International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Approximate Solution for NP Complete Problem

Sumit Sahu[1], Makarand Nale [2], Mohit More[3]

[1] IT (Networking)
VIT University, Tamil Nadu
sumitsahu7211@gmail.com

[2] IT (Networking)
VIT University, Tamil Nadu
makarand.nale1990@gmail.com

[3] IT (Networking)
VIT University, Tamil Nadu
mohit.more9494@gmail.com

Abstract: ***We all know the traveling salesman problem is an optimization problem. In deterministic time, we can find the optimal solutions to the problem through linear programming. However, the TSP is NP Complete problem, it will be very time consuming to solve big scale problem with guaranteed optimality. Setting optimality aside, there is a bunch of algorithms offering comparably fast running time and still yielding near optimal solutions, the proposed algorithm will give approximate solution not more than twice of the optimal solution; this makes sense because it gives the upper bound to an algorithm.***

Keywords: NP complete, greedy algorithm, heuristic function.

1. Introduction

In the traveling salesman problem (TSP), which is closely related to the Hamiltonian cycle problem, a sales person must traverse n nodes or cities. Making the problem like a complete graph with n nodes, we can also say that the sales person is trying to make a tour, or we can also say that a Hamiltonian cycle, visiting each city exactly once and ending at the city where he starts from.

The sales person takes a nonnegative integer cost c(i,k) for traveling from city i to city k, and the salesman wishes to make the tour whose total cost is minimum, where the total cost is the sum of the sum of the individual costs along the edges of the tour.

There are so many ways to finding the optimal length of the TSP instance. The one we are using here is approximate algorithm technique. By using approximation algorithm we are making it run in a polynomial time.

2. Approximate Algorithm

An algorithm that returns near to the optimal solutions is said to be an approximation algorithm. It depends on the problem, what is the problem we have, we may also define an optimal solution as one with maximum possible cost or one with minimum possible cost, that is the problem can be either from maximization or from minimization problem. We say that an algorithm for a problem has an approximation ratio of ρ(n) if, consider for any input of size n, a cost C of the solution produced by the algorithm is under the factor of ρ(n) of the cost C* of an optimal solution:

$$\max(c/c^*, c^*/c) <= \rho(n).$$

If an algorithm achieves an approximate ratio of ρ(n), we can say it a ρ(n) approximation algorithm.

3. Making Tour

3.1 Greedy Algorithm

A greedy algorithm always makes the choice that looks best at the instance. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.

Greedy O(n^2 log(n))

- Start arbitrary vertex as a root vertex.
- Start selecting the vertex, which is optimal at that time.
- Repeat step 2 until all vertices are visited exactly once.

3.2 Closest Point Heuristic

Closest point heuristic routine is clear. The essential thought of this is to begin with unimportant cycle comprising of a discretionarily picked vertex. At each one stage, recognize the vertex u that is not on the cycle that is closest u is vertex v. Amplify the cycle to incorporate u by embeddings u just after v. Rehash until all vertices are on the cycle. Closest point heuristic O(n^2)

- Start with any cycle in the graph and choose any vertex.
- Identify the alternate vertex which is nearest to choose vertices on the graph.
- Add this selected vertex to the cycle.
- Repeat steps 2 and 3 until all vertices are on the cycle.

3.3 Christofides' algorithm

Most heuristics can only guarantee a worst-case ratio of 2. Professor NicosChristofides extended one of these algorithms and concluded that the worst-case ratio of that extended algorithm might be 3/2. This algorithm is

commonly known as Christofides heuristic. Original Algorithm (Double Minimum Spanning Tree), worst-case ratio 2, O(n2log2(n))

- Build a minimal spanning tree (MST) from the set of all cities.
- Duplicate all edges; we can now easily construct an Euler cycle.
- Traverse the cycle, but do not visit any node more than once, taking shortcuts when a node has been visited.

3.4 Christofides' algorithm O(n3)

- Make a minimal spanning tree from the set of all cities.
- Create a minimum-weight matching on the set of nodes having an odd degree. Add the MST together with the MWM.
- Make an Euler cycle from the combined graph, and traverse it taking shortcuts to avoid visited nodes[1].

The main difference is the additional MWM calculation. This part is also the most time consuming one, having a time complexity of $O(n3)$[2]. Tests have shown that Christofides algorithm tends to place itself around 10% above he Held-Karp lower bound. For more information on tour construction heuristics see [2].

4. Detailed Problem Definition

The most compelling reason why theoretical computer scientists believe that P $\neq$ NP comes from the existence of the class of "NP-complete" problem. This class has the intriguing property that if any NP-complete problem can be solved in polynomial time then every problem in NP has a polynomial time solution , that is, P = NP Despite decades of study, though, we do not have any polynomial time algorithm has ever been discovered for any NP-complete problem.

The problem with traveling salesman problem is whenever the node increases its growth of complexity also increases exponentially. For small input of a graph it will not affect anymore.

