

An Analysis of the Hardness of TSP Instances for Two High-performance Algorithms

Thomas Fischer* Thomas Stützle† Holger Hoos‡ Peter Merz*

*Department of Computer Science, University of Kaiserslautern
Postfach 3049, D-67653 Kaiserslautern, Germany
{fischer,pmerz}@informatik.uni-kl.de

†Computer Science Department, Darmstadt University of Technology
Hochschulstr. 10, D-64283 Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

‡Computer Science Department, University of British Columbia
2366 Main Mall, Vancouver, BC, V6T 1Z4, Canada
hoos@cs.ubc.ca

1 Introduction

The task in the Traveling Salesman Problem (TSP) is to find a minimum length closed tour through a given set of n cities with known inter-city distances such that each city is visited exactly once. The TSP has played a central role in the development of many Stochastic Local Search (SLS) algorithms, such as Ant Colony Optimization [5], or Simulated Annealing [9]. Currently, the best performing SLS algorithms for the TSP strongly rely on the exploitation of high-performing local search algorithms, and most top-performers are iterated local search (ILS) algorithms [1, 3, 8, 7] or memetic algorithms [4, 11]. While most ILS algorithms are based on the Lin-Kernighan algorithm (LK) [10], Glover [6] and Rego [13] present an approach based on an edge based ejection chain method, which, compared to LK, tends to find better tours while requiring more computation time. Small and medium size TSP instances ranging from a few hundred to several thousand cities are also solved rather efficiently by exact algorithms, the best example being the branch & cut code of Applegate, Bixby, Chvátal, and Cook [2]. However, a disadvantage of exact algorithms is that they show a very strong variability in computation time between different TSP instances. Performance variability among instances can also be observed for SLS algorithms, in addition to the variability in the performance on any given instance that is caused by the stochastic nature of these algorithms.

It is clear that the structure of TSP instances influences the amount of time needed to solve them to optimality. For the scope of this article, our investigation of structure focuses on the distribution of the cities in the Euclidean plane. In particular, the instances we study are derived from highly structured TSP instances that can be solved in polynomial time, by either removing cities or by perturbing city coordinates. We analyze the impact of these variations in the structure of TSP instances on the run-time of two high-performance algorithms: Applegate *et al.*'s branch & cut code (`concorde`) [2] and Helsgaun's iterated Lin-Kernighan algorithm `lkh` [7]. `concorde` is currently the best performing exact algorithm for the TSP using state-of-the-art B&C methods and is freely available for academic purposes. Helsgaun's `lkh` algorithm

Vienna, Austria, August 22–26, 2005

is currently one of the top performing SLS algorithms for the TSP [7, 8]; it is based on an innovative implementation of the well-known Lin-Kernighan heuristic [10] and uses candidate edge sets that are built by a so-called α -nearness measurement based on 1-trees and are updated upon success in the search algorithm.

2 TSP instances

Probably the best known set of benchmark instances is that of TSPLIB [14] which consists mainly of map instances and drilling problems with up to 85 900 cities. Almost all of these instances are metric instances, where the cities correspond to points in some metric space. The most important property of (metric) TSP instances besides their size is the distribution of cities in the given space. In particular, cities may be distributed in a clearly structured way, which in several cases even allows finding the optimum in polynomial time. Such instances are usually easily solvable by heuristics.

In this article, we study the change in algorithmic behavior when in response to transitioning from highly structured, polynomially solvable TSP instances to instances with increasingly random distributions of cities. We consider two different classes of structured instances: *Fractal instances* have been described and solved to optimality by Moscato and Norman using simple constructive heuristics [12]. Here, Koch, David, MPeano and MNPeano instances in several sizes were built by a Lindenmayer system. *Grid instances* have their cities arranged in a chess board pattern. Here again, optimal solutions can be easily constructed. To introduce randomness into these instances, two perturbation operations were applied. The central idea for the first approach is that by increasingly destroying the structure of the instances, the cost of finding optimal solutions can be expected to increase. For fractal instances, the *reduction* operation removes randomly selected cities from the instance. The strength of this operator is characterised by the fraction of cities removed. Different levels of preservation resulted in instances, where 99%, 98%, 95%, 90%, 85%, 75%, 50%, or 25% of the original instances' cities were kept (cf. Fig. 1). The *shake* operator moves cities from their original position by applying an offset for each coordinate that is determined by a normally distributed random variable with zero mean. The amount of perturbation is determined by the standard deviation of the normal distribution. On the grid instances, we used values of 5%, 10%, 25%, 50%, 75%, 100%, 150%, and 200% of the horizontal (or vertical) distance between two neighboring cities on the original instance's grid. The shake operator was only applied to grid instances, as the exact chess board like arrangement of cities is used for optimal tour construction in polynomial time. For fractal instances optimal tour construction depends on the existence of certain cities, hence removing is an adequate perturbation operator.

