Exploring the Feature Space of TSP Instances Using Quality Diversity

Jakob Bossek
Statistics and Optimization
Dept. of Information Systems
University of Münster, Germany

Frank Neumann
Optimisation and Logistics
School of Computer Science
The University of Adelaide, Australia

ABSTRACT
Generating instances of different properties is key to algorithm selection methods that differentiate between the performance of different solvers for a given combinatorial optimization problem. A wide range of methods using evolutionary computation techniques has been introduced in recent years. With this paper, we contribute to this area of research by providing a new approach based on quality diversity (QD) that is able to explore the whole feature space. QD algorithms allow to create solutions of high quality within a given feature space by splitting it up into boxes and improving solution quality within each box. We use our QD approach for the generation of TSP instances to visualize and analyze the variety of instances differentiating various TSP solvers and compare it to instances generated by established approaches from the literature.

CCS CONCEPTS
- Theory of computation → Evolutionary algorithms.

KEYWORDS
Quality diversity, instance generation, instance features, TSP

ACM Reference Format:
Jakob Bossek and Frank Neumann. 2022. Exploring the Feature Space of TSP Instances Using Quality Diversity. In Genetic and Evolutionary Computation Conference (GECCO ’22), July 9–13, 2022, Boston, MA, USA. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528851

1 INTRODUCTION
Evolutionary algorithms have been used for a wide range of different tasks. This includes solving classical \( \text{NP-Hard} \) optimization problems, optimizing the design for complex engineering problems [11], algorithms for machine learning [36], and approaches in the area of algorithm selection and configuration [17].

In this paper, we focus on a crucial problem in the context of algorithm selection, namely the design of problem instances where two algorithms show a significantly different behaviour. We consider instances for the classical Euclidean traveling salesperson problem (TSP) for which a wide range of algorithms has been introduced over the last 50 years [3]. The problem is to find a round-trip tour of minimal costs through \( n \) cities where the cities are given by coordinates in the 2D Euclidean space and pairwise distances correspond to the Euclidean distances of the points.

The complementary ability of algorithms for solving the TSP has been subjected to a wide range of studies. Center of these investigations is the characterization of features of TSP instances where a considered set of algorithms shows a significantly different behaviour. Studies have considered simplified variants of local search [35], approximation algorithms [23], and state-of-the-art inexact TSP solvers such as EAX and LKH [6, 7, 16].

Evolutionary diversity optimization (EDO) has recently been applied using various diversity measures to create diverse sets of images and TSP instances [1, 24, 25]. For TSP instances, it has been shown that such approaches are able to create TSP instances with desired properties (such as good or bad performance behaviour) with a much wider range in the feature space [13]. Furthermore, recent investigations have used and analyzed EDO methods to create diverse sets of high quality solutions for the TSP and other permutation problems [9, 10, 26, 27].

Quality diversity (QD) [30] follows a similar goal as EDO. Algorithms explore a feature space of possible solutions to a given problem. Simple approaches, e.g., Map-Elites, partition the feature space into boxes and store for each box the best solution that has been found for its range of feature combinations. QD algorithms have mainly been applied in the context of robotics and games [8, 37]. In the context of combinatorial optimization, a QD approach has recently been used for the traveling thief problem and it has been shown that significantly better solutions can be obtained [28]. In this paper, we use for the first time the quality diversity approach in the context of instance generation. We use a Map-Elites approach which for a given set of features divides the feature space into boxes and stores for each box TSP instances that show the maximum performance difference for two given algorithms.

Using this approach, we explore the feature space and optimize instances within each box with respect to solver performance. This allows to get an overview on differences of instances in the feature space in terms of their difficulties for a given set of TSP solvers. In our experiments we evolve instances for the insertion heuristics Farthest-Insertion and Nearest-Insertion as well as for state-of-the-art exact TSP solvers EAX and LKH on different two-dimensional feature spaces. We compare our QD approach against classical \((\mu + 1)-EAs\) and EDO-tailored \((\mu + 1)-EAs\) modified to store their “footprint” in a similar way. Our experimental investigations show that with the proposed QD approach (a) in most cases a much wider range of feature combinations is explored during the evolutionary process and (b) the objective scores of the evolved instances are either comparable or even better in all cases. Put together the QD
The paper is organized as follows. In Section 2 we describe the task of instance generation in detail, discuss challenges and review approaches from the literature. In Section 3 we introduce our quality diversity approach. Experimental studies on simple and state-of-the-art TSP heuristics follow in Section 4 and Section 5 respectively. We conclude the paper in Section 6.

