A Comparative Study of Performance of A*algorithm and AO* algorithm for Solving Travelling Salesman Problem

Shraddha Ramteke
M.Tech Research Scholar, CSE
Chattrapati Shivaji Institute of Technology
Durg, India

Deepty Dubey
Associate Professor, CSE
Chattrapati Shivaji Institute of Technology
Durg, India

Abstract- **According to travelling salesman problem (TSP) given a number of cities and the distances between each couple of cities, the aim is to find the smallest possible route that goes to each city exactly once and returns to the origin city i.e., find a least cost Hamiltonian cycle It is definitely an NP-hard problem, important in operations investigation and theoretical computer scientific discipline. In the theory involving computational complexity, the decision version in the TSP where, given any length L, the task is to decide whether the graph offers any tour shorter than L belongs to the class of NP-complete issues. Thus, it is possible which the worst-case running time for virtually every algorithm for the TSP increases superpolynomially or perhaps exponentially with the quantity of cities. Heuristic search is definitely an AI search technique that employs heuristic to its moves. Heuristic is a rule of thumb that probably leads into a solution. Heuristics play a major role in search strategies because of exponential nature of the most extremely problems. Heuristics help to reduce the quantity of alternatives from an exponential number into a polynomial number. In AI, heuristic search incorporates a general meaning, and an increasingly specialized technical meaning. Within a general sense, the term heuristic is utilized for any advice which is often effective, but just isn't guaranteed to work always. The major aim of this proposed research work is to analyze and compare the performance of A* and AO* heuristic search algorithms for solving TSP on the basis of providing the optimal path according to Computational Time and runtime Memory Consumption**.

Keywords—A* algorithm, AO algorithm*, Travelling Salesman Problem (TSP).

I. INTRODUCTION

Travelling Salesman Problem is a combinational problem consisting of some cities and some edges connecting one city to other. TSP can be represented by a graph G (V, E), where V is the set of vertices (cities) and E is the set of edges(path) between each of the two vertices specific to the graph. TSP is to discover the shortest path in the graph G setting up a least cost Hamiltonian Cycle. If there exists a path between the two cities i and j, then the distance between these cities can be computed as-.

$$d_{ij} = \sqrt{(x_i - x_j)^2 - (y_i - y_j)^2}$$

TSP deals with real time scenario for a traveling salesman, where the most important for him to follow the shortest and optimal path having the minimum cost so as to gain maximum profit and to deliver the goods lesser time as much as possible. Path optimization is one of the major problems while travelling from the source to destination city visiting each city only once. Path can be calculated using many strategies but to decide the best possible strategy according to the number of cities is a difficult task.

TSP Heuristic: Whenever the salesman is at town i this individual chooses as their next city i. e., the city j which is why the c(i, j) charge, is the bare minimum among all c(i, k) charges, where k will be the pointers of the city the salesman has not visited yet. In case more than one city gives the particular minimum cost, the city with the smaller k is going to be chosen. This greedy algorithm selects the lowest priced visit in each step and won't care whether this will lead to a correct solution or not.

Input

Network formed from *n* cities
Cost c(i,j) of traveling from one city to next city, w*here* *i* & *j* = 1,2,3 . . , *n*.
Start with initial city.

Output

A least cost Hamiltonian cycle.

II. PROPOSED METHODOLOGY

A. *Applying TSP with A* Algorithm*

[1] A Star is typically the most popular choice for path finding, because it's reasonably flexible and works extremely well in an array of contexts. A Star is just like Dijkstra's algorithm as it enables you to find a speediest path. A Star is like Greedy Best-First-Search as it can make use of a heuristic to manual itself. In the simple case, it is as fast as Best-First-Search. The secret for its success is that it combines the information that Dijkstra's criteria uses (favoring vertices that are near to the starting point) as well as information that Best-First-Search uses (favoring vertices that are near to the goal). In the standard terminology used when speaking about A Star, g(n) represents the complete cost of the trail from the starting point to any vertex and, and h(n) presents the heuristic estimated cost from vertex n to the goal. Each time A Star chooses the vertex n that has the lowest f(n) = g(n) + h(n).

