ISSN: 0975-766X
CODEN: IJPTFI
Available Online through
Research Article
www.ijptonline.com

COMPARATIVE STUDY OF ALGORITHMS TO SOLVE TRAVELLING SALESMAN PROBLEM

Rajat Khandelwal[1] , Vijayan Ellappan[2], Ajay[3], A.Hamsadhvani[4]
1 2 3 4 School of Information Technology and Engineering(SITE),VIT University, Vellore, Tamil Nadu, India.
Email: evijayan@vit.ac.in

Received on 25-10-2016 *Accepted on 02-11-2016*

Abstract

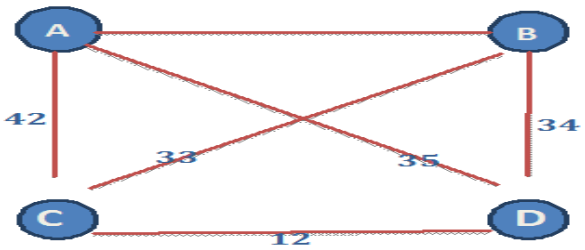
The Travelling Salesman Problem (TSP) is commonly used in the field of combinatorial optimization. As it is an NP-complete problem, there has not been any efficient method to solve the problem and to provide the best results. Many different algorithms can be used to solve the travelling salesman problem. Some algorithms might give an optimal solution whereas others may produce results nearing to the optimal solution. This paper presents the comparative study of different algorithms based on the given problem we arrive at the best optimum solution.

Introduction

Graphs are mathematical structures used to model pair-wise relations between objects. Graph theory can be used to solve problems in branches of Mathematics, Computer Science, and other scientific areas. We can make model an enormous number of real world systems and phenomena using graphs. There are many different problems in graph theory that have attracted much attention. One such problem is the Travelling Salesman Problem (TSP), which refers to a salesman who wants to find a shortest possible tour that visits every city exactly once and returns back to the city from which he started. In terms of graph theory, given a list of nodes (cities) and their pair-wise distance, the task is to find the shortest possible tour that visits every node exactly once. Since the start node and the end nodes are the same the tour creates a Hamiltonian cycle, which is a cycle in the graph that visits each vertex exactly once.

Literature Survey: Travelling salesperson problem is NP-Complete problem so there is no optimized solution for it. Every algorithm has its own pros and cons. A straightforward approach, usually based directly on the problem's statement and definitions of the concepts involved. A brute force solution to a problem involving search for an element with a special property, usually among combinatorial objects such as permutations, combinations, or subsets of a set.

Brute force is a sequential search algorithm. We can use Brute force algorithm to find the minimum cost Hamiltonian circuits. In Travelling salesman problem a salesman need to visit all the cities but without back traversal and return back to the home at the end of the day. For that we are going to apply brute force algorithm so first we need to find the all possible root for given nodes using Brute force algorithm where no back traversal and then the shortest root in all the roots would be our final result.

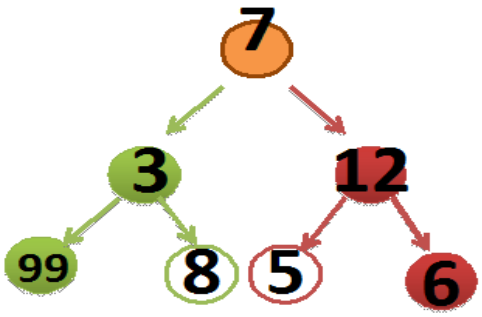


The Brute-Force-Algorithm is optimal: it's guaranteed to find a solution. The algorithm has to find all the (N-1)! Hamilton circuits. For that brute force has to find all possible roots that can take a long time so that's the reason Brute-Force-Algorithm is optimal because it provides optimal solution but it is inefficient algorithm.

- First make all the possibility roots.
- Then calculate the weight of each root by adding up the weights of its edges.
- Finally choose the root with the smallest total weight.

