Lecture 13.1

- Anurag Sharma & Shymal Chandra

Branch & Bound Algorithms

Branch-and-bound algorithms

- Branch-and-bound algorithms are very similar to backtracking algorithms in that a state space tree is used to solve a problem
- The differences are:
  - branch-and-bound method does not limit us to any particular way of traversing the tree (backtracking uses depth-first search)
  - branch-and-bound is used only for optimization problems

Cont.

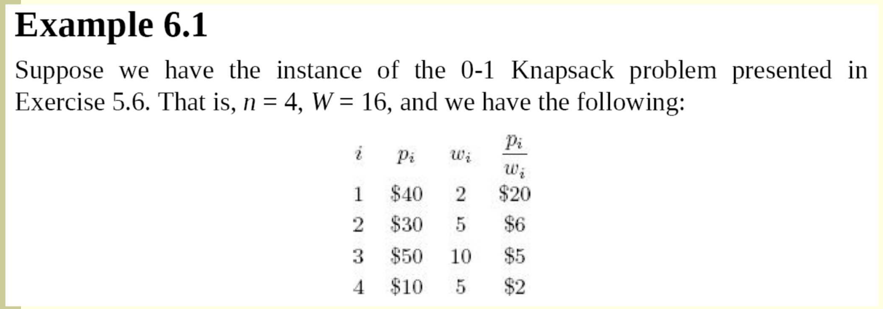
- A branch-and-bound algorithm computes a number (bound) at a node to determine whether the node is promising. This number is a bound on the value of the solution that could be obtained by expanding beyond the node. If that bound is no better than the value of the best solution found so far, the node is non-promising so there is no need to expand beyond that node
- In fact, the last example we saw in backtracking (0-1 Knapsack problem) was an example of branch-and-bound (remember, a bound was calculated at each node…)
- Branch-and-bound however does not visit the nodes in some predetermined order (i.e. no depth-first search)

Breadth-first Search

- In branch-and-bound we use best-first search, which is a modification of another approach, breadth-first search
- Breadth-first search involves visiting the root first, then all nodes at level 1 of the tree, then all nodes at level 2 and so on
- Each node is checked and if it is non-promising, it is eliminated using the same rules as in a backtracking algorithm
- This process of pruning is same as in backtracking which reduces the portion of the state space tree

The 0-1 Knapsack Problem

Same example



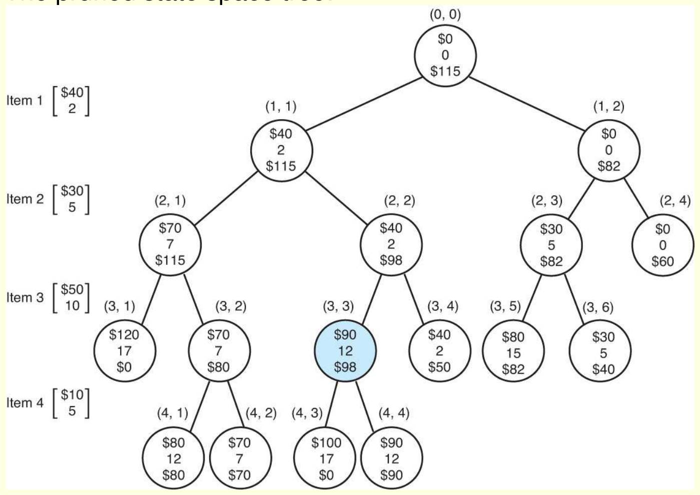
Example 6.1

Suppose we have the instance of the 0-1 Knapsack problem presented in Exercise 5.6. That is, $n = 4$, $W = 16$, and we have the following:

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

The 0-1 Knapsack Problem using Breadth-first Search

- The pruned state space tree:

