Branch & Bound Algorithms

S Kavitha

AP/CSE

SSNCE

Topics

- Define branch & bound
- 0-1 Knapsack problem
  - Breadth-First Search
  - Best-First Search
- Assignment Problem

Introduction

- The branch-and-bound design strategy is very similar to backtracking in that a state space tree is used to solve a problem.

- The differences from backtracking is that

  (1) does not limit us to any particular way of traversing the tree

  (2) is used only for optimization problems.

- A branch-and-bound algorithm computes a number (bound) at a node to determine whether the node is promising.

Introduction …

- The 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 nonpromising. Otherwise, it is promising.
- This approach is called best-first search with branch-and-bound pruning. The implementation of this approach is a modification of the breadth-first search with branch-and-bound pruning.

Branch and Bound

- An enhancement of backtracking
- Applicable to optimization problems
- Uses a lower bound or upper bound for the value of the objective function for each node (partial solution) so as to:
  - guide the search through state-space
  - rule out certain branches as "unpromising"

Breadth-first Search

- We can implement this search using a queue.
- All child nodes are placed in the queue for later processing if they are promising.
- Calculate an integer value for each node that represents the maximum possible profit if we pick that node.
- If the maximum possible profit is not greater than the best total so far, don't expand the branch.

Breadth-first Search

- The breadth-first search strategy has no advantage over a depth-first search (backtracking).
- However, we can improve our search by using our bound to do more than just determine whether a node is promising.

Best-first Search

- Best-first search expands the node with the best bounds next.
- How would you implement a best-first search?
  - Depth-first is a stack
  - Breadth-first is a queue
  - Best-first is a ???

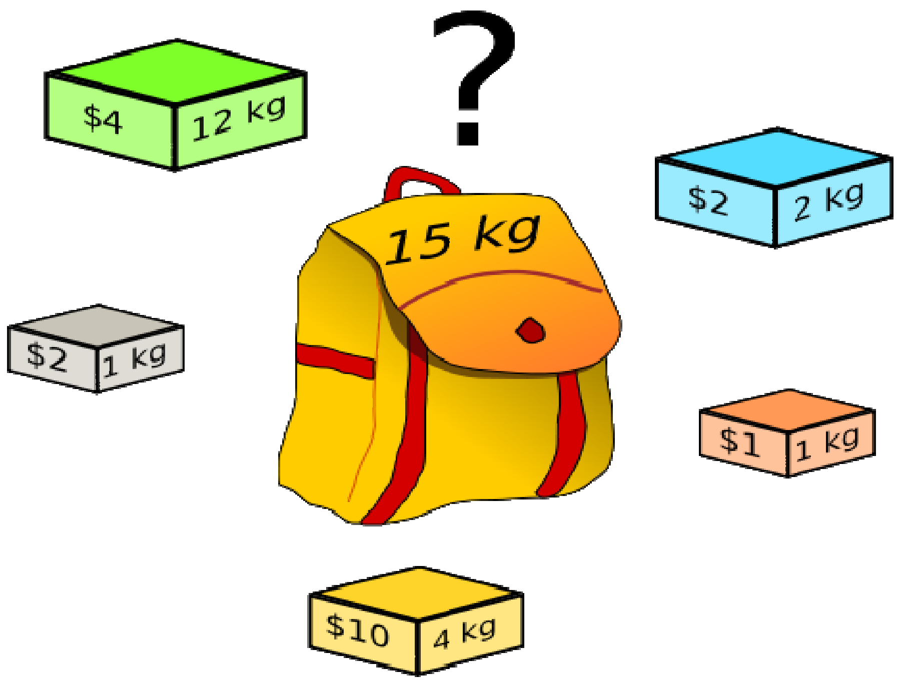
0-1 Knapsack – Problem Statement

- **Input:** Weight of N items $\{w_1, w_2, ..., w_n\}$
  Cost of N items $\{v_1, v_2, ..., v_n\}$
  Knapsack limit W
- **Output:** Selection for knapsack: $\{x_1, x_2, \ldots x_n\}$
  where $x_i \in \{0,1\}$.