We try to look all the possibility and Brute force always gives us optimal output so we called as Brute force is an optimal algorithm because Brute force algorithm always gives a best root to visit with the shortest distance but not an efficient algorithm.

Greedy-Algorithm is simple to understand and quick to processing the result ant easy to implement. It uses to take the locally optimal decision. Greedy-Algorithm makes the best choice at each decision without looking ahead. Greedy algorithm will not give the optimum solution it just make locally optimal decision.



Greedy-Algorithm: 7+12+6 = 25

Actual long path: 7+3+99 = 109

In the travelling salesman problem which is of a high computational complexity. Using Greedy-Algorithm, at each stage visit an unvisited node (city) nearest to the current city. The problem with the Greedy-Algorithm is that it fails to provide the globally optimal solution.

Greedy is a strategy that works wellon optimization problem with some of following characteristics:

- **Greedy-Choice Property**: A global optimum can be arrived by selecting the local optimum decision.
- **Optimal substructure:** An optimal solution for the problem that contains an optimal solution to the sub problem.

  In Greedy-Algorithm, in this Algorithm activities are sorted first according to their completion timing, from the earliest to the latest.

Greedy-Algorithm follows the routing technique as well. Using Greedy routing in Greedy-Algorithm a message would be forwarded to the nest neighbour node which one is closest to the destination node.

Greedy-Algorithm can be characterized as 'short sighted' and also as 'non-recoverable'. There are three types of variation with Greedy-Algorithm:

- Pure Greedy-Algorithms
- Orthogonal-Greedy-Algorithm
- Relaxed-Greedy-Algorithm

Minimum Spanning tree can be used to approximately solve the traveling salesman problem. A convenient formal way of defining this problem is to find the shortest path that visits each point at least once.

Note that if you have a path visiting all points exactly once, it's a special kind of tree. If we have a path visiting some vertices more than once, we can always drop some edges to get a tree. So in general the MST weight is less than the TSP weight, because it's a minimization over a strictly larger set.

On the other hand, if we draw a path tracing around the minimum spanning tree, we trace each edge twice and visit all points, so the TSP weight is less than twice the MST weight. Therefore this tour is within a factor of two of optimal. There is a more complicated wayof using minimum spanning trees to find a tour within a factor of 1.5 of optimal;

Kruskal's algorithm:

- sort the edges of G in increasing order by length
- keep a sub graph S of G, initially empty
- for each edge e in sorted order
- if the endpoints of e are disconnected in S
- add e to S
- return S

Note that, whenever we add an edge (u,v), it's always the smallest connecting the part of S reachable from u with the rest of G, so by the lemma it must be part of the MST.

This algorithm is known as a *greedy algorithm*, because it chooses at each step the cheapest edge to add to S. We should be very careful when trying to use greedy algorithms to solve other problems, since it usually doesn't work. E.g. if we want to find a shortest path from a to b, it might be a bad idea to keep taking the shortest edges. The greedy idea only works in Kruskal's algorithm

Analysis: The line testing whether two endpoints are disconnected looks like it should be slow (linear time per iteration, or O (mn) total). But actually there are some complicated data structures that let us perform each test in close to constant time; this is known as the *union-find* problem .The slowest part turns out to be the sorting step, which takes O(m log n) time.

Prim's algorithm:

- letT be a single vertex x
- while (T has fewer than n vertices)

{

- find the smallest edge connecting T to G-T
- add it to T

}

Since each edge added is the smallest connecting T to G-T, the lemma we proved shows that we only add edges that should be part of the MST.

Again, it looks like the loop has a slow step in it. But again, some data structures can be used to speed this up. The idea is to use a heap to remember, for each vertex, the smallest edge connecting T with that vertex.

