Branch and Bound

- It is variation of general backtracking.
- It takes into account value of some objective function before branching off.
- In backtracking using a good bounding function restricts the generation of many useless nodes in the implicit search tree.
- Yet the search is blind in backtrack. A *live* node is expanded when a deadend is reached through the current path.
- The role of bound function was to determine if the current node of the implicit tree would lead to a solution or not and thus assists the decision to backtrack.
- In branch and bound method the current node will get expanded.
- Only when the current node has been fully explored, other nodes can be explored.
- Before discussing further let us introduce a few notations.

Branch and Bound

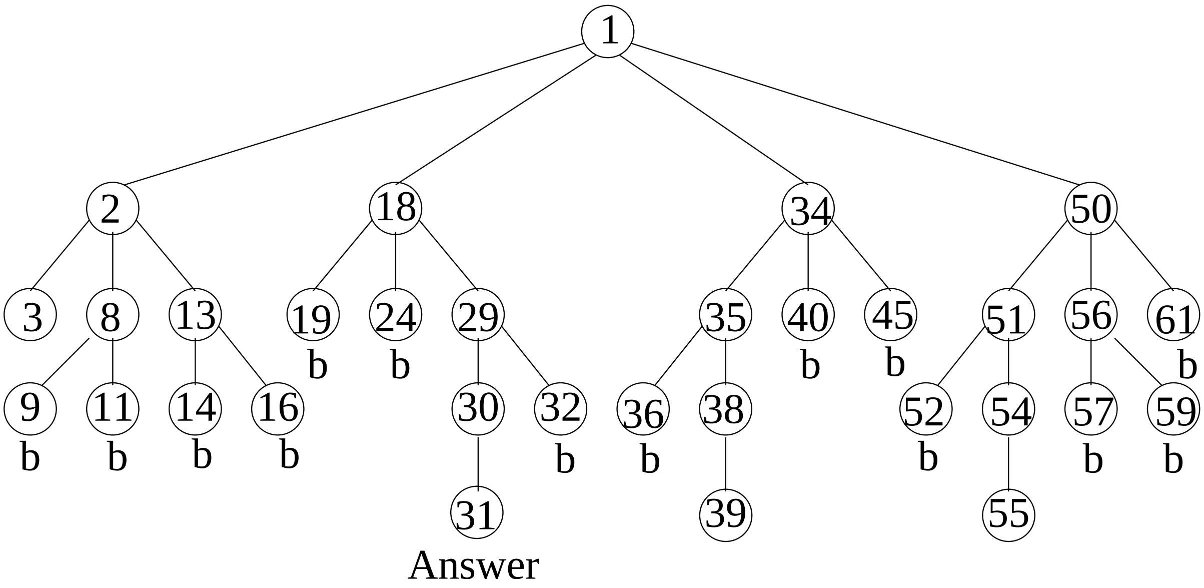
- A node in the implicit tree, can be considered as a
  - `E-node`: when it is being expanded.
  - `Live node`: its children nodes are not generated yet.
  - `Dead node`: this node either can not be expanded or all its children have been generated.
- The role of bound function to assist in pruning the search from current node by ruling out full expansion of the some of the descending paths.
- The bound function essentially optimizes some objective function of some variables.
- The best value of objective function is maintained.
- Using this value, branches which would lead to descending path less than this value under best circumstances can be prevented.

Branch and Bound

- There are two strategies for searching graphs: DFS and BFS.
- In backtrack only DFS like search of state space is carried out.
- In branch and bound both search techniques are used.
- When BFS like search is used then we have FIFO branch and bound.
- When DFS like search is used then we have LIFO branch and bound.

Example of Search

- A FIFO-BB generates the following state space tree for the 4-queen problem.



Branch and Bound

- Initially one 1 node is live – the root.
- The root is expanded – becomes E-node.
- The children of root are live nodes 2, 18, 34, 50.
- Next E-node is 2. It generates 3, 8, 13.
- 3 becomes dead by bounding function.
- 8, 13 are added to queue of live nodes and next E-node is 18.
- 19, 24, 29 are added to queue.
- Next 34 becomes E-node.
- But if backtrack is used then node 34 won't be expanded as the answer node is encountered before it.
- Branch and bound uses a ranking method by which it selects the E-node.

Ranking of Nodes

- Ranks should be such that it is an indicative of the computational effort required to reach an answer node from the live node.
- So, given a node v the cost could the number of nodes which should be generated in subtree under v before the answer node is generated.
- This may simply be equivalent to the depth of the answer node from v.
- In 4-queens example, the answer node is at level 4 from the root. So the cost of the root is 4.
- The cost of some of other nodes are as follows.

Node	Dist from ans	Cost
1	4	4
18, 34	3	3
29, 35, 51	2	2
30, 38, 54	1	1

Ranking of Nodes

- The suggested cost measures when available would help to definitely reduce generation of the number of nodes in the implicit search tree.
- But the difficulty in using the cost function is
  - Cost computation itself would require generation of the tree.
- Therefore, the exact cost computation is not possible. Only estimation of cost can be made.

Ranking of Nodes

- Let the estimated cost to reach answer node from a node v be $g(v)$.
- The cost of reaching v from the root be $h(v)$.
- Then the rank of node v is given by

$$r(v) = f(h(v)) + g(v)$$

- $f(.)$ is any nondecreasing function.
- At first sight the use of $f(.)$ does not seem to make a sense.
- The interest is to minimize the cost of additional computational efforts needed to reach the answer node, so the cost already incurred in expanding of an earlier node should not be considered.
- The purpose of including $f(.)$ is to make sure that the search does not make deep probes in searching the answer.

Ranking of Nodes

- Clearly $g(v) \geq g(w)$ for all $w \in L(v)$ where v is any node in the search tree.
- Hence, following v, w will always become the `E-node`.
- This means the probe will first make deep probes before in subtree of v before it can be directed to search other parts of the tree.
- Since, $g(.)$ is only estimated cost, so it is quite likely that one of the siblings of v can lead to answer more quickly.
- By making $f(.) \neq 0$ we can force probe to recover from making deeper probes should the descending paths from v become deeper and deeper without hitting the answer node.
- Such a strategy for searching is alternatively known as Least Cost search (`LC-search`)

A Puzzle

- Consider the following puzzle.

Initial

1	5	2
4	8	3
7		6

$\longrightarrow$

Goal

1	2	3
4	5	6
7	8	

- The directions to move the blank tile are:
  l: left u: up
  r: right d: down
- FIFO order of search without using any bound function is illustrated next.
- Note that only constraint for abandoning a descending path is repetition of earlier node.