Measuring Algorithm Footprints in Instance Space

Thomas T. Tan
School of Mathematical Sciences
Monash University
Clayton, Victoria 3800 Australia
thomas.tan@monash.edu

Kate Smith-Miles
School of Mathematical Sciences
Monash University
Clayton, Victoria 3800 Australia
kate.smith-miles@monash.edu

***Abstract*—This paper proposes a new methodology to determine the relative performance of optimization algorithms across various classes of instances. Rather than reporting performance based on a chosen test set of benchmark instances, we aim to develop metrics for an algorithm's performance generalized across a diverse set of instances. Instances are summarized by a set of features that correlate with difficulty, and we propose methods for visualizing instances and algorithm performance in this high-dimensional feature space. The footprint of an algorithm is where good performance can be expected, and we propose new metrics to measure the relative size of an algorithm's footprint in instance space. The methodology is demonstrated using the Traveling Salesman Problem as a case study.**

***Keywords*—instance difficulty; heuristics; traveling salesman problem; performance metrics; algorithm footprints.**

I. Introduction

The operations research literature is dominated by a standard methodology in which a new optimization algorithm is introduced and usually claimed to be superior by outperforming previous approaches on a set of well-studied test instances. However, the weaknesses of an algorithm are rarely reported, and it is unlikely that this new algorithm could claim superior performance on the broadest classes of instances. Indeed, the No Free Lunch Theorem [1] warns us against expecting a single algorithm to perform well on all instances of a problem, regardless of their structure and characteristics. Instead, we are likely to find that there are certain types of instances where this algorithm excels because it is exploiting some particular characteristics of those instances. On different instances, other algorithms may dominate, and the new algorithm may even fail miserably.

In this paper we propose a new methodology to enable researchers to report the strengths and weaknesses of optimization algorithms across a diverse set of instances. To define the terms we will use, consider the Traveling Salesman Problem (TSP) as an example. The *problem* is defined by the generic set of constraints and the objective function, which is to visit each of n cities exactly once and return to the starting city while minimizing travel costs. A particular *instance* of the problem is not defined until we provide the set of n city coordinates. Some instances will be easy (all cities arranged in a circle, for example), and others will be hard for a particular algorithm, depending on the properties of the instance that the algorithm exploits. But how can we create a metric space of instances so that we can readily visualize where they lie and their properties? We define the instance space to be the high-dimensional space that summarizes a set of instances by a feature vector. The measurable *features* or properties of an instance of the TSP, for example, might include the cluster structure of the cities, and statistical properties of the distance matrix. The region of instance space where an algorithm can be expected to perform well has recently been described as an *algorithm footprint* [2], but the methodologies for defining, visualizing, and characterizing this footprint across the high-dimensional instance space do not currently exist.

Understanding and reporting the boundary of algorithm performance is critical for good research practice, and would provide a more accurate and powerful description of an algorithm's strengths and weaknesses. It would also help to avoid trial and error selection of algorithms. It is impractical to determine this boundary through exhaustive testing of the infinite number of possible test instances. More efficient would be to define the boundary by generalizing algorithm performance on selected instances across a diverse instance space, projected onto a two-dimensional plane for ease of visualization.

This paper proposes a methodology to visualize algorithm footprints and to measure their relative size. Section II presents the methodology, which is then demonstrated via a case study of the TSP in Section III. We discuss the findings of this approach in Section IV before drawing conclusions and suggesting future research directions in Section V.

II. Proposed Methodology

The Algorithm Selection Problem framework proposed by Rice [3] in 1976 considers the task of predicting the winning algorithm based on past experience of the relationship between features of an instance and algorithm performance. While Rice's original work was focused on using regression models to predict algorithm performance for partial differential equation solvers, we have previously extended the approach to use various supervised and unsupervised learning methods to predict optimization algorithm performance. For a particular optimization problem, careful choice of features is required, and we have previously demonstrated that the performance of optimization algorithms can be predicted with high accuracy for various problems including TSP [4], quadratic assignment problem [5], scheduling [6], timetabling [7], and graph coloring [8]. Related studies include those by Hall and Posner [9] and Cho et al. [10] on the knapsack problem, and the work of Lleyton-Brown and colleagues on propositional satisfiability [11] and combinatorial auction problems [12].

We now extend Rice's framework in a different direction to enable, not just winning algorithm prediction, but visualization and measurement of algorithm footprints. Our methodology starts by defining an instance space as a metric space formed in $\mathbb{R}^n$ based on a coordinate system defined by a set of n feature vectors. Instances are defined as points in $\mathbb{R}^n$ and the distances between instances are defined by a Euclidean metric. We then use principal component analysis to project the instances into a two-dimensional space for ease of visualization. The classes of instances can be identified in this instance space, and the performance of algorithms on the instances can be labeled as good or bad according to some metric. The footprint of an algorithm can then be visualized, and we propose the use of convex and non-convex hulls to measure the area of the footprint relative to the area of the instance space. Details of each stage of the methodology are discussed below.

