Multiple Traveling Salesman Problem [DRAFT]

Navid Kooshkjalali

Autumn 2020

Abstract The multiple traveling salesman problem (mTSP) is a generalization of the famous traveling salesman problem (TSP), where more than one salesman is allowed to be used in the solution. While there is a considerable body of literature surrounding the TSP and variants of it like the vehicle routing problem (VRP), such as branch and bound[1], cutting planes[2], 2-opt[3], particle swarm[4], simulated annealing[5], ant colony[6, 7], neural network[8], tabu search[9], and genetic algorithms [13, 10, 11, 12], mTSP which seems to have various real-life applications is not yet researched thoroughly. The purpose of this survey is to apply a Genetic Algorithm (GA), a stochastic derivative-free method which produces a feasible solution within reasonable time, to solve the mTSP in Scala. Genetic algorithms are evolutionary techniques of optimizing according to survival of the fittest proposed by Holland[13]. They are useful algorithms for NP-Hard problems. The genetic algorithm depends on the selection criteria and the types and proportions of crossover and mutation operators in each generation.

1 Introduction

1.1 Problem Statement

1.1.1 TSP

Given a set of nodes, let there be a salesman located at a single depot node. The remaining nodes (cities) that are to be visited are called intermediate nodes. Then the TSP is finding the tour that starts and ends at the depot, such that such that each intermediate node is visited exactly once and the total cost along that tour is minimized. This is equivalent to finding the least weight Hamiltonian cycle in a complete weighted graph. Since mTSP is an extention of TSP all mTSP solutions are also valid TSP solutions. TSP was documented by Euler in 1759, whose interest was in solving the knight's tour problem. It is the fundamental problem in the fields of computer science, engineering, operations research, discrete mathematics, graph theory, and so forth. In 1972, Richard M. Karp showed that the Hamiltonian cycle problem was NP-Complete, which implies the NP-Hardness of TSP.

1.1.2 mTSP

Given a set of nodes, let there be m salesmen located at a single depot node. The remaining nodes (cities) that are to be visited are called intermediate nodes. Then, the mTSP consists of finding tours for all m salesmen, who all start and end at the depot, such that each intermediate node is visited exactly once and the total cost of visiting all nodes is minimized. This is a relaxation of the VRP, where the capacity restrictions are removed. This means that solutions for the VRP are also applicable to mTSP by giving sufficiently large capacities to the salesmen. However the scope of this paper will be limited to mTSP and solutions for the VRP will not be discussed.

1.2 Exact Solutions

Exact solutions are given by exact algorithms, which always solve an optimization problem to optimality. Unless P=NP, an exact algorithm for an NP-Hard optimation problem cannot run in worst-case polynomial time. In the case of TSP the only work when the number of cities is relatively small. They begin to take progressively longer as the size of the input grows such that using them is functionally impractical.

1.3 Genetic Algorithms

Genetic Algorithms represent a computational model inspired by evolutionary sciences. Usually GAs represent optimization procedures in a binary search space. GAs are based on improving a set of solutions; a population. They then produce a successor population by mutations and recombinations of best solutions of the predecessor population. Thus at each iteration part of the current population is replaced with offspring of the fittest solutions, where the fitness of a solution has to be defined as a scalar measure. This

implies that the population figuratively evolves towards an optimal solution. Genetic algorithms use three class of rules at every iteration to produce the successor population.

- **Selection Rules** select the individual solutions, called parents that whose chromosomes will contribute to the next generation.
- **Crossover Rules** combine two parents to form children to form the next generation.
- **Mutation Rules** apply random changes to individual parents to form children.

A simple pure genetic algorithm consists of the following steps.

1. Create an initial population of K chromosomes.
2. . Evaluate the fitness of each chromosome.
3. Choose $\frac{K}{2}$ parents from the current population via proportional selection.
4. Randomly select two parents to create offspring using crossover operator.
5. Apply mutation operators for minor changes in the results.
6. Repeat steps 4 and 5 until all parents are selected and mated.
7. Replace old population of chromosomes with new one.
8. Evaluate the fitness of each chromosome in the new population.
9. Terminate if the number of generations meets some upper bound, otherwise go to step 3.

2 Traveling Salesman Problem

The decision version of TSP (where given a length K, the task is to decide whether the graph has a tour of at most K) belongs to the class of NP-Complete problems. Therefore the worst case running time of the TSP increases superpolynomially (but no more than exponentially) with increase in the number of cities.

TSP can be formulated as an **integer linear program**[14, 15, 16] with several known formulations, most notable of which are **the Miller-Tucker-Zemlin(MTZ)** and **the Dantzig–Fulkerson–Johnson (DFJ) formulation**.

Traditional approaches to solving TSP are similar to any NP-Hard problem.

