Appears in: *Parallel Problem Solvers from Nature VIII, pages 150–159, 2004*

Phase transition properties of clustered travelling salesman problem instances generated with evolutionary computation

Jano I. van Hemert[1] and Neil B. Urquhart[2]

[1] National Institute for Mathematics and Computer Science, Amsterdam (CWI)
jvhemert@cwi.nl
[2] School of Computing, Napier University, Edinburgh
n.urquart@napier.ac.uk

Abstract. This paper introduces a generator that creates problem instances for the Euclidean symmetric travelling salesman problem. To fit real world problems, we look at maps consisting of clustered nodes. Uniform random sampling methods do not result in maps where the nodes are spread out to form identifiable clusters. To improve upon this, we propose an evolutionary algorithm that uses the layout of nodes on a map as its genotype. By optimising the spread until a set of constraints is satisfied, we are able to produce better clustered maps, in a more robust way. When varying the number of clusters in these maps and, when solving the Euclidean symmetric travelling salesman person using Chained Lin-Kernighan, we observe a phase transition in the form of an easy-hard-easy pattern.

1 Introduction

During the development of new algorithms for solving combinatorial optimisation problems, it is important to discover the limits within which each algorithm operates successfully. Accepted practice has often been to test new techniques on problem instances used in previous studies. Such problem instances include those published by Solomon [1] or those available via OR-LIB [2]. A disadvantage with such an approach is that the techniques developed may become somewhat fitted to the problem instances, in a similar way as classification techniques can be overfitted to data [3]. This phenomenon is well known in the field of neural networks, where is under study since the late eighties [4]. Contrary to benchmark suites, parameterized problem generators provide customisable problem instances, which may involve specific constraints to which each instance must adhere. Furthermore, the investigator may produce instances that differ only in a very specific manner, thus allowing the influence of a particular aspect of the problem to be investigated thereby focusing on the weak or strong points of a technique.

Solomon [1] originally published 56 instances of the vehicle routing problem in 1983, each containing 100 customers, these instances have been extensively used in the literature, in studies on both the vehicle routing problem [5, 6] and the travelling salesman problem TSP [7]. Backer et al. [8] have transformed these instances into a static set of problem instances that can be used for studies on dynamic vehicle routing. A various number of problem instances relating to various vehicle routing problems are available on-line from OR-LIB [2]. The upside of these problem instances is that benchmark results are easy accessible. However, the downside to that this has led to many publications that contribute new techniques which only provide an improvement over previous techniques on these instances without showing the weak and strong points in relation to the problem class itself. Braun and Buhmann [9] provide evidence of what the effect can be of performance studies on a fixed set of Euclidean TSP instances. In their study they attempt to improve their algorithm by learning the underlying common structural properties of problem instances and using it to guide the search. This led to an overfitting effect as soon as the algorithm learned from optimal solutions.

Beck et al. [10] use a simple problem generator to produce specialist vehicle routing problems. The locations for possible delivery point layout are determined from a file containing the (x, y) coordinates of postal codes within the City of Glasgow, distances between them being calculated as Manhattan distances. The remaining items, such as job length, were set using user defined parameters. This makes it a kind of hybrid approach where reality is mixed with artificial data. In this study we shall use a map generator to produce layouts of locations in an artificial way while keeping a link with real routing problem by focusing on a property often observed in real routing problems, that of clustered locations. This may help us to identify what structure in a map would make solving a routing problem more difficult. Instead of working with full routing problems, we simplify our analysis here by focusing on the Euclidean symmetric travelling salesman problem (TSP). The objective in the Euclidean symmetric TSP is to find the shortest Hamiltonian path through all the nodes. These nodes lie in a 2-dimensional space and the distance to travel between two of them is defined as the Euclidean distance. The TSPLIB [11] problem instances repository contains 112 problem instances of this type. It makes a nice starting point for testing new algorithms, as it also contains optimal and best known results. However, besides the aforementioned problem of overfitting, it also contains insufficient problem instances for current research purposes. The total number of instances in this study lies over 27 000.