A. STEP 1: Summarizing instance as feature vectors

In order to visualize the differences between instances of an optimization problem, and the performance of algorithms on these instances, our first challenge is to be able to view two instances in the same space in such a way that if the two instances are similar according to some metric (Euclidean distance), they are close in the instance space, and if they are dissimilar then they are far apart in the instance space. Since we are focused on arranging the instances in a space where the easy instances might be separated from the hard instances, it is natural to represent each instance as a feature vector that considers a number of properties known to correlate with instance difficulty.

While it may seem limiting to build a methodology that relies on appropriate choice of features that summarize the difficulty of instances, fortunately the area of optimization instance difficulty is one that has been researched for many decades, and there is much already known about the properties of optimization problem instances that make them easy or hard. We refer the interested reader to our recent survey paper [13] for a review of various measures of instance difficulty for broad classes of optimization problems including assignment, knapsack, bin-packing, timetabling, graph problems and TSP.

B. STEP 2: Evolving easy and hard instances

Suppose a set of instances can be summarized by a set of n features, so that each instance is a point in the feature space defined in $\mathbb{R}^n$. Two instances are said to be similar if their distance in $\mathbb{R}^n$ is close. It is important for our methodology that our chosen instances are not restricted to only a small region in feature space. If the instances are very similar to each other, then we should not expect the performance of algorithms to be very different, and we will lose the opportunity to see the strengths and weaknesses of algorithms. We must include instances that are diverse and well-spread across the instance space, especially if we expect to be able to generalize algorithm performance well.

If we rely only on the instances that are commonly available (for example OR-library instances) or randomly generated instances, we may find that we do not achieve the diversity we seek. Previous studies [10] have demonstrated that the knapsack benchmark instances are not as diverse in terms of the features that make them difficult as we would need for our purposes, and we have previously shown that randomly generated TSP instances are inadequate to reveal the strengths and weaknesses of some heuristics [4].

Instead of relying on randomly generated or benchmark instances alone, we have adopted a methodology of using evolutionary algorithms to evolve instances that are intentionally easy or hard for certain algorithms [4]. To obtain hard instances, an initial population of random instances is evolved with a fitness function that measures how long it takes the algorithm to find a good solution, or the gap to a known optimal or good solution obtained after a fixed runtime. Since we are maximizing fitness, the final population consists of a collection of instances that take a long to time to solve or return a poor solution after a fixed runtime. These are hard instances, and a similar approach can be used to evolve easy instances by inverting the fitness function.

C. STEP 3: Visualizing instances in feature space

Once we have a diverse set of m instances, and can summarize each instance as an n-dimensional feature vector of relevance to instance difficulty, our challenge is to visualize the instances in the n-dimensional feature space. Our data matrix X is of dimension $m \times n$. We use principal component analysis (PCA) to project the m instances into a 2-dimensional instance space, where the two axes correspond to the two most dominant eigenvectors of the covariance matrix $C = X^{\mathsf{T}}X$. This 2-d instance space will be an adequate representation of the original set of instances provided the first two dominant eigenvectors represent a high enough percentage of the variation in the data, as measured by the ratio of the first two eigenvalues compared to the sum of all n eigenvalues.

At this stage we should be able to clearly see all instances in the instance space, and note the location of different types of instances (random, benchmark, evolved easy, evolved hard). If the features are adequate then we should find the different types of instances grouped together in the feature space, particularly the easy instances well separated from the hard instances. If we cannot achieve this kind of separation, then we must question our choice of features and the degree to which they really do correlate with instance difficulty.

D. STEP 4: Visualizing algorithm footprints in instance space

With an adequate 2-dimensional representation of the instance space, we can now superimpose algorithm performance on each instance. We first need to decide how we will determine the goodness of an algorithm's performance. For example, we may define an algorithm's performance to be good if it achieves a solution that lies within 5% of the known optimal solution after a fixed runtime. Alternatively, a "good" performance might be a runtime less than 5 minutes to achieve a solution of a certain precision.

Once we have determined our criteria for measuring good algorithm performance, we can label all instances for a certain algorithm with a binary class (1 if good and 0 if bad). Our algorithm performance results can be stored in a binary matrix Y of dimension $m \times P$, where P is the number of algorithms we are evaluating.

It should be noted that a traditional interpretation of Rice's algorithm selection problem uses m training data examples to learn the relationship between features and algorithm performance, i.e. builds P logistic regression models to predict each column of Y (dependent variable) in terms of the columns of X (independent variables).

Rather than solving a classification problem in an n-dimensional space though (using logistic regression or support vector machines for example) to accurately predict algorithm performance, we are focused here on visualizing a generalized boundary of algorithm performance (the footprint), and measuring the relative size of the footprint. We attach labels to the instances where good performance is obtained, in the 2-dimensional instance space, and use the boundary of the convex hull [14], [15] of these instances to define the boundary of the algorithm footprint. In order to ensure appropriate generalization, we first remove outliers by clustering the instances labeled within the footprint. In the event that the generalized boundary is still considered too much of a generalization beyond the observable performance from the instances, we propose to calculate a non-convex hull by performing a Delaunay triangulation [16] on the instances within the convex hull, and removing the outer edges. While there are other non-convex hull generation methods that have been proposed to determine the area occupied by a set of points [14], we adopt this simple strategy in the first instance.

