Solving TSP with the branch and bound method using MPI

Piero Marini

July 29, 2021

1 Introduction

The traveling salesman problem consists in minimizing the distance traveled while visiting all nodes. This problem is easily visualized as a directed weighted graph, where the weights can be the distance between nodes. Nodes could be visualized as cities, like in the code provided. Each node can contain the x,y coordinates for example. For this paper, we are using the branch and bound method. This method consists in checking if the currently found path is better than the global best. This pairs well with parallel processing of course, which is the reason why we are implementing this approach using the MPI library.

2 Method

The branch and bound method first distributes the work (based on the amount of nodes) depending on the amount of available processors. On each processor, we assign the first node as the starting node and proceed doing a Depth-First Search to find the nearest node to it using their distance as the selected metric. Said distance is calculated using Euclidean distance with each node's X/Y coordinates. Following the branch and bound method, we can avoid continuing processing a current path based on the best solution found so far. We also update the best solution with a new one if it's distance traveled is less than the current best. We repeat this whole process until all nodes are visited and we get back to the starting node.

3 Results

The supplied program correctly solves the TSP problem using multiple cores thanks to the MPI library. Pairing branch and bound with multi-core processing gives a considerable speed up according to our tests.

Distritos	10	11	12	13	14	15
Tiempo(s)	0.8	4.5	24.4	70.5	105.3	355.3

4 Conclusions

The branch and bound method in itself doesn't gain performance versus the brute-force approach, since in the worst case scenario we won't be able to prune any nodes. Of course, using parallel processing greatly speeds up the processing for big inputs since we have multi-core processing. Each "branch" can be processed asynchronously on separate cores and then gathered in the master node to choose the path that best minimized the problem.