The work of Cheeseman at al. [12] was the first of many studies on the phenomena of phase transitions in constraint satisfaction and constrained optimisation problems. A phase transition occurs in many NP-complete problems, when one or more order parameters are varied. The system that is created this way moves through an easy-hard-easy pattern. For example, in graph k-colouring, when one tries to solve different problem instances where the ratio of edges in the graph is varied, at a certain point the average effort required to solve problem

instances with that ratio of edges, will be the highest. For a while, the consensus was that no such phase transition occurs in the travelling salesman problem. This changed with the work of Gent and Walsh [13], wherein it was shown that, although no phase transition occurs in the optimisation problem, it does occur in the equivalent decision problem with either the tour length or the number of nodes as the order parameter. Here we will show that another order parameter exists, which reveals a phase transition in the optimisation problem of TSP. By varying the number of clusters in generated Euclidean symmetric TSP instances, an easy-hard-easy patter can be identified. Furthermore, a preliminary estimation is given of the location of the, on average, most difficult to solve problem instances.

The general outline of the paper is as follows. First, we shall introduce the concept of generating maps. Then in Section 3, we introduce an evolutionary algorithm that makes layouts of maps with clustered nodes. The tools used in the experiments are discussed in Section 4. Section 5.1 presents a robust set of parameters to be used in the objective function of the evolutionary algorithm. These parameter settings are used in further experiments. In Section 5.2 we compare maps created by two different methods. In Section 5.3 we show that, when the ratio of clusters is varied, a phase transition occurs in the difficulty of solving travelling salesman problems with Chained Lin-Kernighan. We draw conclusions in Section 6.

2 Generating 2-dimensional maps

The evolutionary algorithm discussed in the next section is part of a larger software package, which will facilitate in creating routing problems. It is incorporated in a map generator that can distribute a given number of nodes onto a 2-dimensional plane with a given size. This can be done using uniform sampling or using the evolutionary algorithm. When a number of clusters is supplied it will first create the centres of these clusters, and then, distribute the nodes equally over these clusters. For uniform sampling, the amount of space to which a cluster is confined is equal to a rectangular area defined by the maximum width and height, both divided by the number of clusters. The evolutionary algorithm restricts the size and spread of clusters by trying to satisfy two constraints; first, the maximum distance between nodes and second, the minimum distance between nodes.

Attempting to satisfy both constraints is in many cases impossible, for instance as the minimum distance is increased, it becomes more difficult to fit larger numbers of nodes into smaller maps. As the minimum and maximum distance constraints move closer together so the number of feasible solutions becomes less. A similar relationship exists between the numbers of locations that may be placed within a given area with respect to the maximum distance constraint.

3 Evolutionary algorithm for generating maps

The representation of each individual takes the form of a list of points each representing one node point on the map. The evolutionary algorithm employs a steady-state evolutionary model with 2-tournament selection [14, Part C2.3] to determine the pool of parents, and to determine the replacement. A population size of 20 is used, where 10 new individuals are created at each generation. Two-point crossover is used with a recombination rate of 0.5 [14, Part C3.3.1.2]. The offspring not created by the recombination of two parents are directly cloned from one parent. Mutation is applied to all offspring. A mutation is defined as the translation of a node by a small, randomly chosen, amount j. Initially, all nodes are set to the origin which is located at the centre of the map. At the start, the translations are performed using a random amount in the range $(0, \ldots, j \times 2)$. If, after a translation, a node will be of the map or on top of another node, other translations are tried until this is resolved. The value j is initially set to the maximum inner cluster distance. After 100 generations have passed without an improvement, j is reduced by 10%. This is repeated over the run of the algorithm. Thus the amount of adjustment made to the positions of nodes decreases as the algorithm progresses. This cooling down is a familiar process in evolutionary computation [14, Part E1.2.3]. When either $j = 0$, a solution is found, or 5 000 generations are past the evolutionary algorithm is terminated.