Prim with heaps:

- make a heap of values (vertex,edge,weight(edge))
- initially (v,-,infinity) for each vertex
- let tree T be empty
- while (T has fewer than n vertices)

{

- let (v,e,weight(e)) have the smallest weight in the heap
- remove (v,e,weight(e)) from the heap
- add v and e to T
- for each edge f=(u,v)
- if u is not already in T
- find value (u,g,weight(g)) in heap
- if weight(f) < weight(g)
- replace (u,g,weight(g)) with (u,f,weight(f))

}

Analysis:

We perform n steps in which we remove the smallest element in the heap, and at most 2m steps in which we examine an edge f= (u, v). For each of those steps, we might replace a value on the heap, reducing its weight. (You also have to find the right value on the heap, but that can be done easily enough by keeping a pointer from the vertices to the corresponding values.).

I haven't described how to reduce the weight of an element of a binary heap, but it's easy to do in O (log n) time. Alternately by using a more complicated data structure known as a Fibonacci heap, you can reduce the weight of an element in constant time. The result is a total time bound of O (m + n log n).

Genetic Algorithm was introduced in 1975 by John Holland and later in 1992 John Koza used genetic algorithm to optimize some problems which lead to introduction of Genetic programming. Initially LISP programming language was used to write GA programs.

Components of Genetic Algorithm -

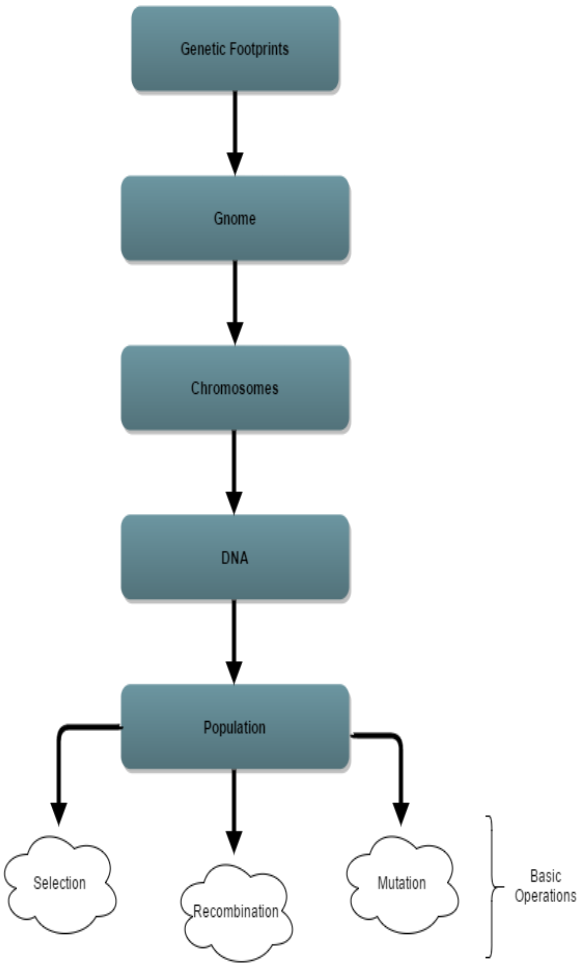


Figure 1: Components of Genetic Algorithm

Genome – Every organism on earth have some trait. These traits come from heredity, or they are generated by biological factors all these are stored or built up in genes. Genes are the basic building block of life.

Chromosome - All living beings on earth has a basic unit cell in there body. Each cell contains same set of chromosomes. Chromosomes are basically long strings of DNA.

Population – it consist of the entire individual, patterns or anything that is participating in optimizing the solution using genetic algorithm.

Operation of Genetic Algorithm:

1. Selection

2. Recombination
3. Mutation

How Genetic Algorithm can be used in solving Travelling Salesman Problem?

We know that as TSP is NP-Complete problem so there is no algorithm that can work for all the cases. Algorithm that are used to solve TSP have their pros and cons. Genetic algorithm find optimized solution by using Heuristic approach which is iterative in nature. It and generates solutions and optimizes this on the go. Due to this behaviour of genetic algorithm we get the results soon and the results are optimized over a period of time.

