Understanding TSP Difficulty by Learning from Evolved Instances

Kate Smith-Miles[1], Jano van Hemert[2], and Xin Yu Lim[1,3]

[1] School of Mathematical Sciences, Monash University, Victoria 3800, Australia
kate.smith-miles@sci.monash.edu.au
[2] School of Informatics, University of Edinburgh, EH8 9AB Edinburgh, UK
j.vanhemert@ed.ac.uk
[3] Mathematical Institute, Oxford University, OX1 3TG Oxford, UK
limxinyuuu@gmail.com

Abstract. Whether the goal is performance prediction, or insights into the relationships between algorithm performance and instance characteristics, a comprehensive set of meta-data from which relationships can be learned is needed. This paper provides a methodology to determine if the meta-data is sufficient, and demonstrates the critical role played by instance generation methods. Instances of the Travelling Salesman Problem (TSP) are evolved using an evolutionary algorithm to produce distinct classes of instances that are intentionally easy or hard for certain algorithms. A comprehensive set of features is used to characterise instances of the TSP, and the impact of these features on difficulty for each algorithm is analysed. Finally, performance predictions are achieved with high accuracy on unseen instances for predicting search effort as well as identifying the algorithm likely to perform best.

Keywords: Algorithm Selection, Travelling Salesman Problem, Hardness Prediction, Phase Transition, Combinatorial optimisation, Instance Difficulty.

1 Introduction

A generic answer to the question of what makes an optimisation problem hard has been elusive, since it is clear that problem specific characteristics determine the intrinsic difficulty of a particular instance of an optimisation problem for a particular algorithm [1]. Consequently, there has been increasing interest in measuring the characteristics of instances for a given optimisation problem to gain insights into which features of an instance make a particular algorithm perform well or poorly. Examples of such studies include Nudelman et al. [2] and Xu et al. [3] for SAT problems, Cho et al. [4] and Hall and Posner [5] for knapsack problems, Smith-Miles [6] for the QAP, and Smith-Miles et al. for job shop scheduling [7]. The methods used to predict and gain insights come from the statistical and machine learning communities, and the task is to learn from performance results to give the optimisation process increased intelligence about algorithm suitability.

C. Blum and R. Battiti (Eds.): LION 4, LNCS 6073, pp. 266–280, 2010.

The algorithm selection framework of Rice [8] provides a convenient representation of this task, as discussed in a recent survey paper [6], and enables us to generalize the important components of the methodology. The algorithm selection problem requires a complete set of meta-data from which to learn, comprising: a large set of diverse instances, suitable features with which to characterise the instances, a diverse portfolio of algorithms, and a performance metric for the algorithms. Once we have sufficient meta-data, we can apply machine learning methods to learn the relationships within the meta-data.

One of the key challenges to successful learning from the meta-data though, is to ensure sufficient diversity of the instances in both feature space and algorithm performance space. There is nothing meaningful that will be learned from the meta-data if all instances map to the same region in feature space (in which case the features are not discriminating enough to model sub-classes of instances), or if all algorithms perform similarly on all instances (in which case the instances are not discriminating enough to test algorithm performance). The instance generation method, and the choice of features are clearly critical to the success of any subsequent analysis of the meta-data.

The most common approach to generating problem instances for meta-data is to randomly generate instances based on statistically varying features known to relate to instance difficulty. The performance of algorithms can then be measured on these instances, and we hope that the instance generation process has created sufficient diversity in the feature space that some meaningful relationships can be inferred. The limitation of this approach is that generating diverse random instances is challenging [4], and randomly generated benchmark datasets are often quite restricted in the spectrum of difficulty.

An alternative approach is to intentionally generate instances that are easy or hard for certain algorithms using an evolutionary algorithm [9], and then examine the variation of interesting features across these sets with a view to understanding which features correlate with instance hardness. In this paper we adopt this latter approach, thereby proposing a methodology for ensuring that the collection of instances contains sufficient diversity to be useful for the analysis of the meta-data. By using an evolutionary algorithm to evolve instances that are hard or easy for certain algorithms, we are better able to test the discriminatory power of candidate feature sets in a more effective manner than relying on random instance generators. We demonstrate this methodology on the symmetric Travelling Salesman Problem. Much is known after many decades of research about features that tend to make particular instances of the TSP easy or hard for certain algorithms. We test the effectiveness of these features as predictors of algorithm performance, using evolved instances, as well as randomly generated instances, on two heuristics: chained Lin-Kernighan and Lin-Kernighan with cluster compensation.

The remainder of this paper is as follows. Section 2 discusses the algorithm selection framework we have adopted for performance prediction and analysis based on meta-data. The critical components of instance generation and feature selection are discussed in more detail, as they apply to the Travelling Salesman

Problem, in the next two sections. Section 3 reviews the characteristics of the TSP that have been shown to correlate with instance difficulty for various algorithms, and forms the basis of our feature set. Section 4 presents the evolutionary algorithm methodology for evolving various instances of the TSP that are easy or hard for the two heuristics. The entire meta-data is presented in Section 5 together with an extensive analysis focused on establishing the suitability of the meta-data for subsequent learning. Results for performance prediction, and analysis revealing the impact of TSP characteristics on algorithm performance are also presented in Section 5, before future research directions and conclusions are drawn in Section 6.