2.1 Exact Aglorithms

Producing exact solutions using exact algorithms (algorithms that solve for optimality of the solution) can only work in reasonable time if the number of cities is small. The most direct of these solutions (the naive approach) is to try all possible permutations and find the cheapest by brute force search. The running time of such approach lies within a polynomial factor of $O(n!)$ where n is the number of cities. The following is a pseudocode for this approach (For implementation see **com.tsp.BruteForce.bruteForce**).

1. Let S be the set of all $(n-1)!$ possible path permutations.
2. Let T be the a mapping of each path of S to a tuple of that path and the total distance of it. $S \to T : \forall s_i \in S,\, \exists! t_i \in T | t_i = (s_i, \text{Cost}(s_i))$
3. Reduce T by taking a by taking the tuple with minimum distance.

By the application concurrency and streaming, one can significantly boost the running time of this approach. The following is a pseudocode for this approach (For implementation see **com.tsp.BruteForce.bruteForceWithStreams**).

1. Let S be the set of all $(n-1)!$ possible path permutations.
2. Let $G = \{X_0, X_1, ..., X_k\}$ such that X_i are subsets of S and $X_0 \cup X_1 \cup ... \cup X_k = S$ and $\forall X_i$ where $0 \leq i < k$, $|X_i| = const$. Tuning this constant based on the hardware can optimize for spatial locality of reference, giving a speed boost thank to caching.
3. Create a thread pool and implement a stream with the following setup:
   - **Source**: G
   - **Flow**: Asynchronously and unordered, m
   - **Sink**:

Despite these efforts such a solution will become impractical even when the number of cities is as small 20. A typical way to improve such brute force approaches is the usage of dynamic programming and memoization.

One of the earliest dynamic programming applications of TSP is the Held-Karp algorithm. This algorithm offers faster (but still exponential time) execution than exhaustive enumeration, with the disadvantage using a lot of space: the worst-case complexity of this algorithm is $O(2^n n^2)$ in time and $O(2^n n)$ in space. Held-Karp takes advantage of the property in TSP that ***every subpath of a path of minimum distance is itself of minimum distance***. Therefore as opposed to the **top-down** naive approach of exhaustive enumeration, a **bottom-up** approach is used such that all the intermediate information required to solve the problem is evaluated only once, and them memoized so that it's value can be later read in constant time. Starting from the smallest sub-path, and building larger paths, using the information previously evaluated. To summerize the Held-Karp algorithm saves time by eliminating redundant evaluation of information necessary to solve the problem.

References

[1] G. Finke, A. Claus, and E. Gunn, "A two-commodity network flow approach to the traveling salesman problem," vol. 41, pp. 167–178.

[2] P. Miliotis, "Using cutting planes to solve the symmetric Travelling Salesman problem," Mathematical Programming, vol. 15, no. 1, pp. 177–188, 1978.

[3] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," Operations Research, vol. 21, pp. 498–516, 1973.

[4] J. Kennedy, R. C. Eberhart, and Y. Shi, Swarm intelligence, morgan kaufmann publishers, Inc., San Francisco, CA, USA, 2001.

[5] S. Kirkpatrick and G. Toulouse, "Configuration space analysis of travelling salesman problems," Le Journal de Physique, vol. 46, no. 8, pp. 1277–1292, 1985.

[6] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 53–66, 1997.

[7] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 26, no. 1, pp. 29–41, 1996.

[8] S. Bhide, N. John, and M. R. Kabuka, "A Boolean Neural Network Approach for the Traveling Salesman Problem," IEEE Transactions on Computers, vol. 42, no. 10, pp. 1271–1278, 1993.

[9] F. Glover, "Artificial intelligence, heuristic frameworks and tabu search," Managerial and Decision Economics, vol. 11, no. 5, pp. 365–375, 1990.

[10] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, New York, NY, USA, 1996.

[11] C. Moon, J. Kim, G. Choi, and Y. Seo, "An efficient genetic algorithm for the traveling salesman problem with precedence constraints," European Journal of Operational Research, vol. 140, no. 3, pp. 606–617, 2002.

[12] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," Annals of Operations Research, vol. 63, pp. 339–370, 1996.

[13] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, University of Michigan Press, Oxford, UK, 1975.

[14] Papadimitriou, C.H.; Steiglitz, K., Combinatorial optimization: algorithms and complexity, Mineola, NY: Dover, pp.308-309. 1998.

[15] Tucker, A. W., "On Directed Graphs and Integer Programs", IBM Mathematical research Project, Princeton University 1960.

[16] Dantzig, George B. (1963), Linear Programming and Extensions, Princeton, NJ: PrincetonUP, pp. 545–7, ISBN 0-691-08000-3, sixth printing, 1974.