Optim Lett
DOI 10.1007/s11590-014-0828-5

SHORT COMMUNICATION

On the empirical time complexity of finding optimal solutions vs proving optimality for Euclidean TSP instances

Holger H. Hoos · Thomas Stützle

Received: 15 April 2014 / Accepted: 5 November 2014

Abstract We investigate the empirical performance of the long-standing state-of-the-art exact TSP solver Concorde on various classes of Euclidean TSP instances and show that, surprisingly, the time spent until the first optimal solution is found accounts for a large fraction of Concorde's overall running time. This finding holds for the widely studied random uniform Euclidean (RUE) instances as well as for several other widely studied sets of Euclidean TSP instances. On RUE instances, the median fraction of Concorde's total running time spent until an optimal solution is found ranges from 0.77 for $n = 500$ to 0.97 for $n = 3{,}500$; on TSPLIB, National and VLSI instances, we pegged it at 0.86, 0.74 and 0.61, respectively, with a tendency of even smaller values for larger instances.

Keywords TSP · Empirical complexity · Exact algorithms

1 Introduction

The travelling salesman problem (TSP) is arguably the most prominent and widely studied NP-hard optimisation problem overall (see, e.g., [3,10,13]). High-performance algorithms for solving the TSP play an important role in academic research and numerous practical applications (see, e.g., [3]), and the development and analysis of such

Electronic supplementary material The online version of this article (doi:10.1007/s11590-014-0828-5) contains supplementary material, which is available to authorized users.

H. H. Hoos (✉)
Computer Science Department, University of British Columbia, Vancouver, BC, Canada
e-mail: hoos@cs.ubc.ca

T. Stützle
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

Published online: 28 November 2014

algorithms has been deeply connected to general insights into solving hard combinatorial optimisation problems.

Two-dimensional Euclidean TSP instances, in which the objective is to find the shortest round trip visiting a given set of locations (typically referred to as 'cities') in the Euclidean plane, represent perhaps the most prominent special case of the general TSP. Although the 2D Euclidean TSP is NP-hard, exact TSP algorithms can solve instances of up to few thousands of cities within hours on a standard CPU [4]; furthermore, the scaling of running time with instance size has recently been shown to follow a function of the form $a \cdot b^{\sqrt{n}}$ with $a = 0.21$ and $b = 1.24194$ for the state-of-the-art exact TSP solver Concorde on random uniform Euclidean (RUE) instances [9].

The running times of exact TSP solvers, such as Concorde [4], for finding optimal solutions and completing the corresponding proof of optimality are often much higher than those required by high-performance inexact solvers to find optimal solutions in cases where those are known [1,7,11]. This is consistent with common perceptions regarding the differences in difficulty between finding optimal solutions to an NP-hard optimisation problem (without a proof of optimality) and proving optimality—and, more generally, between solving NP-hard problems and their co-NP-hard counterparts.[1] The study presented here challenges these perceptions, by investigating the fraction of overall running time spent by Concorde, which represents the state of the art in exact TSP solving (and has held this distinction for over 10 years), for finding an optimal tour to a given 2D Euclidean TSP instance.

The remainder of this article is structured as follows. In Sect. 2 we outline the setup of our empirical study and in Sect. 3, we present the results we obtained. Section 4 provides further discussion of our findings, and in Sect. 5, we draw some overall conclusions and provide a brief outlook on future work.

2 Materials and methods

The TSP solver we chose for our study is Concorde, version 03.12.19 [4]—the best performing exact algorithm currently known for the symmetric TSP. To solve the linear programming problems that arise during the sophisticated branch & cut search carried out by Concorde, we used QSopt 1.01, an LP solver specifically designed for this purpose. Concorde's parameters were left at their default settings, and we used the random number seed 23 throughout our experiments.

We ran the Concorde solver on four sets of prominent TSP instances. The first of these comprises so-called random uniform Euclidean (RUE) instances, which are obtained by placing n points uniformly at random in a square, with integer coordinates between 1 and 1,000,000, each point corresponding to a city to be visited. The distance matrix contains the Euclidean distances between the respective points rounded to the nearest integer. For generating the RUE instances, we used the `portgen` generator from the 8th DIMACS Implementation Challenge. Our instance set comprises 1,000 instances for each instance size $n \in \{500, 600, \ldots, 2{,}000\}$ and 100 instances for each $n \in \{2{,}500, 3{,}000, 3{,}500, 4{,}000, 4{,}500\}$.

[1] We further discuss the reasons for holding such perceptions in Sect. 4.

