PERGAMON

AVAILABLE AT
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Neural Networks 16 (2003) 827–832

Neural Networks

www.elsevier.com/locate/neunet

2003 Special issue

Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks

Samuel A. Mulder[a,*], Donald C. Wunsch II[b]

[a]*Applied Computational Intelligence Lab, Department of Computer Science, University of Missouri–Rolla, Rolla, MO 65401, USA*
[b]*Applied Computational Intelligence Lab, Department of Electrical and Computer Engineering, University of Missouri–Rolla, Rolla, MO 65401, USA*

Abstract

The Traveling Salesman Problem (TSP) is a very hard optimization problem in the field of operations research. It has been shown to be NP-complete, and is an often-used benchmark for new optimization techniques. One of the main challenges with this problem is that standard, non-AI heuristic approaches such as the Lin–Kernighan algorithm (LK) and the chained LK variant are currently very effective and in wide use for the common fully connected, Euclidean variant that is considered here. This paper presents an algorithm that uses adaptive resonance theory (ART) in combination with a variation of the Lin–Kernighan local optimization algorithm to solve very large instances of the TSP. The primary advantage of this algorithm over traditional LK and chained-LK approaches is the increased scalability and parallelism allowed by the divide-and-conquer clustering paradigm. Tours obtained by the algorithm are lower quality, but scaling is much better and there is a high potential for increasing performance using parallel hardware.

Keywords: Traveling salesman problem; Lin–Kernighan algorithm; Adaptive resonance theory

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most studied problems in computer science literature. It is an example of the important class of problems known as NP-complete problems. An NP-complete problem is one that is solvable in polynomial time by a non-deterministic algorithm, but not necessarily by a deterministic one. Another way of considering this class of problems is to say that a correct solution may be checked in polynomial time, but no known algorithm can solve the problem in that time. These problems are interesting because it is possible to draw parallels between any two NP-complete problems and show that finding an algorithm to solve one in polynomial time gives you an algorithm to solve the other. They are also interesting because it has never been proven that NP-complete problems cannot be solved in polynomial time, so it remains as an open question. The most general form of the TSP is to find a Hamiltonian cycle given an arbitrary graph. All known algorithms to solve this problem take greater than polynomial time, but if we have a solution we can check it in $O(n)$ time. The more specific case of the TSP considered in this paper is also NP-complete, but is more complicated to check.

The traveling salesman family of problems is an area in which neural networks have previously been unable to compete with the best non-neural approaches, in accuracy, speed, or scaling. (See (Vishwanathan & Wunsch, 2001) for a critique of the then-state-of-the-art in neural network approaches to TSP. Improvements in the past 2 years notwithstanding, accuracy, speed and scaling were significant limitations.) Accuracy remains an important limitation. In fact, it seems doubtful whether neural networks will ever exceed the accuracy of algorithms like the Lin–Kernighan solution (Lin & Kernighan, 1973) in solving this class of problems. Clustering algorithms in general are limited in the accuracy they can achieve on the TSP Table 1. If highly accurate tours are required, then Lin–Kernighan and variants (Applegate, Cook, & Rohe, 2000) seem likely to remain the algorithms of choice. In situations where accuracy is not the primary determinant, however, neural networks may show some promise. This is especially relevant for situations where significant modifications to (Lin & Kernighan, 1973) and (Applegate et al., 2000) may be required, as opposed to minor modifications and retraining, which is often all that is needed for a neural network.

* Corresponding author.
E-mail addresses: smulder@umr.edu (S.A. Mulder), dwunsch@ece.umr.edu (D.C. Wunsch II).