Proposition. *The convex hull with a Delaunay triangulation can be converted into a non-convex hull by deleting all the boundary edges.*

Proof: Suppose there is an edge $e = (v_i, v_j)$ in the boundary of the Delaunay triangulation (a convex hull), where $i \neq j$ and v_i and v_j are extreme points in the convex hull. The deletion of edge e means the simple path from v_i to v_j is now through some other point v_k. Since v_k was not an extreme point of the convex hull, (v_i, v_k, v_j, v_i) constitutes a triangle. If we delete the edge e and the area of the corresponding boundary triangle from the hull, the straight line segment between v_i and v_j no longer lies entirely within the hull, and so the hull is no longer convex. □

We can now visualize in the 2-d instance space either a broad generalized footprint of an algorithm using the convex hull of the instances where algorithm performance has been measured to be good; or a smaller footprint using a non-convex hull after removing outliers to define a more conservative footprint where algorithm performance is expected to be good only based on strong observational evidence. An out of sample test set can be used to validate that instances that lie within the boundary of a footprint are accurately predicting good performance.

E. STEP 5: Measuring the relative size of algorithm footprints

Once we have a hull that covers all instance points labeled good for a certain algorithm, we can calculate the area of the hull, and take its ratio to the area of the hull covering all instances. This will provide a metric for the relative goodness of the algorithm across the entire instance space. We should subtract from this area any region where we have evidence that the algorithm performance may not be good (i.e. remove good instances that lie within the hull of other instances labeled as not good for the algorithm, which therefore contradict the footprint).

Specifically, the convex hull of a region defined by a set of points $S = \{(x_i, y_i)$ for all $i = 1, \ldots, \eta\}$ we notate as $\mathcal{H}(S)$, with the area given by

$$Area(\mathcal{H}(S)) = \frac{1}{2}\sum_{j=1}^{k}(x_j y_{j+1} - y_j x_{j+1}) + (x_k y_1 - y_k x_1)$$

with the subset $\{(x_j, y_j)$ for all $j = 1, \ldots, k\}$ and $k \leq \eta$ defining the extreme points of $\mathcal{H}(S)$.

Let Ω be the set of m points defining the m instances in the two-dimensional projected feature space, and let Ψ_n be the set of points (instances) where algorithm $\pi \in \{1, \ldots, P\}$ is labeled "good", excluding those that lie in the hull of any points where the algorithm is not good. Then the area of the footprint of algorithm π is given by $Area(\Psi_n)$, and the relative size of this footprint is given by the ratio $Area(\Psi_n)/Area(\Omega)$. We can then compare the relative sizes of each algorithm $\pi \in \{1, \ldots, P\}$ to determine which algorithm has the largest footprint, and explore the degree of overlap of the footprint with particular regions of interest, such as real-world instances.

III. Case Study: The Traveling Salesman Problem

We now demonstrate the proposed methodology on the well-studied Traveling Salesman Problem (TSP). In previous work we have studied how TSP instances can be summarized with features that correlate with difficulty, and have shown that randomly generated TSP instances are not particularly difficult [4]. Here we consider an augmented set of features compared to our previous work. We have earlier used evolutionary algorithms to evolve easy and hard TSP instances based on a fitness function that measures how many iterations of an algorithm are required to converge to a locally optimal solution [4]. Here we extend the ideas to firstly redefine the fitness function to generate hard instances that have a maximal optimality gap, and easy instances with a minimal optimality gap, measured using Concorde [17]. Thus our approach to steps 1 and 2 are different from our previous studies. Steps 3, 4 and 5 of the proposed methodology have been applied here to the TSP for the first time. We also include benchmark instances from the TSPLIB [18], [19] since we are interested to see where they lie in the instance space compared to random, easy and hard instances.

Our choice of algorithms are two variations of the Lin-Kernighan heuristic [20]. Developed more than thirty years ago, the Lin-Kernighan heuristic is still known for its success in efficiently finding near-optimal results. The core of the algorithm and its variants, consists of edge exchanges in a tour. We consider two variants:

- Chained Lin-Kernighan (CLK) is a variant [21] that chains multiple runs of the Lin-Kernighan algorithm to introduce more robustness in the resulting tour. Each run starts with a perturbed version of the final tour of the previous run. The length of the chain depends on the number of cities in the TSP problem.
- Lin-Kernighan with Cluster Compensation (LKCC) is a reaction to the poor performance of the Lin-Kernighan heuristic reported in [22] for clustered instances. It aims to reduce the computational effort, while maintaining the quality of solutions produced for both clustered and non-clustered instances. The cluster distance between cities v and w is calculated as the minimum bottleneck cost (heaviest edge) of any path v and w. These values are then used in the guiding utility function of Lin-Kernighan to prune unfruitful regions (with high bottlenecks) from the search space.

These two algorithms are essentially similar, with subtle differences in how they behave, noticeable particularly for instances that have well defined cluster structures [4]. While we could have chosen any algorithms to demonstrate the proposed methodology, we are interested to see if the distinguishing footprints of two quite similar related algorithms will be clearly visible in the instance space.

TABLE I: Evolutionary Algorithm Parameters

Parameter	Description
Chromosome	Vector of length 200 containing 100 (x, y) city coordinates in 400×400 grid.
Population	30 chromosomes.
Generations	600
Selection	2 tournament with 1-elitism.
Crossover	Uniform random.
Mutation	Random mutation of each (x, y) coordinate with probability $(1 + 99(2^{-g/3}))/200$ where g is the generation number, following the scheme of [23].