The second set comprises 27 instances from TSPLIB [14] with 500 to 3,500 cities with edge types EUC 2D, CEIL 2D and ATT. Larger instances from these sets would require infeasibly long computation times with Concorde, while smaller instances may lead to floor effects due to the limited accuracy of running time measurements.

The third and fourth sets contain National and VLSI instances, respectively, taken from the TSP webpage at http://www.math.uwaterloo.ca/tsp/index.html. The National instances are based on the locations of cities within countries, while the VLSI instances stem from applications in VLSI circuit design. The six National instances with 500 to 3,500 cities were all solved by Concorde within 3 CPU hours on our reference machines (specified below). Of the 41 VLSI instances with n between 500 and 3,500, Concorde was unable to solve 19 within 7 CPU days. These instances are known to be particularly hard for Concorde, and for many of the instances we failed to solve, provably optimal solutions are currently unknown.

To study the behaviour of Concorde on these instance sets, we used a cluster of identical computer nodes, each equipped with two dual-core 2.4 GHz AMD Opteron 2,216 processors with 2 $\times$ 1MB L2 cache and 4GB main memory, running Cluster Rocks Linux, version 4.2.1/CentOS 4. All experiments were run on a single CPU core, using code compiled with gcc-3.4.6-3 and optimization flag settings `-O3`. We note that all instances use integer distances, and therefore Concorde terminates as soon as the difference between the shortest tour and the tightest lower bound falls below a value of one.

Running times were measured based on CPU times reported by the Concorde code. In particular, we measured the overall running time on each TSP instance and the time at which the solution later proven to be optimal is first encountered. We note that Concorde is an anytime optimisation algorithm in that it reports improvements in the tours it constructs when solving a TSP instance. Therefore, an optimal tour, once found by Concorde, is reported and could be used in the context of a practical application, potentially quite long before Concorde completes the proof of its optimality. We note that this way of separating the time required for finding an optimal solution and for completing a run that ultimately produces a proof of optimality applies to arbitrary anytime optimisation algorithms. Furthermore, this approach does not require difficult judgement calls on how to attribute the time spent by components that serve the purpose of finding good solutions quickly and also contribute to a proof of optimality.

3 Results

Figure 1 shows the fraction of Concorde's total running time spent on RUE instances until an optimal solution is found. (Tables containing the numerical results underlying this and the following figures are included in the supplementary material). Surprisingly, the fraction of running time required to find optimal solutions tends to be quite large and *increases* with instance size. This is seen in the median fraction of the additional running time required as well as in the 10th and 90th percentiles. For example, at $n = 2{,}500$, Concorde spends more than 90 % of its total running time on finding optimal solutions on over 50 % of the RUE instances, and more than 40 % on 90 % of the instances. At $n = 3{,}500$, the median fraction of time spent on finding optimal

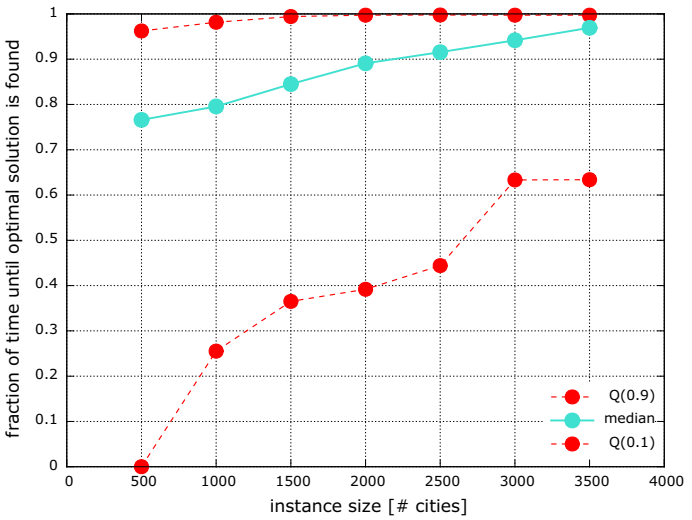

Fig. 1 Concorde on RUE: fraction of total running time spent before an optimal solution is first found