0893-6080/03/$ - see front matter © 2003 Elsevier Science Ltd. All rights reserved.
doi:10.1016/S0893-6080(03)00130-8

Our goal then is to evaluate what advantages a neural clustering algorithm can bring to bear on the problem and decide how to best exploit those advantages. Adaptive resonance theory (ART) provides a very rapid and effective neural clustering model. Functionally, it operates similarly to k-means clustering, with an optimal k determined dynamically by the vigilance factor. The divide and conquer paradigm gives us the flexibility to hierarchically break large problems into arbitrarily small clusters depending on what trade-off between accuracy and speed is desired. In addition, the sub-problems provide an excellent opportunity to take advantage of parallel systems for further optimization. Even without parallel processing, the algorithm developed in this paper has demonstrated that better scaling is possible using a divide and conquer approach.

1.1. Traveling salesman problem

Given a complete undirected graph $G = (V, E)$, where V is a set of vertices and E is a set of edges each relating two vertices with an associated non-negative integer costs $c(u, v)$, the most general form of the TSP is equivalent to finding any Hamiltonian cycle over G where such a cycle is known as a tour. The more common form of the problem is the optimization problem of trying to find the shortest Hamiltonian cycle. Both of these problems have been proven to be NP-complete in (Cormen, Leiserson, & Rivest, 1996). These problems are very useful to consider because they map so easily to many real-world applications in a wide range of fields from network routing to cryptography (Agarwala, Applegate, Maglott, Schuler, & Schaffler, 2000; Bailey, McLain, & Beard, 2003; Turino, 2002).

The variation of the TSP that we (and most neural network approaches to this topic) consider is even more limited. We look only at graphs that can be mapped to a two-dimensional Euclidean coordinate system, where the edge weight between two nodes is the distance between them. This system implies that the triangle inequality holds: $c(u, w) \leq c(u, v) + c(v, w)$. We also require that the XY coordinates of each node be given. This variation can be shown to still be NP-complete and therefore to map onto the more general problem (Papadimitriou, 1977). The Euclidean form of the TSP has been widely studied (Aurora, 1998; Braun & Buhmann, 2002; Cochrane & Cochrane, 1999).

1.2. Adaptive resonance theory

ART was first introduced by Carpenter and Grossberg (1988). The unsupervised variants provide a simple, effective neural clustering algorithm. The original algorithm, designated ART1, used binary input sequences. The variation developed for this solution uses two-input integer sequences where the integers are the XY coordinates of a node. In this form, ART is functionally similar to k-means clustering except that k is increased dynamically as new patterns are introduced.

1.3. Local search techniques

By far the most successful algorithms in tackling large-scale ($1000 +$ cities) TSPs have been the family of algorithms known as local search heuristics (Johnson & McGeoch, 2002). The basic idea of local search is to start with some tour, either randomly generated or generated by some fast but low quality method, and make iterative improvements, hopefully driving it towards the optimal. In many ways this resembles the neural network technique of starting with random initial weights and then making incremental improvements via a technique like steepest descent, in an attempt to minimize errors. In the case of the TSP, the error to be minimized may be seen as the tour length. The local search class of algorithms considers a series of minor changes to determine whether they reduce the tour length.

A simple local search technique that has been easy to implement is to consider the result of swapping any two edges currently in the tour with two new edges in such a way that a valid tour still exists. In a permutation representation of the tour, this change can be seen as the operation of flipping a segment of the tour. By repeatedly searching for the flip that minimizes the tour length, and stopping when no flips can be found that reduce the tour length, a shorter tour may be discovered. Once no shorter tour can be found through this method, the tour is said to be locally optimal. This specific algorithm is referred to in the literature as 2-Opt. There is a general class of algorithms of this nature that involve substituting a fixed number of edges with new edges such that a valid tour is maintained. These are referred to as k-Opt algorithms. An $(n + 1)$-Opt algorithm, where n is the number of cities in the problem, will always find the optimal solution, i.e. the optimal solution is at most $n + 1$ edges away from the current tour (Rego & Glover, 2002)

