CS 312

Vol. 12, Issue 6 (2011)
Pages 101-105

Empirical Performance of Geo-2opt, a Variant of the 2opt Algorithm, on Random Traveling-Salesman Datasets

Greg Philbrick, Matt Seeley, Justin Seliger

Computer Science Department, Brigham Young University, 3361 TMCB Provo, UT 84602

Received on December 7, 2011; awarded an A+ grade on December 7, 2011

ABSTRACT

The 2opt algorithm is a heuristic method that uses a type of localized search to find near-optimal solutions on traveling-salesman datasets in polynomial time. In this paper, we discuss a variation of the 2opt algorithm that we call geo-2opt and we measure its performance on several datasets, comparing it to the performance of the branch-and-bound and greedy algorithms.

INTRODUCTION

The traveling-salesman problem (TSP) is well-known and highly studied.[1] Since it is NP-Complete,[2] exponential time is required for all currently known algorithms that are guaranteed to find a globally optimal solution for a given set of cities. However, there are many useful heuristic algorithms that run in faster time. Many of these heuristic algorithms use local-search techniques to find locally optimal solutions within some subset of solutions; the key is to choose a subset of solutions that is very likely to contain the globally optimal solution or at least a very good solution.

One of these useful heuristic algorithms was first proposed by G. A. Croes of the Shell Development Company in Houston, Texas, in 1957.[3] This algorithm, which is now known as the "2opt" method, proceeds as follows: First, find an initial solution using a fast method, such as a greedy approach. Next, delete two edges from the path; we will refer to these edges as {a, b} and {c, d}. If the sum of the costs of edges {a, d} and {b, c} is less than the sum of the costs of {a, b} and {c, d}, replace {a, b} and {c, d} with {a, d} and {b, c} if the distances between all pairs of cities are symmetric. If there are asymmetric distances in the dataset, it is necessary to do a more thorough check. Specifically, we must verify that the cost of the path {a, b, m_1, m_2, …, m_{j-1}, m_j, c, d} is greater than the cost of the path {a, d, m_j, m_{j-1}, …, m_2, m_1, b, c}, where m_1 … m_j represent the cities in the path between cities b and d in the order in which they appear in our original solution. Repeat this process for some set of pairs of edges in the current solution as desired; the greater the number of edge pairs examined, the greater the improvement over the original solution is likely to be.

The traditional 2opt algorithm has been shown to perform much better in practice than its worst-case theoretical bounds might suggest.[4] In addition, because it has been known for several decades, there is plentiful peer-reviewed literature discussing its variations and its derivative forms. Furthermore, the algorithm itself is intuitive and relatively straightforward to implement. For all these reasons, a variation of the 2opt algorithm was chosen for this project.

DESCRIPTION OF THE ALGORITHM

Since the 2opt algorithm uses a form of local search that moves out from an initial solution, it is preferable that we start with a solution that is likely to be close to the optimal solution. Careful study revealed the fact that optimal paths for geometric TSP problems usually comprise closed polygons with no crossing edges. This intuitively makes sense because, in a geometric TSP problem, "any tour that crosses itself can be shortened by replacing a pair of crossing edges, where an edge is a tour segment going directly from one city to another."[5] With these considerations in mind, we adopted an $O(n^3)$ method from Sedgewick[6] to generate an initial path comprising a closed polygon with no crossing

edges. This method begins by taking three cities and connecting them to form the simplest possible polygon: a triangle. In order to add a new city to the polygon, one edge between two cities that are currently in the polygon must be deleted and two new edges from those two cities to the new city must be added. In order to ensure that the newly expanded path maintains its identity as a closed polygon, the two new edges added must not cross any edges that remain in the path once whichever edge we choose to delete is removed. Figures 1-4 illustrate the first full iteration of the city-adding process. Once the updated polygon path is completed, the process repeats by adding one city at a time until all *n* cities are included in the path. This method is implemented in the source code as GetPolygonPath. On datasets where a closed-polygon path is not found, a path with as few crossing edges as possible is generated instead.

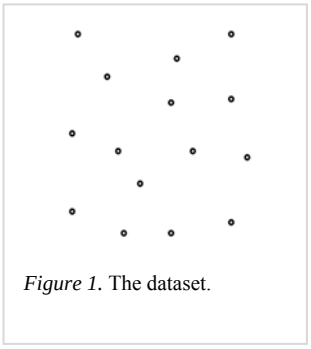

Figure 1. The dataset.

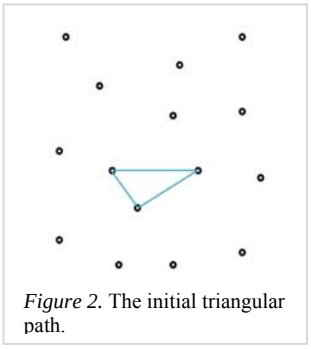

Figure 2. The initial triangular path.