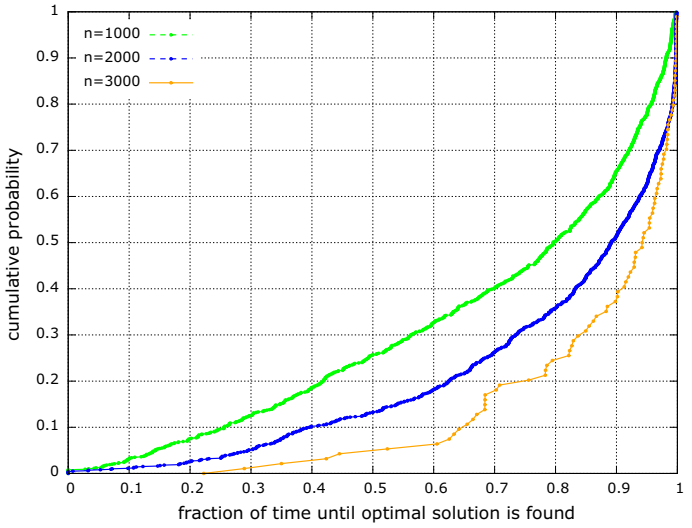

Fig. 2 Concorde on RUE: fraction of total running time spent until an optimal solution is found, CDFs over sets of instances

solutions has increased to more than 96.9 %, and the 10th percentile to more than 63 %.

Further analysis revealed that the same trend is observed for all quantiles of the respective distributions over sets of RUE instances of size n. Figure 2 shows the cumulative distribution functions (CDFs) for the fraction f_p of Concorde's total running time spent until an optimal solution to a given instance is found over three sets of RUE instances. All quantiles of these distributions increase with instance size, as do the

fractions of instances for which f_p exceeds a given threshold; in particular, for only 4 of the 74 instances of size $n = 4{,}000$ solved within our cutoff time we observed fractions smaller than 0.5, compared to 10 of 100 instances for $n = 2{,}500$ and 258 of 1,000 instances for $n = 1{,}000$.

For even higher n, more than 10 % of the 100 instances in our sets could not be solved by Concorde within the 7 CPU day cutoff time used in our experiments; therefore, we could not obtain reliable estimates for the median and other quantile values of the fraction of Concord's running time spent on finding optimal tours. However, there is no indication that those statistics are significantly lower than those for smaller instance sizes; e.g., from the results for the 65 instances at $n = 4{,}500$ solved by Concorde, we can estimate the median fraction of time spent on finding optimal solutions to lie between 0.8141 and 0.9938, and the 90th percentile to be at most 0.9973.

The 10th percentile for $n = 500$ is zero, as we did not consider the time taken by Concorde to run the initial $\max\{n/2{,}500\}$ iterations of the CLK procedure, Concorde's implementation of an iterated Lin-Kernighan algorithm [4], and in a small percentage of instances with $n = 500$ the initial solution obtained from CLK was already optimal. However, the overall effect of this inaccuracy is small: on average, the initial CLK run takes 0.44 s on our machines, while the average solution time for instances of size 500 is 36.2 s; additionally, the deviation is in line with our arguments, namely that the time taken to find an optimal solution is often surprisingly large.[2]

Results for TSPLIB instances are overall consistent with the observations for RUE instances (see Fig. 3). Of the 27 TSPLIB instances we considered, Concorde solved 25 within the 3 CPU day cutoff time used in our experiments (runs on *u1817* and *d2103* did not terminate, and optimal solutions for these instancs were not found by Concorde within this cutoff time). Over those 25 instances, the median fraction of Concorde's running time spent until an optimal solution is found was measured at 0.8573. On 18 instances, the fractions fell within the band between the 10th and 90th percentiles determined on RUE instances, while 4 fell above and 3 below. This is close to the theoretical value of 20 instances expected to fall within this confidence band. Findings for the 6 National TSP instances in the range between $n = 500$ and $n = 3{,}500$ are similar: the median fraction of running time for finding optimal solutions is 0.7402, with 4 of the 6 instances between the 10th and 90th percentile determined on RUE instances (see Fig. 3).

For the 22 VLSI instances on which our runs of Concorde completed, the median time spent for finding optimal solutions was 0.6169—lower than for the RUE, TSPLIB and National instances, but still high. As seen in Fig. 3, unlike in the case of the TSPLIB and National instances, compared to RUE instances of similar sizes, a significant number of the VLSI instances required a lower fraction of Concorde's running time for finding optimal solutions; still, there appears to be the same tendency for that fraction to increase with instance size.

[2] Note that on larger instances, no optimal solutions were found in Concorde's initial CLK phase.