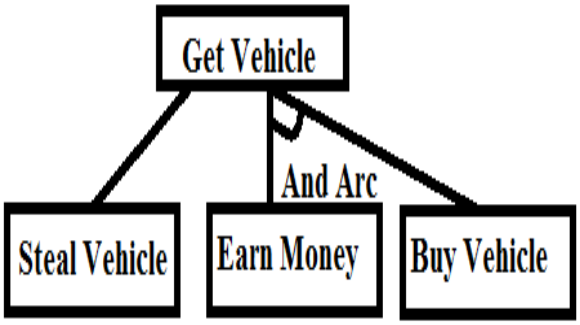
Algorithm-

Procedure Proposed A* algorithm for TSP
Initialize ClosedSet =: empty

Initialize ,OpenSet := start_node
Initialize FromNode := empty
Initialize CoveredNodes := set of all nodes in graph
Initialize g_score[start] := 0
Calculate
f_score[start] := g_score[start] + heuristic_cost_estimation
Loop Until OpenSet is not empty
Choose the current node from OpenSet with lowest f_score
If current node is not equal to start node **Then**
Reconstruct path from current_node
Remove current node from OpenSet
If current node not in covered nodes **Then**
Add current node to ClosedSet
For Each neighbour in neighbour nodes of current
If neighbour in ClosedSet or in Covered Nodes **then**
Continue loop;
Calculate tentative_score:= g_score_current + distance between current and neighbour
If neighbour not in OpenSet or tentative_score < g_score of current **Then**
Update FromNode with neighbour
Update g_score with tentative f_score
Update f_score = g_score + heuristic_cost_estimation
If neighbour not in OpenSet **Then**
Add neighbour to OpenSet
End For
End Loop
End

B. Applying TSP with AO Algorithm*

[2] When a problem can be split into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution. The decomposition of the problem or problem reduction generates AND arcs. [3] One AND are may point to any number of successor nodes. All these must be solved so that the arc will rise to many arcs, indicating several possible solutions. Hence the graph is known as AND - OR instead of AND. Figure shows an example for AND – OR tree.



And Or Graph Example

An algorithm to find a solution in an AND - OR graph must handle AND area appropriately. A Star algorithm cannot work with AND - OR graphs correctly and efficiently.

Algorithm-

Procedure Proposed AO* algorithm for TSP
Let G consists only to the node representing the initial state call this node INIT.
Calculate h' (INIT).
Until INIT is labeled SOLVED or h' (INIT) becomes greater than FUTILITY,
Repeat the following procedure.
Trace the marked arcs from INIT and select an unbounded node NODE.
Generate the successors of NODE. If there are no successors then assign FUTILITY as h' (NODE). This means that NODE is not solvable. If there are successors then for each one called SUCCESSOR, that is not also an ancestor of NODE do the following
(a) Add SUCCESSOR to graph G
(b) If successor is not a terminal node, mark it solved and assign zero to its h ' value.
(c) If successor is not a terminal node, compute it h' value.
Propagate the newly discovered information up the graph by doing the following. Let S be a set of nodes that have been marked SOLVED. Initialize S to NODE. Until S is empty repeat the following procedure;
(a) Select a node from S call if CURRENT and remove it from S.
(b) Compute h' of each of the arcs emerging from CURRENT, Assign minimum h' to CURRENT.
(c) Mark the minimum cost path as the best out of CURRENT.
(d) Mark CURRENT SOLVED if all of the nodes connected to it through the new marked are have been labeled SOLVED.
(e) If CURRENT has been marked SOLVED or its h ' has just changed, its new status must be propagate backwards up the graph. Hence all the ancestors of CURRENT are added to S.
End Loop
End Loop

III. EXPECTED OUTCOME

After the implementation of proposed research work, the expected outcome includes graphical analysis generated on the basis of the following:-
A. Computational Time: Time taken by both the algorithms for solving the Travelling Salesman Problem.
B. Memory Consumption: Runtime memory required by both the algorithms to solve Travelling Salesman Problem individually.

IV. CONCLUSION

Both the heuristic search algorithms A* and AO* are proposed for solving the traveling salesman problem. Theoretically, both the algorithms provide optimal solution to the problem according to the heuristics estimation. But for following the AO* approach, it is must that the selection of nodes for ANDing and ORing at particular level should so done so that it should obtain the unbounded node, avoiding backward propagation as possible and thus acquiring possible minimum time for solving the TSP. Also the FUTILITY value is so chosen in AO*, that it can lead to obtaining the solution with minimizing the complexities. Thus, both A* and AO* algorithms guarantees to provide the solution as the least cost Hamiltonian cycle for Travelling Salesman Problem.

REFERENCES

[1]. http://en.wikipedia.org/wiki/A*_search_algorithm
[2]. http://artificialintelligence-notes.blogspot.in/2010/07/problem-reduction-with-ao-algorithm.html
[3]. Artificial Intelligence, second edition, by Elaine Rich and Kevin Knight, published by Tata MacGraw-Hill Edition.
[4]. Introduction to Algorithms, second edition, by Thomas H. Coremen, Charles E. Leieserson, Ronald L. Rivest, Clifford Stein, published by Asoke Ghosh, Prentice-Hall of India Private Limited.
[5]. http://www.cs.cf.ac.uk/Dave/AI2/node26.html