Branch and Bound Technique (TSP)

Branch and Bound

- Branch and bound is an algorithm design paradigm for discrete and combinatoric optimisation problems, as well as mathematical optimisation.

- A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions. That is, the set of candidate solutions is thought of as forming a rooted tree with the full set at the root.

- The algorithm explores branches of this tree, which represent the subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

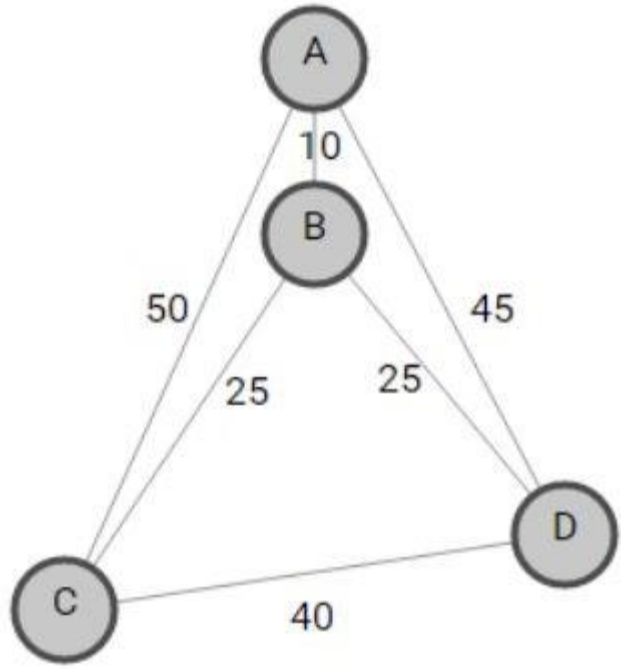
Parameter	Backtracking	Branch and Bound
Approach	Backtracking is used to find all possible solutions available to a problem. When it realises that it has made a bad choice, it undoes the last choice by backing it up. It searches the state space tree until it has found a solution for the problem.	Branch-and-Bound is used to solve optimisation problems. When it realises that it already has a better optimal solution that the pre-solution leads to, it abandons that pre-solution. It completely searches the state space tree to get optimal solution.
Traversal	Backtracking traverses the state space tree by DFS(Depth First Search) manner.	Branch-and-Bound traverse the tree in any manner, DFS or BFS.
Function	Backtracking involves feasibility function.	Branch-and-Bound involves a bounding function.
Problems	Backtracking is used for solving Decision Problem.	Branch-and-Bound is used for solving Optimisation Problem.
Searching	In backtracking, the state space tree is searched until the solution is obtained.	In Branch-and-Bound as the optimum solution may be present any where in the state space tree, so the tree need to be searched completely.
Efficiency	Backtracking is more efficient.	Branch-and-Bound is less efficient.
Applications	Useful in solving N-Queen Problem, Sum of subset.	Useful in solving Knapsack Problem, Travelling Salesman Problem.

