Proceedings of IAM, V.4, N.1, 2015, pp.8-15

A TOUR EXTENDING HYPER-HEURISTIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

Gözde Kızılateş[1], Fidan Nuriyeva[2,3], Hakan Kutucu[4]

[1]Ege University, Department of Mathematics, Izmir, Turkey
[2]Institute of Control Systems of ANAS, Baku, Azerbaijan
[3]Dokuz Eylul University, Department of Computer Science, Izmir, Turkey
[4]Karabuk University, Department of Computer Engineering, Karabuk, Turkey
e-mail: gozde.kizilates@gmail.com, fidan.nuriyeva@gmail.com, hakankutucu@karabuk.edu.tr

Abstract. In this paper, a new hyper-heuristic algorithm is proposed for the Symmetric Traveling Salesman Problem. In the proposed method, we first create a tour selecting a few cities by the Nearest Neighbor algorithm, then we extend the tour by inserting the unselected cities into the tour using shortest path and insertion algorithms. We conducted computational experiments of the proposed algorithm on TSPLIB test problems. Experimental results show that the proposed algorithm is efficient compared to the Nearest Neighbor algorithm for solving The Symmetric Traveling Salesman Problem.

Keywords: symmetric traveling, salesman problem, hyper-heuristic algorithm, nearest neighbor algorithm, insertion algorithm, shortest path algorithm.

AMS Subject Classification: 90C27, 90C59.

1. Introduction

The Traveling Salesman Problem (TSP) is a well-known and extensively studied problem in the field of combinatorial optimization. Since it is an NP-hard problem, many heuristic and meta-heuristic algorithms have been developed to solve it [1].

This problem has many applications. Some of them are [1-3]:

- Drilling of printed circuit boards
- Overhauling gas turbine engines
- X-ray crystallography
- Computer wiring
- The order-picking problem in warehouses
- Vehicle routing

There are different variations of the TSP such as asymmetric and symmetric TSP, the generalized TSP and travelling purchaser problem. In this paper, we focused on only The Symmetric Traveling Salesman Problem (STSP).

The TSP can be defined on a graph $G = (V, A)$, where $V = \{v_1, \ldots, v_n\}$ is a set of n vertices (cities) and $A = \{(v_i, v_j) | v_i, v_j \in V, i \neq j\}$ is a set of arcs, together with a nonnegative cost (or distance, weight) matrix $C = (c_{ij})$ associated with A. The problem is considered to be symmetric (STSP) if $c_{ij} = c_{ji}$ for all $(v_i, v_j) \in A$, and asymmetric (ATSP) otherwise. Elements of A are often called edges (rather than arcs) in the symmetric case. The TSP consists of determining a minimum cost Hamiltonian cycle (circuit), often simply called a tour.

Algorithms for solving the TSP may be divided into two classes: exact algorithms and heuristic algorithms [4-7]. Exact algorithms try all permutations and choose the tour with minimum cost. Dynamic programming, branch and bound, linear programming, cutting plane methods are exact methods with different complexities to solve the TSP problems but they all not applicable for large-scale problems.

Heuristics do not guarantee an optimal tour but propose approximate solutions. These heuristic algorithms can be divided into three classes: tour construction, tour improvement and composite algorithms [3].

Tour construction algorithms iteratively extend a connected partial tour or iteratively combine tour fragments of the best tour. The tour improvement algorithms start with a given tour and replace two links of the tour with other links to obtain a shorter tour. Composite algorithms include both construction and improvement models [1].

In this paper, a new method is proposed based on creating initially a tour using a few vertices and then adding new vertices to the tour.

2. Tour construction heuristics

Tour construction algorithms have one thing in common, they stop when a solution is found and never try to improve it.

In this paper, we focus on only two of these algorithms: Nearest neighbor and Insertion heuristics.

2.1 The nearest neighbor algorithm

This is perhaps the simplest and most straightforward heuristic for the TSP. The key to this algorithm is to always visit the nearest city [8].

Nearest Neighbor:
1. Select a random city.
2. Find the nearest unvisited city and go there.
3. Are there any unvisited cities left? If yes, repeat step 2.
4. Return to the first city.

2.2 Insertion heuristic

Insertion heuristics are quite straightforward, and there are many variants to choose from. The basics of insertion heuristics is to start with a tour of a subset of all cities, and then inserting the rest by some heuristic. The initial subtour is often a triangle or the convex hull of the cities. One can also start with a single edge as subtour [3].