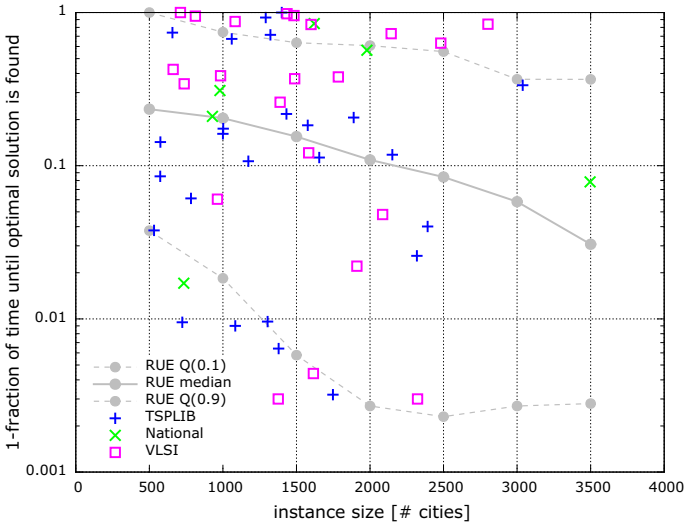

Fig. 3 Concorde on TSPLIB, National and VSLI *vs* RUE: one minus the fraction of total running time spent before an optimal solution is first found (this representation was chosen to optimise readability of the plot). Note that, because of the inverted measure on the *y*-axis, the 10th percentile of the time to find the first optimal solution appears near the top, while the 90th percentile appears near the *bottom* of the plot

4 Discussion

It was surprising for us to see that the long-standing state-of-the-art exact TSP solver, Concorde, when solving Euclidean TSP instances, requires a major part of its running time to find optimal solutions, while once an optimal solution is found, the additional computational burden of completing the proof of optimality is relatively modest. This runs against the widely held intuition that proving optimality of a solution is inherently more difficult than finding optimal solutions, an intuition based on the observation that in a search-based approach, like Concorde's, optimal solutions can be found using good heuristic guidance and may involve only reasoning about a small part of the search space; proving optimality, on the other hand, involves reasoning over the entire space of solutions.

Formally, the decision variant of the TSP is defined as follows: given a TSP instance and bound on tour length l, determine whether there is a tour of length at most l. The finding of high-quality solutions corresponds to the 'yes' instances of this NP-complete decision problem, while proving optimality corresponds to the 'no' instances, and therefore to 'yes' instances of the co-NP-complete problem of deciding whether there exists *no* tour of length at most l. It is widely believed that NP≠co-NP and that, moreover, 'yes' instances of co-NP-hard problems are even harder to solve than 'yes' instances of NP-hard problems. This is consistent with the fact that 'yes' instances of co-NP-hard problems have been shown to require exponential length proofs in the worst case in a number of proof systems underlying practically effective solvers (see, e.g., [5,6,12])—which, of course, does not imply NP≠co-NP, since it might be possible to build effective solvers based on more powerful proof systems.

While our empirical results are not in contradiction with any known results in complexity theory, they suggest that in practice—at least within complex branch & cut algorithms like Concorde, where the tasks of finding optimal solutions and proving optimality are tightly integrated—optimality proofs can often be rather quickly completed, once an optimal solution has been found. The experimental results reported here were obtained with the default version of Concorde, which is the result of a prolonged and intense algorithm engineering effort by a group of world-leading researchers, as described in detail by Applegate et al. [3]. Different behaviour may be obtained by reconfiguring or modifying Concorde.

In particular, it is known that high-quality and even optimal solutions can be found very fast, using incomplete solvers based on stochastic local search, such as the Iterated Lin-Kernighan procedure by Helsgaun [7,8] or the recent edge assembly evolutionary algorithm (EAX) by Nagata and Kobayashi [11]. For example, EAX has been reported to frequently find optimal solutions to TSPLIB instances of up to 18,512 cities within computation times of about 4,000 CPU seconds on a Xeon 2.93 GHz CPU, while Concorde cannot solve many of these instances within a week of CPU time. Furthermore, it has been observed that knowledge of high-quality tours is crucial for Concorde's ability to solve large Euclidean TSP instances (including proof of optimality) [2].[3] Therefore, one might conjecture that (1) the initial CLK local search phase is important for Concorde's ability to solve Euclidean TSP instances effectively, and (2) by seeding Concorde with even better solutions than obtained from the limited-length CLK run it performs by default, its running time can be improved.

However, a preliminary investigation of these conjectures, in which we performed longer runs of the initial CLK phase (of 10,000 iterations), indicates that this is not necessarily the case, and even providing Concorde with optimal tours at the start of a run does not yield substantial, consistent speedups (see supplementary material). We still see value in strengthening the initial local search phase in Concorde to improve its anytime performance, even if this does not result in substantial reductions in overall running time, or to simply run a state-of-the-art inexact TSP solver concurrently.