To compute the fitness of an individual we first calculate the Euclidean distance between all of nodes, during which the smallest distance (min) and the largest distance (max) is noted. The fitness is then computed as follows:

$$
\begin{aligned}
&\mathit{fitness} = 0 \\
&\texttt{if } \min < \mathit{minimum\text{-}distance} \texttt{ then} \\
&\qquad \mathit{fitness} = \mathit{fitness} + \mathit{minimum\text{-}distance} - \min \\
&\texttt{if } \max > \mathit{maximum\text{-}distance} \texttt{ then} \\
&\qquad \mathit{fitness} = \mathit{fitness} + \max - \mathit{maximum\text{-}distance}
\end{aligned}
$$

This fitness needs to be minimised, where $\mathit{fitness} = 0$ means a solution is found and the algorithm will terminate.

4 Tools

Here we discuss two tools, which will be used in the experiments. First, the clustering algorithm GDBSCAN and second, the Chained Lin-Kernighan algorithm for solving travelling salesman problems.

4.1 Density-Based Clustering Method Based on Connected Regions with Sufficiently High Density

To measure how well maps are clustered we use the clustering algorithm GDBSCAN [15]. This algorithm uses no stochastic process, assumes no shape of clusters, and works without a predefined number of clusters. This makes it an ideal candidate

to cluster 2-dimensional spatial data, as the methods suffers the least amount of bias possible. It works by using an arbitrary neighbourhood function, which in this case is the minimum Euclidean distance. It determines clusters based on their density by first seeding clusters and then iteratively collecting objects that adhere to the neighbourhood function. The neighbourhood function here is a spatial index, which results in a run-time complexity of $O(n \log n)$. This method relies on the way nodes are spread solely in terms of distance, which is our reason for using it to test whether the maps generated contain the number of clusters we have requested. The settings for the minimum Euclidean distance will be set in the experimental setups.

4.2 Chained Lin-Kernighan

To determine the difficulty of problem instances, expressed in how much time is required to get a good solution, we employ the well known Lin-Kernighan [16] algorithm. Here we use Chained Lin-Kernighan with the implementation from the Concorde system [17]. This latter algorithm performs multiple runs of Lin-Kernighan, where for each next run the initial tour is the result of the previous run after a slight perturbation. The number of runs of Lin-Kernighan depends on the size of the input, it equals to the number of cities. The authors have shown that it is an efficient method, with a high probability of achieving optimal routes, even on very large problem instances.

The stochastic nature of Chained Lin-Kernighan requires multiple runs. For each map in a test, we let it perform 50 independent runs. During a run we count the number of elementary steps performed. As opposed to measuring processor time usage, this measure is independent of hardware, software, and programming specifics. The elementary step of Lin-Kernighan, which makes up for more than 99% of the execution time, consists of flipping a segment in the tour.

5 Experiments

In all the experiments we fix the size of a map to 500×500. In this section we shall first determine a robust parameter setting for the constraints in the fitness function of the evolutionary algorithm. Then, we will show the improvement of the EA over uniform sampling. Finally, we shall provide evidence for the occurrence of a phase transition with varying amounts of clusters.

5.1 Finding robust parameter settings

The number of nodes is set to 100 and the number of clusters to 10. Because of the maximum size of the map, the maximum distance constraint of the evolutionary algorithm is set to 500 to make sure the whole area will be used. This leaves us with finding a good value for the minimum distance constraint. As the setting of this parameter determines the spread of the nodes within clusters, it is mostly responsible for the quality of the map. To determine a robust value we try

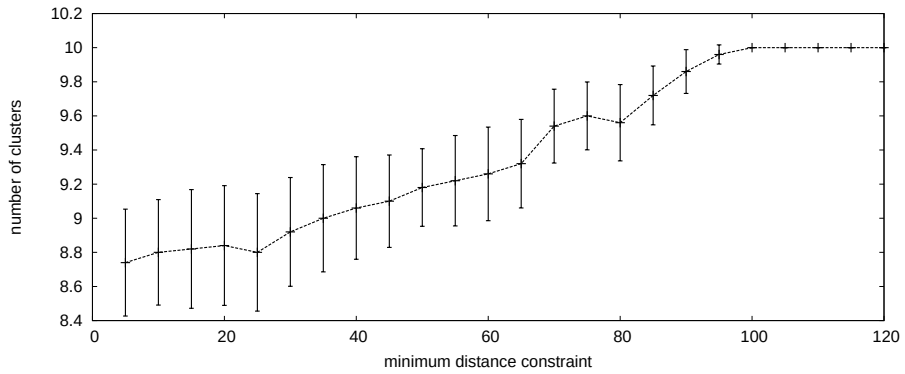


Fig. 1. The effect of the minimum distance constraint on the forming of clusters measured by the average number of clusters found by the GDBSCAN clustering algorithm. Confidence intervals of 95% are shown where for each setting of the minimum distance constraint 50 maps were generated.