Travelling Salesman Problem using Branch and Bound

https://www.youtube.com/watch?v=1FEP_sNb62k&t=672s

Traveling Salesman Problem using Branch And Bound

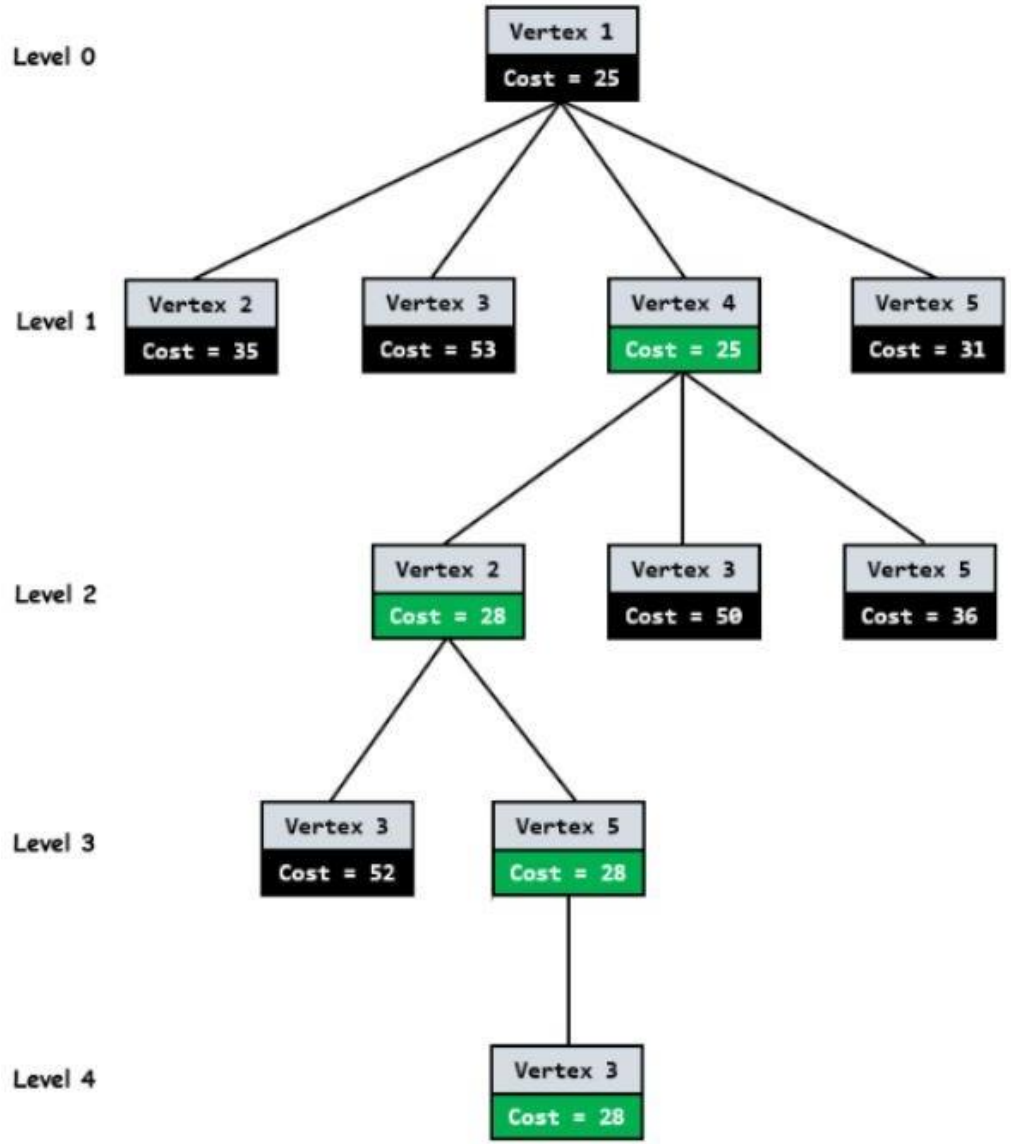
- Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.



	C0	C1	C2	C3	C4
C0	INF	20	30	10	11
C1	15	INF	16	4	2
C2	3	5	INF	2	4
C3	19	6	18	INF	3
C4	16	4	7	16	INF

C(i, j) = W(i, j), if there is a direct path from C_i to C_j
= INFINITY, if there is no direct path from C_i to C_j

Following is the state-space tree for the above TSP problem, which shows the optimal solution marked in green:



State Space Diagram

Let's start from the root node.