Finally, we note that our results reported here for RUE, TSPLIB, National and VLSI instances reinforce a recent finding regarding the scaling of Concorde's running time on the same sets of instances [9]: Results for TSPLIB and National instances are consistent with those for similarly sized RUE instances, which suggests that the randomly structured RUE instances are a good model for more structured classes of Euclidean TSP instances as far as Concorde's performance is concerned. On the VLSI instances, as also found in our earlier scaling study [9], Concorde's performance deviates from that observed on RUE, TSPLIB and National instances. Our scaling study indicated that these instances tend to be harder to solve than other Euclidean TSP instances of similar size; here, we find that the fraction of Concorde's running time spent on proving optimality after an optimal solution has been found tends to be

[3] We note that for solving very large TSP instances, non-standard settings of Concorde are used, and special techniques are exploited to obtain feasible computation times on clusters of parallel machines; for details, see Chapter 16 of [3]. In fact, for the proof of optimality for the instance `sw24978` (which involves the locations of cities in Sweden), it was stated explicitly that *"the long Sweden computation benefited greatly form the accurate upper bounds provided by Helsgaun's tour"* [3], p. 517.

higher for these instances. We believe that the reason for both of these observations lies in the fact that the VLSI instances differ markedly from other types of Euclidean TSP instances in that they contain many short edges of identical length.

5 Conclusions and future work

The empirical results for the long-standing state-of-the-art TSP solver Concorde reported here challenge commonly held beliefs regarding the fraction of running time required by an exact optimisation algorithm for finding optimal solutions vs completing optimality proofs. It will be interesting to investigate to which extent state-of-the-art algorithms for other NP-hard optimisation problems, such as mixed integer programming, show similar behaviour, and to determine what causes this phenomenon. We are hopeful that it may be possible to improve the anytime behaviour, and possibly also to reduce the overall running time of exact algorithms, by better exploiting knowledge of the constructive bounds and respective solutions obtained from state-of-the-art inexact algorithms.

Acknowledgments We gratefully acknowledge helpful input from David Mitchell on connections with complexity theory. Furthermore, we thank the anonymous reviewers for their useful comments. This work was supported by the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. H. H. acknowledges support through an NSERC Discovery Grant. T. S. acknowledges support from the Belgian F. R. S.-FNRS, of which he is a senior research associate.

References

1. The traveling salesman problem. Version visited last on 15 April 2014. http://www.math.uwaterloo.ca/tsp/ (2014)
2. Applegate, D.: Personal communication (2009)
3. Applegate, D., Bixby, R.E., Chvatal, V., Cook, W.J.: The traveling salesman problem: a computational study. Princeton University Press, Princeton (2006)
4. Applegate, D., Bixby, R.E., Chvatal, V., Cook, W.J.: Concorde TSP solver. Version visited last on 15 April 2014. http://www.math.uwaterloo.ca/tsp/concorde.html (2014)
5. Beame, P., Pitassi, T.: Propositional proof complexity: past, present, and future. In: Current trends in theoretical computer science: entering the 21st century, pp. 42–70. World Scientific Publishing (2001)
6. Dash, S.: Exponential lower bounds on the lengths of some classes of branch-and-cut proofs. Math. Oper. Res. **30**(3), 678–700 (2005)
7. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur. J. Oper. Res. **126**, 106–130 (2000)
8. Helsgaun, K.: General k-opt submoves for the Lin-Kernighan TSP heuristic. Math. Program. Comput. **1**(2–3), 119–163 (2009)
9. Hoos, H.H., Stützle, T.: On the empirical scaling of run-time for finding optimal solutions to the traveling salesman problem. Eur. J. Oper. Res. **238**(1), 87–94 (2014)
10. Lawler, E.L., Lenstra, J.K., Rinnooy Kan., A.H.G, Shmoys, D.B.: The traveling salesman problem. Wiley, Chichester (1985)
11. Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS J. Comput. **25**(2), 346–363 (2013)
12. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artif. Intell. **175**, 512–525 (2011)
13. Reinelt, G.: The traveling salesman: computational solutions for TSP applications. Lecture Notes in Computer Science, vol. 840. Springer, Heidelberg, Germany (1994)
14. Reinelt, G.: TSPLIB. Version visited last on 15 June 2012, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95 (2012)