Design and Analysis of Algorithm

Lecture-23:
Branch and Bound

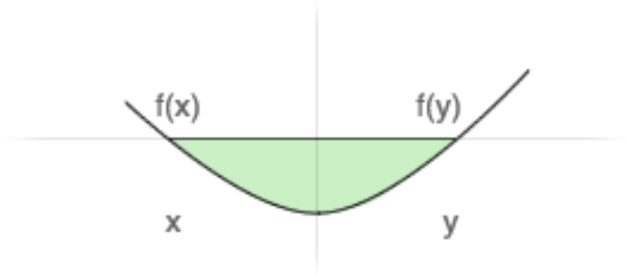
Contents

Optimization Problem Types

Convex and Non Convex optimization

A convex optimization problem is a problem where all of the constraints are convex functions, and the objective is a convex function if minimizing

If we can draw a line segment between any two points on the graph of a function such that there is no point of this graph that is above this line segment between these two points then the function is called a convex function.

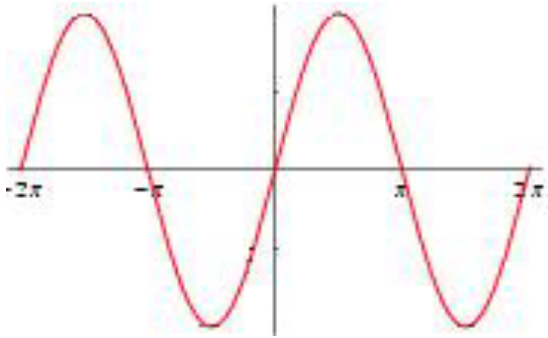


Optimization Problem Types

Convex and Non Convex optimization

A non-convex optimization problem is any problem where the objective or any of the constraints are non-convex

A non-convex function "curves up and down" -- it is neither convex nor concave.



Branch and Bound

Branch and Bound is a state space search method in which all the children of a node are generated before expanding any of its children.

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

Branch-and-Bound traverse the tree in any manner, FIFO or LIFO

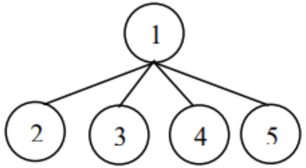
Important Definition

Live node is a node that has been generated but whose children have not yet been generated.

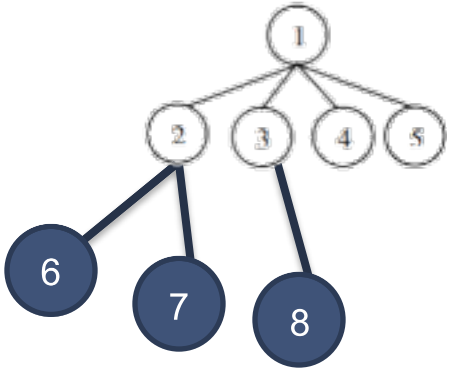
E-node is a live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.

Dead node is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded

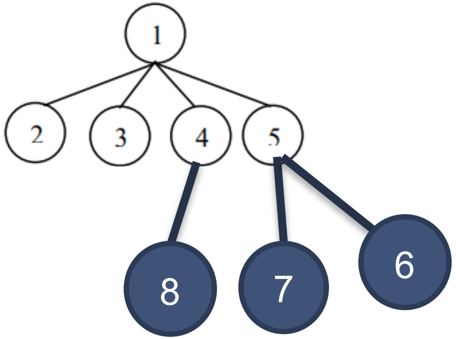
Example



Live Node: 2, 3, 4, and 5



FIFO Branch & Bound (BFS)
Children of E-node are inserted in a queue.



LIFO Branch & Bound (D-Search)
Children of E-node are inserted in a stack.

Concept

Set up a bounding function, which is used to compute a bound (for the value of the objective function) at a node on a state-space tree and determine if it is promising

Nonpromising: if the bound is no better than the value of the best solution so far then do not expand beyond the node (pruning the state-space tree).

Promising: if the bound is better than the value of the best solution so far: expand beyond the node.

0-1 Knapsack

Capacity W=10

Item	Weight	Value	Value / weight
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

0-1 Knapsack

Compute Upper Bound

The maximum upper bound is

– pick no items, take maximum profit item
– ub = (10 – 0)*($10) = $100

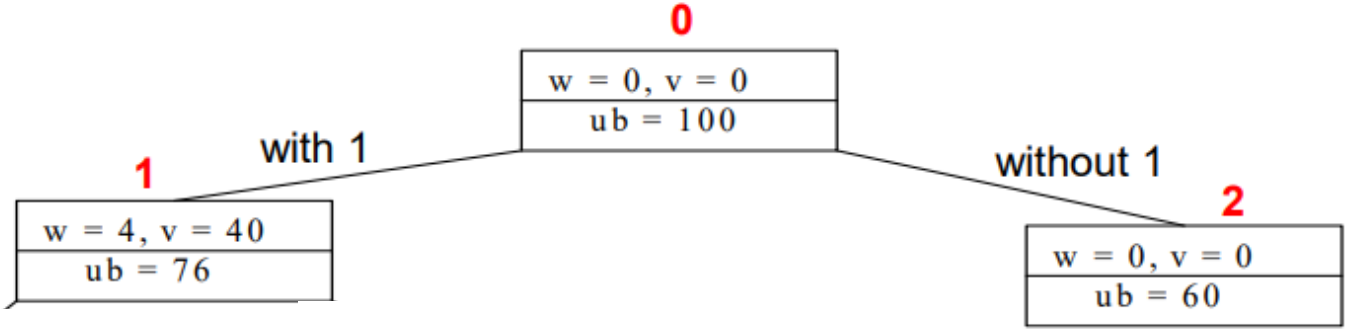
After we pick item 1,

-- all of item 1 (4, $40) + partial of item 2 (7, $42)
– $40 + (10-4)*6 = $76

If we don’t pick item 1:

– ub = (10 – 0) * ($6) = $60

0-1 Kn



Assignment problem

Consider the problem of assigning $\boldsymbol{n}$ people to $\boldsymbol{n}$ jobs so that the total cost of the assignment is as small as possible.

$$C = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} J_1 & J_2 & J_3 & J_4 \end{array} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array}$$

It can be observed that the cost of any solution, including an optimal one, cannot be smaller than the sum of the smallest elements in each of the matrix’s rows.

Assignment problem

$$C = \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{cccc} J_1 & J_2 & J_3 & J_4 \end{array} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array}$$