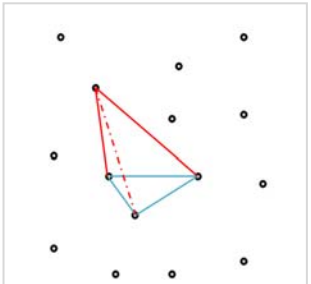

Figure 3. Deciding which two edges to add. Note that the crossing edge is excluded.

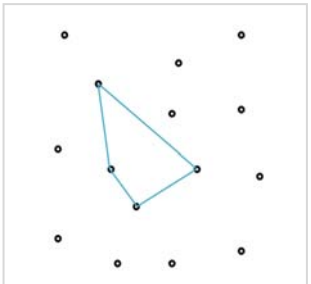

Figure 4. The newly formed polygon path including the new edge.

Before moving on to the next step, the cost of the reverse path is also calculated; the better of the two paths is used as the initial path.

With the initial path chosen, the next important matter to consider is how to define a useful set of pairs of edges in the initial path such that using the 2opt method on all these pairs of edges would be likely to yield a worthwhile improvement. Because the basic 2opt method is most likely to find a better local solution if a large number of edge pairs are examined, we decided to use the set of all possible pairs of edges from the original solution. With a dataset of size *n*, a given solution must have exactly *n* edges (though it technically only has n-1 degrees of freedom; once the first n-1 edges have been chosen, there is only one permissible option for the last edge). The number of possible ordered pairs of two edges taken from a set of *n* edges without replacement is ——— ——— ———, ($n \in \mathbb{N}$) ∩ ($n \geq 2$). Thus, this set of edge pairs can be defined in polynomial time. In order to increase the likelihood of finding more improvements, we also decided to consider replacing edges a given pair {a, b}, {c, d} with the second alternate pair {c, a}, {d, b} at each iteration in addition to the first alternate pair {a, c}, {b, d} that is suggested in the original 2opt algorithm. That way, the number of possible improvements examined is increased by a linear factor. For this project we arbitrarily decided to terminate once *n* improvements have been made in order to limit the running time One iteration of this modified 2opt approach is illustrated in figures 5-7.

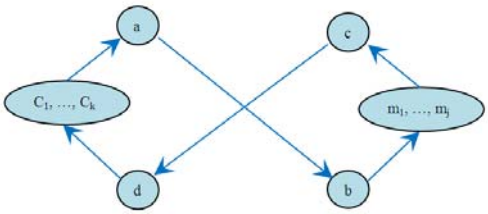

Figure 5. The state of a current path before edges {a, b} and {c, d} are replaced.

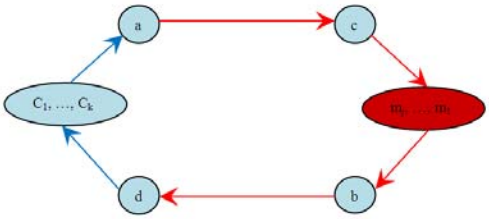

Figure 6. The state of the path if {a, b}, {c, d} is replaced with the first alternative pair {a, c}, {b, d}. Note that the intermediate edges connecting cities m_1, …, m_j that make up the sub-path between city b and city c are replaced with the reversed-order sub-path m_j, …, m_1. For datasets with asymmetric distances, the cost of the reversed-order sub-path must be calculated because it may not equal the cost of the original-order sub-path.

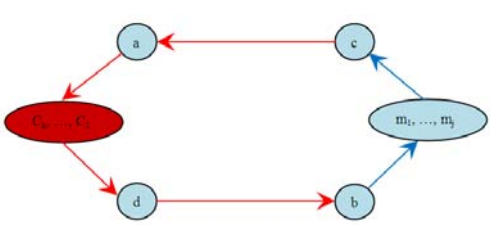

Figure 7. The state of the path if {a, b}, {c, d} is replaced with the first alternative pair {c, a}, {d, b}. Note that the intermediate edges connecting cities C_1, …, C_k that make up the sub-path between city b and city c are replaced with the reversed-order sub-path C_k, …, C_1.

	15 Speed	15 Path	15 Imp	40 Speed	40 Path	40 Imp	60 Speed	60 Path	60 Imp
Random	0.0008	8878.1		0.001	20525.1		0.0007	31614.3	
Greedy	0.0016	4115.4	0.46355	0.0030003	6688.2	0.32585	0.005701	7962.7	0.25187
B & B	0.648	3322.3	0.80728	TB	TB	TB	TB	TB	TB
Geo-2Opt	0.0239	3438.1	0.83542	0.335	5346.7	0.79942	0.988	6587.8	0.82733

Figure 9. Mean results for datasets of size 15, 40, and 60.