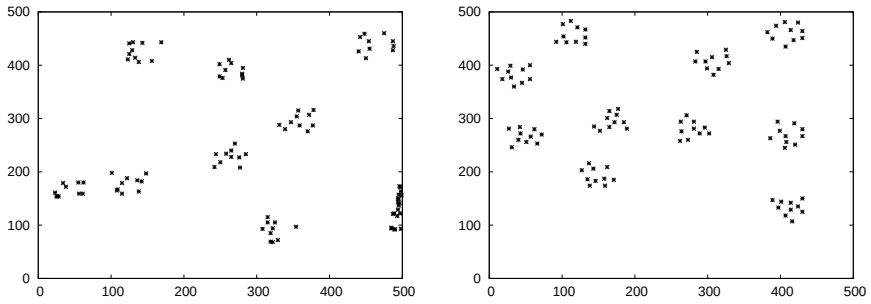
different settings and assess the quality of each setting using GDBSCAN. We vary the minimum distance from 5 to 120, with steps of 5. For each setting we generate 50 maps, and run GDBSCAN with a minimum Euclidean distance of 50. Figure 1 shows the average number of clusters found by GDBSCAN in these maps together with 95% confidence intervals.

At a setting of 100 the confidence intervals only maps with 10 clusters are generated. The almost monotone-increasing plot, together with the decrease of the confidence intervals, make this a save setting for further experiments. Higher values will result in sparser clusters, and eventually, overlapping clusters, which is not what we are interested in.

5.2 Comparison with uniform randomly generated

We compare the maps generated with the evolutionary algorithm to maps created uniform randomly as explained in Section 2. First, in Figure 2 we give a visual presentation of a map created with both the evolutionary algorithm and uniform random method. These maps contain 100 nodes and 10 clusters. Generally speaking, the clusters of the map created with the evolutionary algorithm show less deformations than the ones in a map generated uniform randomly. Also, the map created uniform randomly suffers from the size restriction on the map, which is apparent from the cluster at $(500, 125)$. This cluster is squashed against the edge of the map.

Although visual presentations help identify the problems in maps, they do not provide scientific evidence that the evolutionary algorithm creates better maps in a more robust way. To show how well the maps are laid out we use again the GDBSCAN clustering algorithm. Every clustering algorithm needs a definition



(a) Uniform randomly (b) Evolutionary algorithm

Fig. 2. Examples of maps with ten clusters

of what makes a cluster, and such definitions always involve a parameter. In the case of GDBSCAN this parameter is the neighbourhood function, which is defined here as the minimum Euclidean distance. By varying this parameter we get an idea of the sensitivity of the clustering algorithm to the given clustering problems.

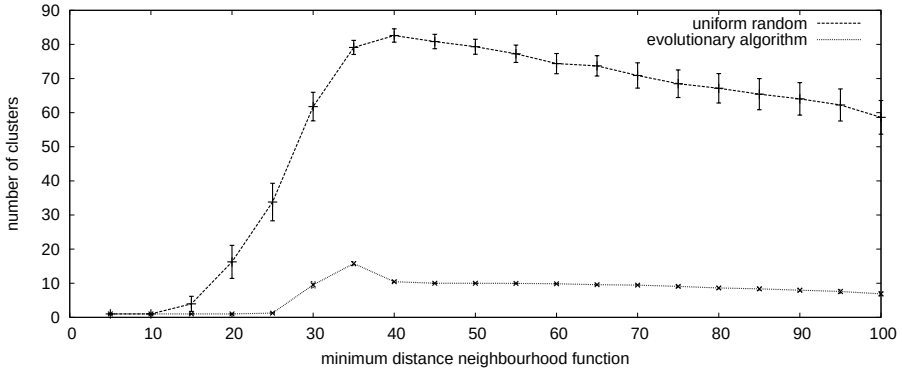


Fig. 3. Analysing the sensitivity of both the evolutionary algorithm and the uniform random method to the clustering parameter. For every setting we generated 50 maps. 95% confidence intervals included

