**CSE 6140 Team Final Project – TSP Problem**

|  |  |  |  |
| --- | --- | --- | --- |
| Hung-Yi Li  hli603@gatech.edu |  |  |  |

**ABSTRACT**

…

**Keywords**

…

**1. Introduction**

…

**2. Branch and Bound Algorithm**

**2.1 Subproblem**

The subproblem of branch and bound is to determine the traveling order for the remaining nodes to minimize the traveling costs of visiting all the remaining nodes and going back to the starting node given the order of previously chosen nodes. In the initial stage, no node is selected. Any node can be picked as the starting node since in the end all nodes are connected as a cycle. Assume there are n nodes. After the first node 0 is chosen, there are (n – 1) node options for the next hop, (n - 2) for the hop next to the next hop, and so on until the last node. Then the final node should link back to the first node. The size of the solution space is (n – 1)!.

**2.2 Lower Bound Computation**

For the lower bound computation, three kinds of lower bound functions are adopted. First, the sum of the minimum cost of leaving every node. Second, the sum of the two shortest adjacent edges (incoming and outgoing) divided by 2 for a symmetric graph. Third, the sum of the smallest value for each row and column of the cost matrix, which is similar to the process of reduced matrix.

**2.3 How to Choose A Subproblem to Expand?**

Each iteration, the deepest branch with the lowest lower bound is processed to find the next hop with the lowest cost. The branch with lower bound updated after taking into account the next hop will be put back into the candidate pool again to be considered for the next iteration with the other branches. Every time a branch reaches to a leaf node, which means the length of nodes along the branch is equal to the number of all nodes, the upper bound could be updated if the overall cost of this branch is the minimum among all completed branches. Then the branches with the lower bound higher than this upper bound can be pruned. Also, the branch with the updated cost higher than the upper bound will not be put back into the candidate pool for consideration afterwards.

**2.4 How to Expand A Subproblem?**

As the deepest branch with lowest lower bound is picked, the costs will be computed for traveling next to each of the remaining nodes. The way of expanding a subproblem depends on the choice of lower bound function. For the first lower bound function – the sum of the minimum cost of leaving every node, the updated lower bound will be the original lower bound plus the true cost of traveling between the current node and the chosen next node minus the minimum cost of leaving current node. For the second lower bound function – the sum of the two shortest adjacent edges divided by 2, the updated lower bound will be the original lower bound plus the true cost of traveling between the current node and the chosen next node minus the sum of the second shortest edge of the current node and the shortest edge of the next node divided by 2. For the third lower bound function – the sum of the smallest value for each row and column of the cost matrix, the updated lower bound will be the original lower bound plus the true distance of traveling between the current node and the chosen next node plus the reduced costs of the current matrix.

**2.5 Pseudocode**

Input locations with x, y

For node in nodes:

For node in nodes:

Compute distance and put in matrix

Calculate lower bound with reduced matrix or the sum of shortest edges

Initiate min heap with node 0

While min heap:

Take current node from the top of min heap

For next node from current node:

Calculate updated lower bound

Put the branch back into min heap

If reaching the leaf 🡪 new solution is found:

Update the solution

Prune the solution tree by removing branches whose current lower bound   
 larger than the updated solution from min heap

Return solution

**2.6 Time and Space Complexity**

The worst time complexity is O(n^3 \* n!). For every implementation, there must be O(n^2) for the initial pair distance calculation. For reduced matrix cost, matrix reduction takes O(n^2) each. In total, there are (n – 1)! branches. For each branch, there are n steps to the leaf node. There are n possible options for the next hop. So it takes O(n^3 \* n!) in total.

For minimum edge cost or 2 shortest adjacent edges, it takes O(1) to calculate the updated lower bound. In total, there are (n – 1)! branches. For each branch, there are n steps to the leaf node. There are n possible options for the next hop. So it takes O(n \* n!) in total.

The worst space complexity is bounded by O(n^2 \* n!) for a distance matrix in each branch.

For minimum edge cost or 2 shortest adjacent edges, the space complexity is O(n \* n!) for the traveling order in each branch.

**2.7 Strengths and Weaknesses**

The strength of branch and bound algorithm is that it examines over all the possible solutions with pruning, so it is more efficient than pure brute-force. The weakness is that the whole solution space is still huge if there is a large number of nodes. But if time and memory allow, branch-and-bound would finally generate the optimal solution.

In our implementation, we found 2 shortest adjacent edges divided by 2 outperforms the other two lower bound functions in most cases. The reason could be that the calculation of updated lower bound of 2 shortest adjacent edges is faster than reduced matrix cost and that 2 shortest adjacent edges divided by 2 better presents the lower bound than minimum cost of leaving each node does. Therefore, adopting 2 shortest adjacent edges allows us to explore more branches than adopting reduced matrix cost and to prioritize more promising branches than adopting minimum cost of leaving each node.

To expedite the process to reach at least one leaf, we adopt depth-first-search instead of breadth-first-search. That is, the min heap sort the deepest branch first and then the lowest lower bound, so that the deepest branch will be processed earlier.

