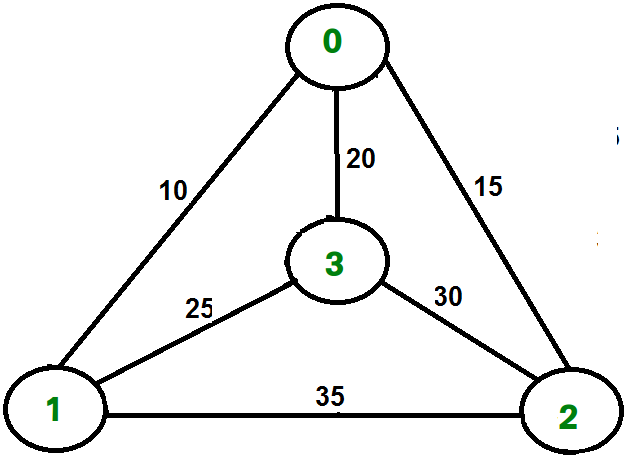
# **Traveling Salesman Problem using Branch And Bound**

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.

[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/TSP.png)

For example, consider the graph shown in figure on right side. A TSP tour in the graph is 0-1-3-2-0. The cost of the tour is 10+25+30+15 which is 80.

**Branch and Bound Solution**

As seen in the previous articles, in Branch and Bound method, for current node in tree, we compute a bound on best possible solution that we can get if we down this node. If the bound on best possible solution itself is worse than current best (best computed so far), then we ignore the subtree rooted with the node.

Note that the cost through a node includes two costs.

1) Cost of reaching the node from the root (When we reach a node, we have this cost computed)  
2) Cost of reaching an answer from current node to a leaf (We compute a bound on this cost to decide whether to ignore subtree with this node or not).

* In cases of a **maximization problem**, an upper bound tells us the maximum possible solution if we follow the given node. For example in [0/1 knapsack we used Greedy approach to find an upper bound](https://www.geeksforgeeks.org/branch-and-bound-set-2-implementation-of-01-knapsack/).
* In cases of a **minimization problem**, a lower bound tells us the minimum possible solution if we follow the given node. For example, in [Job Assignment Problem](https://www.geeksforgeeks.org/branch-bound-set-4-job-assignment-problem/), we get a lower bound by assigning least cost job to a worker.

In branch and bound, the challenging part is figuring out a way to compute a bound on best possible solution. Below is an idea used to compute bounds for Traveling salesman problem.

Cost of any tour can be written as below.

Cost of a tour T = (1/2) \* ∑ (Sum of cost of two edges

adjacent to u and in the

tour T)

where u ∈ V

For every vertex u, if we consider two edges through it in T,

and sum their costs. The overall sum for all vertices would

be twice of cost of tour T (We have considered every edge

twice.)

(Sum of two tour edges adjacent to u) >= (sum of minimum weight

two edges adjacent to

u)

Cost of any tour >= 1/2) \* ∑ (Sum of cost of two minimum

weight edges adjacent to u)

where u ∈ V

For example, consider the above shown graph. Below are minimum cost two edges adjacent to every node.

Node Least cost edges Total cost

0 (0, 1), (0, 2) 25

1 (0, 1), (1, 3) 35

2 (0, 2), (2, 3) 45

3 (0, 3), (1, 3) 45

Thus a lower bound on the cost of any tour =

1/2(25 + 35 + 45 + 45)

= 75

Now we have an idea about computation of lower bound. Let us see how to how to apply it state space search tree. We start enumerating all possible nodes (preferably in lexicographical order)

**1. The Root Node:** Without loss of generality, we assume we start at vertex “0” for which the lower bound has been calculated above.

**Dealing with Level 2:** The next level enumerates all possible vertices we can go to (keeping in mind that in any path a vertex has to occur only once) which are, 1, 2, 3… n (Note that the graph is complete). Consider we are calculating for vertex 1, Since we moved from 0 to 1, our tour has now included the edge 0-1. This allows us to make necessary changes in the lower bound of the root.

Lower Bound for vertex 1 =

Old lower bound - ((minimum edge cost of 0 +

minimum edge cost of 1) / 2)

+ (edge cost 0-1)

How does it work? To include edge 0-1, we add the edge cost of 0-1, and subtract an edge weight such that the lower bound remains as tight as possible which would be the sum of the minimum edges of 0 and 1 divided by 2. Clearly, the edge subtracted can’t be smaller than this.

**Dealing with other levels:** As we move on to the next level, we again enumerate all possible vertices. For the above case going further after 1, we check out for 2, 3, 4, …n.  
Consider lower bound for 2 as we moved from 1 to 1, we include the edge 1-2 to the tour and alter the new lower bound for this node.

Lower bound(2) =

Old lower bound - ((second minimum edge cost of 1 +

minimum edge cost of 2)/2)

+ edge cost 1-2)

Note: The only change in the formula is that this time we have included second minimum edge cost for 1, because the minimum edge cost has already been subtracted in previous level.

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. [https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/#:~:text=Branch%20and%20Bound-,Traveling%20Salesman%20Problem%20using%20Branch%20And%20Bound,returns%20to%20the%20starting%20point.](https://www.geeksforgeeks.org/traveling-salesman-problem-using-branch-and-bound-2/" \l ":~:text=Branch%20and%20Bound-,Traveling%20Salesman%20Problem%20using%20Branch%20And%20Bound,returns%20to%20the%20starting%20point.)

**Lecture Video:**

1. <https://youtu.be/1FEP_sNb62k>
2. <https://youtu.be/Ch77w90gG10>

**Online Notes:**

1. <http://vssut.ac.in/lecture_notes/lecture1428551222.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.

**In addition: PPT can be also be given.**