Branch and Bound

General method:

Branch and Bound is another method to systematically search a solution space. Just like backtracking, we will use bounding functions to avoid generating subtrees that do not contain an answer node. However branch and Bound differs from backtracking in two important manners:

1. It has a branching function, which can be a depth first search, breadth first search or based on bounding function.

2. It has a bounding function, which goes far beyond the feasibility test as a mean to prune efficiently the search tree.

Branch and Bound refers to all state space search methods in which all children of the E-node are generated before any other live node becomes the E-node

Branch and Bound is the generalisation of both graph search strategies, BFS and D-search.

- A BFS like state space search is called as FIFO (First in first out) search as the list of live nodes in a first in first out list (orqueue).

- A D search like state space search is called as LIFO (Last in first out) search as the list of live nodes in a last in first out (orstack).

Definition 1: Live node is a node that has been generated but whose children have not yet been generated.

Definition 2: E-node is a live node whose children are currently being explored. In other words, an E-node is a node currently being expanded.

Definition 3: 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.

Definition 4: Branch-an-bound refers to all state space search methods in which all children of an E-node are generated before any other live node can become the E-node.

Definition 5: The adjective "heuristic", means" related to improving problem solving performance". As a noun it is also used in regard to "any method or trick used to improve the efficiency of a problem solving problem". But imperfect methods are not necessarily heuristic or vice versa. "A heuristic (heuristic rule, heuristic method) is a rule of thumb, strategy, trick simplification or any other kind of device which drastically limits search for solutions in large problem spaces. Heuristics do not guarantee optimal solutions, they do not guarantee any solution at all. A useful heuristic offers solutions which are good enough most of thetime.

Least Cost (LC) search:

In both LIFO and FIFO Branch and Bound the selection rule for the next E-node in rigid and blind. The selection rule for the next E-node does not give any preference to a node that has a very good chance of getting the search to an answer node quickly.

The search for an answer node can be speeded by using an "intelligent" ranking function $\hat{c}'(.)$ for live nodes. The next E-node is selected on the basis of this ranking function. The node x is assigned a rank using:

$$\hat{c}'(x) = f(h(x)) + \hat{g}(x)$$

where, $\hat{c}'(x)$ is the cost of x.

h(x) is the cost of reaching x from the root and f(.) is any non-decreasing function.

$\hat{g}(x)$ is an estimate of the additional effort needed to reach an answer node from x.

A search strategy that uses a cost function $\hat{c}'(x) = f(h(x)) + \hat{g}(x)$ to select the next E-node would always choose for its next E-node a live node with least $\hat{c}(.)$ is called a LC–search (Least Cost search)

BFS and D-search are special cases of LC-search. If $\hat{g}(x) = 0$ and f(h(x)) = level of node x, then an LC search generates nodes by levels. This is eventually the same as a BFS. If f(h(x)) = 0 and $\hat{g}(x) > \hat{g}(y)$ whenever y is a child of x, then the search is essentially a D-search.

An LC-search coupled with bounding functions is called an LC-branch and bound search

We associate a cost c(x) with each node x in the state space tree. It is not possible to easily compute the function c(x). So we compute a estimate $\hat{c}(x)$ of c(x).

Control Abstraction for LC-Search:

Let t be a state space tree and c() a cost function for the nodes in t. If x is a node in t, then c(x) is the minimum cost of any answer node in the subtree with root x. Thus, c(t) is the cost of a minimum-cost answer node in t.

A heuristic $\hat{c}(.)$ is used to estimate c(). This heuristic should be easy to compute and generally has the property that if x is either an answer node or a leaf node, then $c(x) = \hat{c}'(x)$.

LC-search uses $\hat{c}$ to find an answer node. The algorithm uses two functions Least() and Add() to delete and add a live node from or to the list of live nodes, respectively.

Least() finds a live node with least c(). This node is deleted from the list of live nodes and returned.

Add(x) adds the new live node x to the list of live nodes. The list of live nodes be implemented as a min-heap.

Algorithm LCSearch outputs the path from the answer node it finds to the root node t. This is easy to do if with each node x that becomes live, we associate a field *parent* which gives the parent of node x. When the answer node g is found, the path from g to t can be determined by following a sequence of *parent* values starting from the current E-node (which is the parent of g) and ending at node t.

```
Listnode = record
{
        Listnode * next, *parent; float cost;
}

Algorithm LCSearch(t)
{       //Search t for an answer node
        if *t is an answer node then output *t and
        return; E  :=  t;       //E-node.
        initialize the list of live nodes to be
        empty; repeat
        {
                for each child x of E do
                {
                        if x is an answer node then output the path from x to t and
                        return; Add   (x);               //x is a new live node.
                        (x □  parent)  := E;             // pointer for path to root
                }
                zif there are no more live nodes then
                {
                        write ("No answer
                        node"); return;
                }
                E := Least();
        } until (false);
}
```

The root node is the first, E-node. During the execution of LC search, this list contains all live nodes except the E-node. Initially this list should be empty. Examine all the children of the E-node, if one of the children is an answer node, then the algorithm outputs the path from x to t and terminates. If the child of E is not an answer node, then it becomes a live node. It is added to the list of live nodes and its parent field set to E. When all the children of E have been generated, E becomes a dead node. This happens only if none of E"s children is an answer node. Continue the search further until no live nodes found. Otherwise, Least(), by definition, correctly chooses the next E-node and the search continues from here.

LC search terminates only when either an answer node is found or the entire state space tree has been generated and searched.

Bounding:

A branch and bound method searches a state space tree using any search mechanism in which all the children of the E-node are generated before another node becomes the E-node. We assume that each answer node x has a cost c(x) associated with it and that a minimum-cost answer node is to be found. Three common search strategies are FIFO, LIFO, and LC. The three search methods differ only in the selection rule used to obtain the next E-node.

A good bounding helps to prune efficiently the tree, leading to a faster exploration of the solution space.

A cost function $c'(.)$ such that $c'(x) \leq c(x)$ is used to provide lower bounds on solutions obtainable from any node x. If upper is an upper bound on the cost of a minimum-cost solution, then all live nodes x with $c'(x) \geq c(\dot{x}) >$ upper. The starting value for upper can be obtained by some heuristic or can be set to ∞ .

As long as the initial value for upper is not less than the cost of a minimum-cost answer node, the above rules to kill live nodes will not result in the killing of a live node that can reach a minimum-cost answer node. Each time a new answer node is found, the value of upper can be updated.

Branch-and-bound algorithms are used for optimization problems where, we deal directly only with minimization problems. A maximization problem is easily converted to a minimization problem by changing the sign of the objective function.

To formulate the search for an optimal solution for a least-cost answer node in a state space tree, it is necessary to define the cost function c(.), such that c(x) is minimum for all nodes representing an optimal solution. The easiest way to do this is to use the objective function itself for c(.).

- For nodes representing feasible solutions, c(x) is the value of the objective function for that feasible solution.
- For nodes representing infeasible solutions, c(x) = ☐.
- For nodes representing partial solutions, c(x) is the cost of the minimum-cost node in the subtree with root x.

Since, c(x) is generally hard to compute, the branch-and-bound algorithm will use an estimate $c(\dot{x})$ such that $c(\dot{x}) \leq c(x)$ for all x.