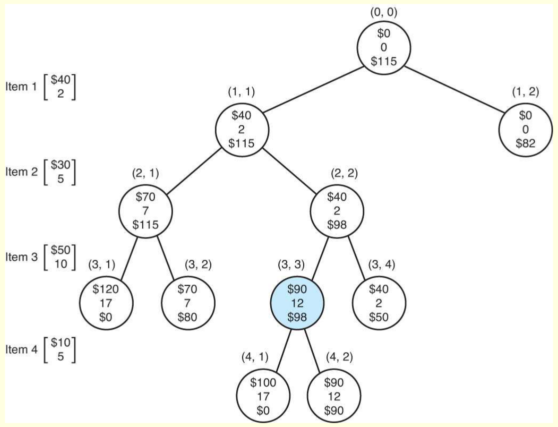


Best-first Search

- In general, the breadth-first search strategy has no significant advantage over a depth-first search (backtracking)
- However, this can be improved by using our bound to do more than just determine whether a node is promising
- This is where best-first search comes in…
- After visiting all children of a given node, we can look at all the promising, unexpanded nodes and expand beyond the node with the best bound. In this way an optimal solution is arrived more quickly than if some predetermined order is used
- In the 0-1 Knapsack problem example, we pick only that child node to expand which has the highest maximum possible profit (bound)

The 0-1 Knapsack Problem using Best-first Search

- The pruned state space tree with branch-and-bound pruning (note – this is a smaller tree than the previous one which is the result of best-first search)



Lecture 13.2

- Anurag Sharma & Shymal Chandra

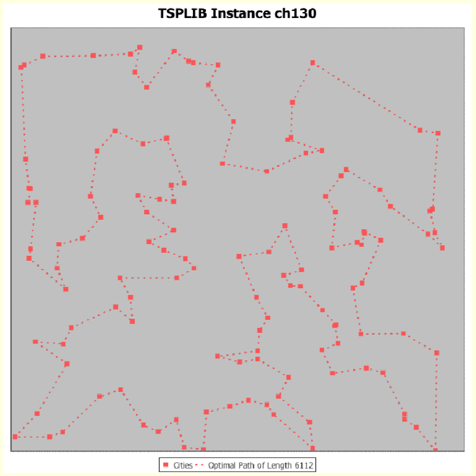
Branch & Bound Algorithms: TSP

Travelling Salespersons Problem (TSP)

- Recall: The goal in the TSP is to find the shortest path in a graph that starts at a given vertex, visits each vertex exactly once, and ends up at the starting vertex. Such a path is called optimal tour.
- Previously we have seen a dynamic programming algorithm to solve TSP; now we will see how the branch-and-bound technique can be used to solve TSP

GPS?

TSP



Cont.

- The state space tree for TSP can be as follows:
  - Start with a starting vertex on level 0 of the tree
  - Level 1 can have child nodes for each remaining vertex other than the starting vertex, level 2 can have child nodes for remaining vertices than do not fall on each path up to level 1 and so on
- To use best-first search, determine a bound for each node
- The bound in TSP can be a lower bound on the length of any tour that can be obtained by expanding beyond a given node - we can mark a node promising only if its bound is less than current minimum tour length

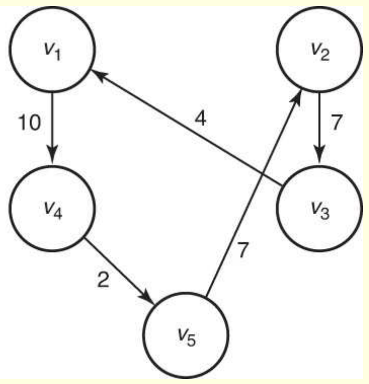
Cont.

- The method used in TSP to calculate the bound is to add up the minimum edges to every remaining vertex
- Calculate the bounds for the various children of the root, so you know which node to expand further i.e. the node with the minimum value needs to be expanded

TSP State Space Tree Example

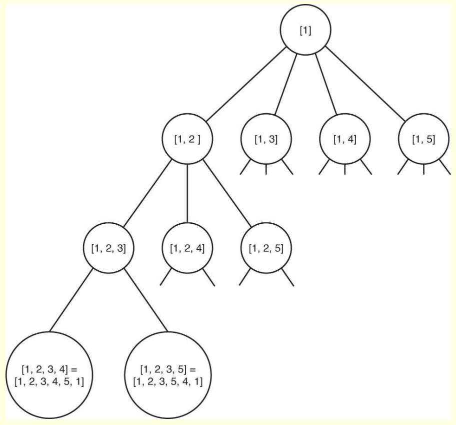
- Adjacency matrix for a graph with 5 vertices:

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



If the starting vertex is 1

- Goal is to find the shortest path in a directed graph that starts at a given vertex, visits each vertex in the graph exactly once, and ends up back at the starting vertex. Such a path is called an optimal tour.