$$T \subseteq \{1, 2, \ldots, n\}$$

maximizes $\sum_{i \in T} v_i,$

subject to $\sum_{i \in T} w_i \leq W.$



0-1 Knapsack – given problem

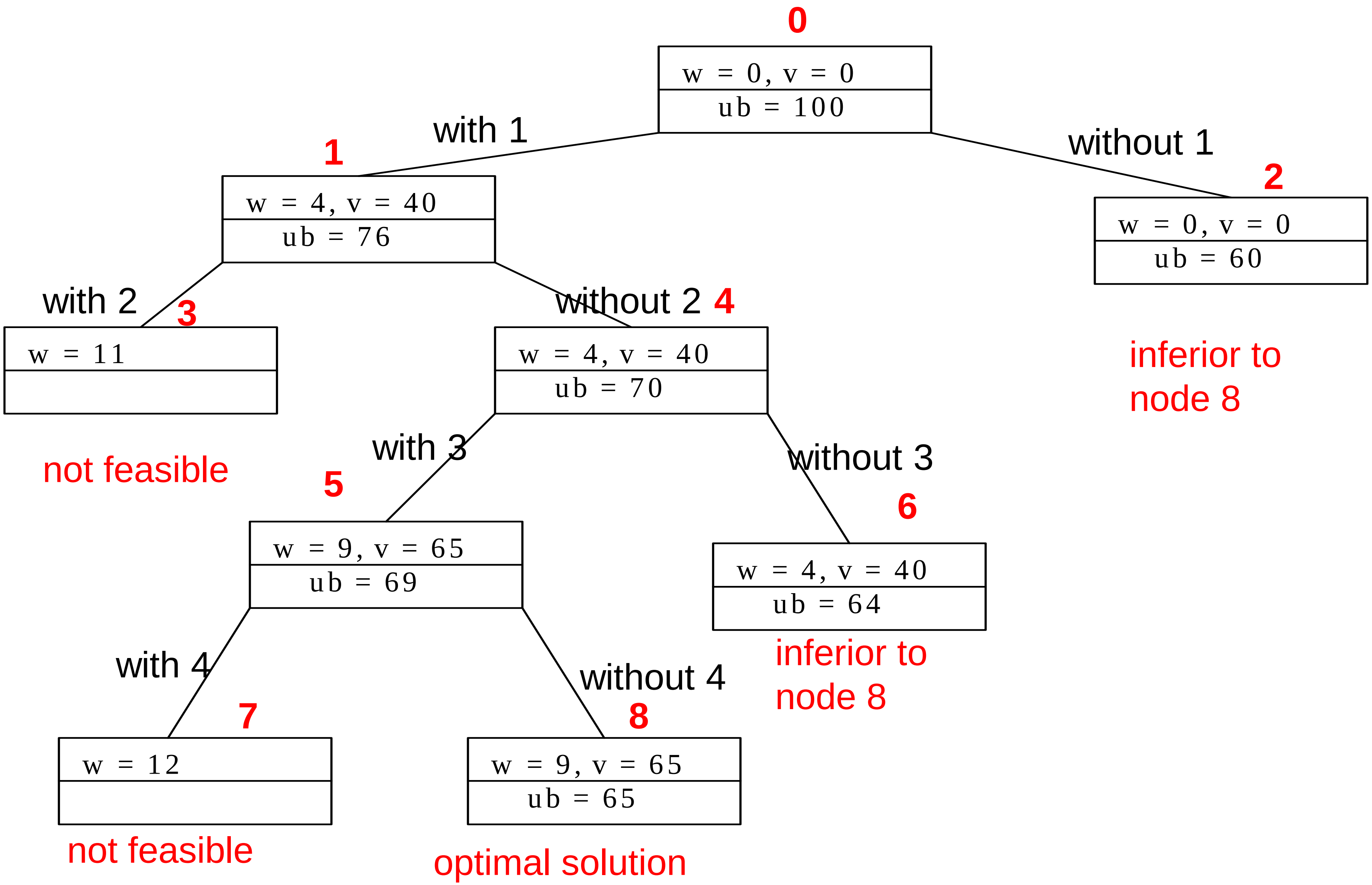
- Capacity W is 10
- Upper bound is $100

Item	Weight	Value	Value / weight
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

Computing Upper Bound

- To compute the upper bound, use
  - ub = v + (W – w)(v_{i+1}/w_{i+1})
- So the maximum upper bound is
  - pick no items, take maximum profit item
  - ub = (10 – 0)*($10) = $100
- After we pick item 1, we calculate the upper bound as
  - all of item 1 (4, $40) + partial of item 2 (7, $42)
  - $40 + (10-4)*6 = $76
- If we don't pick item 1:
  - ub = (10 – 0) * ($6) = $60

State Space Tree



Bounding

- A bound on a node is a guarantee that any solution obtained from expanding the node will be:
  - Greater than some number (lower bound)
  - Or less than some number (upper bound)

- If we are looking for a maximal optimal (knapsack), then we need an upper bound
  - For example, if the best solution we have found so far has a profit of 12 and the upper bound on a node is 10 then there is no point in expanding the node
    - The node cannot lead to anything better than a 10

Bounding

- Recall that we could either perform a depth-first or a breadth-first search
  - Without bounding, it didn't matter which one we used because we had to expand the entire tree to find the optimal solution
  - Does it matter with bounding?
    - Hint: think about when you can prune via bounding

Bounding

- We prune (via bounding) when:

  (currentBestSolutionCost >= nodeBound)

- This tells us that we get more pruning if:
  - The currentBestSolution is high
  - And the nodeBound is low

- So we want to find a high solution quickly and we want the highest possible upper bound
  - One has to factor in the extra computation cost of computing higher upper bounds vs. the expected pruning savings

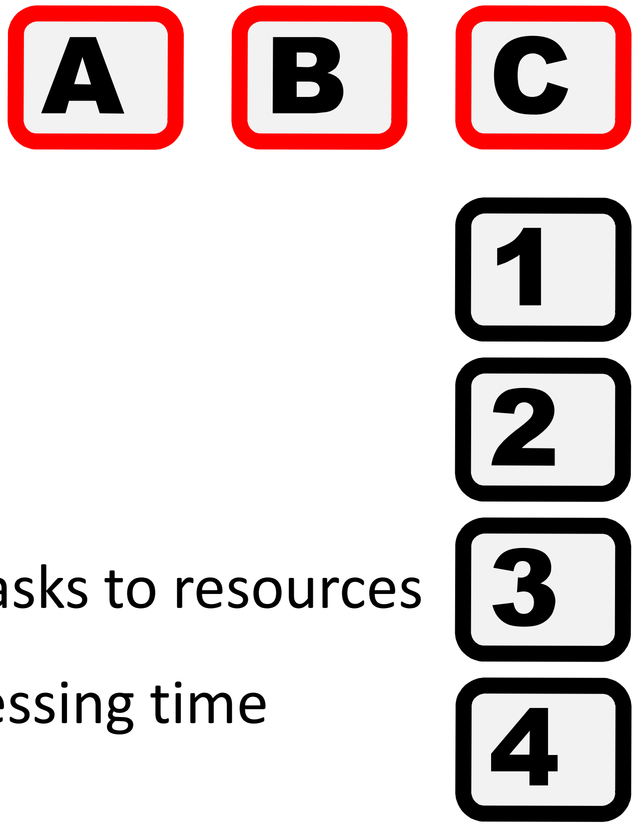
The assignment problem

- We want to assign *n* people to *n* jobs so that the total cost of the assignment is as small as possible (lower bound)