Considering the time complexity of k-Opt algorithms, it immediately becomes obvious that large values of k are not practical. A naïve implementation of 2-Opt requires $O(n^2)$ time, and in general a k-Opt algorithm requires $O(n^k)$ time. A number of improvements to this family of algorithms have been considered to reduce this running time. The primary improvement in terms of running time has been the use of bounded neighbor lists. With this modification, the nearest h neighbors are calculated for each city, and the search for edges to swap is restricted to this list of neighbors for each city. This neighbor list may be generated in a number of different ways, but in the general case require time $O(n^2 \log h)$, which may be reduced to $O(n \log n)$ in the case of Euclidean TSPs using k–d trees (Johnson & McGeoch, 2002). The main problem with this improvement is that it limits the quality of tours generated, since it restricts which edges are considered for improvement. In practice, the speed-up is so large and the difference in solution is so small that this improvement is

almost universally applied. Cases may be constructed, however, where it causes very poor tours to be found.

1.4. The Lin–Kernighan local optimization algorithm

The current best results on large-scale TSP instances come from variations on an algorithm proposed by Lin and Kernighan (1973), (Rego & Glover, 2002). This algorithm takes a randomly selected tour and optimizes it by stages until a local minimum is found. This result is saved and a new random tour selected to begin the process again. Given enough time, the optimal tour can be found for any instance of the problem, although in practice this is limited to small instances for an absolute result due to time constraints. Even with extremely large instances, however, reasonably short tours can be found (Johnson & McGeoch, 2002).

The optimizations performed by the Lin–Kernighan (LK) algorithm use the concept of λ-optimality. If n is the number of cities in a TSP, then we know that given any tour we can reach an optimal tour by swapping at most n edges. Using λ-optimality, we consider the optimal tour that can be reached by swapping only λ edges. As λ grows, the chance that the λ-optimal tour is actually optimal increases. Unfortunately, the processing time required to find λ-optimal tours is intractable for large λ. The largest commonly used values for λ are 2 and 3, although some attempts have been made with λ as 4 or 5 (Lin & Kernighan, 1973).

The algorithm proposed by Lin and Kernighan examines variable λ optimization. In brief, an initial 2-edge swap is chosen. The algorithm then examines whether it would be profitable to make a 3-edge swap, and continues increasing the number of edges as long as the total swap remains an improvement over the initial tour. This process is repeated with each node being considered in turn as a starting point for the initial 2-edge swap until no further improvements can be found. The original algorithm then started with a different random tour and worked from the beginning, always saving the best tour seen. This original algorithm is what is used in our current algorithm. Applying more advanced variants of LK will likely yield even better results (Johnson & McGeoch, 2002).

Table 1
Algorithm Pseudocode

```
Read data from file
Apply ART algorithm to cluster data
For each cluster
   Apply LK optimization
   Apply intersection removal
   Apply LK optimization
   Merge with final tour
End for
Save final tour and time
```

2. Algorithm-divide and conquer

The algorithm developed in this paper combines ART and the LK local optimization algorithm to divide and conquer instances of the TSP. We begin by reading in the cities, stored in TSP–LIB format (Moscoto, 2002). The ordering of the cities in memory represents the current tour at any time. This is known as a permutation representation and obviously only works with fully connected variants of the TSP, as an arbitrary permutation of a non-fully connected graph would not necessarily represent a valid tour. This permutation representation is chosen because weight-matrix representations become intractable for large values of n, i.e. a 250k-city problem would require around 62.5 GB of memory to store an edge-weight matrix. Since the problem is Euclidean, it is sufficient to store (X, Y) coordinates for each city and calculate distances on the fly.