A. TSP instances and features

We consider only 100 city TSP instances, and have three types of instances: randomly generated, TSPLIB benchmark instances, and instances that we have evolved to be easy or hard for each algorithm. A description follows:

1) Randomly generated instances: We have generated 190 instances randomly placing 100 cities in a 400×400 plane.

2) TSPLIB instances: We have selected 6 instances from TSPLIB with exactly 100 cities, namely instances *kroA100*, *kroB100*, *kroC100*, *kroD100*, *kroE100*, and *rd100*.

3) Evolved instances: Following the approach used in our previous work [4], and summarized in Table I, we have evolved instances that are designed to be easy or hard for a given algorithm from our set (CLK and LKCC). The seed for the evolutionary algorithm is the same of randomly generated 190 instances defined in *1)* above.

For hard instances, our fitness function is the ratio of the optimality gap produced by the algorithm compared to the exact tour length found using Concorde [17], after the algorithm is run to convergence. For easy instances we take the inverse of this ratio, so that the evolutionary algorithm can maximize the fitness and we achieve the desired result. In addition to evolving instances that are easy for CLK, hard for CLK, easy for LKCC, hard for LKCC, we also evolve instances that are uniquely hard for CLK (and simultaneously easy for LKCC), and uniquely hard for LKCC (and simultaneously easy for CLK). The fitness function for the uniquely hard instances for a specified algorithm is the ratio of the algorithm's optimality gap to the other algorithm's optimality gap which, when maximized, will create instances that are hard for the specified algorithm, and easy for the other algorithm. For uniquely easy instances we invert this ratio. In total we have 6 sets of evolved instances, with 190 instances per set. All details of the evolutionary algorithm implementation are the same as those used in our previous work [4] and summarized in Table I, except for the different definition of the fitness function. Thus, in total we study 1336 instances of 100 city TSPs.

A comprehensive set of $n = 40$ measurable characteristics were used to summarize the properties of these TSP instances, based on 28 features from our previous work [4], 10 features from Kanda et al. [24], and two new features that correlate with tour length. These features are:

1. Standard deviation of inter-city distances
2–3. The (x, y) centroid of the TSP instance
4. Radius of TSP instance (mean distance from cities to centroid)
5–8. Fraction of distinct distances in the distance matrix, to 1, 2, 3 and 4-decimal places
9. Rectangular area in which the cities lie
10–11. Normalized variance and coefficient of variation of the nNNd's (normalized nearest neighbor distances)
12. Number of clusters found using GDBSCAN, max=10
13. Cluster ratio (number of clusters to the number of cities)
14–23. Number of cities in each of 10 potential clusters
24. Outlier ratio (number of outliers to number of cities)
25. Variance of the number of cities in each cluster
26. Number of outliers
27. Ratio of nodes near the edges of the plane
28. Mean radius of the clusters
29. Value of the smallest edge (distance)
30. Value of the largest edge
31. Value of the edge average
32. Value of the edge standard deviation
33. Quantity of values in the edge mode set
34. Frequency in the edge mode set
35. Modal average of the edges
36. Value of the edges median
37. Number of edges with value lower than the edge average
38. Sum of smallest 25% of edges (distances)
39. Number of edges with value higher than the edge average
40. Sum of the largest 25% of edges (distances).

B. *TSP instances in feature space*

Rather than visualizing these 1336 TSP instances in a 40 dimensional feature space, we use PCA to project the instances into a two-dimensional instance space. The top two eigenvectors account for 74% of the variation in the data, with a 3rd eigenvector increasing this to 79%. For ease of visualization we will utilize two-dimensional data reduction, but naturally the methodology generalizes easily to 3-dimensional footprints, where the volume is calculated rather than the area. Fig. 1 shows the different sets of instances located in the 2-d instance space. It reveals that the easy and hard instances for each algorithm are at opposite ends of the instance space, and that the random instances and TSPLIB instances are in the center. Clearly the evolved instances have enabled us to create a broader instance space than would have been possible had we only relied on random or benchmark instances. We now have a set of instances that enable the relative strengths and weaknesses of the two algorithms to be understood in the feature space.

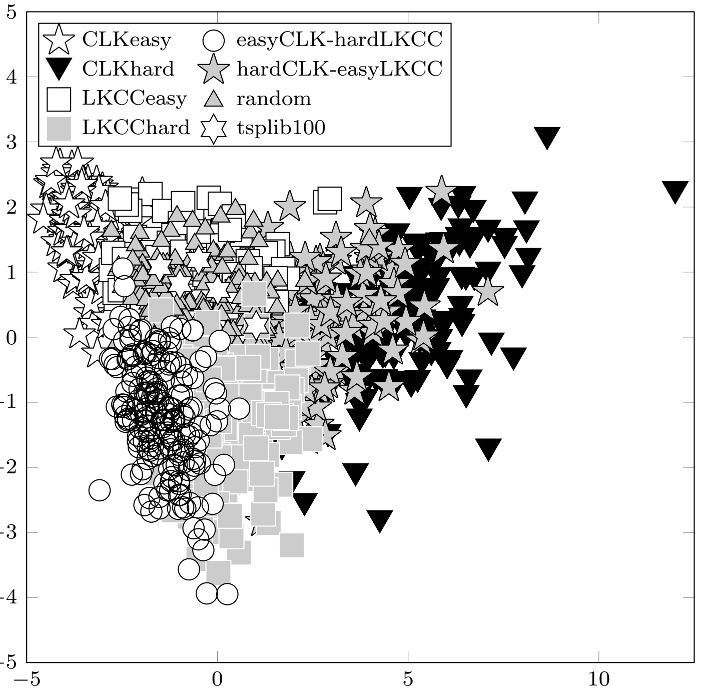

Fig. 1: Instance space after PCA showing instance sets.