2 PROBLEM FORMULATION AND HISTORY

Let \( \mathcal{A} \) be a portfolio of algorithms which show complementary performance on a set of problem instances \( I \). Further, let \( F : I \rightarrow \mathcal{F} \subseteq \mathbb{R}^p \), \( p \geq 1 \) be a feature-mapping which maps instances to a \( p \)-dimensional real-valued numeric vector in the feature-space \( \mathcal{F} \) that ideally captures the characteristics of the instance which makes it easy or hard for the algorithms in the portfolio. In the context of the Euclidean TSP, features are for example the share of points on the convex-hull of the point coordinates or structural properties of transformations of the original instance, e.g., statistics on the weak/strong connected components of a \( k \)-nearest neighbor graph (\( k \)-NNG). The idea of (per-instance) algorithm selection (AS) is to learn an algorithm selector \( S : \mathcal{F} \rightarrow \mathcal{A} \) which – for a previously unseen instance – predicts the algorithm which will likely perform best on it based on its cheap-to-evaluate features only [17, 32].

In order to understand the strengths and weaknesses of algorithms and to be able to learn reasonable selectors, the instance set \( I \) should ideally be diverse on two different levels: (D1) diverse/complementary with respect to algorithm performance and (D2) diverse with respect to the structure of the instances which is captured by the coverage of the feature-space. Finding such a set of instances is challenging [35]. Since the availability of real-world instances for research is limited, a systematic generation process is desirable. In this paper we focus on the problem of evolving a large set of diverse instances that satisfy both diversity requirements. For the TSP, early approaches focused on diversity in the performance space (D1). To this end Mersmann et al. [20] introduced a \((\mu+1)\) EA to generate instances which are hard/easy for an algorithm \( A \) by minimizing/maximizing the gap to a known optimum; they evolved small instances for classical approximation algorithms and local-search algorithms. Subsequent works evolved instances which are easy for one algorithm \( A_1 \) and hard(er) for another algorithm \( A_2 \) by minimizing the ratio of performances measured by the penalized average runtime (PAR [4]) for state-of-the-art inexact TSP solvers EAX and LKH [6, 7]. The EAs used for these tasks – leaving apart details – very much follow the pseudo-code given in Algorithm 1. In a nutshell the algorithm initializes a population \( P \) with \( \mu \) random uniform Euclidean (RUE) instances. I.e., each of the \( n \) city coordinates is placed uniformly at random within a bounding-box usually set to \([0,1]^2\) (see left-most plot in Figure 1). Within the evolutionary loop one instance is selected to produce an offspring. The classical mutation operators either add some Gaussian noise to the feature vector of a subset of the points of the parent as illustrated in Figure 1 (center) or re-locate a subset of the points by sampling new coordinates, again, uniformly at random. An elitist \((\mu+1)\)-strategy assures that the best individuals with respect to the objective function survive. On termination the best instance is usually returned. It turned out that even though goal (D1) was successfully met, (D2) was not; in fact no difference to RUE instances was visible to the naked eye and as a consequence the feature space \( \mathcal{F} \) was not well covered. Later, this issue was approached by different groups of researchers. Gao et al. [12] used a bi-level approach which aimed to evolve a feature-diverse population of instances with minimal performance quality requirements (see also the recent journal extension [13]). To this end they developed a diversity measure based on a weighted sum approach to calculate each individuals contribution to the feature diversity in the population. Though appealing, the bi-level character delayed the search for low objective values. Later TSP instance generation was approached in the context of evolutionary diversity optimization (EDO). In EDO, given a (close-to-)optimal solution OPT and a parameter \( \alpha > 1 \) the goal is to evolve a population \( P \) such that \( f(l) \leq (1+\alpha) \cdot OPT \) for all \( l \in P \) and a diversity measure \( D : P \rightarrow \mathbb{R} \) is maximized. Promising contributions in this field adopted the star-discrepancy [24] or well-known performance indicators from multi-objective optimization [25] as feature-space diversity measure. D. Bossek et al. [5] recently took a different path and introduced a sophisticated set of “creative” mutation operators. These operators addressed the short-comings of the aforementioned operators (re-location and Gaussian noise) by strongly affecting the parents’ point coordinates. An easy to illustrate example is the implosion mutation where all points within a random radius around a randomly sampled center of implosion are attracted towards the center (see right-most plot in Figure 1). The authors showed in their study, adopting a \((5+1)\) EA where in each iteration one of these creative mutation operators was applied to the parent, that this approach – applied many times to produce a set of instances – covered a much wider part of the feature space without explicit feature-space diversity preservation at all.