In Figure 3, we show the sensitivity of the GDBSCAN algorithm to the minimum Euclidean distance parameter when clustering maps from both the evolutionary algorithm and the uniform random method. For very small distances,

GDBSCAN is not able to find a clustering. Where maps of the evolutionary algorithm are concerned, from 40 onwards GDBSCAN starts finding the correct number of clusters. It does this with a very low variance, visible from the small confidence intervals. For maps created uniform randomly, only at one setting, which is at approximately 18, the correct number of clusters may be identified. For larger settings, a much too high number of clusters is identified, which shows that the clusters in those maps are not spread in a satisfactory way.

5.3 Phase transition

We investigate the difficulty of solving the Euclidean symmetric travelling salesman problem, where the problem instances are based on clustered maps generated by the evolutionary algorithm with the robust parameter settings described in Section 5.1. In the experiments reported next, every map, i.e., TSP problem instance, is solved using Chained Lin-Kernighan in 50 independent runs. For each setting of the number of clusters, 50 problem instances are generated, thus making 2500 runs per setting. We perform the same test for 50, 100, 150, and 200 nodes. To measure the difficulty of solving problem instances we count the average amount of Lin-Kernighan steps needed to reach good solutions.

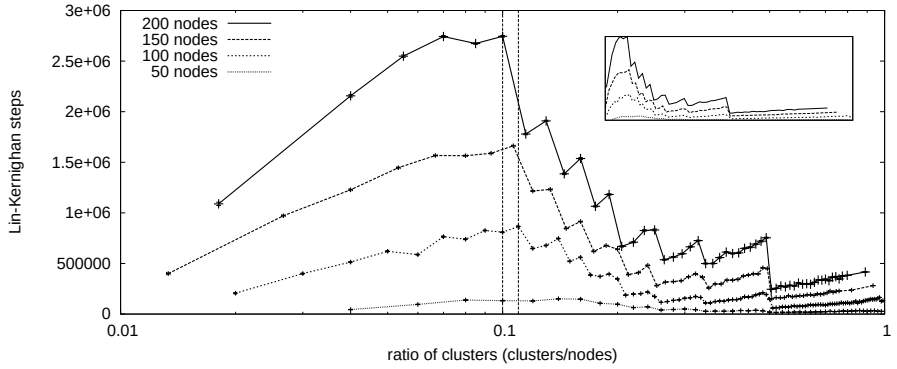


Fig. 4. The difficulty of solving Euclidean symmetric TSP for clustered maps of 50, 100, 150, and 200 nodes, generated with the evolutionary algorithm. At each setting of the number of clusters, 50 maps are generated. 95% confidence intervals included, which are very narrow

Figure 4 shows the four plots corresponding to the different settings of the number of nodes. As the number of nodes differ for each of the settings, we take as the x-axis the ratio of clusters, i.e., the number of clusters divided by the number of nodes. With increasing ratio of clusters the average difficulty increases to, up until a highest point, after which it decreases with a pattern of short increases and sharp drops. In the easy-hard-easy pattern that appears, we

clearly see that the problem instances that are on average the most difficult to solve are found when this ratio lies approximately in the interval $(0.10, 0.11)$.

The pattern of short increases and sharp drops after the peak is related to how nodes are distributed over the clusters. When the number of clusters increases, the difficulty increases until a point is reached when $0 = nodes \pmod{clusters}$. By adding one more cluster, the difficulty drops sharply. Thus, an evenly distributed set of loads seems to lead to more difficult to solve problem instances. Note that as the ratio of clusters approaches one, we are essentially creating random problem instances, where most clusters contain only one, and sometimes two, nodes. Such problem instances are much easier to solve than those found near the peak of the phase transition. On average, problem instances at the peak are approximately 5.9 times as difficult as randomly distributed ones, with a standard deviation of 0.6.