C. *TSP algorithm footprints*

By evaluating the performance of each of the 1336 instances using both CLK and LKCC, we can label each instance as "CLK good", "CLK bad", "LKCC good" or "LKCC bad". We define goodness arbitrarily for this case study as the algorithm obtaining a tour length that is within a certain percentage of the known optimal found using Concorde [17]. We used discriminant analysis on the evolved easy and hard instances for each algorithm to obtain the optimality gap that best differentiates the easy and hard instances. For CLK this was 3.4% and for LKCC this was 0.07%. A visual inspection of the distributions of optimality gaps for these two classes of instances had already revealed that 1% was a reasonable threshold for both.

Fig. 2 shows the regions in instance space where each algorithm's performance is "good" or "bad" (with "bad" meaning the tour length is greater than 1% of the optimal tour length).

Fig. 3 breaks down the performance of each algorithm according to how the other algorithm performed on the same instances. The algorithm footprint for CLK is the union of the green and yellow regions (top row), with the yellow region alone (top right) showing where CLK is uniquely good. The algorithm footprint for LKCC is the union of the green and blue region (left column), with the blue region alone (bottom left) showing where LKCC is uniquely good.

Of course, it is natural to now wonder how the characteristics of certain parts of the instance space

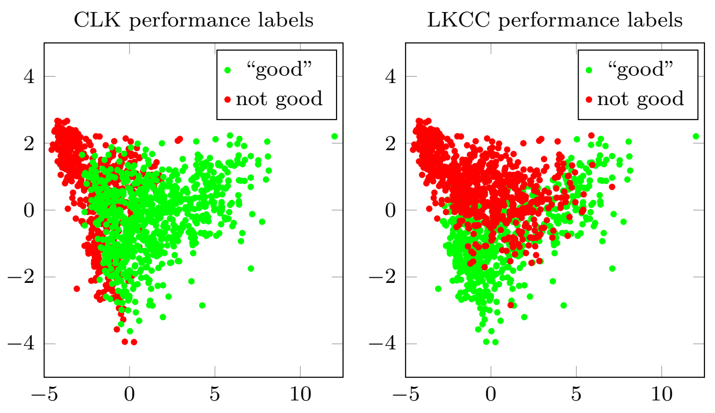

Fig. 2: Good performance (green) and bad performance (red) for CLK (left) and LKCC (right) in instance space.

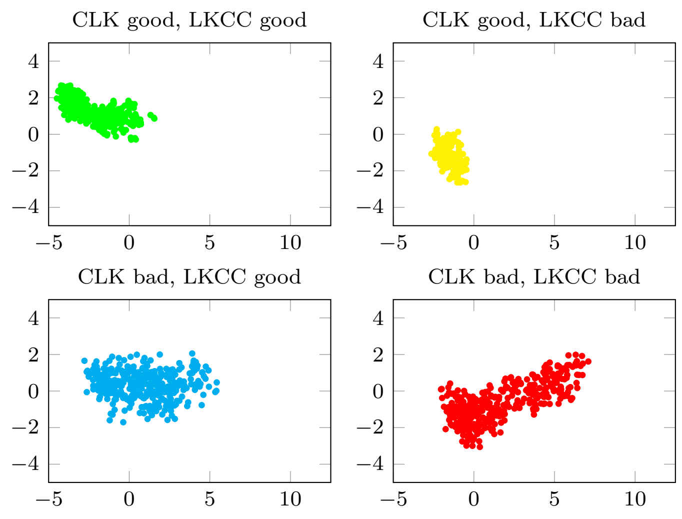

Fig. 3: Performance in instance space showing unique footprints.

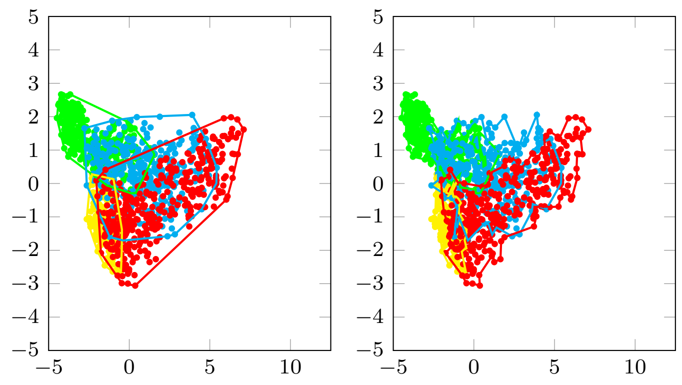
Fig. 4: Convex hulls (left) and non-convex hulls (right) for each combination of good and bad performance from the two algorithms shown in Fig. 3.

contribute to good or bad performance (i.e. which features and what values of those features are found in certain regions of the instance space where algorithm performance is good or bad). It is straightforward to use machine learning methods to derive rules explaining how certain values of the features (stored in the matrix X) affect the performance labels (stored in the matrix Y). We have done this in our previous work [4], and it is not the focus of the present study. Instead we focus here on measuring the relative performance of an algorithm across a broad instance space.