For each of 13 fractal and 3 grid instances, 100 instances were built for each of 8 perturbation levels resulting in a total of 12 816 instances. To compare the performance of the algorithms on these instances, for each occurring instance size (total of 111), we constructed 100 random Euclidean TSP instances. In these random Euclidean TSP instances the cities are uniformly distributed within a square.

3 Computational study

The TSP instances were solved on a dual processor machine (AMD Athlon 1.2 GHz, SuSE Linux 7.1) by Helsgaun's `1kh` and the `concorde` code using `cp1ex6` (B&C algorithm). Different analyses were performed on the algorithms' results. First, we studied the distribution of search costs over each set of 100 instances for `1kh` and `concorde` in order to analyze instance hardness. The resulting search cost distributions illustrate the variation of the computation time required

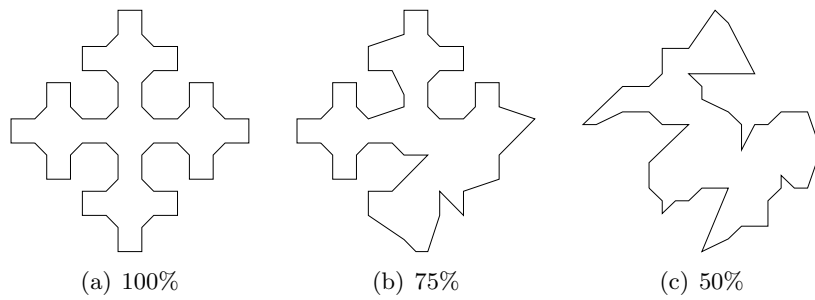


Figure 1: Shape of optimal tours for MNPeano instances with different preservation levels.

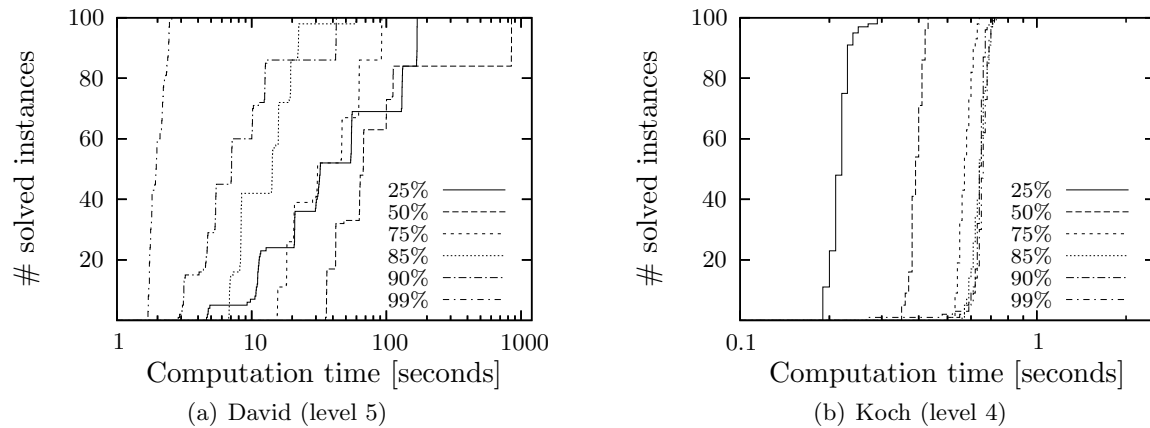


Figure 2: Search cost distribution (*concorde*) for David and Koch instances at different preservation levels.

for solving differently sized instances generated through the same process. Second, we plotted the size of these instances that were generated by using the reduction operator with various preservation rates versus their search cost; these plots illustrate how the perturbation influences the computation times. Furthermore, the number of 2-exchange steps required to reach a local optimum starting from a random tour (gradient analysis) were put into relation to the computation times of the instances.