Scheduling Problem

Input of the problem:

- A number of resources
- A number of tasks



Output of the problem:

- A sequence of feeding the tasks to resources to minimize the required processing time

Example: The assignment problem

Select one element in each row of the cost matrix *C* so that:

- **no two selected elements are in the same column; and**
- **the sum is minimized**

For example:

	Job 1	*Job* 2	*Job* 3	*Job* 4
Person *a*	9	2	7	8
Person *b*	6	4	3	7
Person *c*	5	8	1	8
Person *d*	7	6	9	4

<u>Lower bound</u>: Any solution to this problem will have total cost of <u>at least</u>:

sum of the smallest element in each row = 10

Assignment problem: lower bounds

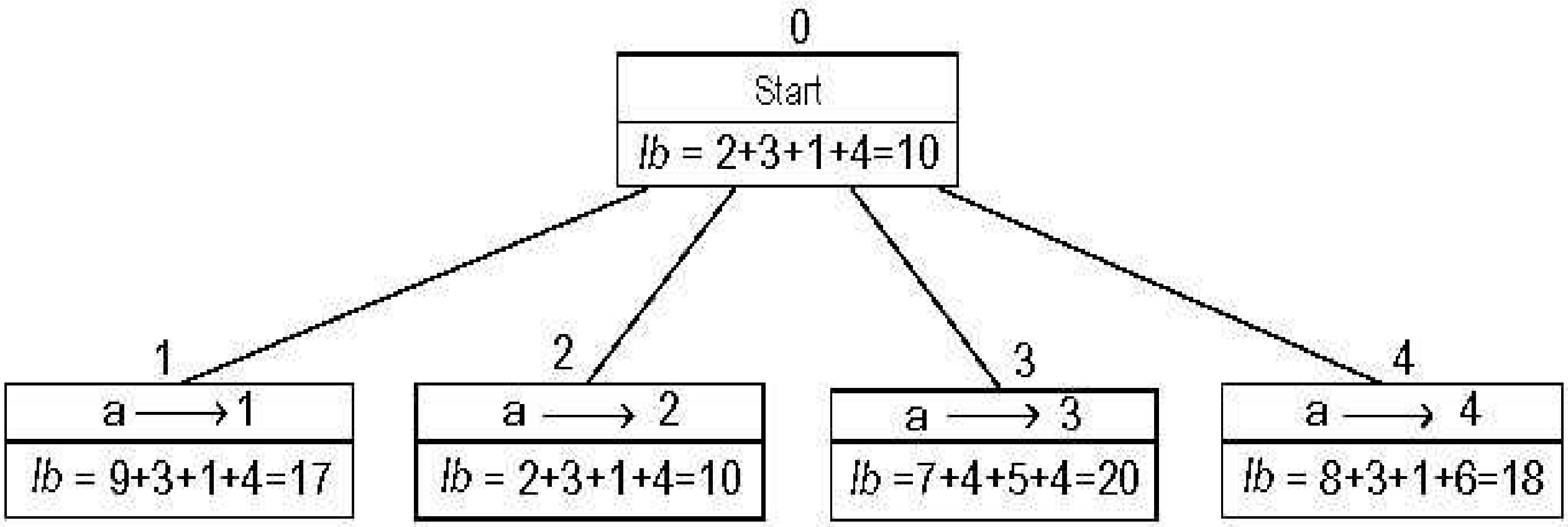


Figure 11.5 Levels 0 and 1 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm. The number above a node shows the order in which the node was generated. A node's fields indicate the job number assigned to person a and the lower bound value, lb, for this node.