The first stage of the algorithm involves sorting the cities into clusters using the ART algorithm described previously. Our variation of ART uses the vigilance parameter to set a maximum distance from the current pattern. A vigilance parameter between 0 and 1 is considered and used as a percentage of the global space to determine the vigilance distance. Values were chosen based on the number and size of individual clusters desired, but typical values ranged from 0.80 to 0.97. The learning rate was set to 0.02. The clusters at this point were still in a random order. The individual clusters were then each passed to a version of the LK algorithm also described above. Since the size of the tours was controlled and kept under a thousand cities, we allowed the LK search depth to be infinite as long as the total improvement for a given swap series remained positive. We also did not restrict the neighborhood size as specified in the original algorithm. This required more time than a limited depth search, but resulted in better overall tour quality. After a first pass through the LK algorithm, a simple intersection removal algorithm was applied. This algorithm is based on the idea that any tour containing an intersection between two edges is demonstrably sub-optimal. Typically after LK is run, few intersections remain, but those that do remain often involve edges crossing large distances. Swapping these edges changes the global search space sufficiently that a new run of LK can often find additional improvement. In short, the intersection removal algorithm is capable of making double-bridge swaps that the LK algorithm is unable to discover. This double-bridge property is the same as that used in the chained LK algorithms described in (Applegate et al., 2000).

Now we are faced with the problem of combining a number of sub-tours back into one complete tour. This may be accomplished by adding the cluster tours into a combined tour, one at a time. Obviously the order in which the tours are added back is important. To minimize the cost added by inter-tour edges, it is important to add tours that are adjacent to the current combined tour. Our method for accomplishing this is to add the tours in order of increasing distance from

the origin. It is not clear whether this is the optimal solution, and this is an area for future research. The other major factor involved with merging the tours is running time. Since this a potentially global operation, care must be exercised in the nature of the algorithm. For example, attempting to find an optimal linking between the two tours could be at least an $O(n_g^2)$ algorithm, which is unacceptable, because the n involved would be the total number of cities, not just the cities in a tour. To avoid the $O(n_g^2)$ global operation, we first find the centroid of the cluster to be added. This is just the average of the X and Y coordinates of each city in the cluster, and is easily calculated in $O(n_c)$, where the n involved is the size of an individual cluster. We then find the k nearest cities to that centroid in the combined tour. Clearly, this operation requires $O(n_g)$ time. Next, we consider each of the k cities from the main tour to determine the cost of inserting the cluster tour in place of the following edge. This involves the comparison of k cities to n_c cities to determine the lowest cost matching, yielding a running time of $O(k^* n_c)$, where $k \ll n_g$. Finally, the cluster tour is inserted into the merged tour at the best location discovered.

3. Results

The results from our experiments were very encouraging. In all these results, the cities were distributed randomly (uniform distribution), generated by the generation algorithm provided at the TSPLIB web site (Moscoto, 2002). All experiments were run on a 2 GHz AMD Athlon MP processor with 512 M of DDR RAM. In Tables 2 and 3 below, we compare our clustered variation of the LK algorithm to the original LK algorithm. The LK algorithm used for these comparisons is that same algorithm that we implemented and used on the clusters of our main algorithm. Since the focus of these experiments was to determine the effects of clustering, the LK implementation is not heavily optimized and the running times reflect are only to be used in comparison to each other. In this implementation of the LK algorithm, neighborhoods were not used. Instead, the algorithm compared all possible swaps at a given level. This resulted in relatively slow running times. Further below, a more representative LK implementation is benchmarked against our technique. However, in Tables 2 and 3 results, some noteworthy items are already apparent.

Table 2
original Lin–Kernighan algorithm implemented by author

TSP size	Tour length	Time (s)
1000	2.63808e7	5
2000	3.64058e7	21
4000	5.07807e7	82
20000	1.12221e8	4814

Table 3
Clustered Lin–Kernighan

TSP size	Tour Length	Time (s)	Off LK (%)
1000	2.69785e7	1	2
2000	3.80334e7	3	4
4000	5.35437e7	8	5
20000	1.17e8	98	4

Table 4
Concorde chained LK

TSP size	Tour length	Time (s)
10000	7.20532e7	38
20000	1.01468e8	85
250000	3.58274e8	1380

Obviously the clustered variation is scaling much better than the original, and tour quality is remaining within 5% or so. Also, with the clustered version we can smoothly trade time for quality in either direction by adjusting the vigilance factor.