3.1 Influence of structure on computation time

Computation times for instances based on fractal curves solved by *concorde* show two kinds of behaviors for different preservation levels. The solution times of David, MPEano and MNPeano instances grow steadily from the original or only marginally perturbed instances to lower preservation levels that keep only 50% to 75% of the original cities. However, removing even more cities leads then to reductions in computation times. This behavior can be observed in Fig. 2(a). On the other side, Koch instances show a linear relationship between instance size and computation time. Additionally, computation times within the set of Koch instances were very homogenous with a very small deviation (see Fig. 2(b)).

For grid instances subject to shaking roughly similar behavior to David instances was observed: Computation times increase with the degree of shaking reaching a maximum at a shaking level of about 25%; for higher shaking levels the computation times were found to decrease slightly. The shape of the search cost distribution was similar for all sets of grid instances. Within the group of 100 instances, most instances were solved within small computation times, while the last 5% to 10% required significantly more time. Instances with

high computation times required many branches in the branch & bound tree during their search. For example, two grid instances with 625 cities and a shaking level of 100% required in one case 1211 CPU seconds and 77 branches, in the other case only 9 seconds and one branch.

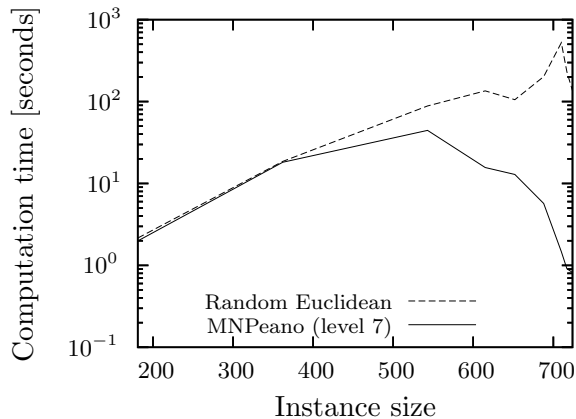


Figure 3: Median run-times (*concorde*) for MNPeano instances compared to random Euclidean instances of the same size.

Computation times for the instances mentioned above were compared with random Euclidean instances of the same size. Interestingly, the computation times grow exponentially for the random instances. In contrast, fractal and grid instance in their original form or with only slight variations through reduction or shaking had much lower solving times. Hence, minor modifications in strongly structured instances lead only to rather minor effects on the required computation times. Only for high degrees of reduction/shaking the computation times reach a similar level as those for random instances of the same size; hence, a strong change in the structure of the instances makes them more difficult to solve (see Fig. 3).

All instances were also solved using Helsgaun's *1kh*. As the *1kh* program was not able to solve instances to optimality in every run, time penalties $\varphi_r(s, t_{\max}) = (r/(r-s) - 1) \cdot t_{\max}$ were introduced. Here r denotes the total number of runs (always 100), s denotes the number of successful runs ($0 \leq s < 100$) and t_{\max} denotes the largest time for a single, successful run. David fractals required the same amount of computation time, regardless of instance size, if the instance could be solved by *1kh*. With decreased preservation levels the instances became harder for *1kh* and caused increased computation times due to the penalties caused by unsuccessful runs. Koch instances were much harder for *1kh* than for *concorde*, requiring 5 to 10 times higher computation times. However, this effect may be mainly due to the costly preprocessing steps applied by *1kh*. MPeano and MNPeano instances required less than 3 CPU seconds for instances with less than 1000 cities, although instances with preservation levels between 75% and 90% were not solved in every run. The search cost distribution for most fractal instance sets are similar, and about 70 out of 100 instances within a group were solved to optimality rapidly. The remaining instances had significantly smaller or larger computation times (due to unsuccessful runs). As is the case for the computation times measured by *concorde*, run-times for *1kh* are also higher for random Euclidean instances than for their fractal counterparts with the same size. Generally, the more the instances are perturbed, and, hence, the more structure is lost, the closer the computation times for perturbed fractal instances approach these random Euclidean instances. Computation times for shaken grid instances increase with the shaking level up to a level of 50%. Higher shaking levels do not result in further increases in difficulty, and *1kh* computation times stay below those required for solving random Euclidean instance of the same size.

When comparing the computation times of *1kh* and *concorde* (*concorde*'s times here only include the time to find the optimum), we can note that for about 18% of all instances *concorde* was faster than *1kh*. The instances that *concorde* solved faster than *1kh* were mainly Koch curves (78% faster with *concorde*), MNPeano and MPeano (each 43%). This time comparison includes time penalties for incomplete *1kh* runs.