All approaches discussed so far are very wasteful in the sense that only the best individual of the final population is returned as output.
or the whole usually small\textsuperscript{1} final population for the approaches with diversity preservation. Nevertheless, many potentially interesting intermediate instances become extinct in the course of optimization. This is undesirable since the entire process is obviously very time consuming.\textsuperscript{2} Hence, there is a need for a systematic means to memorize instances of interest.

3 A QUALITY DIVERSITY APPROACH

A straight-forward idea is to simply save all intermediate instances, i.e., the footprint of the EA so to say. This approach is not target-oriented however since instances with large performance gaps are desirable; see diversity requirement (D1). We now present an idea that borrows ideas from quality diversity (QD). QD is a recently emerged branch of evolutionary computation. In the so-called Map-elites approach the target space is partitioned into boxes where each box stores the best solution found for all target space realizations closest to its box-center. This approach is highly successful, e.g., in evolving different robot behavior [8] where the target space corresponds to the behavioral space of the robots. In our setting the target space corresponds to the feature space $\mathcal{F} \subseteq \mathbb{R}^k$ we aim to cover. Each box of our QD-algorithm corresponds to one possible feature combination. The algorithm is designed for discrete features where, given a fixed instance size $n$, the number of possible realizations of each feature is countable and bounded. This is indeed the case for many TSP-features of interest, e.g., $k$-NNG based features or depth-based MST features [29] which are frequently among the top discriminating features in AS-models [18, 29, 34].\textsuperscript{3} We refer the interested reader to recent work by Heins et al. [14] who derived theoretical lower and upper bounds for a large collection of such features for feature normalization. For each box our algorithm \textit{in classical QD-fashion} stores the TSP instance found over time with the best performance value. More precisely, the algorithm proceeds as shown in Algorithm 2. First, an empty map is initialized. The map takes the role of the “population” in Algorithm 1. Next, an RUE instance $I$ is created at random and stored in the respective box of the map indexed by $I$’s feature-vector $F(I)$.\textsuperscript{4} From there on the following steps are performed until a stopping condition is met: an instance $I$ from the subset of covered boxes, i.e., those that are already assigned an instance, is sampled uniformly at random. Mutation is applied to $I$ which yields another instance $I’$. Next, if the box at $F(I’)$ is empty (we say the box is \textit{hit for the first time} in the following), $I’$ is saved in any case regardless of its objective value. Otherwise, if the box is already covered, the objective function $f$ comes into play. Let $I''$ be the instance assigned to the box, i.e., $F(I’') = F(I’)$. If $f(I’)$ is not worse than $f(I’’)$ the box is updated by replacing $I’’$ with $I’$. The algorithm returns all instances stored in the map upon termination. Hence, the algorithm follows an elitist strategy in each box and thus box-wise the objective function is monotonically decreasing (w.l.o.g., we assume minimization). The approach is simple but nevertheless very generic and powerful and by no means restricted to the TSP domain that we adopt here for a proof-of-concept study in this contribution.

4 EXPERIMENTS ON SIMPLE HEURISTICS

In this and the next section we continue with an extensive empirical evaluation of the proposed approach. Our experimental setup is two-fold: In the \textit{cheap} setting we evolve small instances for simple TSP heuristics whereas in the \textit{expensive} setting we evolve larger instances with the goal to maximize performance differences between state-of-the-art heuristics. The cheap experiments – due to fast evaluation of the objective function – allow to explore the differences obtained after a huge number of iterations as a proof-of-concept. In the expensive setting we can afford only a fraction of the iterations due to computational limitations, but tackle interesting algorithms.

4.1 Experimental setup

In a first series of experiments we evolve small instances with $n = 100$ nodes. We consider two insertion heuristics: \textit{Farthest-Insertion} (FI) and \textit{Nearest-Insertion} (NI) [33]. FI starts with a partial tour consisting of one randomly selected node. In $(n-1)$ iterations the remaining nodes are added until a TSP tour is constructed. In the selection phase the algorithm selects the node furthest away from the currently maintained sub-route. This node is injected into the existing partial route such that the least extension of the previous partial route is created. (NI) works very much alike with a slight difference in the selection phase: NI selects the nearest node. The goal is to evolve instances which span a two-dimensional feature space nicely and minimize the tour length ratio $F(I)/NI(I)$ or $NI(I)/FI(I)$ respectively. Ratios below 1 are desirable as this indicates a performance advantage for the “numerator” algorithm. The feature combinations consist of combinatorial features calculated on basis of a MST of the instance or its $k$-NNG as introduced in [29]; for details we refer the reader to the original papers. More precisely, we consider feature combinations (FCs) that were frequently selected by AS-models in various studies (see, e.g., [5, 29]).

\begin{algorithm}
\caption{QD approach for instance generation.}
\begin{algorithmic}[1]
\State \textbf{input}: Objective $f$, feature-mapping $F$.
\State Initialize empty map $M$;
\State Initialize instance $I$ at random;
\State Calculate $F(I)$ and store $I$ in $M[F(I)]$;
\While {Stopping condition not met}
\State Select an instance $I$ from $M$ uniformly at random;
\State Generate $I’$ by mutating $I$;
\If {Box $M[F(I’)]$ is not covered}
\State Store $I’$ in $M[F(I’)]$;
\Else
\State Let $I’’$ be the instance stored in box $M[F(I’)]$;
\If {f($I’$) is not worse than f($I’’$)}
\State Replace $I’’$ with $I’$ in $M[F(I’)]$;
\EndIf
\EndIf
\EndWhile
\State Return $M$;
\end{algorithmic}
\end{algorithm}

\textsuperscript{1}Due to the computational costs of the instance generation process the population is usually small in order to allow for as many generations as possible.

\textsuperscript{2}In recent studies on state-of-the-art inexact TSP solvers, for each generated instance the exact optimum has to be calculated in order to calculate the PAR-score. E.g., in [5] the generation of a single instance was given a time budget of 48h CPU time.

\textsuperscript{3}E.g., given $n$ cities, the number of weak connected components in the $k$-NNG can take values in $\{1, 2, \ldots, n-k\}$ only [14].

\textsuperscript{4}The map can be implemented as a hash-map allowing for very fast look-up operations.
Algorithm 3: $(\mu+1)$ EA (see Alg. 1) with archive.

```
input: Objective $f$, population size $\mu$, feature-mapping $F$
1 Initialize empty map $M$;
2 Initialize population $P$ of $\mu$ instances at random;
3 Store $I \in P$ at $M[F(I)]$ for all $I \in P$;
4 while Stopping condition not met do
5     Select random instance $I \in P$;
6     Generate $I'$ by mutating $I$;
7     if Box $M[F(I')]$ is not covered then
8         Store $I'$ in $M[F(I')]$;
9     else
10        Let $I''$ be the instance stored in box $M[F(I')]$;
11        if $f(I')$ is not worse than $f(I'')$ then
12            Replace $I''$ with $I'$ in $M[F(I')]$;
13        if $f(I')$ is not worse than $f(I)$ then
14            Replace $I$ with $I'$ in $P$;
15 return $M$;
```

FC1) Maximum size of a strongly connected component in the 3-NNG (nng_3_strong_components_max) and the number of weak connected components in the 3-NNG (nng_3_n_weak).

FC2) The number of strongly connected components in the 5-NNG (nng_5_n_strong) and the median depth of nodes in a MST of the input graph (mst_depth_median).

In the following we will make heavy use of the abbreviations FC1 and FC2 for ease of writing. For each algorithm pair (Fl vs. NI) and (NI vs. Fl) and each of the two FCs we evolve instances by ten algorithms in total: QD implements Algorithm 2. We compare QD against classic evolutionary algorithms ($1 + 1$) EA and ($50 + 1$) EA as well as EDO-focused EAs ($50 + 1$) ED-IGD and ($50 + 1$) EA-HV from [25] which use inverted generational distance (IGD) and the dimension-doubling hyper-volume indicator for feature-diversity maximization. These algorithms were among the best-performing EAs in [25] (see the reference for details). EDO-based algorithms require a good initial population. Hence, we initialize each run with clones of a solution obtained by running ($1 + 1$)-EA for additional 10,000 iterations. All algorithms are run with either only simple mutation operators (re-location, Gaussian mutation) from Mersmann et al. [20] (suffix [simple]) or the full set of disruptive mutation operators introduced in Bossek et al. [5] (suffix [all]; see also Section 2).\(^3\) Note that the baseline algorithms return at most 50 instances and thus it is unfair to compare directly against these; the winner would be determined in advance — QD — since it stores all novel instances with not yet seen feature- combinations. Hence, for a fair comparison we modify the EAs to store instances the same way the proposed QD approach does (see Algorithm 3). We stress that this way of saving the footprint of the EAs has also not been done before in the literature. Each experiment was repeated 30 times with different random numbers generators seeds and a budget of $1,000,000$ function evaluations. Within each objective function evaluation the tour length of each TSP heuristic is calculated as the

mean over 5 independent runs to account for stochasticity in the choice of the initial node for tour construction. All algorithms were implemented in the R programming language [31]. Code and data will be publicly available in a GitHub repository upon acceptance.\(^6\)

4.2 Results

We first study the number of boxes covered, i.e., the number of instances / feature combinations hit by the algorithms over time. To this end we logged the load of the map $M$ over time.

Figure 2 shows the progress of the mean number of boxes covered (with one standard-deviation error bars). Note that we show ($50 + 1$) EA-HV in all subsequent plots as a representative EDO-EA since it performs very much like the EA-HV version. We observe that both QD approaches are highly superior to the classic EA-baselines with respect to feature space coverage. After termination the mean number of boxes covered by QD [all] surpasses the best classic EAs’ value by a factor of $> 2$ for FC1 and up to a factor of $> 4$ for FC2 and the algorithm pair (NI vs. Fl). The EDO-EAs perform much better than their classic counterpart. However, expect for (Fl vs. NI) on FC2 they produce less instances than QD [all] and QD-algorithms show a high robustness / very low variation. Table 1 shows the mean and standard deviation of boxes covered among other statistics that will be discussed later. The observations are confirmed. A Kruskal-Wallis test with Bonferroni-correction rejects the zero hypothesis $H_0: \text{median}(X) > \text{median}(QD [\text{all}])$ for every competitor evolver $X$ at significance level $\alpha = 0.001$ for all settings expect for the mentioned (Fl vs. NI) on setting FC2; the results are highly significant. Another important observation is that QD [all] dominates the whole field of competitors significantly in the first few thousand iterations. This is important for computationally more involved instance generation task (see Section 5). As expected, all algorithms tend to perform much better

