ANADOLU ÜNİVERSİTESİ

Bilim ve Teknoloji Dergisi B-Teorik Bilimler
Cilt: 2 Sayı: 2 2013
Sayfa:143-148

ARAŞTIRMA MAKALESİ / **RESEARCH ARTICLE**

Gözde KIZILATEŞ[1], Fidan NURİYEVA[2]

A NEW HYBRID HEURISTIC ALGORITHM FOR SOLVING TSP

ABSTRACT

The Traveling Salesman Problem (TSP) is an important and well known combinatoriyal optimization problem. The goal of the problem is to find the shortest tour that visits each city in a given list exactly once and then returns to the starting city. Although the definition of the travelling salesman problem is easy, it belongs to NP-Hard class. In this paper, a new hybrid heuristic algorithm based on Nearest Neighbour (NN) and Greedy heuristic algorithms is proposed for solving the TSP. This proposed hybrid heuristic algorithm is compared with NN and Greedy heuristics. The experimental results show that the proposed algorithm is efficient.

Keywords: Traveling salesman problem, Hybrid heuristic algorithm, Nearest neighbour algorithm, Greedy algorithm

GEZGİN SATICI PROBLEMİNİ ÇÖZMEK İÇİN YENİ BİR HİBRİD SEZGİSEL ALGORİTMA

ÖZ

Gezgin Satıcı Problemi (GSP) önemli ve iyi bilinen kombinatoriyal optimizasyon problemidir. GSP'de amaç, bir satıcının, bulunduğu şehirden başlayıp, her şehre sadece bir kez uğradıktan sonra başladığı noktaya geri dönen en kısa turu bulmaktır. GSP kolay görünmesine rağmen, NP-zor sınfına ait bir problemdir. Bu çalışmada, GSP'nin çözümü için Yakın Komşu ve Açgözlü algoritmalarına dayanan yeni bir hibrid sezgisel algoritma önerilmiştir. Önerilen yeni hibrid sezgisel algoritma, Yakın Komşu ve Açgözlü algoritmaları ile kıyaslanmıştır. Hesaplama denemeleri önerilen algoritmanın verimli olduğunu göstermektedir.

Anahtar Kelimeler: Gezgin satıcı problemi, Hibrid sezgisel algoritma, Yakın komşu algoritması, Açgözlü algoritma

[1], Ege University, Faculty of Science, Department of Mathematics
Tel: (554) 7084143,E-mail: gozde.kizilates@gmail.com

[2] İnstitute of Cybernetics, Azerbaijan National Academy of Sciences
Tel: (0531) 0805209, E-mail: nuriyevafidan@gmail.com

Recieved: 28 November 2012 **Revised:** 28 March 2013 **Accepted:** 19 June 2013

1. INTRODUCTION

The Traveling Salesman Problem (TSP) is an NP-hard problem and one of the most famous and well studied problems in combinatorial optimizational field (Gutin and Punnen, 2002). In standard TSP, the goal is to find the minimum length Hamiltonian cycle through a set of *n* cities, given the distances between all pairs of cities (Lawler and others, 1986). In other words, TSP can be considered as a graph problem in which cities are represented by vertices and distances between cities are represented by edges. There are many variations of the problem. In this work, we examine the classic symmetric TSP.

Solving TSP is an important part of many applications in different fields including vehicle routing, computer wiring, machine sequencing and scheduling, frequency assignment in communication networks as well as data analysis in psychology and clustering in biostatistics (Lenstra, 1974) (Johnson and Liu, 2006). For instance, data analysis applications in psychology ranging from profile smoothing to finding an order in developmental data are presented by (Hubert and Baker, 1978). Clustering and ordering using TSP solvers are currently becoming popular in biostatistics (Climer and Zhang, 2006).

2. APPROACHES FOR SOLVING TSP

The algorithms for solving TSP can be divided into four classes: exact algorithms, heuristic algorithms, approximate algorithms and metaheuristic algorithms (Land and Doi1g, 1960).