2 Algorithm Selection and Performance Prediction

The Algorithm Selection Problem, first posed by Rice [8], seeks to predict which algorithm from a portfolio is likely to perform best based on measurable features of a collection of problem instances. There are four essential components of the model:

- the problem space $\mathcal{P}$ represents the set of instances of a problem;
- the feature space $\mathcal{F}$ contains measurable characteristics of the instances generated by a computational feature extraction process applied to $\mathcal{P}$;
- the algorithm space $\mathcal{A}$ is the set (portfolio) of all considered algorithms for tackling the problem;
- the performance space $\mathcal{Y}$ represents the mapping of each algorithm to a set of performance metrics.

In addition, we need to find a mechanism for generating the mapping from feature space to algorithm space. The Algorithm Selection Problem can be formally stated as: For a given problem instance $x \in \mathcal{P}$, with features $f(x) \in \mathcal{F}$, find the selection mapping $S(f(x))$ into algorithm space $\mathcal{A}$, such that the selected algorithm $\alpha \in \mathcal{A}$ maximizes the performance mapping $y(\alpha, x) \in \mathcal{Y}$. The collection of data describing $\{\mathcal{P}, \mathcal{A}, \mathcal{Y}, \mathcal{F}\}$ is known as the *meta-data*.

This framework provides a convenient mechanism to describe the numerous contributions to performance prediction found in many disciplines, as reviewed in [6]. When applying this framework to optimisation, key questions revolve around what the features should be to ensure the meta-data contains meaningful relationships that can be learned. Features that correlate with difficulty may include generic (problem independent) metrics based on identifying challenges in the search space such as isolation, deception, multi-modality [10,11,12], and features such as the size of basins of attraction [13], as well as landscape metrics based on analysis of autocorrelation structures and number and distributions of local minima [14,15]. Obviously these features can only be measured after an extensive analysis of the landscape, and are not suitable as inputs to a performance prediction model. A requirement of the feature set for performance prediction purposes is to ensure that the set of features used to characterise problem instances can be calculated faster than running all of the algorithms in

the portfolio, otherwise there is no time saving. Problem specific features that have been shown to correlate with difficulty are reviewed in [16] for a variety of combinatorial optimisation problems. Additional features may come from the concept of landmarking [17], related to hyper-heuristics [18], whereby the performance of simple heuristics is used as a feature to characterise the relative difficulty of an instance.

Regardless of which features are selected for a given optimisation problem, we need to test the degree to which they help to discriminate between instances in the meta-data, prior to embarking on any learning. Various information theory approaches have been proposed [19,20] for identifying a subset of features that provide the most value for the resulting classification problem (learning the mapping S to classify the best algorithm). In this paper we are interested in assessing the sufficiency of the meta-data by decomposing this question into two parts: Do the selected features create distinct classes of problem instances in feature space? and Are these features useful predictors of algorithm peformance?

3 Measuring TSP Hardness

The Travelling Salesman Problem (TSP) involves finding the shortest path tour visiting each of N cities exactly once and returning to the starting city. The distance between city i and city j is notated as $D_{i,j}$ and we assume symmetry of the distance matrix $\mathbf{D}$. The difficulty of this problem has been well-studied for many years, with properties of the distance matrix $\mathbf{D}$ naturally governing developed metrics of difficulty. For example, if all of the inter-city distances were identical, the problem is extremely easy to solve and there are multiple equally minimal cost solutions. By the early 1990's, the AI community had started to explore the question of whether all NP-complete problems could be characterised as easy or hard depending on some critical parameter embedded within the problem. Cheeseman et al. [21] conjectured that this was the case, that phase transitions exist for all NP-complete problems including the TSP, and contain at least one critical control parameter around which the most difficult problem instances are clustered. The variance of the distance matrix $\mathbf{D}$ has been demonstrated to correlate with difficulty for both exact approaches [21] and heuristics such as ant colony optimisation [22]. Zhang and colleagues have examined the distribution of distances and shown that the number of distinct distance values affects algorithm performance [23], and that phase transitions exist controlled by the average fraction of distinct distances [24]. Some researchers have converted the TSP optimisation problem into a binary decision problem (can an algorithm find a solution with tour length less than l?), and have shown that phase transitions exist for the TSP decision problem [25,26]. If the N cities are distributed randomly within a square of area A, then the critical parameter controlling the phase transition from easy to hard problem is given by $(\frac{l}{\sqrt{NA}} - 0.78).N^{1/1.5}$ where the constants were determined based on experimental modelling of TSP instances ($N \leq 30$) with the branch and bound algorithm [25]. Gaertner and Clark [27] have proposed a property that takes the uniformity of the problem instance into account by considering the distance between any city and its closest

neighbouring cities. Normalized values are used to eliminate differences due to scaling. The standard deviation of the normalised nearest-neighbour distances (nNNd) is then taken as an indication of how uniform the problem is. These authors also propose a coefficient of variation metric for the nNNd which is the standard deviation divided by the mean, providing a relative magnitude of variation in relation to the mean [27]. There are numerous other studies that have attempted to characterise the difficulty of a TSP instance by more computationally expensive metrics, such as landscape metrics [28], and measures of the number of backbone variables [29], but these metrics cannot readily be calculated until all optimal solutions have been found, and are therefore not practical as features for performance prediction. The final contribution towards characterising the difficulty of TSP instances comes from those who have been seeking to generate hard instances, and then analyse their properties. Van Hemert [9] has used evolutionary algorithms to evolve TSP instances that are difficult for the Lin-Kernighan algorithm [30] and its variants to solve, revealing the importance of the cluster distribution of the cities, and the location and distribution of outliers. The ratio of the number of clusters to the number of cities was demonstrated experimentally ($N \leq 200$) to create an easy-hard-easy phase transition, with instance difficulty maximized when the ratio is in the range [0.1,0.11] [31].

This paper extends the earlier work of Van Hemert [32,31,9] on the TSP, and considers not just hard instances, but also easy instances, for two variations of the Lin-Kernighan algorithm. The meta-data is augmented to include other features suggested as significant by previous studies, and a methodology is proposed and tested to tackle the question of whether the meta-data is sufficient for analysis and performance prediction.

4 Evolving TSP Instances

In this section we describe the part of the methodology related to generating the set of problem instances, and demonstrate the effectiveness of this approach in producing instances that are discriminating of algorithm performance.

Methodology

The method to evolve problem instances is the same as in [33], where an evolutionary algorithm was used to evolve difficult to solve travelling salesman problem instances. Here, we use the same method to evolve instances that are easy for each algorithm. A TSP instance is represented by a list of $N = 100$ (x, y) city coordinates on a 400×400 grid. The list directly forms the chromosome representation with which the evolutionary algorithm works. For each of the 30 initial TSP instances, we create a list of 100 cities, by uniform randomly selecting (x, y) coordinates on the grid. This forms the first step in the process, depicted in Figure 1. Then the process enters the evolutionary loop: Each TSP instance is awarded a fitness equal to the search effort (defined in the next section) required by an optimisation algorithm to find a near-optimal shortest tour. Using two-tournament selection, we repeatedly select two parents, which create

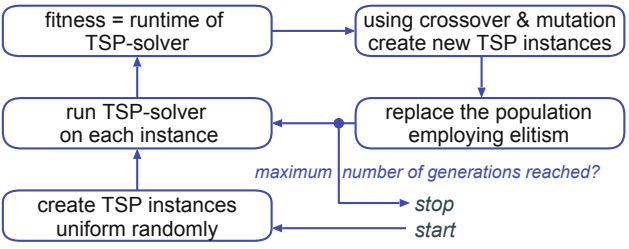

Fig. 1. The process of evolving TSP instances

one offspring using uniform crossover. Every offspring is subjected to mutation, which consists of replacing each one of its nodes with a probability pm, with uniform randomly chosen (x, y) coordinates. This generational model is repeated 29 times, and together with the best individual from the current population (1-elitism), a new population is formed. The loop is repeated for 600 generations. The mutation rate pm and related parameters are based on the settings in [34].

Algorithms

Our chosen algorithms are both based on the Lin-Kernighan heuristic method [35]. Developed more than thirty years ago, it is still known for its success in efficiently finding near-optimal results. The core of Lin-Kernighan, and its descending variants, consists of edge exchanges in a tour. It is precisely this procedure that consumes more than 98% of the algorithm's run-time. Therefore, in order to measure the *search effort* of Lin-Kernighan-based algorithms we count the number of times an edge exchange occurs during a run. Thus, this measure of the time complexity is independent of the hardware, compiler and programming language used.

Chained Lin-Kernighan (CLK) is a variant [36] that aims to introduce more robustness in the resulting tour by chaining multiple runs of the Lin-Kernighan algorithm. 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 [37] for clustered instances. It aims to reduce the computational effort, while maintaining the quality of solutions produced for both clustered and non-clustered instances. Cluster compensation works by calculating the cluster distance for nodes, which is a quick pre-processing step. The cluster distance between node v and w equals the minimum bottleneck cost of any path between v and w, where the bottleneck cost of a path is defined as the heaviest edge on that path. These values are then used in the guiding utility function of Lin-Kernighan to prune unfruitful regions, i.e., those involved with high bottlenecks, from the search space.

Experimental Settings

In our experiments, five sets of problem instances were generated, each containing 190 instances. The first set is RANDOM where each instance is created by scattering uniform randomly 100 cities in the 400×400 space. We evolve two sets of instances where for each set the evolutionary algorithm's objective is to maximise the search effort required by one of the two Lin Kernighan variants. This gives us the sets Hard_CLK and Hard_LKCC. Similarly, we generate Easy_CLK and Easy_LKCC by minimising the search effort required.

