/10.1007/s10472-011-9230-5

[31] Renato Tinós, Keld Helsgaun, and L. Darrell Whitley. 2018. Efficient Recombination in the Lin-Kernighan-Helsgaun Traveling Salesman Heuristic. In *Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN XV) (Lecture Notes in Computer Science (LNCS))*, Anne Auger, Carlos Manuel Mira da Fonseca, Nuno Lourenço, Penousal Machado, Luis Paquete, and L. Darell Whitley (Eds.), Vol. 11101. Springer, 95 – 107. https://doi.org/10.1007/978-3-319-99253-2_8

[32] Tamara Ulrich and Lothar Thiele. 2011. Maximizing population diversity in single-objective optimization. In *Genetic and Evolutionary Computation Conference, GECCO*. ACM, 641–648.

[33] David Hilton Wolpert and William G. Macready. 1997. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation (TEVC)* 1, 1 (1997), 67 – 82. https://doi.org/10.1109/4235.585893