	100 Speed	100 Path	100 Imp	200 Speed	200 Path	200 Imp	300 Speed	300 Path	300 Imp
Random	0.001	53119.8		0.001	104962		0.0014	154551	
Greedy	0.010201	11159.3	0.21008	0.037404	15663.6	0.14923	0.057306	20248.4	0.13101
B & B	TB	TB	TB	TB	TB	TB	TB	TB	TB
Geo-2Opt	4.106	8564.4	0.76747	40.212	12139.4	0.77501	160.511	15156.5	0.74853

Figure 10. Mean results for datasets of size 100, 200, and 300.

EMPIRICAL PERFORMANCE

With our modified version of the 2opt algorithm thus explained, its empirical performance will now be discussed. Our first task was to gather data on the performance of (1) an algorithm that generates a random path, (2) a greedy algorithm, (3) a branch-and-bound algorithm, and (4) our modified 2opt algorithm (Geo-2opt). We chose to measure how each algorithm performed in terms of elapsed time and path length using datasets of 15, 40, 60, 100, 200, and 300 cities. For each different dataset size, ten different trials with randomly generated datasets of the respective size were performed. All trials were performed on computers in the 1066 TMCB lab in order to minimize variation due to processor speeds and other hardware issues. The mean values for elapsed time and path length were calculated and recorded in figures 8-9.

The differences in the table values are visible, but it is first necessary to test whether or not they are statistically significant. This baseline consideration is important because any difference that could plausibly be attributed to sampling error could cast any conclusion we could make from this data into doubt. For our null hypothesis, let $H_0 = \mu_{greedy}=\mu_{geo\text{-}2opt}$ (i.e., there is no difference in the mean path distances of the greedy algorithm and the Geo-2opt algorithm for all datasets of size *n*). For the alternative hypothesis, let $H_a = \mu_{greedy}>\mu_{geo\text{-}2opt}$ (i.e., the mean path distance for the Geo-2opt algorithm is less than the mean path distance for the greedy algorithm for all datasets of size *n*). To test this hypothesis, we will use a two-sample *t* test to compare the mean path distances. The two-sample t statistic is defined as:[7]

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} - \frac{s_2^2}{n_2}}}$$

Using this equation and a *t*-distribution table for nine degrees of freedom, the p-values for the problems of each size are shown in figure 10.

Dataset Size	μ_{greedy}	$\mu_{geo\text{-}2opt}$	Difference	*t*-statistic	p-value
15	4115.4	3438.1	677.3	2.97	<0.01
40	6688.2	5346.7	1341.5	7.02	0.00
60	7962.7	6587.8	1374.9	7.45	0.00
100	11159.3	8564.4	2594.9	12.39	0.00
200	15663.6	12139.4	3524.2	9.17	0.00
300	20248.4	15156.5	5091.9	13.26	0.00

Figure 10. Tests for statistical significance to ensure that the improvements are not due to sampling error.

Given these p-values for each dataset, we can safely reject the null hypothesis and conclude that, on average, the mean path distances produced by the Geo-2opt algorithm on datasets of these sizes are smaller than those produced by the greedy algorithm. Thus, the Geo-2opt algorithm can be considered useful for finding paths that are regularly better than those found by the Greedy algorithm.

With the question of statistical significance answered, we now consider what the results suggest about the efficiency and accuracy of the Geo-2opt algorithm. In terms of speed, the Geo-2opt algorithm clearly runs in much less time than the branch-and-bound algorithm. While the branch-and-bond algorithm did not return a result after approximately 15 minutes of running time on a dataset of size 40, the geo-2opt algorithm returned a

solution in about one third of a second. It should come as no surprise that the branch-and-bound algorithm's running time quickly became unmanageable; its exponential rate of increase simply makes it an impractical method for datasets of large sizes. However, for the data that we were able to collect (i.e., problems of size 15), the path distance found by geo-2opt compared very favorably with the path distance found by branch-and-bound: geo-2opt returned a path distance that was about 83.54% of the greedy distance, while branch-and-bound returned a path distance that was about 80.72% of the greedy distance. In other words, branch-and-bound had a less than three percent advantage over geo-2opt in terms of path optimization. However, the running time required for geo-2opt was about 3.69% of the running time required for branch-and-bound. This suggests that the geo-2opt algorithm offers a trade-off of a relatively small sacrifice in accuracy for a substantial gain in running-time efficiency.

Another fascinating result was that the geo-opt algorithm actually seemed to perform better in terms of its improvement percentage with the larger datasets than with the smaller ones. For the three smaller problem sizes, the geo-2opt path distance was, on average, about 82.07% of the greedy path. For the three larger problem sets, the average dropped to 76.37%. Because these results represent a relatively small sample size of improvement percentages, it is harder to say with certainty whether the trend observed in these results was simply due to noise or chance. This would be an interesting question to explore in future research.