5 Analysis of the TSP Meta-data

Description of the Meta-data

The meta-data described by $\{\mathcal{P}, \mathcal{A}, \mathcal{Y}, \mathcal{F}\}$ can now be compiled as follows:

- a set of instances $\mathcal{P}$ comprised of five classes (Easy_CLK, Easy_LKCC, Hard_CLK, Hard_LKCC, and RANDOM), each with 190 TSP instances evolved to be either hard or easy for the two algorithms (with the random instances based on random selection of 190 of the initial randomly generated TSP instances used to seed the evolutionary algorithms),
- a set of algorithms $\mathcal{A}$ comprised of the CKL and LKCC algorithms,
- a performance metric $\mathcal{Y}$ being the log of the mean search effort of the algorithm on the best instance from each of the 99 trials,
- a set of 12 features $\mathcal{F}$ for a TSP instance consisting of: the standard deviation of the distances in $\mathbf{D}$; the (x, y) coordinates of the instance centroid; the radius of the TSP instance (defined as the mean distance from each city to the centroid); the fraction of distinct distances in $\mathbf{D}$ (precision to 2 decimal places); the rectangular area within which the cities lie; the variance of the normalized nearest neighbour distances (nNNd's); the coefficient of variation of the nNNd's; the cluster ratio (the ratio of the number of clusters to the number of cities, with clusters generated using the GDBSCAN algorithm [38]); the outlier ratio (ratio of number of outliers to cities); the ratio of cities near the edge of the plane (from GDBSCAN); the mean radius of the clusters (defined as the mean distance from any city in a cluster to the cluster centroid, averaged over all clusters). It should be noted that all TSP instances generated for the meta-data have the same number of cities ($N = 100$) so we do not need to use size as a feature.

Sufficiency of the Meta-data

A number of questions need to be answered before we can feel confident that the meta-data is sufficient for learning:

1. Do we have distinct classes of instances based on algorithm performance? In other words, are the instances relevant for distinguishing between algorithm performance?

2. Do we have distinct classes of instances based on features? In other words, are the features relevant for distinguishing between problem instances?
3. Can the features be used to predict algorithm performance?

The answer to all three of these questions is yes as will be shown in this following analysis, mostly due to the methodology of evolving instances that intentionally distinguish between algorithm performance, and selection of a set of features that have been shown to correlate with difficulty. The sufficiency of meta-data based on random instance generation methods cannot be so readily established, reinforcing the benefits of our chosen methodology.

First we answer the question of whether we have distinct classes of problem instances based on algorithm performance. Figs. 2 and 3 show the distribution of our chosen performance metric (log of search effort) for the CLK and LKCC algorithms respectively, grouped by the five classes of TSP instances. It is clear that the evolutionary algorithm has been successful in generating TSP instances that match the requirement. For example, the instances that were evolved to be hard for CLK (labelled Hard_CLK) show a considerably higher search effort required for the CLK algorithm compared to all other instances (see Fig. 2), including those evolved to be hard for the LKCC algorithm, and random instances. Likewise the instances evolved to be easy for each algorithm, require considerably shorter search effort compared to all other instances. All of these differences in search effort have been tested to be statistically significant (with p values $<<0.001$). It should be noted that the randomly generated instances, compared to the spectrum of difficulty, are quite limited in the pressure they place on each algorithm, and are not sufficient on their own to generate useful meta-data. Thus we conclude that the instances in the meta-data generated by the evolutionary approach are useful for distinguishing between algorithm performance.

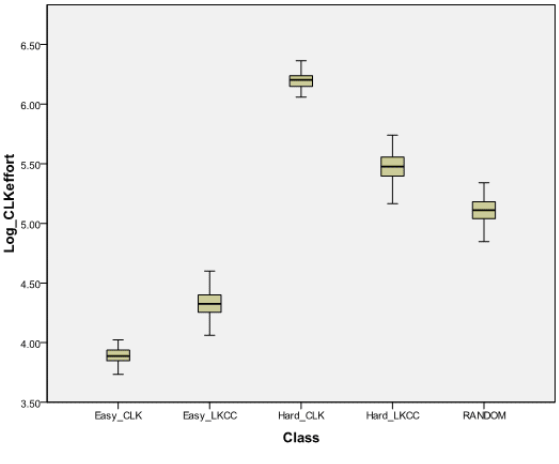

Fig. 2. Boxplot showing distribution of CLK search effort for instance classes

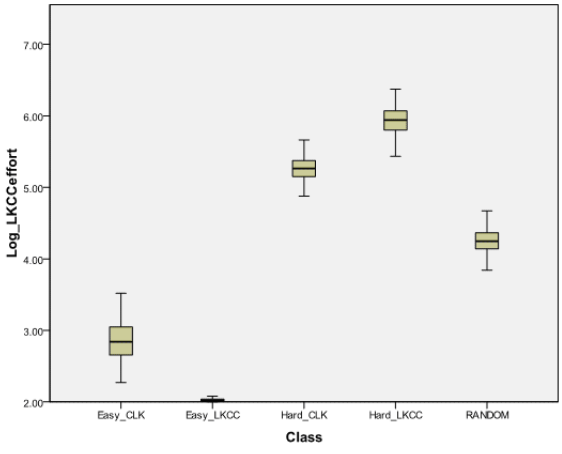

Fig. 3. Boxplot showing distribution of LKCC search effort for instance classes

Next we answer the question of whether we have distinct class of instances based on our chosen feature set. In other words, are the features relevant for distinguishing between problem instances? To answer this question, we map the high dimensional feature space to a 2-dimensional plane where the relationships between the features and the instances can be more readily visualized. A self-organizing map (SOM) [39] is used to achieve this dimensional reduction and visualization, using the Viscovery SOMine tool [40]. Based only on the 12 features, the SOM finds similarities and differences between the 950 instances, and determines that four distinct clusters of instances exist. The quantization error is sufficiently small (0.03) to show that each instance has been well assigned to one of the clusters in a manner that maximizes within cluster similarity and minimizes between cluster similarity. The fact that distinct clusters can be formed based only on the features is reassuring that the chosen feature set is useful for distinguishing between problem instances, but we perform further analysis on these clusters. Fig. 4 shows the four clusters in the two-dimensional map, as well as the distribution of search effort for each algorithm across the clusters. Recall that no algorithm performance information was used to cluster the instances, only features, and yet it is clear that Cluster 3 contains mostly instances that are the hardest for the CLK algorithm, Cluster 2 mostly contains instances that hardest for the LKCC algorithm, Cluster 4 contains mostly instances that are easy for both algorithms, and Cluster 1 contains a collection of instances that have mixed (and mostly average) performance by the algorithms. This is compelling evidence that the chosen features are not only capable of distinguishing between problem instances, but are also likely to be predictive of algorithm performance based on their location in the feature space. The distribution of the 5 problem classes within these 4 clusters is shown in Table 1, and shows strong "purity" of all clusters, with the exception of Cluster 1 which contains a mixture of the random instances and the EASY_LKCC instances.

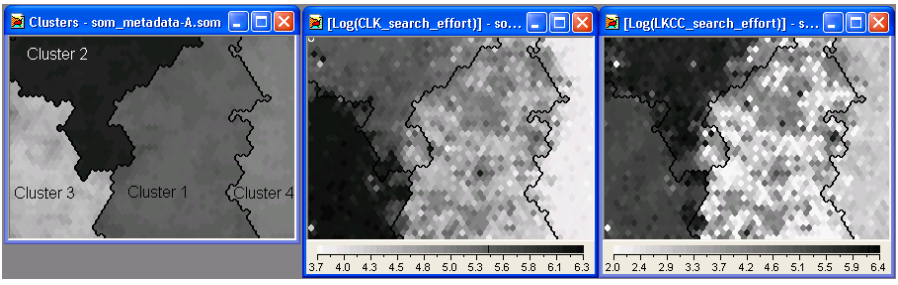

Fig. 4. Clusters (left), and distribution of search effort over clusters for the CLK algorithm (middle) and the LKCC algorithm (right). The grey scale indicates the range from minimal values (white) to maximal values (black) of the distribution.

Table 1. Distribution of each problem class over clusters

Cluster (N)	Easy_CLK	Easy_LKCC	Hard_CLK	Hard_LKCC	Random
1 (N=347)	7.89%	88.42%	0.53%	6.32%	79.47%
2 (N=209)	0.00%	5.79%	6.84%	85.26%	12.11%
3 (N=196)	0.00%	0.53%	92.63%	8.42%	1.58%
4 (N=198)	92.11%	5.26%	0.00%	0.00%	6.84%
	100.00%	100.00%	100.00%	100.00%	100.00%

Learning from Meta-data

Now that we have confirmed the sufficiency of the meta-data, we can conduct further analysis in the form of learning models to predict performance, as well as gaining insights into which features of an instance make it easy or hard for certain algorithms to solve. We can now analyze the meta-data with a view to understanding which features make particular classes of instances hard or easy for each algorithm. Fig. 5 shows the distribution of 9 of the features across the map. Several of the other features, such as the rectangular area, were distributed uniformly across the map, and therefore add no value to the analysis. From these distributions we can infer some relationships. Instances tend to be easy for both algorithms, but especially for CLK, in Cluster 4 where the mean radius of the TSP clusters is large, with a high ratio of cities near the edge of the plane, a low number of clusters and outliers. In other words, the instances are easy if the cities are spread quite uniformly across the plane. Further, if only a small fraction of the inter-city distances are distinct (measured to 2 decimal places of precision) then the instances are also easy. The instances become more challenging for both algorithms as the coefficient of variation of the nNNd's increases, and the instances become more clustered with the presence of outliers. The instances that are most challenging for CLK are those in Cluster 3 with a higher fraction of distinct distances, and cities that form a larger number of tight (small radius) clusters. On the other hand, the LKCC algorithm, handles such instances better, but has a longer search effort to solve instances that have less clustering structure