State-space levels 0, 1, 2

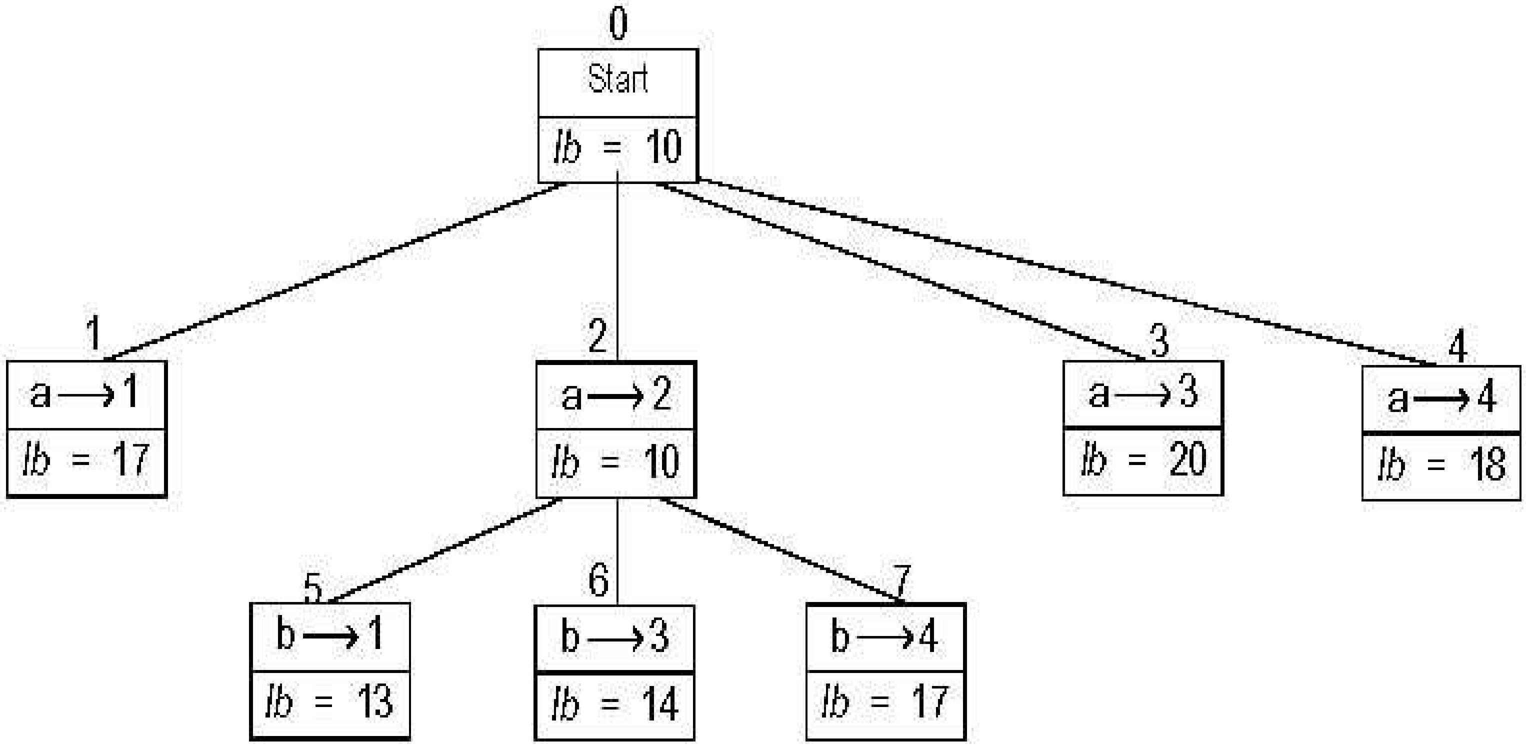


Figure 11.6 Levels 0, 1, and 2 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm

Complete state-space

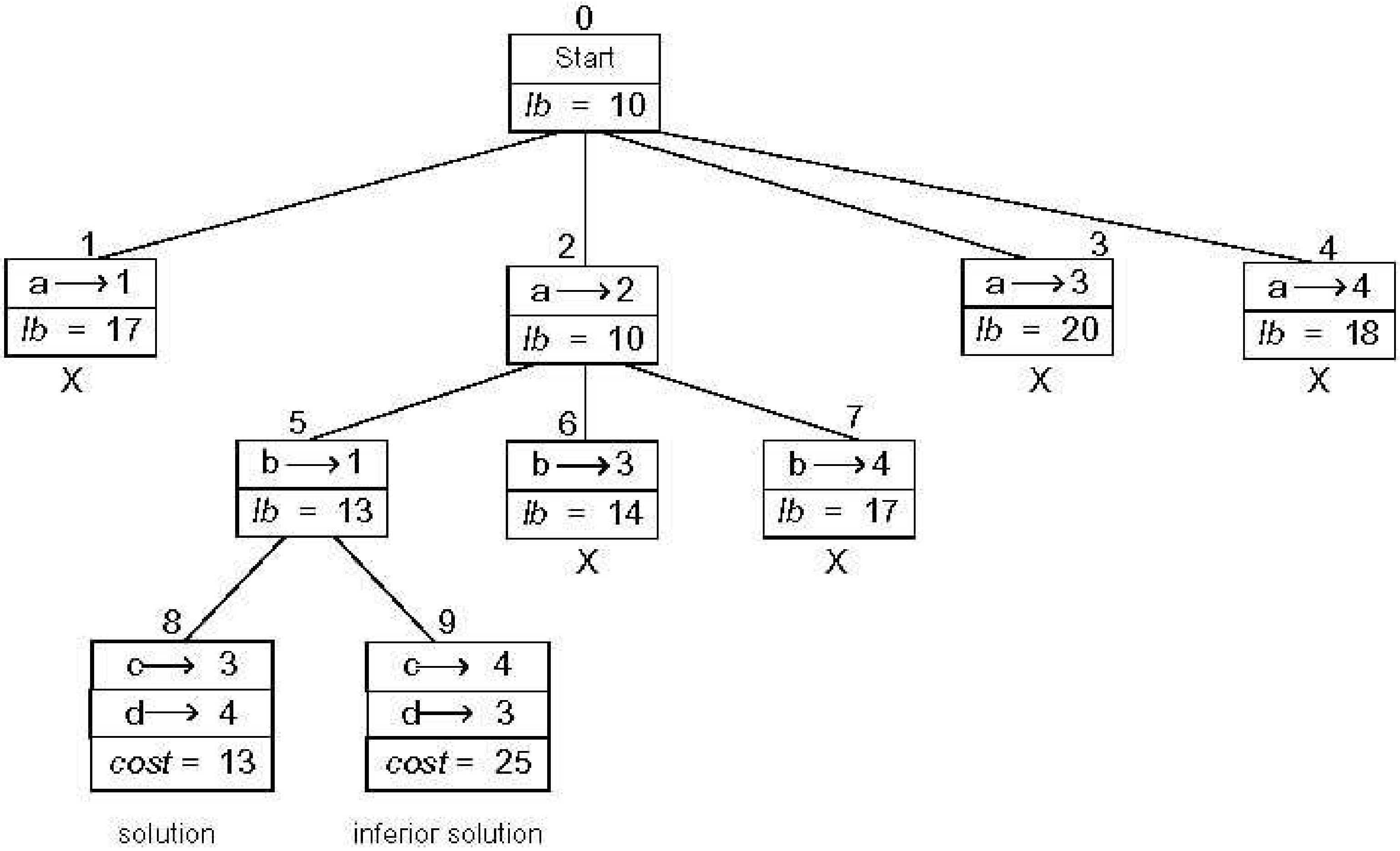


Figure 11.7 Complete state-space tree for the instance of the assignment problem solved with the best-first branch-and-bound algorithm