In the second set of experiments, we compare our algorithm with an improved version of the LK algorithm, the Concorde software package. This version incorporates the limited neighborhood structure that most implementations of the LK algorithm use, as well as additional optimizations. Concorde is widely regarded as the fastest TSP solver, for large instances, currently in existence. Concorde is described in more detail in (Applegate, Bixby, Chvatal, & Cook, 2001). Essentially, it uses the chained LK algorithm. Search depth for LK is greatly limited, but high quality is obtained by repeatedly applying double-bridge transformations and optimizing. The Concorde (Table 4) package also applies a simple algorithm to generate an initial tour of fair quality before beginning that provides a large speed boost to the LK algorithm. These techniques could also be implemented in our clustered variation giving similar speed and quality improvements. For now we are just interested in the scalability of the relative algorithms.

In Table 5, we can clearly see that while our algorithm is below the quality of tour that the chained-LK algorithm produces, it is scaling nicely. Our quality-of-result numbers, reported in the last column of Table 5, are comparable with previously published neural net results for much smaller TSP problems, see (Vishwanathan & Wunsch, 2001). In fact, our algorithm gives us additional flexibility in terms of changing the vigilance factor to generate different numbers of clusters. To illustrate this, Table 6 shows the results of the 250k city problem as the vigilance parameter is adjusted.

Table 5
Clustered Lin–Kernighan

TSP size	Tour length	Time (s)	Off CLK (%)
10000	8.21277e8	57	14
20000	1.17e8	98	15
250000	4.27169e8	693	19

Table 6
Clustered Lin–Kernighan 250k city problem

Vigilance	Tour length	Time (s)	% off CLK
0.95	4.09444e8	2451	14
0.96	4.18222e8	1077	17
0.97	4.27169e8	693	19
0.98	4.47744e8	778	25

As is clear from the table, increasing the vigilance parameter increases the speed at the expense of tour quality. We also observe the limit of 0.97 for this particular problem past, which the time begins to increase instead of decrease. This is the point where the individual constant factors involved with processing each cluster begin to outweigh the advantages of having smaller clusters. The actual vigilance parameter chosen depends largely on the quality of tour needed and the time available.

As a further experiment, a version of the program was developed which used a neighborhood structure similar to that used in the Concord package, as well as other algorithm optimizations described in (Johnson & McGeoch, 2002). This resulted in significant speed and quality increases are seen in the comparison provided by Tables 7 and 8. It also gives an opportunity to illustrate the potential scalability of our technique. At the 1,000,000-city level, our solution is just 11% off of the Concorde algorithm results, but only takes 16% as long to run. (Because of the much longer run time of the Concorde algorithm, some intermediate size problems are omitted, thus the blank spaces in the last column of Table 7 and the shorter Table 8.)

Table 7
New clustered LK algorithm results for randomly distributed TSP

Cities (#)	Tour length	Time	Vigilance	Off (%)
1000	2.58499E + 07	0.422	0.7	10.40
2000	3.61019E + 07	1.031	0.7	10.64
4000	5.07564E + 07	2.484	0.7	
6000	6.20307E + 07	5.141	0.72	
8000	7.13997E + 07	8.328	0.72	10.97
10,000	7.96673E + 07	11.359	0.75	10.57
20,000	1.12156E + 08	24.641	0.8	10.53
30,000	1.37686E + 08	35.407	0.82	
50,000	1.78731E + 08	61.985	0.83	
85,000	2.33053E + 08	104.094	0.85	
100,000	2.53306E + 08	122.407	0.86	
250,000	3.99968E + 08	315.078	0.92	11.64
500,000	5.64177E + 08	667.375	0.95	
750,000	6.91278E + 08	1015.890	0.96	
1,000,000	7.93896E + 08	1468.165	0.97	11.03

Table 8
Concorde results for randomly distributed TSP

Cities (#)	Tour length	Time
1000	2.34148E + 07	1.670
2000	3.26303E + 07	3.500
8000	6.43398E + 07	26.570
10,000	7.20532E + 07	37.620
20,000	1.01468E + 08	84.830
250,000	3.58274E + 08	1379.540
1,000,000	7.15058E + 08	9013.530