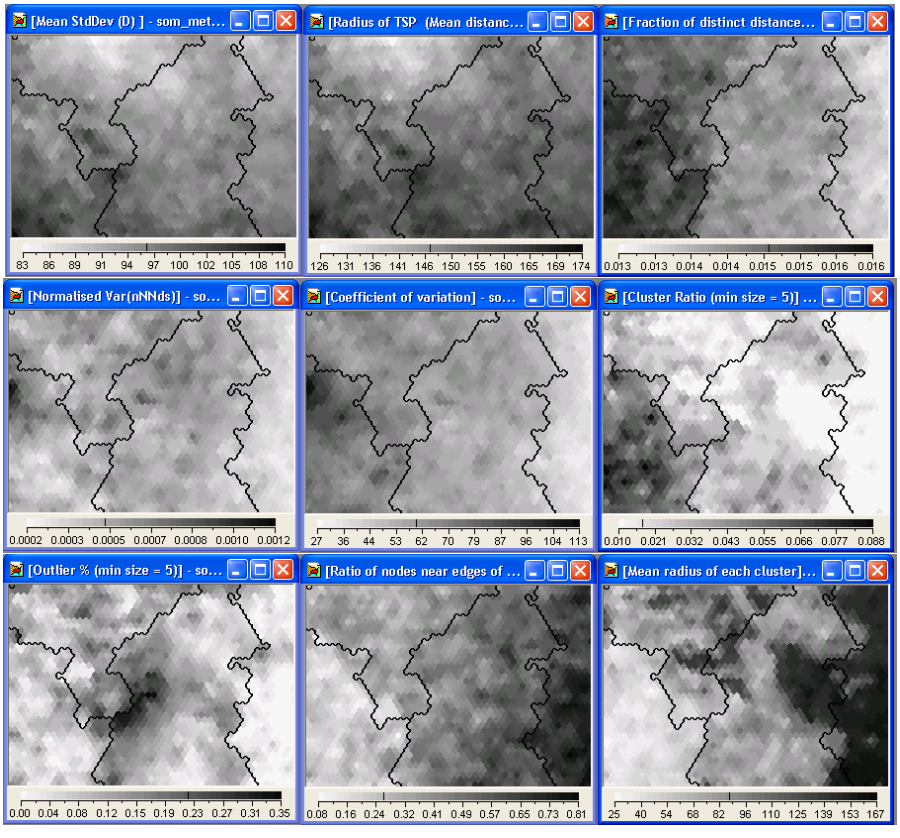

Fig. 5. Distribution of various features of the TSP instances over the four clusters

(as would be expected for an algorithm designed to exploit cluster structure). Certainly, many of the features that have been selected for the meta-data show strong relationships to algorithm performance.

A decision tree (using default implmentations of the C4.5 algorithm) has also been used to determine rules describing the 4 classes of instances evolved based on the features. These rules (shown in Table 2), evaluated of a randomly extracted test set comprising 20% of the instances, produce correct classification of an instance's class in 91.5% of cases.

These results indicate some clear phase transition boundaries around a cluster ratio of 0.05, depending on the nature of the clusters (their size and the presence of outliers), with the fraction of distinct distances around 1.4% also serving as a phase transition parameter. It should be noted that the cluster ratio phase transition observed in these experiments is about half the value of the one observed in [31] and is due to differences in the parameter settings controlling the formation of clusters (our most clustered TSP instance had only 10 clusters, producing a cluster ratio of 0.1 as a maximum of the distribution). These differences in

Table 2. Rules generated by Decision Tree showing relationship between features and classes

Rule 1	IF coeff. variation < 45.34 THEN Easy_CLK
Rule 2	IF coeff. variation >= 45.34 AND fraction distinct < 0.014 THEN Easy_LKCC
Rule 3	IF cluster ratio >= 0.05 THEN Class = Hard_CLK
Rule 4	IF fraction distinct >= 0.014 AND mean cluster radius >= 63.91 THEN Hard_LKCC
Rule 5	IF cluster ratio < 0.05 AND outlier ratio < 0.12 THEN Hard_CLK
Rule 6	IF cluster ratio < 0.05 AND outlier ratio >= 0.12 THEN Hard_LKCC

implementation aside, it is noteworthy that the evolved instances here confirm the features that have been reported to play critical roles in phase transitions in other studies.

Performance Prediction Results

We now use a supervised learning algorithm to learn to predict the log of search effort for each algorithm, as well as classifying which of the two algorithms is likely to find the optimal solution with less search effort, based on the features. A multilayered feedforward neural network, trained with the backpropagation learning algorithm, is used in both cases. For the prediction of search effort and classification of winning algorithm, the same architecture is employed comprising 12 inputs (features), 35 hidden neurons, and 2 output neurons. A test set of size 20% was randomly extracted for validating the models, built on the remaining 80% of instances. The neural network was implemented in Neuroshell®, with a learning rate and momentum factor both set to 0.1. Training is terminated when the average error on the test set falls below 0.015 (to avoid over-fitting). The accuracy of the neural network model for predicting the log of search effort, measured on the test set, is $R^2 = 0.89$ for the CLK algorithm, and $R^2 = 0.73$ for the LKCC algorithm. The classification problem of predicting which of the two algorithms is likely to require less search effort is handled using a winner-take-all interpretation of the 2 output neurons to identify the winning algorithm as the one with the higher probability of being best. The classification accuracy of this model is 96.8% on the test set. There are only 6 instances that are misclassified by the model, all of which fall in cluster 1 on the self-organizing map, and most of which are random instances.