D. *TSP algorithm footprint metrics*

For each region of interest shown in Fig. 3, we can calculate the area of the region, using either the convex hull, or a non-convex hull if we are concerned that we have insufficient instances near the boundary to justify the generalization. After removing outliers, the algorithm for calculating the area of the hulls described in the methodology is used, and the resulting areas are shown in Fig. 4. It is clear that these regions are not disjoint, and there are overlapping regions (such as near the boundary between yellow and red) where there is no crisp generalization that can be made. While LKCC is consistently bad in this region for example, the performance of CLK is sometimes good and sometimes bad, and no doubt our threshold of 1% optimality gap to define goodness, if adjusted, would affect the boundaries of the hulls.

When considering the relative size of the areas of good performance, we must therefore consider any overlap with contradictory performance. For example, if we wish to know the size of the unique footprint for CLK (yellow), then we should not include any points inside the hull of any region where CLK is shown to be bad, since it contradicts the footprint. Thus we are left with only a thin sliver of yellow along the left edge in Fig. 4 where the unique footprint of CLK exists. The total (non-unique) footprint for CLK is the union of this thin yellow sliver and the green region at the top that is not within the blue region's boundary.

Likewise, for LKCC the footprint is the blue region where there is no overlap with the red region (since that contradicts good performance for LKCC). The unique footprint for LKCC also has to exclude the blue regions where there is overlap with the green regions (where CLK is also good). For the convex hull, this leaves no unique footprint for LKCC, while the non-convex hull reveals a small group of instances in the top-center of the instance space where LKCC is uniquely good.

Suppose we define the following sets:

- S_{gg} is the set of instances where the performance of both CLK and LKCC is good
- S_{bb} is the set of instances where the performance of both CLK and LKCC is bad
- S_{gb} is the set of instances where the performance of CLK is good and LKCC is bad
- S_{bg} is the set of instances where the performance of CLK is bad and LKCC is good.

Mathematically, we can describe the footprint of CLK as:

$$\mathcal{H}\big((S_{gg} \setminus \{(x_i, y_i) \in \mathcal{H}(S_{bg})\}) \cup (S_{gb} \setminus \{(x_i, y_i) \in \mathcal{H}(S_{bb})\})\big)$$

which gives us the hull of the points where CLK is shown to be good (S_{gg} and S_{gb}) after removing those instances that lie in the hulls of S_{bg} and S_{bb} which contradict expectation of good performance.

Likewise, the *unique* footprint of CLK can be described as:

$$\mathcal{H}(S_{gb} \setminus \{(x_i, y_i) \in \mathcal{H}(S_{gg}) \cup \mathcal{H}(S_{bg}) \cup \mathcal{H}(S_{bb})\})$$

which gives us the hull of the points where CLK is shown to be uniquely good (S_{gb}) after removing those instances that lie in the hulls of contradictory regions (S_{gg}, S_{bg}, S_{bb}).

Similar descriptions are easily found for the LKCC footprint and LKCC unique footprint, and then the areas of these footprints can be calculated.

Table II shows relative size of the areas of the derived footprints, using both the convex hull and non-convex hull approaches for the optimality gap of 1% defining good performance.

IV. Discussion

The results presented in the previous section assumed a definition of good as an algorithm's tour length to be within 1% of the known optimal tour length. Obviously the size of the footprints depends on our definition of goodness. Fig. 5 shows how the area of the footprints for CLK varies with the optimality gap used to define goodness. While the area of the footprint where CLK is expected to perform well increases monotonically as the definition of goodness is increasingly relaxed to include a larger optimality gap, we see that the size of the unique footprint for CLK is maximized with a definition of goodness within 1.1% of optimal. Any more relaxed definition of goodness means that, while CLK will perform well, so will LKCC. Similar footprint areas for LKCC are shown in Fig. 6, where we observe the same behaviors.

While this paper has focused on a comparison of only two algorithms (CLK and LKCC), the extension of the ideas to more than two algorithms presents no difficulties aside from the minor issue of subtracting the areas of the overlapping regions, which will grow combinatorially with more algorithms.

This paper has also chosen to represent the instance space as a two-dimensional projection of the feature space, but the methodology generalizes easily to three-dimensions, where 3 principal components are used to represent the instance space, and the volume of the footprint is calculated, rather than the area. There are some computational issues associated with calculating the hyper-volume of unions and intersections of regions in higher dimensions, but approximation heuristics are available [25], and we do not expect that three-dimensions will cause problems.

It should be noted that our decision to represent the instance space using a data reduction method like PCA has meant that we are inevitably discarding some information that may be useful to more accurately represent the relationship between features and algorithm performance. In the case study presented in this paper, we lost 26% of the variation in the original feature space by projecting to two-dimensions. Naturally, an optimal feature selection strategy could be employed to ensure that the principal components retain as much of the variation in the data as possible, and new features could be considered as well.