6 Conclusions

We have shown that the proposed evolutionary algorithm to layout nodes in a 2-dimensional space is able to create well clustered maps in a robust way. The quality of the spread of nodes in clusters is measured by using GDBSCAN, a clustering algorithm that makes no assumptions about the shape of clusters and the total number of clusters. The resulting maps are of a better quality than those generated using random uniform sampling.

The maps created using the evolutionary algorithm also represent Euclidean symmetric travelling salesman problem instances. Through the use of the Chained Lin-Kernighan algorithm we can show how hard it is to solve these instances, measured in time complexity. When varying the number of clusters a typical phase transition appears, where we get an easy-hard-easy pattern. For the values used in the experiments presented here, i.e., 50, 100, 150, and 200 nodes on a 500×500 map, the most difficult to solve instances occur when $clusters/nodes$ lies in $(0.10, 0.11)$. This suggests that the position of the phase transition depends only on the ratio of clusters. Moreover, it seems that the distribution of nodes over the clusters has a large impact on the difficulty of solving travelling salesman using the Lin-Kernighan algorithm, where more evenly distributed maps are more difficult to solve.

Future research will be on vehicle routing problems, where clustering is an essential part of the problem. The knowledge gained here will be used to create appropriate maps that will form the basis for non-trivially solved routing problems. Furthermore, research on the travelling salesman problem continues whereby focusing on the difference between using uniform and evolutionary methods as well as by observing larger problem instances.

Acknowledgements

This work is supported by DEAL (Distributed Engine for Advanced Logistics) as project EETK01141 under the Dutch EET programme.

References

1. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research **35** (1987) 254–264
2. Beasley, J.E.: OR-library: distributing test problems by electronic mail. Journal of the Operational Research Society **41** (1990) 1069–1072
3. Ng, A.Y.: Preventing "overfitting" of cross-validation data. In: Machine Learning: Proceedings of the Fourteenth International Conference. (1997) 245–253
4. Baum, E., Haussler, D.: What size net gives valid generalization? Neural Computation **1** (1989) 151–160
5. Golden, B., Assad, A.: Vehicle Routing: methods and studies. Elsevier Science Publishers B.V. (1988)
6. Toth, P., Vigo, D.: The Vehicle Routing Problem Discrete Math. Society for Industrial and Applied Mathematic (2001)
7. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem. John Wiley & Sons, Chichester (1985)
8. Backer, B., Furnon, V., Kilby, P., Prosser, P., Shaw, P.: Solving vehicle routing problems using constraint programming and metaheuristics. Journal of Heuristics **6** (2000)
9. Braun, M.L., Buhmann, J.M.: The noisy euclidean traveling salesman problem and learning. In Dietterich, T.G., Becker, S., Ghahramani, Z., eds.: Advances in Neural Information Processing Systems 14, Cambridge, MA, MIT Press (2002)
10. Beck, J.C., Prosser, P., Selensky, E.: Vehicle routing and job shop scheduling: What's the difference? In: To appear in Proc. of the 13th International Conference on Automated Planning and Scheduling (ICAPS'03). (2003)
11. Reinelt, G.: Tsplib, a library of sample instances for the tsp (and related problems) from various sources and of various types (2002) Available at http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.
12. Cheeseman, P., Kenefsky, B., Taylor, W.M.: Where the really hard problems are. In: Proceedings of the IJCAI'91. (1991) 331–337
13. Gent, I., Walsh, T.: The TSP phase transition. Artificial Intelligence **88** (1996) 349–358
14. Bäck, T., Fogel, D., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York (1997)
15. Sander, J., Ester, M., Kriegel, H.P., Xu, X.: Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. In Data Mining and Knowledge Discovery **2** (1998) 169–194
16. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the travelling salesman problem. Operations Research **21** (1973) 498–516
17. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: On the solution of traveling salesman problems. Documenta Mathematica Extra Volume Proceedings ICM III (1998) 645–656