6 Conclusions

This paper has contributed a methodological advance to the study of problem difficulty and learning from optimisation algorithm performance. While the algorithm selection framework of Rice [8] provides a useful reference point for gathering relevant information to support this learning, stored as meta-data, it provides little guidance about how the instances and features should be selected to ensure that the resulting meta-data is sufficient for subsequent learning

of relationships between features of instances and algorithm performance. This paper has discussed criteria for assessing the suitability of the meta-data, and methodological approaches that can be taken for instance generation to ensure these criteria are met.

The suitability of the meta-data depends on the degree to which the instances are able to discriminate between the performance of algorithms in the portfolio, and the degree to which the feature set is able to discriminate between instances. Instead of randomly generating instances and hoping that they are discriminating of algorithm performance, we have proposed to utilise an evolutionary approach to instance generation, intentionally evolving random instances into very hard or very easy instances for particular algorithms by measuring the fitness of instances according to search effort. This approach has been demonstrated on the Travelling Salesman Problem, and two variants of the Lin-Kernighan heuristic, to great effect. The criteria for the meta-data to contain instances that discriminate between algorithms is easily met, and the results show that the randomly generated instances are not very discriminatory, as has previously been reported for other problems [4]. The choice of features for characterising instances of the TSP has been informed by much literature over many decades, and a set of 12 features has been used in this study. By clustering the instances in feature space we have demonstrated the existence of four clear clusters (distinct groups of instances), and have met the requirement that the features discriminate between instances.

Having established the sufficiency of the meta-data, we have then sought to gain insights into the relationships between features and algorithm performance by examining the distribution of features and algorithm performances over the clusters and classes of instances. A decision tree has also been used to generate rules describing the critical values of features that affect algorithm performance. Most of these results have supported the extensive studies that have been previously conducted into the existence of phase transitions for the TSP. Finally, we have used the features to predict algorithm performance on unseen instances, achieving strong correlations with predicting search effort, and high accuracy when predicting the best performing algorithm.

The methodology adopted here can be readily applied to other optimisation problems, with careful consideration of the feature set, and a review of suitable features for a variety of combinatorial optimisation problems has been presented in [16]. Extending the algorithm portfolio to include a wider range of methods, both heuristic and exact, is also a future direction worth pursuing.

References

1. Macready, W., Wolpert, D.: What makes an optimization problem hard. Complexity 5, 40–46 (1996)
2. Nudelman, E., Leyton-Brown, K., Hoos, H., Devkar, A., Shoham, Y.: Understanding random SAT: Beyond the clauses-to-variables ratio. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 438–452. Springer, Heidelberg (2004)

3. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla-07: The design and analysis of an algorithm portfolio for SAT. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 712–727. Springer, Heidelberg (2007)
4. Cho, Y., Moore, J., Hill, R., Reilly, C.: Exploiting empirical knowledge for bi-dimensional knapsack problem heuristics. International Journal of Industrial and Systems Engineering 3(5), 530–548 (2008)
5. Hall, N., Posner, M.: Performance Prediction and Preselection for Optimization and Heuristic Solution Procedures. Operations Research 55(4), 703 (2007)
6. Smith-Miles, K.: Towards insightful algorithm selection for optimisation using meta-learning concepts. In: IEEE International Joint Conference on Neural Networks, IJCNN 2008. IEEE World Congress on Computational Intelligence, pp. 4118–4124 (2008)
7. Smith-Miles, K., James, R., Giffin, J., Tu, Y.: Understanding the Relationship between Scheduling Problem Structure and Heuristic Performance using Knowledge Discovery, LNCS. Springer, Heidelberg (in press, 2009)
8. Rice, J.: The Algorithm Selection Problem. Advances in computers 65 (1976)
9. van Hemert, J.: Evolving combinatorial problem instances that are difficult to solve. Evolutionary Computation 14(4), 433–462 (2006)
10. Gras, R.: How efficient are genetic algorithms to solve high epistasis deceptive problems? In: IEEE Congress on Evolutionary Computation, CEC 2008. IEEE World Congress on Computational Intelligence, pp. 242–249 (2008)
11. Locatelli, M., Wood, G.: Objective Function Features Providing Barriers to Rapid Global Optimization. Journal of Global Optimization 31(4), 549–565 (2005)
12. Xin, B., Chen, J., Pan, F.: Problem difficulty analysis for particle swarm optimization: deception and modality. In: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, pp. 623–630 (2009)
13. Bachelet, V.: Métaheuristiques parallèles hybrides: application au problème d'affectation quadratique. PhD thesis, Universite des Sciences et Technologies de Lille (1999)
14. Reeves, C.: Landscapes, operators and heuristic search. Annals of Operations Research 86, 473–490 (1999)
15. Schiavinotto, T., Stützle, T.: A review of metrics on permutations for search landscape analysis. Comput. Oper. Res. 34(10), 3143–3153 (2007)
16. Smith-Miles, K.A., Lopes, L.B.: Measuring Combinatorial Optimization Problem Difficulty for Algorithm Selection. Annals of Mathematics and Artificial Intelligence (under review, 2009)
17. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. In: Proceedings of the Seventeenth International Conference on Machine Learning table of contents, pp. 743–750. Morgan Kaufmann Publishers Inc., San Francisco (2000)
18. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. International Series in Operations Research and Management Science, pp. 457–474 (2003)
19. Battiti, R.: Using mutual information for selecting features in supervised neural net learning. IEEE Transactions on neural networks 5(4), 537–550 (1994)
20. Vasconcelos, N.: Feature selection by maximum marginal diversity: optimality and implications for visual recognition. In: Proceedings of 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1 (2003)
21. Cheeseman, P., Kanefsky, B., Taylor, W.: Where the really hard problems are. In: Proceedings of the 12th IJCAI, pp. 331–337 (1991)