\(^6\)GitHub repository: https://github.com/jakobbossek/GECCO2022-QD-TSP

---

\(^3\)In each mutation step one of the mutation operators is selected uniformly at random.
| Feature Combination | Algorithm | Objective | NI vs. FI | FC1 vs. NI |
|---------------------|-----------|-----------|-----------|-----------|
| Feature 1           | (1+1) EA [all] | 527.90 | 263.00 | 700.120.00 |
|                     | (50+1) EA−HV [all] | 920.43 | 475.00 | 1148.50 |
|                     | QD [all] | 856.10 | 437.50 | 1776.00 |
|                     | (1+1) EA [all] | 156.03 | 240.00 | 514.35 |
|                     | QD [all] | 387.50 | 244.90 | 720.00 |
| Feature 2           | (1+1) EA−HV [simple] | 478.83 | 240.00 | 514.35 |
|                     | QD [simple] | 485.90 | 243.00 | 519.00 |
|                     | (1+1) EA−HV [all] | 870.03 | 475.00 | 1285.00 |
|                     | QD [all] | 899.77 | 444.00 | 1134.00 |
|                     | (1+1) EA [simple] | 527.90 | 263.00 | 700.120.00 |
|                     | QD [simple] | 533.00 | 244.90 | 720.00 |

Exploring the Feature Space of TSP Instances Using Quality Diversity
GECCO '22, July 9–13, 2022, Boston, MA, USA

Table 1: Table of different aggregated values of interest for the experiments on the simple insertion heuristics. Best values (for the statistics of covered or objective values) or maximal values (for box update statistics) are highlighted.

Figure 3: Representative feature space coverage obtained by the different algorithms for feature combination FC1 (top row) and FC2 (bottom row). White crosses mark the boxes hit by the initial population for (FI vs. NI; left) and (NI vs. FI; right). The tiles share a common continuous color gradient with the same limits to make the differences in the objective values visible.

if all mutation operators are used due to the disruptive nature of the creative mutation [5]. However, the benefit of disruptive over simple mutation is surprisingly low for FC1 and very strong for FC2. Figure 3 shows the feature space (normalized to [0, 1] [14]) coverage by means of exemplary but representative runs for FC1 and FC2. The covered boxes are colored by their objective value (smaller is better) and white crosses represent the boxes hit by the initial solutions. We see a slight benefit for the [*all] algorithms and FC1. However, the feature space coverage is indeed very similar. Explanation: consider the spread of initial solutions in Figure 3 for the (50 + 1) EA variants on FC1. Recall that each initial solution is generated by placing \( n \) points uniformly at random within \([0, 1]^2\) (RUE; see Section 2). These initial instances already show a nice distribution of this particular feature space indicating that simple mutation operators (which produce again RUE-like mutants) are sufficient to further explore the FC1-space. The FC2 space coverage...
in Figure 3 shows a different picture. Here, the initial population is very close to the origin. I.e., for all initial solutions the feature vectors in FC2-space are quite similar. This is plausible since an RUE instance is likely to have a low number of strong connected components in the 5-NNNG and likewise a rather small median depth of nodes in an MST. In the FC2-space, the strong mutation of the "[all]" variants allows all algorithms to explore the feature space much more thoroughly.

Comparing the "[simple]" and "[all]" plots in Figure 3 for FC2 and also the respective progress over time in Figure 2 gives rise to another question: why do the classic and EDO-based \((\mu+1)\) EAs [all] variants cover much more of the FC2 feature space when the objective is to evolve instances which are easier to solve for FI than for NI? And why is this not the case for their "[simple]" counterparts? One might think that this has to be a bug since an instance is always saved if its feature vector was not seen so far; this first hit of a box is independent of the objective value. However, the reason lies in the way the survival selection of the EAs works: the mutant replaces the parent if and only if its objective value is not worse than that of the parent (see Algorithm 3) or for the EDO-EAs – diversity maximization takes place only if the instance satisfies the minimum quality constraint. Since it is much more challenging to produce instances which are harder for FI than for NI (see Figure 5 where the objective values barely fall below 1) this is an unlikely event to happen. Consequently, in particular \((1+1)\) EA suffers from few updates of its "population" and its only population member serves as the template for the vast majority of the produced offspring. A single mutation (regardless of the type: explosion, implosion etc.) might be insufficient in producing a previously unseen feature combination. This lack of variation leads to the poor performance on (NI vs. FI). In the reverse case, (FI vs. NI), things are different and the population members get updated more often which allows for a more thorough exploration of the feature space since the "source material" changes more frequently. This phenomenon is not observed for FC1. As pointed out already the FC1-space can be explored more reliably with little variation. The observations are the same for all other settings.