In order to determine the empirical big-O bound of the geo-2opt algorithm, we used the Texas Instruments TI-89 graphing calculator to generate a regression curve to fit the data points of running time in seconds plotted against problem size. The graph is shown in figure 11. We predicted that it would be $O(n^4)$ because the number of possible edge pairs for each path was $O(n^2)$ edge pairs, the path reversals needed to calculate the alternative moves for each edge pair were $O(n)$, and we had arbitrarily set the program to continue seeking improvements to each successive complete path until n improvements were made. As a result, we chose to generate a quartic regression curve. With coefficients rounded to four decimal places, the curve that best fit the data was:

$$\textit{Running time} = (6.9954 \cdot 10^{-9})n^4 + (4.507 \cdot 10^{-6})n^3 - (2.6295 \cdot 10^{-4})n^2 + 0.0170n - 1.974$$

This curve actually fit the data so accurately that the R^2 value was listed as one (1); in other words, the curve fit the data so perfectly that, insofar as the TI-89 could compute, 100% of the variation in running time could be explained by the problem size. With such an accurate equation to describe the geo-2opt algorithm's behavior, it becomes even clearer that the constant coefficients are very favorable for this apparently $O(n^4)$ algorithm. (It should be noted that such a highly accurate curve would not be as trustworthy if there had been only five data points instead of six because the quartic regression equation for any bivariate dataset of size five would simply return the Legrange polynomial for that dataset.)

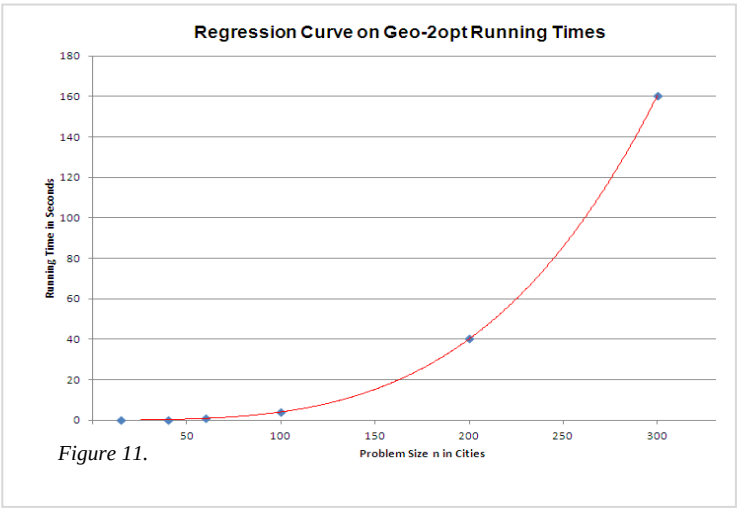

Figure 11.

CONCLUSION

The geo-2opt algorithm, a modified version of the original 2-opt algorithm that uses a closed-polygon path as the initial path for its local search, performed very well on the datasets used in this study. Its path-distance improvement over the greedy algorithm was statistically significant, leaving little doubt that it handily outperforms the greedy algorithm in the average case. In addition, for the dataset of size 15, its improvement percentage over the greedy algorithm's path distance was less than three percent less optimal than the improvement percentage of the branch-and-bound algorithm. The data also suggest that geo-2opt's improvement percentage may increase as the size of the dataset increases, though further research will be needed to verify this observation. There is

strong evidence that its running time is $O(n^4)$ with extremely favorable constant factors, which suggests it offers an excellent trade-off between accuracy and running time. These data all suggest that the geo-2opt algorithm is likely a powerful and valuable tool for finding highly optimal solutions to large TSP problems.

REFERENCES

[1] M. R. Garey, R. L. Graham, and D. S. Johnson. 1976. "Some NP-complete geometric problems." In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing* (STOC '76). ACM, New York, NY, USA, 10-22. DOI=10.1145/800113.803626 http://doi.acm.org/10.1145/800113.803626.

[2] C. H. Papadimitriou. "The Euclidean travelling salesman problem is NP-complete." *Theoretical Computer Science* vol. 4 (1977): 237-244.

[3] G. A. Croes. "A method for solving traveling salesman problems." *Operations Res.* vol 6 (1958): 791-812.

[4] David S. Johnson and Lyle A. Mcgeoch. "The Traveling Salesman Problem: A Case Study in Local Optimization." In *Local Search in Combinatorial Optimization*. Princeton: Princeton University Press, 2003.

[5] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton: Princeton University Press, 2006,

[6] Robert Sedgewick and Kevin Wayne, *Algorithms*, 4th Ed. Boston: Pearson Education, Inc., 2011, available online at http://algs4.cs.princeton.edu/91primitives/.

[7] David S. Moore, *The Basic Practice of Statistics*, 5th Ed. New York: W. H. Freeman and Company, 2010.