22. Ridge, E., Kudenko, D.: An Analysis of Problem Difficulty for a Class of Optimisation Heuristics. In: Cotta, C., van Hemert, J. (eds.) EvoCOP 2007. LNCS, vol. 4446, pp. 198–209. Springer, Heidelberg (2007)
23. Zhang, W., Korf, R.: A study of complexity transitions on the asymmetric traveling salesman problem. Artificial Intelligence 81(1-2), 223–239 (1996)
24. Zhang, W.: Phase transitions and backbones of the asymmetric traveling salesman problem. Journal of Artificial Intelligence Research 21, 471–497 (2004)
25. Gent, I., Walsh, T.: The TSP phase transition. Artificial Intelligence 88(1-2), 349–358 (1996)
26. Thiebaux, S., Slaney, J., Kilby, P.: Estimating the hardness of optimisation. In: ECAI, pp. 123–130 (2000)
27. Gaertner, D., Clark, K.: On optimal parameters for ant colony optimization algorithms. In: Proceedings of the 2005 International Conference on Artificial Intelligence, Citeseer, vol. 1, pp. 83–89 (2005)
28. Stadler, P., Schnabl, W.: The landscape of the traveling salesman problem. Phys. Lett. A 161(4), 337–344 (1992)
29. Kilby, P., Slaney, J., Walsh, T.: The backbone of the travelling salesperson. In: International Joint Conference on Artificial Intelligence, vol. 19, p. 175 (2005)
30. Lin, S., Kernighan, B.: An efficient heuristic algorithm for the traveling salesman problem. Operations Research 21(2) (1973)
31. van Hemert, J.: Property analysis of symmetric travelling salesman problem instances acquired through evolution. In: Raidl, G.R., Gottlieb, J. (eds.) EvoCOP 2005. LNCS, vol. 3448, pp. 122–131. Springer, Heidelberg (2005)
32. van Hemert, J., Urquhart, N.: Phase transition properties of clustered travelling salesman problem instances generated with evolutionary computation. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 151–160. Springer, Heidelberg (2004)
33. van Hemert, J.: Property analysis of symmetric travelling salesman problem instances acquired through evolution. In: Raidl, G.R., Gottlieb, J. (eds.) EvoCOP 2005. LNCS, vol. 3448, pp. 122–131. Springer, Heidelberg (2005)
34. Kratica, J., Ljubić, I., Tošic, D.: A genetic algorithm for the index selection problem. In: Raidl, G.R., Cagnoni, S., Cardalda, J.J.R., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E., Meyer, J.-A., Middendorf, M. (eds.) EvoIASP 2003, EvoWorkshops 2003, EvoSTIM 2003, EvoROB/EvoRobot 2003, EvoCOP 2003, EvoBIO 2003, and EvoMUSART 2003. LNCS, vol. 2611, pp. 281–291. Springer, Heidelberg (2003)
35. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. Operations Research 21, 498–516 (1973)
36. Applegate, D., Cook, W., Rohe, A.: Chained lin-kernighan for large travelling salesman problems (2000), `http://www.citeseer.com/applegate99chained.html`
37. Johnson, D., McGeoch, L.: The traveling salesman problem: a case study. In: Aarts, E., Lenstra, J. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. John Wiley & Sons, Inc., Chichester (1997)
38. Sander, J., Ester, M., Kriegel, H., Xu, X.: Density-based clustering in spatial databases: The algorithm gdbscan and its applications. Data Mining and Knowledge Discovery 2(2), 169–194 (1998)
39. Kohonen, T.: Self-organization maps. Proc. IEEE 78, 1464–1480 (1990)
40. SOMine, V.: Enterprise Edition Version 3.0, Eudaptics Software Gmbh (1999)