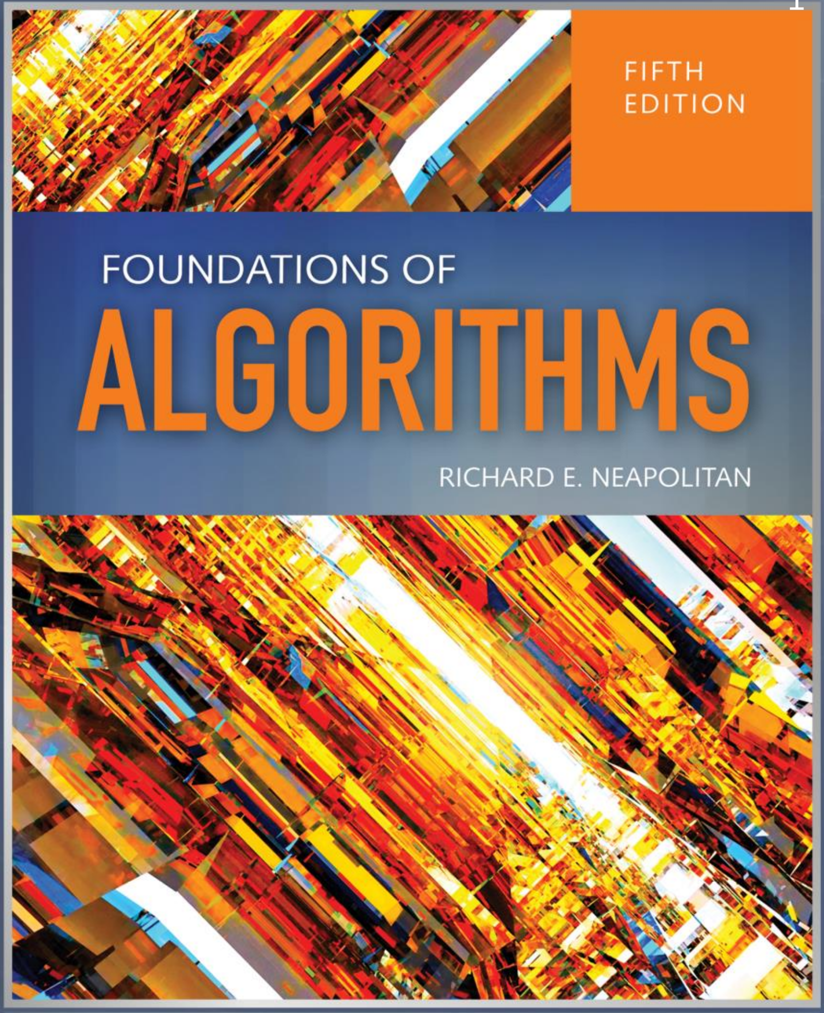


Branch-and-Bound

Chapter 6

Objectives

- Describe the branch-and-bound technique for solving optimization problems
- Contrast the branch-and-bound technique for solving problems with the backtracking technique
- Identify when it is appropriate to use the branch-and-bound technique to solve a particular problem.
- Apply the branch-and bound technique to solve the N-Queen Problem

Objectives

- Apply the branch-and-bound technique to solve the 0-1 Knapsack Problem
- Apply the branch-and-bound technique to solve the Traveling Salesperson Problem

Branch-and-Bound Design Strategy

- Branch-and-bound design strategy is similar to backtracking
- State space tree used to solve problem
- Difference between branch-and-bound and backtracking:
  - 1. branch-and-bound is not limited to a particular tree traversal
  - 2. branch-and-bound is used only for optimization problems

Branch- and-Bound Design Strategy

- Bound computed at a node to determine whether the node is promising
- Bound on the value of the solution that could be obtained by expanding beyond the node.

Bound

- If bound is no better than the value of the best solution found so far, node is non-promising
- Otherwise, the node is promising
- Better – optimal is minimum in some problems and maximum in others

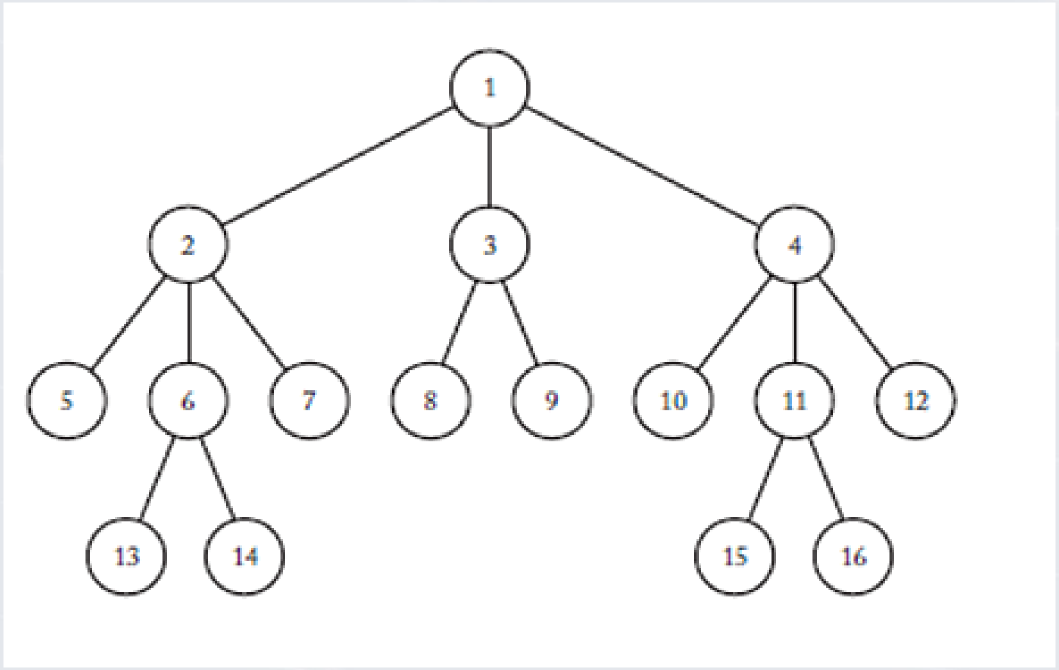
Breadth-first Tree Search

- Use Queue - FIFO
- Visit root first
- Visit all nodes at level 1 next
- Visit all nodes at level 2 next . . .
- Visit all nodes at level n

```
void beadth_first_tree_search(tree T)
{
        queue_of_node Q;
        node u, v;
        initialize(Q); //initialize queue to
be empty
        r = root of T;
        visit v;
        enqueue(Q, v);
        while (!empty(Q))
        {
                dequeue(Q, v);
                for (each child u of v)
                {
                        visit u;
                        enqueue(Q, u)
                }
        }

}
```

Figure 6.1



0-1 Knapsack Problem

- First solution: Breadth-First Search with Branch-and-Bound Pruning
- Let *weight* and *profit* be the total weight and total profit of the items that have been included up to a node.

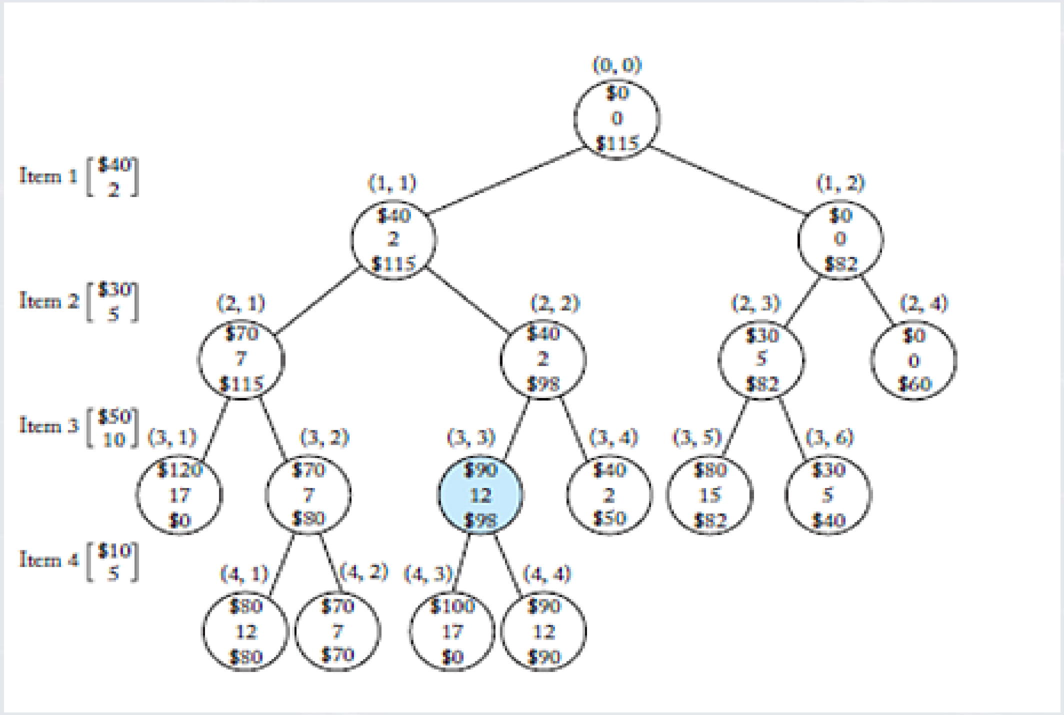
Determine if a node is promising

- Initialize *totweight* to *weight* and *bound* to *profit*
- Greedily grab items
  - Add item’s weight to *totwewight* and items profit to *bound*
- Until an item’s weight would bing totweight above W
- Grab the fraction of the item allowed by available weight and add profit of the fraction to *bound*

Determine if a mode is promising

- A node is non-promising if this bound is <= *maxprofit* – the value of the best solution found up to that point
- A node is also non-promising if weight >= W

Figure 6.2


Item 1 [$40 2]
Item 2 [$30 5]
Item 3 [$50 10]
Item 4 [$10 5]
(0, 0) $0 0 $115
(1, 1) $40 2 $115
(1, 2) $0 0 $82
(2, 1) $70 7 $115
(2, 2) $40 2 $98
(2, 3) $30 5 $82
(2, 4) $0 0 $60
(3, 1) $120 17 $0
(3, 2) $70 7 $80
(3, 3) $90 12 $98
(3, 4) $40 2 $50
(3, 5) $80 15 $82
(3, 6) $30 5 $40
(4, 1) $80 12 $80
(4, 2) $70 7 $70
(4, 3) $100 17 $0
(4, 4) $90 12 $90

Expanding from level i to level i+1

- Left branch: include item i in the solution set
- Right branch: exclude item i from the solution set

Summary Branch-and-Bound

- State space pruning
- Breadth-first traversal of state space tree
- Calculate a bound for a node
  - Bound best possible solution obtainable by expanding this node
  - Choose technique for calculating bound that produces best solution
- If bound of a node not better than the best solution found so far or current node conditions exceed problem instance limits, do not expand the node

Summary Branch and Bound

- BFS branch-and-bound pruning, decision on visiting node’s children is made at time node is visited. Maxprofit can change from time decision is made to visit until node is actually visited wasting time