**2.8 Empirical Evaluation**

When the number of nodes is small enough (under 20), most lower bound functions work well to find the optimal solutions. The results tightly fit the optimal lower bound. As the number goes up, we found that the best solution under time constraint is far away from the optimal solution when the number of nodes increases.

An interesting finding is that the relative error is not proportional to the number of nodes. The reasons could be that the hand-picked starting node 0 has a significant influence on the following branch development and that the smaller node is ordered ahead in the min heap when two nodes have the same lower bounds. Some good paths starting with large costs might grow costs slowly in a later stage. Also, a better path could connect to a larger node in an early stage. Thus, if we are lucky enough, an better solution would appear before time-out, but still the algorithm would know the solution is optimal after all branches were examined or pruned.

The comprehensive tables of all lower bound function implementations are shown below by the order of (1) reduced matrix cost, (2) minimum cost of leaving each node, and finally (3) 2 shortest adjacent edges divided by 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Branch and Bound – Reduced Matrix Cost | | | | | |
| Dataset | Number of Nodes | Time (s) | Solution Quality | Optimal Quality | Relative Error |
| Cincinnati | 10 | 0.07 | 277952 | 277952 | 0.0000 |
| UKansasState | 10 | 0.26 | 62962 | 62962 | 0.0000 |
| Atlanta | 20 | 51.56 | 2003763 | 2003763 | 0.0000 |
| Philadelphia | 30 | 0.11 | 1773796 | 1395981 | 0.2706 |
| Boston | 40 | 175.39 | 1177460 | 893536 | 0.3178 |
| Berlin | 52 | 50.73 | 8972 | 7542 | 0.1896 |
| Champaign | 55 | 196.06 | 62591 | 52643 | 0.1890 |
| NYC | 68 | 376.83 | 1759917 | 1555060 | 0.1317 |
| Denver | 83 | 353.48 | 125963 | 100431 | 0.2542 |
| SanFrancisco | 99 | 12.95 | 1032713 | 810196 | 0.2746 |
| UMissouri | 106 | 212.31 | 175383 | 132709 | 0.3216 |
| Toronto | 109 | 169.30 | 1526770 | 1176151 | 0.2981 |
| Roanoke | 230 | 42.69 | 894260 | 655454 | 0.3643 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BnB – Minimum Cost of Leaving Each Node | | | | | |
| Dataset | Number of Nodes | Time (s) | Solution Quality | Optimal Quality | Relative Error |
| Cincinnati | 10 | 0.00 | 277952 | 277952 | 0.0000 |
| UKansasState | 10 | 0.01 | 62962 | 62962 | 0.0000 |
| Atlanta | 20 | 31.59 | 2003763 | 2003763 | 0.0000 |
| Philadelphia | 30 | 204.80 | 1494896 | 1395981 | 0.0709 |
| Boston | 40 | 538.33 | 1032691 | 893536 | 0.1557 |
| Berlin | 52 | 0.28 | 8947 | 7542 | 0.1863 |
| Champaign | 55 | 0.02 | 59299 | 52643 | 0.1264 |
| NYC | 68 | 0.15 | 1799063 | 1555060 | 0.1569 |
| Denver | 83 | 140.71 | 117713 | 100431 | 0.1721 |
| SanFrancisco | 99 | 438.56 | 1024069 | 810196 | 0.2640 |
| UMissouri | 106 | 19.30 | 148734 | 132709 | 0.1208 |
| Toronto | 109 | 41.72 | 1544025 | 1176151 | 0.3128 |
| Roanoke | 230 | 115.87 | 899136 | 655454 | 0.3718 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BnB – 2 Shortest Adjacent Edges Divided by 2 | | | | | |
| Dataset | Number of Nodes | Time (s) | Solution Quality | Optimal Quality | Relative Error |
| Cincinnati | 10 | 0.05 | 277952 | 277952 | 0.0000 |
| UKansasState | 10 | 0.02 | 62962 | 62962 | 0.0000 |
| Atlanta | 20 | 586.79 | 2003763 | 2003763 | 0.0000 |
| Philadelphia | 30 | 212.59 | 1461817 | 1395981 | 0.0472 |
| Boston | 40 | 383.56 | 937443 | 893536 | 0.0491 |
| Berlin | 52 | 112.7 | 8809 | 7542 | 0.1680 |
| Champaign | 55 | 128.13 | 58523 | 52643 | 0.1117 |
| NYC | 68 | 1.79 | 1812960 | 1555060 | 0.1658 |
| Denver | 83 | 242.53 | 120921 | 100431 | 0.2040 |
| SanFrancisco | 99 | 330.17 | 948100 | 810196 | 0.1702 |
| UMissouri | 106 | 40.06 | 161628 | 132709 | 0.2179 |
| Toronto | 109 | 374.49 | 1245760 | 1176151 | 0.0592 |
| Roanoke | 230 | 0.87 | 870656 | 655454 | 0.3283 |