The exact algorithms usually utilize the integer linear programming model of the TSP. “Branch & Bound” is one of the examples for this category (A. Land andA. Doig, 1960). One approach that comes to mind first is to try all possibilities. Other approach can be dynamic programming (M. Held and R. Karp, 1962).

In general, the heuristic algorithms are subdivided into the following three classes: tour construction algorithms, tour improvement algorithms and hybrid algorithms (Johnson and McGeoch, 1997). The tour construction algorithms gradually build a tour by adding a new city at each step, such as the nearest neighbor algorithm, the insertion algorithm, algorithm based on spanning tree, the saving algorithm and the random algorithm. The tour improvement algorithms improve a tour by performing various exchanges, such as 2-opt and 3-opt.

Approximation algorithms give us a guarantee as to how bad solutions we can get, normally specified as *c* times the optimal value. The best known approximate algorithms for TSP are Christofides Algorithm (guaranteed value is 3/2), Minimum-Spanning Tree (MST) based algorithms (guaranteed value is 2), and others (S. Lin and B. Kernighan, 1973).

Metaheuristic algorithms are the techniques which try to improve iteratively the candidate solution (or solutions) found by a specific approach for hard optimization problems. Metaheuristic algorithms accept the heuristic approach for solving the problem as a black box and don’t care about the details (Rego and Glover, 2002). They only try to optimize the functions used to solve the problem. These functions are named as goal functions or objective functions. Tabu search, genetic algorithms, simulated annealing, artificial neural networks, ant colony algorithm and similar artificial intelligence approaches are the examples for this category.

3. SOME HEURISTIC ALGORITHMS

In this study, we will focus on only Nearest Neighbour and Greedy Algorithms since the proposed algorithm is based on them.

3.1 Nearest Neighbour

This is perhaps the simplest and most straightforward TSP heuristic.

A nearest neighbor (NN) algorithm produces a tour by sequentially adding a connection from the most recent city visited to the nearest city that has not yet been visited. This procedure is repeated until all of the cities in the problem have been visited, at which time a final connection is made to the original/starting city. The algorithm uses each city in the problem as a start/end point to produce N such tours and records the shortest solution. The computational complexity of this algorithm is $O(n^2)$ (Johnson and Papadimitriou, 1985a).

The steps of the algorithm are as following:

Step 1. Select a random city.

Step 2. Find the nearest unvisited city and go there.

Step 3. Are there any unvisited cities left? If yes, go to step 2; otherwise go to step 4.

Step 4. Return to the first city.

We can obtain the best result out of this algorithm by starting the algorithm over again for each city and repeat it for *n* times.

3.2 Greedy Algorithm

The Greedy heuristic gradually constructs a tour by repeatedly selecting the shortest edge and adding it to the tour as long as it doesn't create a cycle with less than *n* edges, or increase the degree of any city by more than 2. We must not add the same edge twice of course.

The computational complexity of this algorithm is $O(n^2 \log_2 n)$ (Johnson and Papadimitriou, 1985b).

The steps of the algorithm are as following:

Step 1. Sort all edges.

Step 2. Select the shortest edge and add it to our tour if it doesn't violate any of the above constraints.

Step 3. Do we have *n* edges in our tour? If no, go to step 2, otherwise go to step 4.

Step 4. Terminate the algorithm.

4. A NEW HYBRID HEURISTIC ALGORITHM

The algorithm that we have proposed is a hybrid of the traditional NN and Greedy heuristic algorithms. We start the algorithm with NN for each city. Each time the algorithm is applied, we give a "priority" to the edge according to the result of the solution. Let the "priority" of the selected edges in the first solution be 1 and all the others be 0. Suppose that the length of the first tour is D_1. We add $\frac{D_1}{D_i}$ (Here, D_i is the length of the tour, which is found at step *i*) to the "priorities" of the selected edges. At the next steps, the edges are sorted in descending order by their updated "priorities", and then, we solve the problem with Greedy algorithm. This process continues until there is no change on the sorting anymore. The result of the algorithm is the best solution found during this process.