Nearest Insertion:
1. Select the shortest edge, and make a subtour of it.
2. Select a city not in the subtour, having the shortest distance to any one of the cities in the subtour.
3. Find an edge in the subtour such that the cost of inserting the selected city between the edge's cities will be minimal.
4. Repeat step 2 until no more cities left.

2.3. Shortest path algorithm

Traditional shortest path algorithms such as Dijkstra's and Bellman-Ford algorithms have complexities $O(n^2)$ and $O(mn)$, respectively, where *n* is the number of vertices and m is the number edges [9]. The following proposed algorithm finds the shortest path between two vertices by trying the unused vertices and this part of the algorithm runs in $O(n)$ time.

3. A new method for solving the Symmetric TSP

The proposed method for solving the Symmetric TSP consists of three main parts.

3.1 The first part

In this part, a tour is constructed with selected vertices by the NN algorithm. To select the vertices, firstly a central vertex should be found.

In order to find a central vertex, firstly, the middle point with $x = \left(\sum_{i=1}^{n} x_i\right)\Big/n$ and $y = \left(\sum_{i=1}^{n} y_i\right)\Big/n$ coordinates should be calculated. Then, the nearest vertex to the middle point (x, y) among all vertices will be our central vertex. All edges incident to the central vertex are sorted in a list so that their weights are monotonically decreasing. The vertices other than the central vertex of the first *k* edges are selected, where *k* is an arbitrary parameter. Let *c* be a constant which equals the half of the weight of the first edge in the sorted list. Another *k* vertices other than the central vertex of the first *k* edges whose weights are less than or equal to *c* are selected. That is, totally 2*k*vertices are selected. Then a tour is constructed for the selected 2*k* vertices using the NN algorithm.

3.2 The second part

In the second part, a new tour is constructed by adding some vertices (if possible) to the tour which is formed by the NN algorithm in the first part. The longest edge is found. A shortest path connecting the end vertices of this edge excluding the edge itself is found. If the edges on the shortest path are smaller than the longest edge, then the longest edge is deleted and the tour is updated as this shortest path.

3.3 The third part

In the last part, if there exist vertices which are not added to the tour, these vertices are added to the tour using insertion algorithm. Finally, we have a tour containing all vertices.

The algorithm of the method proposed above is as follows:

Step 1. Find the central vertex on the graph.

Step 2. Find 2*k* vertices as described in Section 3.1. Then add these 2*k* vertices to the selected list.

Step 3. Construct a tour using the NN algorithm with selected vertices.

Step 4. If there are no unused vertices go to Step 10.

Step 5. Find the longest edge in the tour constructed in Step 3.

Step 6. Find the shortest path connecting the end vertices of this edge excluding the edge itself.

Step 7. If the edges on the shortest path are smaller than the longest edge, then delete the longest edge, and add edges, vertices on the shortest path to the selected list and update the tour.

Step 8. Otherwise, find the next longest edge in the constructed tour and go to Step 6. If there is no any edge to be selected, then go to Step 9.

Step 9. If there are unused vertices, then add these vertices to the tour using insertion algorithm.

Step 10. Stop.

We investigate the time complexity of the algorithm given above as follows: Step 1 requires $O(n)$ time. Step 2 is implemented by a well-known sorting algorithm called quicksort which runs in $O(n \lg n)$. The NN algorithm in Step 3 has a running time $O(k^2)$ to form a tour using 2*k* vertices. Step 5 takes linear time. Since the test problems are complete graphs with Euclidian edge costs, the shortest path between two vertices v_1 and v_2 is the edge incident to these vertices, i.e., (v_1, v_2). However, we find the shortest path in Step 6 without using (v_1, v_2). This can be achieved by trying all the other vertices to form a path from v_1 and v_2 and finding the minimum one. So, Step 6 is handled in $O(n)$ time which is linear. Step 7 takes constant time. In Step 8, if there are unselected edges, the next longest edge can be found in $O(k)$ time and we go to Step 6 which means a loop. This loop is bounded by $O(k)$.The last two steps are implemented in constant time. Thus, the overall running time of the algorithm is $O(n^2)$ which is polynomial.

4. Computational tests of the proposed method for the TSP

To test the effectiveness of the method, 14 TSP problems taken in [10] are examined for benchmarking.

In Table 1, first column includes the name of the test instances having from 51 to 783 cities in TSPLIB. Second column shows the optimal solutions. The third column shows the solutions obtained by the NN algorithm. The other columns except the last two ones shows the solutions using the proposed method for the different values of the parameter *k*. Shaded (grey) areas in Table 1 are the best solutions found by the proposed method. Gap1 indicates the relative error between the solution obtained by the NN algorithm and the optimal solution. Gap2 indicates the relative error between our best solution and the optimal solution.

Table 1. Experimental results

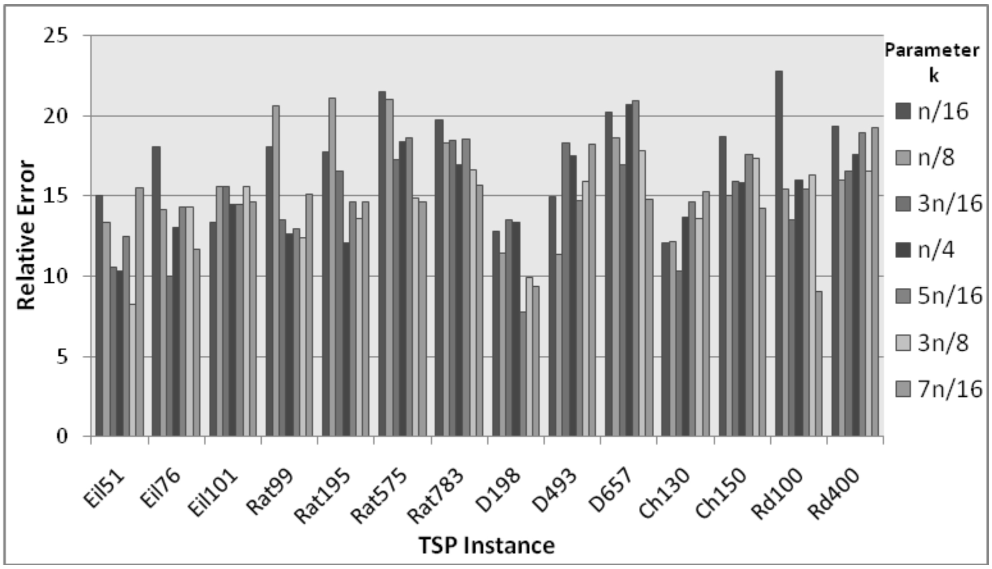
TSP Instances	**Optimum**	**NN**	**Paramater *k***							**Gap1 (%)**	**Gap2 (%)**
			$\left\lfloor \frac{n}{16} \right\rfloor$	$\left\lfloor \frac{n}{8} \right\rfloor$	$\left\lfloor \frac{3n}{16} \right\rfloor$	$\left\lfloor \frac{n}{4} \right\rfloor$	$\left\lfloor \frac{5n}{16} \right\rfloor$	$\left\lfloor \frac{3n}{8} \right\rfloor$	$\left\lfloor \frac{7n}{16} \right\rfloor$		
Eil51	426	514	490	483	471	470	479	461	492	20,7	8,2
Eil76	538	712	635	614	592	608	615	615	601	32,3	10,0
Rat99	1211	1565	1430	1461	1375	1364	1368	1361	1394	29,2	12,4
Rd100	7910	9941	9715	9133	8976	9174	9128	9197	8626	25,7	9,1
Eil101	629	825	713	727	727	720	720	727	721	31,2	13,4
Ch130	6110	7575	6847	6853	6740	6944	7006	6941	7041	24,0	10,3
Ch150	6528	8195	7747	7510	7565	7561	7677	7659	7455	25,5	14,2
Rat195	2323	2762	2735	2814	2708	2604	2662	2638	2662	18,9	12,1
D198	15780	18655	17805	17583	17912	17887	17008	17343	17259	18,2	7,8
Rd400	15281	19168	18239	17728	17809	17967	18172	17806	18229	25,4	16,0
D493	35002	43646	40241	38988	41406	41142	40145	40560	41382	24,7	11,4
Rat575	6773	8449	8227	8195	7943	8021	8037	7781	7763	24,7	14,6
D657	48912	61874	58788	58023	57197	59045	59155	57646	56134	26,5	14,8
Rat783	8806	11255	10548	10418	10431	10297	10437	10269	10189	27,8	15,7

Relative error is calculated as follows:

$$Gap = \frac{best\ \ solution - optimal\ \ solution}{optimal\ \ solution} \times 100 \cdot$$

It can be seen in Table 1 that the proposed algorithm gives better results compared to the NN algorithm.

Figure 1. Graphical representation of the experimental results



5. Conclusion

In this study, a new hyper-heuristic method is proposed for The Symmetric Traveling Salesman Problem. In the proposed method, the NN algorithm and Insertion heuristic are also used. Proposed method is compared with the NN algorithm. The experiment results show the new algorithm is effective for solving the STSP.

In Figure 1, the relative errors between the optimal solution and the best solution using the various values of the parameter *k* are illustrated for each of 14 TSP instances. One can see that there is no a constant value of *k* to obtain the best solution for each problem. To find the best result, algorithm must be applied for different values of the parameter *k*, and the minimum one should be selected among them. The method for solving the TSP presented in this paper is relatively simple and useful.

References

1. Appligate D.L., Bixby R.E., Chavatal V. and Cook, W.J., The Travelling Salesman Problem, A Computational Study, Princeton University Press, Princeton and Oxford, 2006, 593p.

2. Punnen, A., The Traveling Salesman Problem: Applications, Formulations and Variations, The Traveling Salesman and Its Variations, Gutin and Punnen (eds), Kluwer Academic Publishers, Netherlands, 2002, pp.1-28.
3. Rajesh M., Mittal M.L., and Singh S., Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches, Traveling Salesman Problem, Theory and Applications, Davendra D. (eds), INTECH Open Access Publisher, 2010, pp.1-24.
4. Kızılateş G., Nuriyeva F., A parametric hybrid method for the traveling salesman problem, Mathematical and Computational Applications, Vol.18 No.3, 2013, pp.459-466.
5. Nuriyeva F., Kızılateş G., Berbeler M.E., Improvements on heuristic algorithms for solving traveling salesman problem, International Journal of Advanced Research in Computer Science, Vol. 4 No. 11, 2013, pp.1-8.
6. Akay B., Aydogan E., and Karacan L., 2-opt based artificial bee colony algorithm for solving traveling salesman problem, Information Technology & Computer Science , 2012, pp.666-672.
7. Johnson, D.S., and Mc Geoch, L.A., Experimental analysis of heuristics for the STSP, The Traveling Salesman and Its Variations, Gutin and Punnen (eds), Kluwer Academic Publishers, 2002, pp. 369-443.
8. Marinakis Y., Migdalas A., Expanding neighbourhood gras for the traveling salesman problem, Computational Optimization and Applications Vol. 32 No. 3, 2005, pp.231-257.
9. Cormen T.H., Leiserson C.E., Rivest R.L. and Stein, C., Introduction To Algorithms, The MIT Press, Cambridge, 2009, 1180.http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Tacir məsələsi üçün tur genişlədən Hiperevristik alqoritm

Gözde Kızılateş, Fidan Nuriyeva, Hakan Kutucu

XÜLASƏ

Simmetrik Tacir məsələsi üçün hiperevristik bir alqoritm təklif edilmişdir. Bu üsulda öncə az sayda şəhər seçilərək "Ən yaxın qonşu" alqoritmi ilə bu şəhərlərdən keçən bir devrə müəyyənləşdirilir, sonra isə "Ən qısa yol" və "Uyğun şəhər əlavə etmə" alqoritmlərindən istifadə edilərək bu devrə bütün şəhərlərdən keçən tura genişlədilir. Təklif edilmiş alqoritm ilə test kitabxanası məsələləri üzərində hesablama eksperimentləri aparılaraq "Ən yaxın qonşu" alqoritmi ilə müqayisə edilmişdir. Hesablama eksperimentlərinin nəticələri təklif edilmiş alqoritmin effektiv olduğunu göstərir.

Açar sözlər: simmetrik tacir məsələsi, hiperevristik alqoritm, "ən yaxın qonşu" alqoritmi, "uyğun şəhər əlavə etmə" alqoritmi, "ən qısa yol" alqoritmi.

Расширяющий тур гипер эвристический алгоритм для решения задачи коммивояжера

Гозде Кызылатеш, Фидан Нуриева, Хакан Кутуджу

РЕЗЮМЕ

Предложен гиперэвристический алгоритм для симметричной задачи коммивояжера. Этим методом сначала с использованием алгоритма “ближайший сосед” определяется цикл, проходящий через определенные города, потом этот цикл с применением алгоритмов “кратчайший путь между парами вершин” и “в наилучший из подходящих” расширяется до тура проходящего через всего города. Проведены вычислительные эксперименты с предложенным алгоритмом над тестовыми библиотечными задачами и приведены сравнения с алгоритмом “ближайший сосед”. Результаты вычислительных экспериментов показывают эффективность предложенного алгоритма.

Ключевые слова: симметричная задача коммивояжера, гиперэвристический алгоритм, “ближайший сосед” алгоритм, “в наилучший из подходящих” алгоритм “кратчайший путь между парами вершин”.