Branch and Bound
Edmonds, Chapter 25

Neil Rhodes
CSE 101
UC San Diego

Keep value of best solution so far
- As a global, perhaps. No need to consider solutions which will give a worse value.

Compute bound as we visit each node
- Will be a bound on the value of solution found in subtree.
  - Bound will be upper bound if we have a maximization problem or lower bound if we have a minimization problem.

Extend idea of promising
- If bound of a node is better than best solution, node is promising, otherwise it isn’t.

Bounding allows us to prune more nodes

Integer Knapsack with bounding

Example: $W = 16$

At a given node, we have a weight so-far and a profit so-far. A bound on the total profit would be if we greedily took remaining items with the best profit/weight (until an item won’t fit), and then take a fraction of that item that won’t fit.
- Example:
  - We’ve decide to not include item 1 and to include item 2. Weight-so-far = 5, profit-so-far = $30$. A greedy upper bound on the total profit is to take all of item 3 and 1/5 of item 4. Weight = 16 and bound on profit is $30 + 50 + 1/5(10) = $82.
Knapsack Branch and Bound with Breadth-First Search

Example from Foundations of Algorithms by Neapolitan and Naimipour

Breadth-First Search

- Speedup
  - When we remove a node from the queue, its bound may be lower than current best value. If so, skip it.

Best-First Search

- Idea
  - Of all nodes in the queue to be visited, pick the one with the best bounds. It might give better results than others.

- Implementation
  - Rather than the simple queue used for breadth-first search, use a priority queue.
Knapsack Branch and Bound with Best-First Search

**Master Algorithm for Branch and Bound with Best-First Search**

BestFirstBranchAndBound(T)
Precondition: T is decision tree for problem. Nodes have bound and value.
Postcondition: returns best value of a solution

v = root of T
best = value(v)
Add v to priority queue (keyed on bound(v))
while priority queue is not empty
remove element v from priority queue
if bound(v) is better than best then
foreach child u of v
if value(u) is better than best then best = value(u)
if bound(u) is better than best then insert u into priority queue
end foreach
end if
end while
return best

Knapsack Backtracking Branch and Bound (Best-First Search)

Knapsack(w, profit, W)
root.bound = bound(root) bound returns upper bound on profit
maxprofit = root.profit = root.weight = root.level = 0
insert root into priority queue
while priority queue is not empty
remove minimum element, v, from priority queue
if v.bound > maxprofit then
u.level = v.level + 1
u.weight = v.weight + wu.weight
u.profit = v.profit + profitu.level
if u.weight <= W and u.profit > maxprofit then
maxprofit = u.profit
u.bound = bound(u)
if u.bound > maxprofit then insert u into priority queue
u.weight = v.weight; u.profit = v.profit; u.bound = bound(u)
if u.bound > maxprofit then insert u into priority queue
end if
end while Postcondition: maxprofit is value of most profitable solution

Branch and Bound Summary

Don’t branch from a node whose bound is worse than the best solution found so far
- An extension of the idea of non-promising: impossible to find a valid solution from this node
A Best-first search searches along a frontier of nodes, choosing the one with the best bound first
- This is a greedy approach which will often find an optimal solution more quickly than using a predetermined order (like DFS or BFS). It will not always be quicker, but it is a reasonable heuristic.