We reduce the minimum value in each row from each element in that row. The minimum in each row of cost matrix `M` is marked by blue `[10 2 2 3 4]` below.

```
INF  20  30  10  11
 15 INF  16   4   2
  3   5 INF   2   4
 19   6  18 INF   3
 16   4   7  16 INF
```

After reducing the row, we get the below reduced matrix.

```
INF  10  20   0   1
 13 INF  14   2   0
  1   3 INF   0   2
 16   3  15 INF   0
 12   0   3  12 INF
```

We then reduce the minimum value in each column from each element in that column. Minimum in each column is marked by blue `[1 0 3 0 0]`. After reducing the column, we get below the reduced matrix. This matrix will be further processed by child nodes of the root node to calculate their lower bound.

```
INF  10  17   0   1
 12 INF  11   2   0
  0   3 INF   0   2
 15   3  12 INF   0
 11   0   0  12 INF
```

The total expected cost at the root node is the sum of all reductions.

```
Cost = [10 2 2 3 4] + [1 0 3 0 0] = 25
```

Let's consider an edge from `0 -> 1`.

1. As we add an edge `(0, 1)` to our search space, set outgoing edges for city `0` to `INFINITY` and all in edges to city `1` to `INFINITY`. We also set `(1, 0)` to `INFINITY`.

So in a reduced matrix of the parent node, change all the elements in row `0` and column `1` and at index `0)` to `INFINITY` (marked in red).

```
INF  10  17   0   1
 12 INF  11   2   0
  0   3 INF   0   2
 15   3  12 INF   0
 11   0   0  12 INF
```

The resulting cost matrix is:

```
INF INF INF INF INF
INF INF  11   2   0
  0 INF INF   0   2
 15 INF  12 INF   0
 11 INF   0  12 INF
```

2. We try to calculate the lower bound of the path starting at node `1` using the above resulting cost mat lower bound is `0` as the matrix is already in reduced form, i.e., all rows and all columns have zero value.

Therefore, for node 1, the cost will be:

```
Cost = cost of node 0 +
        cost of the edge(0, 1) +
        lower bound of the path starting at node 1
     = 25 + 10 + 0 = 35
```

Let's consider an edge from `0 –> 2`

1. Change all the elements in row `0` and column `2` and at index `(2, 0)` to `INFINITY` (marked in red).

```
INF  10  17   0   1
 12 INF  11   2   0
  0   3 INF   0   2
 15   3  12 INF   0
 11   0   0  12 INF
```

The resulting cost matrix is:

```
INF INF  INF  INF  INF
12  INF  INF    2   0
INF  3   INF    0   2
15   3   INF  INF   0
11   0   INF   12  INF
```

2. Now calculate the lower bound of the path starting at node `2` using the approach discussed earlier. The resultant matrix will be:

```
INF INF INF INF INF
  1 INF INF   2   0
INF   3 INF   0   2
  4   3 INF INF   0
  0   0 INF  12 INF
```

Therefore, for node 2, the cost will be

```
Cost = cost of node 0 +
       cost of the edge(0, 2) +
       lower bound of the path starting at node 2
     = 25 + 17 + 11 = 53
```

Let's consider an edge from `0 –> 3` .

1. Change all the elements in row `0` and column `3` and at index `(3, 0)` to `INFINITY` (marked in red).

```
INF  10  17   0   1
 12 INF  11   2   0
  0   3 INF   0   2
 15   3  12 INF   0
 11   0   0  12 INF
```

The resulting cost matrix is:

```
INF  INF  INF INF  INF
 12  INF  11  INF   0
  0   3   INF INF   2
 INF  3   12  INF   0
 11   0    0  INF  INF
```