4. Conclusions

Our main result is the potential of combining neural clustering with traditional local search techniques. A significant speedup was shown by using clustering in combination with the original Lin–Kernighan algorithm. This speedup was offset by a 2–5% loss in tour quality that is the result of the inherent limitations of clustering when applied to the TSP. Clustering approaches will typically be unable to locate a global minimum because they assume a certain structure on the underlying data that may be sub-optimal. In most real-world situations, however, finding the absolute global minimum is not a requirement. Most applications prefer speed to tour quality within limitations. While our current implementation doesn't beat the latest chained-LK techniques, with heavy optimization, developed over many years of research, it does scale better, and gives additional flexibility when trading tour quality for speed. The modified version of the algorithm, incorporating many of the latest improvements in the LK algorithm, appears to scale on the order of $O(n^* \lg n)$. Scaling was shown through one million cities. In addition, since the number of global operations is limited in our algorithm, it should scale smoothly with parallel hardware and show significant improvement over current solutions.

Acknowledgements

The authors gratefully acknowledge support from the National Science Foundation and M.K. Finley endowment.

References

Agarwala, R., Applegate, D.L., Maglott, D., Schuler, G.D., & Schaffler, A.A (2000). *A fast and scalable radiation hybrid map construction and integration strategy*. http://www.ncbi.nlm.nih.gov/genome/rhmap/

Applegate, D., Bixby, R., Chvatal, V., & Cook, W (2001). *Concorde—A code for solving traveling salesman problems*. http://www.math.princeton.edu/tsp/concorde.html

Applegate, D., Cook, W., & Rohe, A (2000). *Chained Lin–Kernighan for large traveling salesman problems*. Technical report http://www.keck.caam.rice.edu/reports/chained lk.ps

Aurora, S. (1998). Polynomial time approximation schemes for {Euclidean} traveling salesman and other geometric problems. *Journal of the ACM*, *45*(5), 753–782.

Bailey, C. A., McLain, T. W., & Beard, R. W. (2003). Fuel saving strategies for dual spacecraft interferometry missions. *Journal of the Astronatical Science*, in press.

Braun, M. L., & Buhmann, J. M. (2002). The noisy Euclidean traveling salesman problem and learning. *Advances in neural information processing systems*. Cambridge, MA: MIT Press.

Carpenter, G., & Grossberg, S. (1988). The art of adaptive pattern recognition by a self-organizing neural network. *IEEE Computer*, *March*, 47–88.

Cochrane, E. M., & Cochrane, J. M. (1999). Exploring competition and co-operation for solving the Euclidean traveling salesman problem by using the self-organizing map. *Artificial Neural Networks*, *1*, 180–185.

Cormen, T., Leiserson, C., & Rivest, R. (1996). *Introduction to algorithms*. Cambridge, MA: MIT Press, pp. 954–960.

Johnson, D., & McGeoch, L. (2002). Experimental analysis of heuristics for the STSP. In G. Gutin, & A. Punnen (Eds.), *The traveling salesman problem and its variations* (pp. 369–487). Boston: Kluwer Academic Publishers.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, *21*, 498–516.

Moscoto, P (2002). *TSPBIBHome Page*. http://www.densis.fee.unicamp.br/~moscato/TSPBIB_home.html

Papadimitriou, C. H. (1977). The Euclidean traveling salesman problem is NP-complete. *Theoretical Computer Science*, *4*, 237–244.

Rego, C., & Glover, F. (2002). Local search and metaheuristics. In G. Gutin, & A. Punnen (Eds.), *The traveling salesman problem and its variations* (pp. 309–368). Boston: Kluwer Academic Publishers.

Turino, J (2002). *ASIC DFT and BIST Alternatives*. http://www.chipcenter.com/asic/tn004.html

Vishwanathan, N., & Wunsch, D (2001). A Hybrid Approach to the TSP. *IEEE International Joint Conference on Neural Networks, Washington, DC*.