The steps of the algorithm are as following:

Step 1. We start the algorithm with NN for each city and repeat it for *n* times.

Step 2. Assign the best solution as record solution.

Step 3. Suppose that the length of the record tour is D_r. Then, we add $\frac{D_r}{D_i}$ (Here, D_i - is the length of the tour, which is found at step *i*) to the "priorities" of the selected edges. Thus, each edge has a "priority" after *n* steps.

Step 4. Sort the edges in descending order by "priority".

Step 5. Solve the problem by greedy algorithm.

Step 6. Update the priority of the first edge which is not in the solution.

Step 7. If the solution is better (shorter) than the record solution, update the record and update the priority. Subsract *n* from "priorities" of the elements that are in the record solution but are not in the current solution.

Step 8. If the solution is worse (longer) than the record solution, update the priority. Subsract *n* from "priorities" of the elements that are not in the record solution but are in the current solution.

Step 9. Repeat this procedure until *3*n* iteration is complete.

4.1 Computational Complexity of the Proposed Algorithm

The worst-case complexity of the algorithm can be calculated as follows. The cost of finding a tour according to NN algorithm is $O(n^2)$. In this algorithm, *n* tours are found by NN algorithm starting from each vertex. The total complexity for these operations becomes $O(n^3)$. Then, "priority" value is calculated for each edge that is used in the found tours and these values are put in order. Because each edge has a "priority" value for sorting operation as well, *n*(n-1)/2* data are sorted in sorting operation. Therefore, complexity of the sequencing process becomes $O(n^2 \log n)$. Aftewards, a result is found by the Greedy algorithm with respect to these "priorities". The complexity of the Greedy algorithm is $O(n^2 \log n)$. The result found by the Greedy algorithm and "priority" values are updated. The Greedy algorithm and updating process are iterated *3*n* times. As a result, the general complexity of the algorithm will be $O(n^3 \log n)$.

5. COMPUTATIONAL EXPERIMENTS

This section presents the results of the computational experiments for the proposed hybrid heuristic algorithm. The computer program of the proposed algorithm has been coded in C++. The computational experiments have been implemented using 2.6 GHz Intel Core 2 Duo CPU processor and 2 GB RAM, 32-bit windows.

The sample problems used in these experiments are taken from (www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/) and the optimum solutions for each of these problems are taken from (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ST).

Table I shows the length of the tour computed by NN ve Greedy heuristics and new hybrid heuristic algorithm that we have proposed. Since it is based on Nearest Neighbour (NN) and Greedy heuristic algorithms, the proposed algorithm has been compared with these algorithms.

In Table I, selected cells show the best results that algorithms have found.

Table I Computational Experiments

G	**Optimal**	**NN Time (s)**	**Greddy Time (s)**	**Hybrid Algorithm Time (s)**
ulysses22	75.665	86.905 0.000	89.436 0.010	79.157 0.014
bayg29	9074.148	9964.781 0.000	9886.208 0.015	9413.957 0.054
att48	33523.708	39236.885 0.000	38849.621 0.125	37737.693 0.662
eil51	429.983	505.774 0.016	481.518 0.125	487.448 0.813
berlin52	7544.365	8182.192 0.000	9954.062 0.281	7791.375 0.887
st70	678.597	761.689 0.000	746.044 0.485	754.760 3.893
eil76	545.387	612.656 0.016	617.131 0.672	599.682 5.858
gr96	512.309	603.302 0.015	580.101 1.609	592.573 17.938
rat99	1211	1369.535 0.016	1528.308 1.875	1328.376 20.498
kroA100	21236.951	24698.497 0.016	24197.285 1.937	24447.780 22.392
kroB100	22141	25882.973 0.015	25815.214 2.469	25464.985 23.158
kroC100	20750.762	23566.403 0.016	25313.671 2.610	22984.858 22.806
kroD100	21294.290	24855.799 0.016	24631.533 2.359	24224.136 22.554
kroE100	22068	24907.022 0.016	24420.355 2.609	24411.041 23.018
rd100	7910.396	9427.333 0.015	8702.605 2.922	9321.775 22.891
eil101	642.309	736.368 0.015	789.112 2.609	731.943 23.901
lin105	14382.995	16939.441 0.015	16479.785 3.187	16068.265 27.923
pr107	44303	46678.154 0.016	48261.816 2.109	46122.818 31.726
gr120	1666.508	1850.263 0.032	1915.918 4.282	1830.913 56.169
ch130	6110.860	7198.741 0.016	7142.045 7.688	7014.895 82.407

6. CONCLUSION

In this paper, we have proposed a hybrid heuristic algorithm for solving TSP based on traditional Nearest Neighbour and Greedy algorithms. As it is seen in Table I, in which the obtained results from computational experiments are shown, comparing with NN and Greedy algorithms, the proposed algorithm generally gives solutions closer to the optimum. However, the proposed algorithm lags behind NN and Greedy algorithms with respect to running time. We aim to arrange more efficiently the usage of NN algorithms in the first part of the algorithm in order to decrease the running time.

REFERENCES

Climer, S., Zhang, W. (2006). Rearrangement Clustering: Pitfalls, Remedies, and Applications. *Journal of Machine Learning Research*, 7, 919 – 943.

Gutin, G., Punnen, A. (eds.). (2002). *The Traveling Salesman Problem and Its Variations*. Vol. 12 of Combinatorial Optimization. Kluwer, Dordrecht.

Held, M., Karp, R. (1962). A Dynamic Programming Approach to Sequencing Problems. *Journal of SIAM*, 10, 196 – 210.

Hubert, L. J., Baker, F. B. (1978). Applications of Combinatorial Programming to Data Analysis: The Traveling Salesman and Related Problems. *Psychometrika*, 43(1), 81-91.

Johnson, D., Papadimitriou, C. (1985a). *Computational Complexity. In Lawler et al, Chapter* 3, 37-86.

Johnson, D., Papadimitriou, C. (1985b). *Performance Guarantees for Heuristics. In Lawler et al, Chapter* 5,145-180.

Johnson, D. S. and McGeoch, L. A. (1997). "The Traveling Salesman Problem: A Case Study", *Local Search in Combinatorial Optimization*, 215-310, John Wiley & Sons.

Johnson, O., Liu, J. (2006). A Traveling Salesman Approach for Predicting Protein Functions. *Source Code for Biology and Medicine*, 1(3), 1-7.

Land, A., Doig, A. (1960). An Automatic Method for Solving Discrete Programming Problems. *Econometrica*, 28, 497-520.

Lawler, E. L., Lenstra, J. K., Rinnoy, Kan A. H. G., Shmoys D. B. (1986). The Traveling Salesman Problem*: A Guided Tour of Combinatorial Optimization*, John Wiley & Sons.

Lenstra, J. K. (1974). Clustering a Data Array and The Traveling-Salesman Problem. *Operations Research*, 22(2), 413-414.

Lin, S., Kernighan, B. (1973). An Effective Heuristic Algorithm for The Traveling-Salesman Problem. *Operations Research*, 21(2), 498-516.

Ray, S. S., Bandyopadhyay, S., Pal, S. K. (2007). Gene Ordering in Partitive Clustering using Microarray Expressions. *Journal of Biosciences*, 32(5), 1019-1025.

Rego, C., Glover, F. (2002). *Local Search and Metaheuristics. In Gutin and Punnen* (2002), Chapter 8, 309-368.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, Germany.

http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/