We should also mention that, while the footprints presented in this paper have been contiguous regions in

TABLE II: Areas of Footprints for Goodness Defined as 1% Optimality Gap

Footprint	**Convex Hull**		**Non-Convex Hull**	
	Points	**Area (%)**	**Points**	**Area (%)**
CLK	202	11.27 (21.22 %)	223	9.94 (20.57 %)
CLK unique	17	3.12 (5.87 %)	23	3.06 (6.33 %)
LKCC	188	8.84 (16.64 %)	222	7.59 (15.71 %)
LKCC unique	3	0.70 (1.32 %)	22	0.71 (1.47 %)
All instances	1336	53.11 (100 %)	1336	48.32 (100 %)

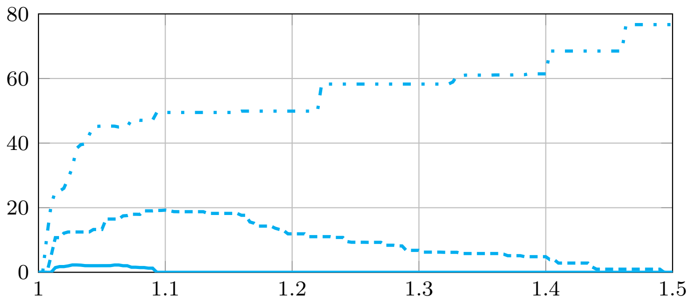

Fig. 5: Area of footprints (vertical axis) for different optimality gaps (horizontal axis). Footprints measured are where CLK is good (top line), where CLK is uniquely good (middle line), and where CLK is uniquely good and there are no contradictions (bottom line).

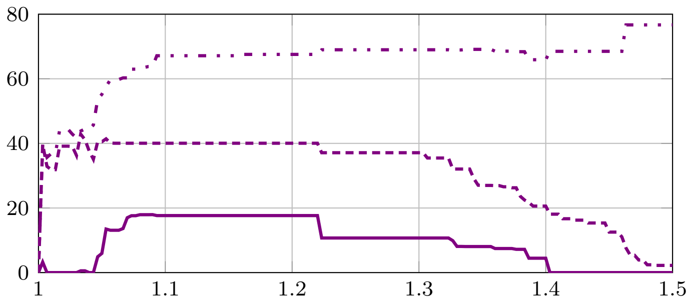

Fig. 6: (horizontal axis). Area of footprints (vertical axis) for different optimality gaps (horizontal axis). Footprints measured are where LKCC is good (top line), where LKCC is uniquely good (middle line), and where LKCC is good and there are no contradictions (bottom line).

instance space, there is no theoretical reason why we should always expect this to be the case. In fact, we should expect that an algorithm's footprint may consist of a number of regions throughout instance space where the performance is good for different types of instances. Rather than measuring the area of the convex hull around all good regions, and then suffering with many contradictions in the middle that need to be subtracted, it may be preferable to first cluster the good labeled instances, and then calculate the area of each cluster component of the footprint.

Finally, it should be noted that the approach taken in this paper has been to treat the labeling of instances as good or bad in a very binary manner. In reality, we would not define algorithm performance to be bad if it fails to meet the goodness criteria by a small amount. A gray buffer region is probably needed, in addition to definitions of bad and good. The ideas presented here are easily extended to consider this ternary classification approach.

V. Conclusions

This paper has presented a new methodology to be able to assess the relative power of various optimization algorithms. We represent a diverse set of instances as points in a high-dimensional feature space, then project to a lower dimensional instance space for ease of visualization. Within this diverse instance space we can then seek greater understanding of the strengths and weaknesses of optimization algorithms. We can define good performance, and then see which parts of the instance space yield good performance from different algorithms. This involves statistical generalization from the training data. The boundaries in instance space between good and bad performance define the edge of the footprint of an algorithm. We have proposed a methodology for measuring the size of the footprint of an algorithm, that is, the region where good performance can be expected, statistically.

The proposed approach has been illustrated using the Traveling Salesman Problem, solved by two common heuristics. The results show clearly where each algorithm is good, according to a specified optimality gap, and where it is uniquely good. The relative sizes of these footprints reveal the power of each algorithm across instance space.

We have shown that the 100 city TSPLIB instances are not particularly discriminating of algorithm performance for these two heuristics, with all TSPLIB instances falling within the footprint of both CLK and LKCC. The same is true for the randomly generated TSP instances. Only when we intentionally generate easy or hard instances do we see where the true strengths of an algorithm lie in instance space.

We believe this methodology is an important first step towards providing researchers with a tool to report the strengths and weaknesses of their algorithms, and to show the relative power of an algorithm either across the entire instance space, or on a particular region of interest (e.g. real world problems). Our current work is focused on developing the key components of the methodology (evolved instances, feature sets) for a number of broad classes of optimization problems [13], which will then become available as tools for researchers. The generalization of the approach to consider not just algorithm selection, but parameter selection (for example, evolutionary operator selection [26] is a natural extension for the evolutionary computation community.

Acknowledgment

This work is supported by the Australian Research Council under grant DP120103678.