So far we investigated the ability of the algorithms to cover the feature space, one of the two goals of the evolving procedure. We now turn the focus towards the second equally if not more important goal: minimizing the tour length ratio. Figure 5 shows the distribution of the objective values on all instances over all runs produced by the algorithms. The panels split the data by feature combination (rows) and objective "direction" ((FI vs. NI) and (NI vs. FI); columns). Recall that the algorithms strive to minimize the tour length ratio. Thus, values in the green shaded region below 1 are desirable. We observe that QD [all] dominates the field for all combinations of features and target directions with respect to the median objective value. In particular in the (NI vs. FI) setting, where finding instances which are harder to solve for NI seems difficult, the median objective values are clearly located in the greenish area whereas the median objective scores of the competitor algorithms are closer or even above 1. With the exception of (NI vs. FI) on FC2, the upper quartile for QD [all] is even lower than the median of the other algorithms. This observation highlights the need for stronger exploration of the feature space and shows the superiority of the QD-approach. Note also that the median objective values produced by the EDO-EAs are worse than the respective QD-values. This is particularly visible in (FI vs. NI) setting and both FC1 and FC2. Recall that QD [all] covered less boxes in the FC2 scenario than the EDO-EAs. However, it shines when it comes to the median objective values. This is also confirmed by the precise numbers in Table 1. Keep in mind that the data used for Figure 5 is the union of all instances of all 30 algorithm runs. Considering the advantage of QD [all] with respect to feature space coverage the QD approaches in consequence produced significantly more easy/hard instances; a clear benefit. Regarding the overall "best" objective values, in most cases the \((1+1)\) EAs dominates. This is in line with our expectation. Recall that (1) the \((1+1)\) EAs explores a significantly
smaller share of the feature space and (2) has the same budget of function evaluations. Hence, the search process is much more focused and boxes are thus hit much more often. This increases the chances to update respective boxes with better instances. Figure 6 shows the relation between the (log-scaled) number of hits on the abscissa and the objective value of the instance stored in the respective boxes after termination on the ordinate. The plots show the result of one particular run of the algorithms. However, the patterns are the same for all other runs. We see obvious negative linear relationships. In addition, we see that for a subset of the feature combinations in the \((\mu + 1) \text{EAs}\) the number of hits surpasses the maximum number of hits for the respective QD algorithm by orders of magnitude (recall that the abscissa is on logarithmic scale). An interesting observation is that the number of box-hits of the EDO-EAs is also much higher. Nevertheless, QD remains the winner with respect to the objective value since the number of box-hits of the EDO-EAs which may lead to frequent box-hits which do not improve upon the objective however. Figure 7 in the top row shows this narrowed search by means of example in the FC2-space explored by \((1 + 1) \text{EA [all]}\), \((50 + 1) \text{EA-HV [all]}\) and QD [all] where boxes are colored by the number of actual updates. While for QD [all] the updates are distributed across the whole explored feature space (leave apart the border regions) only a few boxes are updated more than 30 times in the \((1 + 1) \text{ EA [all]}\) run. The \((50 + 1) \text{ EA-HV [all]}\) updates cells less often which explains its rather poor performance with respect to the objective values discussed before. The box statistics in Table 1 support these observations. They show the maximum number of updates and hits of boxes over all runs. \((1 + 1) \text{EA}\) shows a huge maximum of hits up to \(\approx 52\%\) of the total number of iterations on (NI vs. FI) and FC2. The values for QD in comparison are minuscule. The decreased number of updates at the borders can be explained by the hardness to produce instances with such “extreme” feature vectors. The bottom row in Figure 7 shows the same data colored by the iteration number of the boxes’ first hit. Some of the boxes were only discovered after more than 800 000 iterations. This observation together with the increasing trend in Figure 2 indicates that even longer runs would hit even more boxes.

Figure 6: Scatter-plot of the number of box-hits on log-scale and the objective value of the instance stored in the box. The plot shows the first run by example.

