Branch and bound

Branch and bound is a systematic method for solving optimization problems. Branch and bound is a rather general optimization technique that applies where the greedy method and dynamic programming fail.However,it is much lower. Indeed, it often leads to exponential time complexities in the worst case.On the other hand,if applied carefully,it can lead to algorithms that run reasonably fast on average.

Branch-and-bound procedure requires two tools.

1. Branching
   - a way of covering the feasible region by several smaller feasible sub-regions (ideally,splitting into sub-regions).
   - since the procedure may be repeated recursively to each of the sub regions and all produced sub-regions naturally form a tree structure,called search tree or branch-and-bound-tree. Its nodes are the constructed sub-regions.
2. Bounding
   - which is a fast way of finding upper and lower bounds for the optimal solution within a feasible sub-region

The Branch and bound technique like Backtracking explores the implicit graph and deals with the optimal solution to a given problem.In this technique at each stage we calculate the bound for a particular node and check whether this bound will be able to give the solution or not .If we find that at any node the solution so obtained is appropriate but the remaining solution is not leading to a best case then we leave this node without exploring.

Terminologies

- Each node in this tree defines a **problem state**
- All paths from the root to other nodes define the **state space** of the problem
- **Solution states** are those problem states "S" for which the path from the root to 's' defines a tuple in the solution space.
- **Answer states** are those solution states 's' for which the path from the root to "S"defines a tuple that is a member of the set of solutions.
- The tree organization of the solution space referred to as the **state space tree.**
- A node which has been generated and all of whose children have not yet been generated is called a **live node**.
- The live node whose children are currently being generated is called the **E-node** (node being expanded).
- A **dead node** is a generated node which is not to be expanded further or all of whose children have been generated.

The term branch and bound refers to all state space search methods in which all children of E-node are generated before any other live node can become the E-node.

Difference between Backtracking and Branch-and-Bound

	Backtracking	Branch-and-Bound (BB)
1.	It is used to find all possible solutions available to the problem.	It is used to solve optimization problem.
2.	It traverse tree by DFS (Depth First Search).	It may traverse the tree in any manner, DFS or BFS.
3.	It realizes that it has made a bad choice and undoes the last choice by backing up.	It realizes that it already has a better optimal solution that the pre-solution leads to, so it abandons that pre-solution.
4.	It search the state space tree until it found a solution.	It completely searches the state space tree to get optimal solution.
5.	It involves feasibility function.	In involves bounding function.

BRANCH AND BOUND FOR TSP (Traveling Salesman Problem)

TSP includes as a salesperson who has to visit a number of cities during a tour and the condition is to visit all the cities exactly once and return back to the same city where the person started.

Basic steps

Let G=(V,E) be a direct graph defining an instance of the TSP.

1. This graph is first represented by a *cost matrix* where
   c_{ij} = the cost of edge , if there is a path between vertex i and vertex j
   $c_{ij} = \infty$, if there is no path
2. Convert cost matrix into *reduced matrix* i.e.,every row and column should contain at least one zero entry.
3. Cost of the reduced matrix is the sum of elements that are subtracted from rows and columns of cost matrix to make it reduced.
4. Make the *state space tree* for reduced matrix.
5. To find the next E-node, find the *least cost valued node* by calculating the reduced cost matrix with every node.
6. If (i, j) edge is to be included, then there are three conditions to accomplish this task:
   1. Change all entries in row i and column j of A to ∞ .
   2. set A [j, 1] =∞
   3. Reduce all rows and columns in resulting matrix except for rows and columns containing ∞.
7. Calculate the cost of the matrix where
   cost = L + cost (i, j) + r
   where L = cost of original reduced cost matrix and
   r= new reduced cost matrix
8. Repeat the above steps for all the nodes until all the nodes are generated and we get a path.