References

[1] D. H. Wolpert and W. G. Macready, “No Free Lunch Theorems for Optimization,” IEEE Transactions on Evolutionary Computation, vol. 1, pp. 67-82, 1997.

[2] D. Corne and A. Reynolds, “Optimisation and Generalisation: Footprints in Instance Space,” Parallel Problem Solving from Nature, PPSN XI pp. 22-31, 2010.

[3] J. R. Rice, “The Algorithm Selection Problem,” Advances in Computing, vol. 15, pp. 65-118, 1976.

[4] K. Smith-Miles and J. van Hemert, “Discovering the Suitability of Optimisation Algorithms by Learning from Evolved Instances,” Annals of Mathematics and Artificial Intelligence, vol. 61, no. 2, pp. 87-104, 2011.

[5] K. Smith-Miles, “Towards insightful algorithm selection for optimisation using metalearning concepts,” IEEE International Joint Conference on Neural Networks, pp. 4118-4124, 2008.

[6] K. Smith-Miles, R. James, J. Giffin and Y. Tu, “A Knowledge Discovery Approach to Understanding Relationships between Scheduling Problem Structure and Heuristic Performance,” Proceedings of the 3rd Learning and Intelligent Optimization conference, Lecture Notes in Computer Science, vol. 5851, pp. 89-103, 2009.

[7] K. Smith-Miles and L. Lopes, “Generalising algorithm performance in instance space: a timetabling case study,” Proceedings of the 5th Learning and Intelligent Optimization conference, Lecture Notes in Computer Science, vol. 6683, pp. 524-538, 2011.

[8] K. Smith-Miles, B. Wreford, L. Lopes and N. Insani, “Predicting metaheuristic performance on graph coloring problems using data mining,” in E. Talbi, Ed., Hybrid Metaheuristics, Springer Series in Computational Intelligence, in press.

[9] N. Hall and M. Posner, “Performance Prediction and Preselection for Optimization and Heuristic Solution Procedures,” Operations Research, vol. 55, pp. 703-716, 2007.

[10] Y. Cho, J. Moore, R. Hill and C. Reilly, “Exploiting empirical knowledge for bi-dimensional knapsack problem heuristics,” International Journal of Industrial and Systems Engineering, vol. 3, pp. 530-548, 2008.

[11] L. Xu, F. Hutter, H. Hoos and K. Leyton-Brown, “SATzilla-07: The design and analysis of an algorithm portfolio for SAT,” Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 4741, pp. 712-727, 2007.

[12] K. Leyton-Brown, E. Nudelman and Y. Shoham, “Empirical hardness models: Methodology and a case study on combinatorial auctions,” Journal of the ACM, vol. 56, pp. 1-52, 2009.

[13] K. Smith-Miles and L. Lopes, “Measuring Instance Difficulty for Combinatorial Optimization Problems”, Computers and Operations Research, vol. 39, pp. 875-889, May 2012.

[14] A. Galton and M. Duckham, “What is the region occupied by a set of points?,” GIScience, Lecture Notes in Computer Science, vol. 4197, pp. 81-98, 2006.

[15] M. Duckham, L. Kulik, M. Worboys and A. Galton, “Efficient generation of simple polygons for characterizing the shape of

a set of points in the plane," Pattern Recognition, vol. 41, pp. 3224-3236, 2008.

[16] D. T. Lee and B. J. Schachter. "Two algorithms for constructing a Delaunay triangulation," International Journal of Parallel Programming, vol. 9, no. 3, pp. 219242, June 1980.

[17] Concorde TSP Solver, http://www.tsp.gatech.edu/concorde.html

[18] G. Reinelt, "TSPLIB – A traveling salesman problem library" INFORMS Journal in Computing, vol. 4, pp. 376-384, 1991.

[19] TSPLIB, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95

[20] S. Lin and B. Kernighan, "An efficient heuristic algorithm for the traveling salesman problem," Operations Research, vol. 21, pp. 498-516, 1973.

[21] D. Applegate, W. Cook and A. Rohe, "Chained Lin-Kernighan for large traveling salesman problems," INFORMS Journal on Computing , vol. 15, pp. 82-92, 2003.

[22] D. Johnson and L. McGeoch, "The traveling salesman problem: a case study," in Local Search in Combinatorial Optimization, E. Aarts and J. Lenstra, Eds. John Wiley & Sons, Inc, 1997, pp. 215-310.

[23] J. Kratica, I. Ljubic and D. Toic, "A genetic algorithm for the index selection problem." In: G. Raidl, et al. (eds.) Applications of Evolutionary Computation, vol. 2611, pp. 281-291. Springer-Verlag, 2003.

[24] J. Kanda, A. Carvalho, E. Hruschka and C. Soares, "Using meta-learning to classify traveling salesman problems," 2010 Eleventh Brazilian Symposium on Neural Networks (SBRN), pp. 73-78, 2010.

[25] K. Bringmann and T. Friedrich, "Approximating the volume of unions and intersections of high-dimensional geometric objects," Computational Geometry: Theory and Applications, vol. 43, no. 6-7, pp. 601-610, 2010.

[26] S. Shirakawa, N. Yata, and T. Nagao, "Evolving search spaces to emphasize the performance difference of real-coded crossovers using genetic programming," 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1-8, 2010.