If starting from v1

- Lower bounds:

v_1 $minimum\,(14, 4, 10, 20) = 4$
v_2 $minimum\,(14, 7, 8, 7) = 7$
v_3 $minimum\,(4, 5, 7, 16) = 4$
v_4 $minimum\,(11, 7, 9, 2) = 2$
v_5 $minimum\,(18, 7, 17, 4) = 4$

Because a tour must leave every vertex exactly once, a lower bound on the length of a tour is the sum of these minimums. Therefore, a lower bound on the length of a tour is

$$4 + 7 + 4 + 2 + 4 = 21.$$

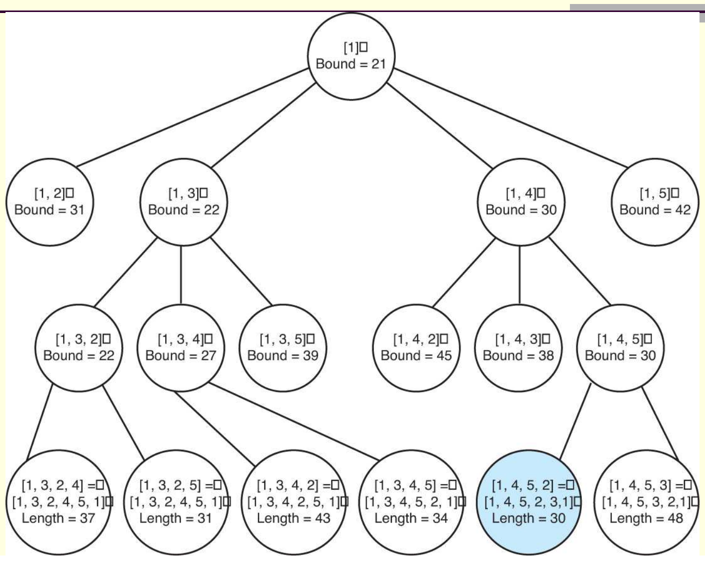
Suppose v1 → v2 is committed

v_1 14
v_2 $minimum\,(7, 8, 7) = 7$
v_3 $minimum\,(4, 7, 16) = 4$
v_4 $minimum\,(11, 9, 2) = 2$
v_5 $minimum\,(18, 17, 4) = 4$

To obtain the minimum for v_2 we do not include the edge to v_1, because v_2 cannot return to v_1. To obtain the minimums for the other vertices we do not include the edge to v_2, because we have already been at v_2. A lower bound on the length of any tour, obtained by expanding beyond the node containing [1, 2], is the sum of these minimums, which is

$$14 + 7 + 4 + 2 + 4 = 31.$$

Deterministic solution?


[1]
Bound = 21
[1, 2]
Bound = 31
[1, 3]
Bound = 22
[1, 4]
Bound = 30
[1, 5]
Bound = 42
[1, 3, 2]
Bound = 22
[1, 3, 4]
Bound = 27
[1, 3, 5]
Bound = 39
[1, 4, 2]
Bound = 45
[1, 4, 3]
Bound = 38
[1, 4, 5]
Bound = 30
[1, 3, 2, 4] =
[1, 3, 2, 4, 5, 1]
Length = 37
[1, 3, 2, 5] =
[1, 3, 2, 4, 5, 1]
Length = 31
[1, 3, 4, 2] =
[1, 3, 4, 2, 5, 1]
Length = 43
[1, 3, 4, 5] =
[1, 3, 4, 5, 2, 1]
Length = 34
[1, 4, 5, 2] =
[1, 4, 5, 2, 3,1]
Length = 30
[1, 4, 5, 3] =
[1, 4, 5, 3, 2,1]
Length = 48