3.2 Influence of instance properties on computation time

Local search steps required to reach an optimum in the 2-exchange neighborhood are a criterion to estimate the difficulty for some instance types. For comparison purposes, the number of steps was scaled by the number of cities in a given instance. Independent of the algorithm used (`1kh` or `concorde`), Koch instances are harder the more local search steps (normalized) are required. For example, for instances with $1.5n$ steps, the `1kh` algorithm needs less than 0.15 CPU seconds on average, while for instances with $3n$ steps about 5 CPU seconds are needed. For MPeano and MNPeano, a similar exponential relation can be observed. But for `concorde`, there is a broad distribution of computation times, and `1kh` times are somewhat obscured by the additional penalties for unsuccessful runs. For David instances, no clear relation between number of steps and computation times is observed. Interestingly, grid instances of the same size all have nearly the same number of steps to reach a local optimum, regardless of structure perturbation (shaking level). In contrast, random Euclidean instances show a clear relation between number of steps and computation times, although the distribution gets broader as more steps are required. For more than $2n$ steps the computation times for `concorde` and `1kh` for the same type of instance differ up to a factor 100 and 1000, respectively.

The classical nearest neighbor relation for successors has been also investigated. Less than 3.4% of all instances' cities required a neighbor for their optimal tour that has a nearest neighbor list position above 20. Instances having successors above list position 20 include strongly shaken grid instances, David curves as well as large MNPeano and MPeano instances. Considering all instances, the 0.95 quantile over the list indices has a maximum of 110, but the third quartile is only 7. In a statistical analysis, however, we did not find a significant correlation between the maximum of the nearest neighbor list positions and computation times (both `1kh` and `concorde`).

A total of 64 instances (0.4% of all instances) were not solved to optimality in all 10 `1kh` runs. Possible reasons for this behavior have been investigated. Instances that are hard to solve for `1kh` have in their optimal tour city neighbors that are more distant than neighbors of the other instances. For an average instance, the average tour successor is the 10th nearest neighbor, compared to the 32nd nearest neighbor for instances with incomplete runs. Performing a statistical analysis of all successors within the tours shows that all quantiles are significantly higher for hard instances than for average instances.

4 Conclusions

The structure of a TSP instance has an important impact on its hardness. As shown, fractal and grid instances without any modification are solved efficiently by both `concorde` and `1kh`. Subjecting these instances to an increasing amount of perturbation, and hence reducing their structure, renders them significantly harder to solve, which leads ultimately to instances whose hardness (and structure) approaches that of random Euclidean instances. This also suggests that random Euclidean TSP instances can be relatively hard to solve compared to more structured instances.

When subjected to very strong perturbations, instances tend to become easier again. For reduction, this effect can be explained by the decrease in the number of cities. As for any combinatorial problem, reduced problem size has a strong effect on solving time; this affects both fractal and grid instances for `1kh` and `concorde`.

To summarize the analysis of properties, it is clear that there is no single attribute that determines the hardness of an instance. Nevertheless, it is possible to estimate `1kh`'s or

concorde's computation times for certain types of instances. For Koch instances, there is a strong relation between problem size and the computation times for both *lkh* and *concorde*. The exact algorithm's running times grow linear with the number of cities, while the running times for Helsgaun's program increase much more strongly. For fractal and random Euclidean instances, the number of 2-exchange steps to a local optimum provides a basis for run-time estimation. However, for instances with larger number of steps (scaled by the instance size) this estimation becomes increasingly unreliable. Especially the occasional occurrence of unsuccessful runs make it hard to predict running times for Helsgaun's program.

References

- [1] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding tours in the TSP. Technical Report 99885, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.
- [2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. *Mathematical Programming Series B*, 97(1–2):91–153, 2003.
- [3] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.
- [4] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.
- [5] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, USA, 2004.
- [6] Fred Glover. New ejection chain and alternating path methods for traveling salesman problems. In Osman Balci, R. Sharda, and S. Zenios, editors, *Computer Science and Operations Research*, pages 449–509. Pergamon Press, New York, NY, USA, 1992.
- [7] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.
- [8] D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
- [9] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [10] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.
- [11] P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297–345, 2001.
- [12] P. Moscato and M. G. Norman. On the performance of heuristics on finite and infinite fractal instances of the Euclidean traveling salesman problem. *INFORMS Journal on Computing*, 10(2):121–132, 1998.
- [13] César Rego. Relaxed tours and path ejections for the traveling salesman problem. *European Journal of Operational Research*, 106(1):522–538, 1997.

Vienna, Austria, August 22–26, 2005

- [14] G. Reinelt. TSPLIB. <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>, November 2004.