Branch-and-Bound

The branch and bound algorithm is similar to backtracking but is used for optimization problems. It performs a graph transversal on the space-state tree, but general searches BFS instead of DFS.

During the search **bounds** for the objective function on the partial solution are determined. At each level the best bound is explored first, the technique is called **best bound first**. If a complete solution is found then that value of the objective function can be used to prune partial solutions that exceed the bounds.

The difficult of designing branch and bound algorithm is finding good bounding function. The bounding the function should be inexpensive to calculate but should be effective at selecting the most promising partial solution.

Assignment Problem

The root of the tree can be no assignments then the partial solution at the first level considers assigning the first work to different tasks, the second level assigns the second worker, and so on to the lowest level and last worker.

We need a lower bound for the objective function. The lower bound can be the smallest cost for each remaining worker over all jobs. Note that this assignment is generally not feasible because two workers may not be assigned to the same job. A better solution would be the smallest cost for each worker over the remaining jobs but this would cost O(*n*).

Illustrate

Knapsack Problem

It is convenient to order the items by there value to weight ration

$$v1/w1 \geq v2/w2 \geq \ldots \geq v_n/w_n$$

The root of the state-space tree is the empty knapsack. Each level, *i*, of the state-space tree can consider adding or not adding the item *i* to the knapsack. The nodes can contain the total weight, *w*, and value, *v*, of the knapsack.

We need an upper bound, *up*, for the objective function. We can consider the filling the remaining knapsack (*W*-*w*) with the best remaining item, *i*+1.

$$up = v + (W-w)(v_{i+1}/w_{i+1})$$

Illustrate the algorithm - A group will demonstrate this problem

Traveling Salesman Problem

Each node of the state space tree is a list of the cities visited currently. The root can be any single city. Traveling a long and edge in the graph adds the destination city to the list at the next level node.

We need a lower bound for the tour distance. First consider a lower bound function for the root of the state tree.

One lower bound could be multiplying the smallest intercity distance by the number of cities. This lower bound is clearly too low.

A better algorithm would be to use the sum of the average distance between the city and the two nearest adjacent cities for each city.

Show example

This is a lower bound because any tour must approach and leave the city by paths at least this long. It also clear that this bounding function is larger than the previous.

As cities are added to the tour, the bounding function uses the average of the used edge with the remaining nearest city.

Show example

We can use additional trick. Because the graph is undirected we can eliminate half of the paths by consider path that travel only one direction between two cities. To implement the algorithm picks two cities, for example *b* and *c*, and considers only path with *b* proceeding *c*.

Complete example