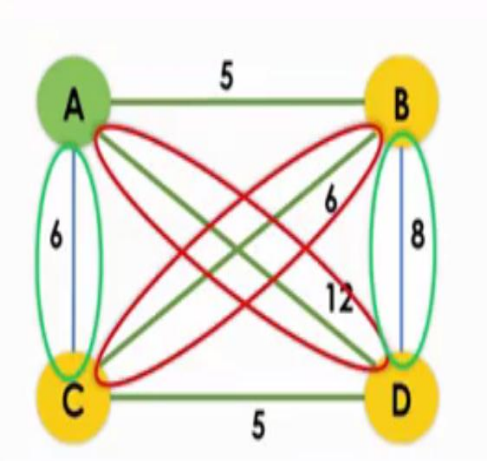


Figure 2

In Figure 2 we have a weighted graph with 4 nodes. Consider these nodes as 4 cities and weights between two cities as distance in K.M. Now if we use greedy algorithm we will get the shortest path as

A -> B -> C -> D -> A

Distance: 28 KM

Now in Heuristic based greedy algorithm we try and remove some edges from the graph in such a way that none of the node is isolated from the network. If we remove edge from A – D and B – C. Now if we traverse the graph using greedy algorithm we get the following path

A -> B -> D -> C -> A

Distance: 24

References

1. Adamchik, Victor. "Graph Theory". 2 April 2010.

http://www.cs.cmu.edu/~adamchik/2127/lectures/graphs_4_print.pdf

2. Arora S, Polynomial Time Approximation Schemes for Euclidian Traveling Salesman and Other GeometricProblems, Journal of the ACM., vol. 45, No. 5, pp.753-782, 1998.
3. Bentley J J, Fast Algorithms for Geometric Traveling Salesman Problem, ORSA J. Com-put., vol. 4, No. 4, pp.387-411, 1992.
4. Maredia A, History, Analysis and implementation of TSPUniversity of Houston-Downtown, pp. (1-40), Spring-2010.
5. Lin S, Computer Solutions for The Traveling Salesman Problem, Bell Syst. Tech J., vol. 44, pp. 2245-2269, 1965.
6. Boffey, T. Graph Theory in Operations Research. 1st ed. London and Basingstoke: The Macmillan Press Ltd,1982. pp. 148-185. 25 November 2009.
7. Saloni Gupta and PoonamPanwar, "Solving Travelling Salesman Problem Using Genetic Algorithm", International Journal of Advanced Research in Computer Science andSoftware Engineering, Volume 3, Issue 6, June 2013
8. VarshikaDwivedi, Taruna Chauhan, SanuSaxena and Princie Agrawal, "Travelling Salesman Problem using Genetic Algorithm", National Conference on Development of Reliable Information Systems, Techniques and Related Issues 2012. kumar, Karambir and Rajiv Kumar
9. Mouhammd Al kasassbeh, Ahmad Alabadleh and Tahsen Al-Ramadeen, "Shared Crossover Method for Solving Traveling Salesman Problem", IJICS Volume 1, Issue 6 September 2011
10. Coursera Course on data structure https://www.coursera.org/learn/advanced-data-structures/lecture/TO6DT/core-traveling-salesperson-problem-tsp
11. Distances/shortest paths between all pairs of vertices : doc.sagemath.org/html/en/reference/graphs/sage/graphs/distances_all_pairs.html
12. I. Gutman, Y.-N. Yeh, S.-L. Lee, and Y.-L. Luo.. Indian Journal of Chemistry, 32A:651–661, 1993.
13. Computing Many-to-Many Shortest Paths Using Highway Hierarchies Sebastian Knopp Peter Sanders Dominik Schultes Frank Schulz Dorothea Wagner epubs.siam.org/doi/pdf/10.1137/1.9781611972870.4.