5. Solution Methodology

This is an optimization problem in which each potential solution has a positive cost, and we wish to find a near-optimal solution.

[3]Many problem of practical significance are NP-complete, yet they are too important to abandon merely because we do not know how to find an optimal solution in polynomial time. Even if a problem is NP-completeness. First and foremost, if the real inputs are little, a calculation with exponential running time may be superbly palatable. Second, we may be able to isolate important special cases that we can solve in polynomial time. Third, we might come up with approaches to find near-optimal solutions in polynomial time. Close optimality is regularly sufficient. We call a calculation that returns close ideal arrangements an estimated calculation. Depending on the problem, we may define an optimal solution as one with maximum possible cost; that is, the problem may maximization or minimization problem.

We say that an algorithm for a problem has an approximation ratio of ρ(n) if, consider for any input of size n, a cost C of the solution produced by the algorithm is under the factor of ρ(n) of the cost C* of an optimal solution:

$$\max\left(\frac{C}{C*},\frac{C*}{C}\right) <= \rho(n)$$

If an algorithm achieves an approximation ratio of $\rho(n)$,we call it a $\rho(n)$- approximation algorithm. The definitions of the approximation ratio and of a $\rho(n)$-approximation algorithm apply in the problems. For a maximization problem, 0<C$\leq$C*, and the ratio C*/C gives the factor by which the cost of an optimal solution is larger than the cost of the estimated arrangement. Likewise, for a minimization issue, 0 < C* ≤ C, and the degree C/C* gives the variable by which the expense of the inexact solution is larger than the cost of an optimal solution. The approximation ratio of an approximation algorithm will never less than 1, since C/C* $\leq$ 1 implies C*/C $\geq$ 1. Therefore, a 1-approximation algorithm produces an optimal solution, and an approximation algorithm with a large approximation algorithm produces an optimal solution, and an approximation algorithm with a large approximation ratio may return a solution that is much worse than optimal.

[3]In traveling salesman problem we have complete undirected graph G = (V,E) that has a non-negative integer cost c(u,v) associated with each edge C(u,v) $\in$E, and we must find a Hamiltonian cycle (a tour) of G with minimum cost. As an expansion of our documentation, let C(a) signify the aggregate expense of the edges in the subset A subset E:

$$C(A) = \sum_{(u,v)\in A} C(u,v).$$

In many practical situations, the least costly way to go from a place u to a place w is to go directly, without any intermediate steps. Put another way, cutting out an intermediate stop never increases the cost. We can make a notion by saying that the cost function c satisfies the triangle inequality if, for all vertices u,v,w$\in$ V, C(u,w)$\leq$C(u,v) + C(v,w).

6. Proposed Work

After working such a long time, we got that if we are able to reduce the time complexity of the minimum spanning tree will greatly affect the complexity of the TSP.

So we are thinking to make such an algorithm which is to be heuristic, so find minimum spanning tree with a heuristic function.

Algorithm is as follows:

- Select any of a vertex in the graph as a root vertex.
- Compute minimum spanning tree with heuristic function.
- In this tree apply preorder tree walk, and make the list of vertices according to the order in which vertices are visited.
- Make a Hamiltonian cycle and return it.

7. Result, Analysis and Discussion

Even if we take simple implementation of minimum spanning tree, the running time our algorithm gives O(n^2). Now we are showing that if the cost of the function for an instance of the traveling salesman problem satisfies the triangle inequality. Then our returns a cycle or tour whose cost is not more than twice the cost of an optimal cycle.

Just now we have seen that this approximation algorithm takes polynomial time. Let Hamil* denote an optimal tour for the given set of vertices. We obtain a spanning tree by deleting any edge from a tour, and each edge cost should be non negative. Therefore, the weight of the minimum spanning tree T computed in line 2 of our approximate TSP tour gives a lower bound on the cost of an optimal tour

$$C(T) \leq C(Hamil^*)$$

A entire walk of T lists the vertices when they are first visited and also whenever they are returned to after a visit to a sub tree. Let us call his entire walk as full walk W.

As the full walk traverses every edge of exactly twice,

$$C(W) \leq C2C(Hamil^*)$$

So the cost of W is within a factor of 2 of the cost of an optimal tour. By triangle inequality, we can delete a visit to any vertex from W and the cost does not increase. The ordering would always be same as obtained by pre order walk. Let Hamil be the cycle, since every vertex is visited by once, and it is the cycle computed by our algorithm. Since Hamil is obtained by deleting vertices from full walk W, we have

$$C(Hamil) \leq C(W)$$

After combining inequalities it gives

$$C(Hamil)) \leq 2C(Hamil^*).$$

8. Conclusion

This algorithm will not give the optimal solution; however, it will give the upper bound of an algorithm. This algorithm will give the complexity under twice the optimal solution.

References

[1] chrni794@student.liu.se ”Heuristics for the Traveling Salesman Problem”.
[2] D.S. Johnson and McGeoch, “The Traveling Salesman Problem A Case Study Local Optimization”, November 20, 1995.
[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms.