COT 6405
ANALYSIS OF ALGORITHMS

Branch-and-Bound

Computer & Electrical Engineering and Computer Science Department
Florida Atlantic University

Branch-and-Bound

- Branch-and-bound technique
- Methods to construct the search tree
  - Breadth-First-Search (BreadthFS)
  - Best-First-Search (BestFS)
- Two problems:
  - Knapsack Problem
  - Traveling Salesman Problem (TSP)

Branch-and-Bound

- ***Optimization problem*** – problem that seeks to minimize or maximize an objective function
- ***Feasible solution*** – point in the problem's search space that satisfies the problem's constraints
- ***Optimal solution*** – feasible solution with the best value of the objective function
- Branch-and-bound is used for solving optimization problems

Branch-and-Bound

Compared to backtracking, branch-and-bound requires two additional items:

- For every node in the search tree, provide ***a bound on the best value of the objective function*** on any solution that can be obtained by adding further components to the partially constructed solution represented by the node
- ***The value of the best solution so far***

Branch-and-Bound Technique

Basic idea: a node is ***nonpromising*** (i.e. the branch is *pruned*) if the node bound value is not better than the best solution seen so far

- Not smaller for a minimization problem
- Not larger for a maximization problem

Branch-and-Bound Technique

A search path terminates at the current node for one of the following reasons:

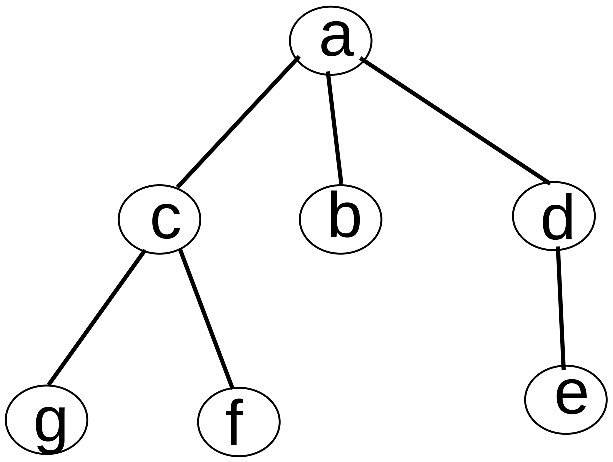
1. The value of the node’s bound is not better than the value of the best solution seen so far
2. The node represents no feasible solution
3. This node represents a feasible solution; compare the value of its objective function with the value of the best solution seen so far; if the new solution is better, then update the best solution seen so far

Branch-and-Bound

- How to generate nodes in the search tree?
  - Breadth-first-search (**BreadthFS**) with branch-and-bound pruning
  - Best-first-search (**BestFS**) with branch-and-bound pruning

Breadth-first-search (BreadthFS) - REVIEW

```
BreadthFS(T)
Q = ∅
r = T.root
ENQUEUE (Q, r)
visit r
while Q ≠ ∅
    v = DEQUEUE(Q)
     for each child u of v
        visit u
        ENQUEUE (Q,u)
```



- BreadthFS has RT = $\Theta(V+E)$
- since G is a tree, RT = $\Theta(V)$

Knapsack Problem

- Given n items:
  - weights: $w_1 \quad w_2 \ \dots \ w_n$
  - profit: $p_1 \quad p_2 \ \dots \ p_n$
  - a knapsack of capacity W
- Find most valuable subset of items that fit into the knapsack

- Example: Knapsack capacity W=16

item	weight	profit	p_i/w_i
1	2	\$40	\$20
2	5	\$30	\$6
3	10	\$50	\$5
4	5	\$10	\$2

BreadthFS with Branch-and-Bound pruning

- ***weight***, ***profit*** – total weight and total profit of the items that have been included up to a node
- compute an upperbound at each node: greedily grab items until ***totalweight*** > W
- assume current node is at level *i* and that item *k* would bring the weight above W:

$$totalweight = weight + \sum_{j=i+1}^{k-1} w_j$$

$$bound = (profit + \sum_{j=i+1}^{k-1} p_j) + (W - totalweight) \times \frac{p_k}{w_k}$$

upperbound on the profit of the current partial solution

Algorithm for BreadthFS with Branch-and-Bound General Framework

```
BreadthFS-Branch-and-Bound(T, best)
Q = ∅
r = T.root
ENQUEUE(Q,r)
best = value(r)
while Q ≠ ∅
    v = DEQUEUE(Q)
    for each child u of v
        if value(u) is better than best
            best = value(u)
        if bound(u) is better than best
            ENQUEUE(Q,u)
```

- *bound( )* and *value( )* are application dependent

Knapsack problem

- Each node is an object with fields:

  v.level – node's level in the tree

  v.profit – total profit of the items that have been included up to the node v

  v.weight – total weight of the items that have been included up to the node v

Knapsack with BreadthFS with Branch-and-Bound pruning(n,p[ ],w[ ],W,maxprofit)

```
Q = ∅
r.level = 0; r.profit = 0; r.weight = 0
maxprofit = 0
ENQUEUE(Q,r)
while Q ≠ ∅
   v = DEQUEUE(Q)
   u.level = v.level +1
   u.weight = v.weight + w[u.level]
   u.profit = v.profit + p[u.level]
   if (u.weight ≤ W and u.profit > maxprofit)
            maxprofit = u.profit
   if bound(u) > maxprofit
            ENQUEUE(Q,u)
   u.weight = v.weight
   u.profit = v.profit
   if bound(u) > maxprofit
            ENQUEUE(Q,u)
```

set u as the child of v that includes the next item

set u as the child of v that does not include the next item

Knapsack with BreadthFS with Branch-and-Bound pruning

```
bound(u)
if u.weight ≥ W
   return 0
else
   result = u.profit
   totalweight = u.weight
   j = u.level + 1
   while (j ≤ n) and (totalweight + w[j] ≤ W)
         totalweight = totalweight + w[j]
         result = result + p[j]
         j = j + 1
   k = j
   if k ≤ n
         result = result + (W – totalweight) × p_k/w_k
   return result
```

RT Analysis

- number of nodes:

$$\leq 1 + 2 + 2^2 + \ldots + 2^n = O(2^n)$$

- bound() takes O(n)

$\Rightarrow$ total RT = $O(n \cdot 2^n)$

BestFS with Branch-and-Bound Pruning

- Basic idea: when it comes to pick-up a new node in the search, choose the one with the best bound among all *promising* unexpanded nodes
- Often arrives at an optimal solution more quickly
  - there is no guarantee that the node that appears to be the best will actually lead to the optimal solution

BestFS with Branch-and-Bound Pruning

- Instead of using a queue, we use a *priority queue* PQ
- Operations:

  insert(PQ, v) – adds v to the PQ

  remove(PQ) – remove the node with the best bound

Algorithm for BestFS with Branch-and-Bound General Framework

```
BestFS-Branch-and-Bound(T, best)
PQ = ∅
r = T.root
best = value(r)
insert(PQ,r)
while PQ ≠ ∅
   v = remove(PQ)
   if bound(v) is better than best
         for each child u of v
               if value(u) is better than best
                     best = value(u)
               if bound(u) is better than best
                     insert(PQ, u)
```

Knapsack problem

- Each node is an object with fields:

  v.level – node's level in the tree

  v.profit – total profit of the items that have been included up to the node v

  v.weight – total profit of the items that have been included up to the node v

  v.bound – upperbound on the profit of the current partial solution up to node v

Knapsack-BestFS-Branch-and-Bound(n,p[ ],w[ ],W,maxprofit)

```
PQ = ∅
r.level = r.profit = r.weight = 0
maxprofit = 0
r.bound = bound(r)
insert(PQ,r)
while PQ ≠ ∅
   v = remove(PQ)
   if v.bound > maxprofit
         u.level = v.level +1
         u.weight = v.weight + w[u.level]
         u.profit = v.profit + p[u.level]
         if (u.weight ≤ W and u.profit > maxprofit)
                  maxprofit = u.profit
         u.bound = bound(u)
         if bound(u) > maxprofit
                  insert(PQ,u)
         u.weight = v.weight
         u.profit = v.profit
         u.bound = bound(u)
         if u.bound > maxprofit
                  insert(PQ,u)
```

set u as the child of v that includes the next item

set u as the child of v that does not include the next item

- function bound() is the same