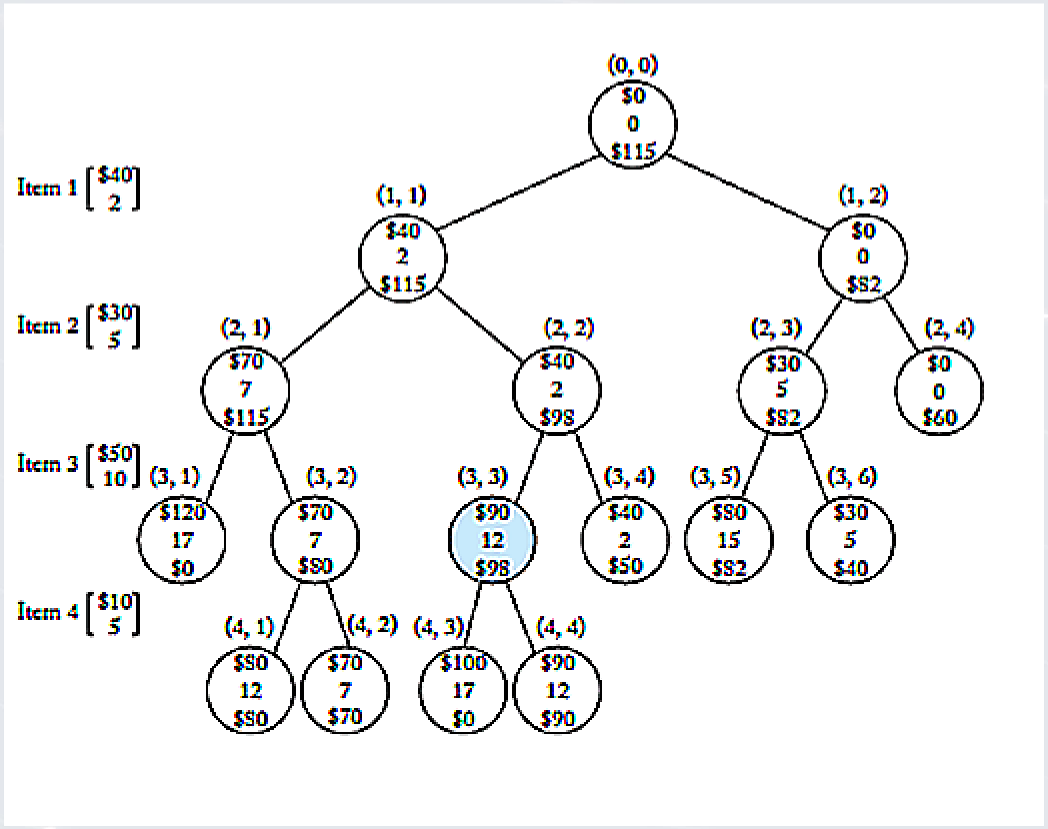
Best-First Search with Branch-and-Bound Pruning

- No advantage of BFS over DFS (backtracking)

Use bound to Improve Search

- After visiting all of the children of a given node, expand the one with the best bound
- Determine promising unexpanded node with greatest bound
- Order determined by best bound rather than pre-determined
- Use priority queue for implementation

Insert Figure 6.2



Traveling Salesperson Problem

- Goal: find an optimal tour
  - Starts at a given city
  - Visits every city exactly once
  - Returns to the starting city
- Such that the total distanced traveled is minimal

Bound

- Determine lower bound on the length of any tour obtainable by expanding beyond given node
- Adjacency Matrix to represent distances between cities
- Length of an edge taken when leaving a vertex >= length of the shortest edge leaving that vertex
- From Figure 6.4: W[1,2] = 6; W[1,3] = 14; W[1,4] = 5
  - Cost of edge taken when leaving v_1 >= 5 (minimum)

Bound

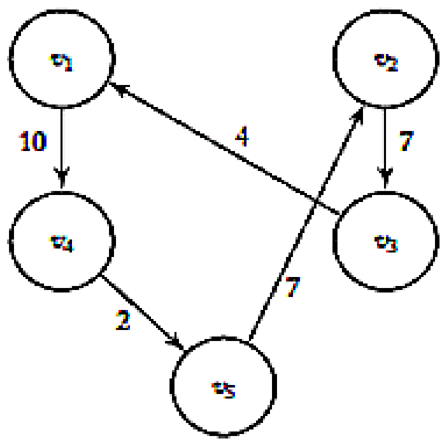
- Lower bound on any tour of n cities:
  - Minimum(W[1,j]) + minimum(W2,j]) + . . . + minimum W[I,j]) + . . Minimum(W[n,j]) where 1 <= j <=n and j !=1
  - The sum of the minimum entry in each row of the adjacency matrix
  - Expand node i representing a sub-path in the state space tree

Bound

- Bound = cost of traveling the sub-path to arrive at node i plus the cost of the minimum sub-path to travel from the last vertex in the path back to v_i
- A node is non-promising of bound >= minlength

Figure 6.4

$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$


v_1
v_2
10
4
7
v_4
v_3
2
7
v_5