Figure 7: FC2 with boxes colored by the number of updates, i.e., the number of times the instance stored in the box was replaced with an instance not worse with respect to the objective score (top row). In the bottom row boxes are colored by the first hitting time / (first discovery time).

Figure 8: Representative feature space coverage obtained by the different algorithms for feature combination FC2 and state-of-the-art heuristics LKH and EAX.

5 EXPERIMENTS ON STATE-OF-THE-ART INEXACT TSP SOLVERS

We now consider EAX and LKH and the feature combination FC2 only since the results in Section 4 indicate that this space is harder to cover. Both algorithms are state-of-the-art in inexact TSP solving. EAX [21, 22] is a sophisticated genetic algorithm based on the edge-assembly-crossover operator and entropy population diversification preservation. LKH [15] is a local-search algorithm whose main driver is the Lin-Kernighan heuristics for \(k\)-opt moves. Both algorithms are capable of solving even large instances to optimality in reasonable time. Complementary behavior on a wide range of instances motivated algorithm selection studies [5, 14, 16, 19].

5.1 Experimental setup

Preliminary experiments show that EDO-based algorithms perform worse than QD in this computationally demanding domain with respect to both box-coverage and objective value since the number of iterations is severely reduced to \(\approx 2000\). This is in-line with
Table 2: Table of aggregated values of interest for the experiments on state-of-the-art TSP solvers EAX and LKH. Best values (for the statistics of covered or objective values) or maximal values (for box update statistics) are highlighted.

| FC | Algorithm | EAX vs. LKH | LKH vs. EAX |
|----|-----------|-------------|-------------|
|    | Nr. of boxes | Box statistics | Objective | Nr. of boxes | Box statistics | Objective |
|    | mean | std | med | upd. | hits | best | median | mean | std | med | upd. | hits | best | median |
| FC1 | (1+1) EA [all] | 150.00 | 20.00 | 140.00 | 8.00 | 237.00 | 0.00 | 0.99 | 722.00 | 242.13 | 321.00 | 0.00 | 0.99 |
|    | (50+1) EA-HV [all] | 116.60 | 22.91 | 89.00 | 8.00 | 237.00 | 0.00 | 0.99 | 722.00 | 242.13 | 321.00 | 0.00 | 0.99 |
| QD [all] | 281.00 | 54.50 | 144.00 | 8.00 | 44.00 | 0.00 | 0.94 | 353.00 | 60.79 | 174.00 | 7.00 | 72.00 | 0.04 | 0.38 |

Figure 9: Scatter-plots of PAR10 scores of the two competing algorithms EAX and LKH. Points are colored by the evolving algorithm. The left plot shows the results for the experiments with the goal to produce instances which are easy for EAX and hard for LKH; the right plot shows the vice versa. Points on the diagonal indicate equal performance of both algorithms on the respective instance. Points above/below the diagonal indicate a runtime-advantages for one of the algorithms. The gray dashed lines indicate the maximum possible PAR10-score of 10 · 180 [s].

6 CONCLUSION

The generation of TSP problem instances which are diverse with respect to solver performance and instance features is of utmost importance, e.g., in the field of algorithm selection. We introduced a flexible quality diversity (QD) approach to evolve instances. Instances are mapped to their feature vectors. In an iterative evolutionary process not yet seen instances / feature combinations are stored in any case; instances are overwritten if newly generated instances share the same feature combination, but show a stronger performance difference for two competing TSP algorithms. We compared our approach to classical (μ + 1) EAs and versions adapted to feature-diversity optimization from the literature modified to store their footprints in a similar way. The results show impressively the capability of the QD approach in covering a wide range of feature combinations – and thus producing much more instances in a single, less wasteful run – while competing or even outperforming the classical approaches with respect to objective scores. These results motivate a broad avenue of future work. Here, we only name a few of our most promising ideas: application to > 2 features and other domains, introducing selection bias towards border regions, or storing multiple instances per box.

ACKNOWLEDGMENTS

J. Bossek acknowledges support by the European Research Center for Information Systems (ERCIS). This work has been supported by the Australian Research Council (ARC) through grants DP190103894 and FT200100536. The authors thank the anonymous reviewers for their immensely valuable feedback.