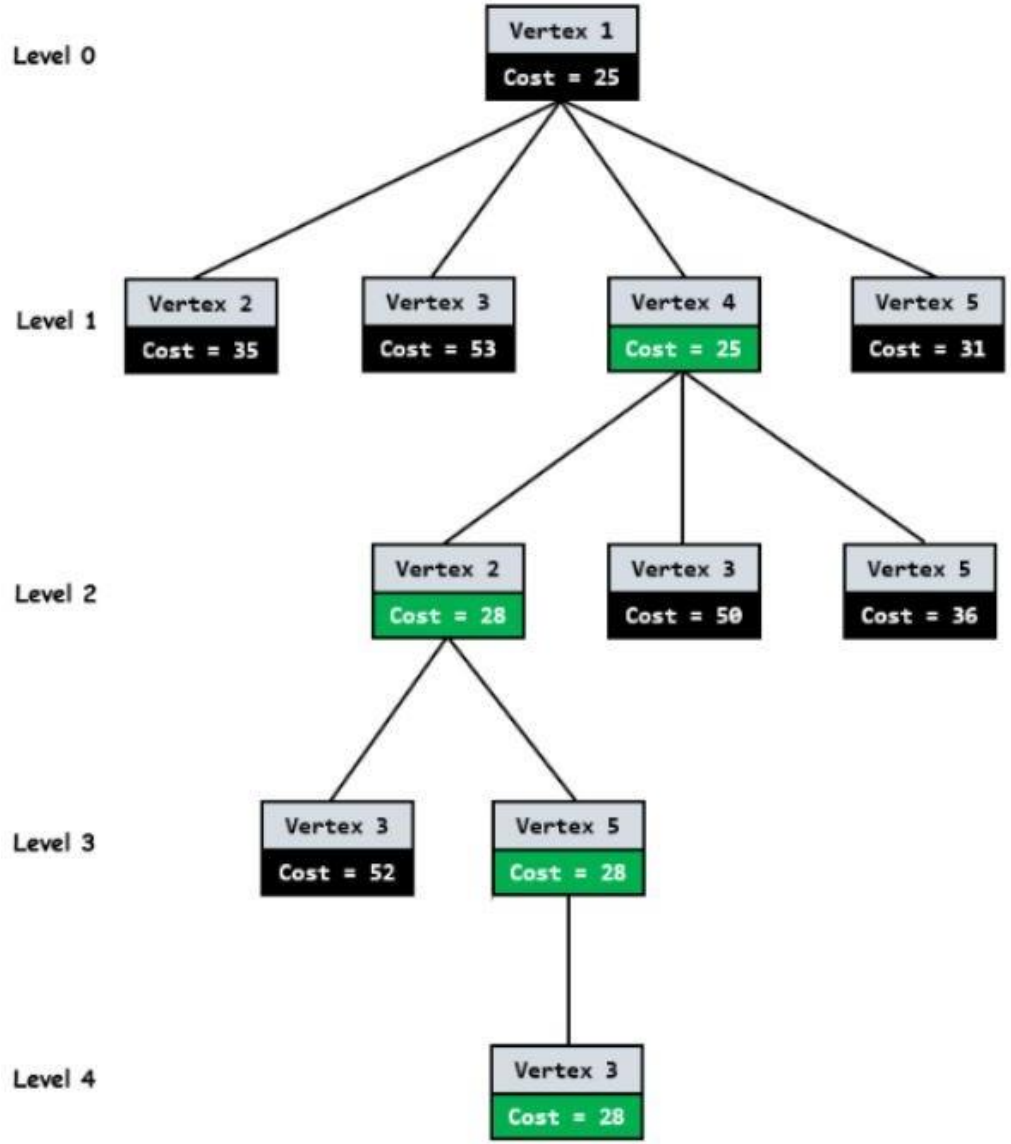
2. Now calculate the lower bound of the path starting at node `3` using the approach discussed earlier. The lower bound of the path starting at node `3` is `0` as it is already in reduced form, i.e., all rows and all columns have zero value.

Therefore, for node 3, the cost will be

```
Cost = cost of node 0 +
       cost of the edge(0, 3) +
       lower bound of the path starting at node 3
     = 25 + 0 + 0 = 25
```

Similarly, we calculate the cost of `0 -> 4`. Its cost will be `31`.

Now find a live node with the least estimated cost. Live nodes `1`, `2`, `3`, and `4` have costs `35`, `53`, `25`, and `31`, respectively. The minimum among them is node `3`, having cost `25`. So, node `3` will be expanded further, as shown in the state-space tree diagram. After adding its children to the list of live nodes, find a live node with the least cost and expand it. Continue the search till a leaf is encountered in the space search tree. If a leaf is encountered, then the tour is completed, and we will return to the root node.



State Space Diagram