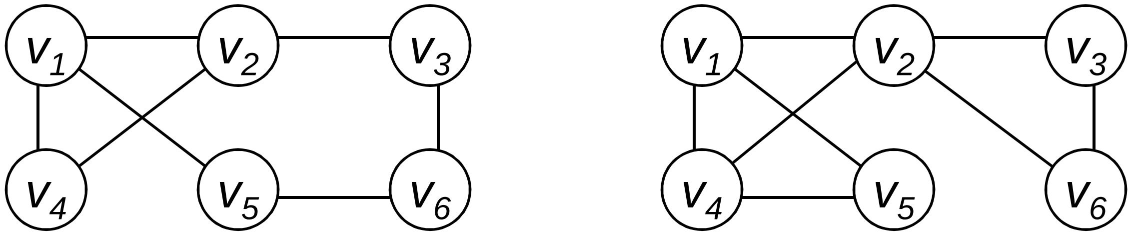
Summary: Branch and bound

- Feasible solution
- Optimal solution
- Breadth-First Search
- Best-First Search (with branch-and-bound pruning)

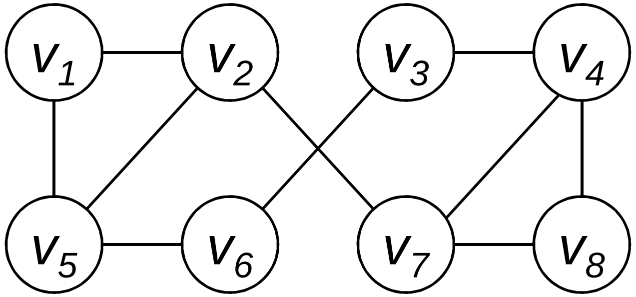
Backtracking - Hamiltonian Circuit Problem

- *A Hamiltonian circuit or tour of a graph is a path that starts at a given vertex, visits each vertex in the graph exactly once, and ends at the starting vertex.*
- *Some graphs do not contain Hamiltonian circuits.*

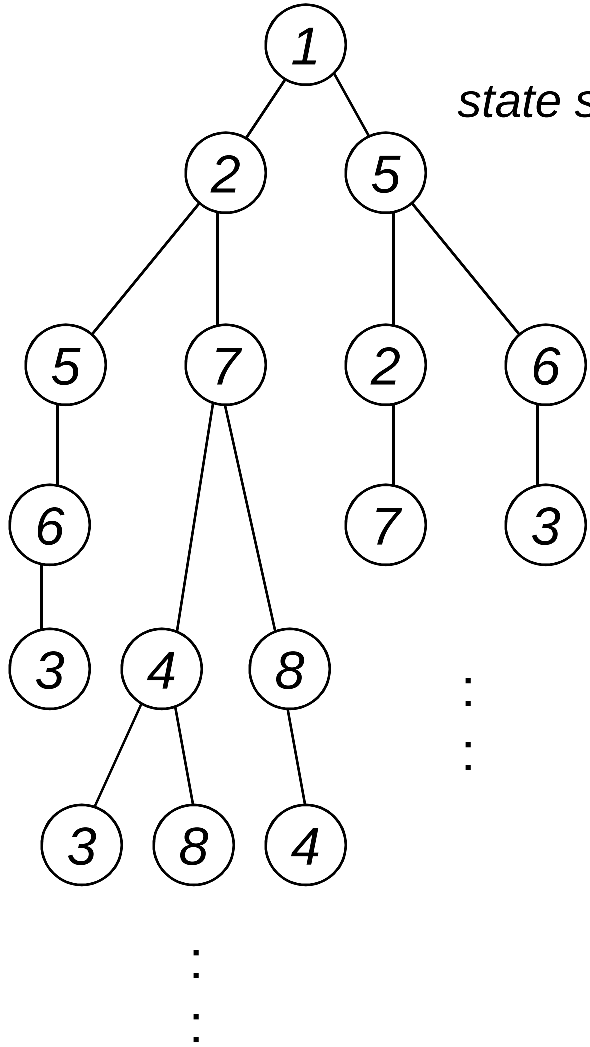
Hamiltonian Circuits Problem ?



Example



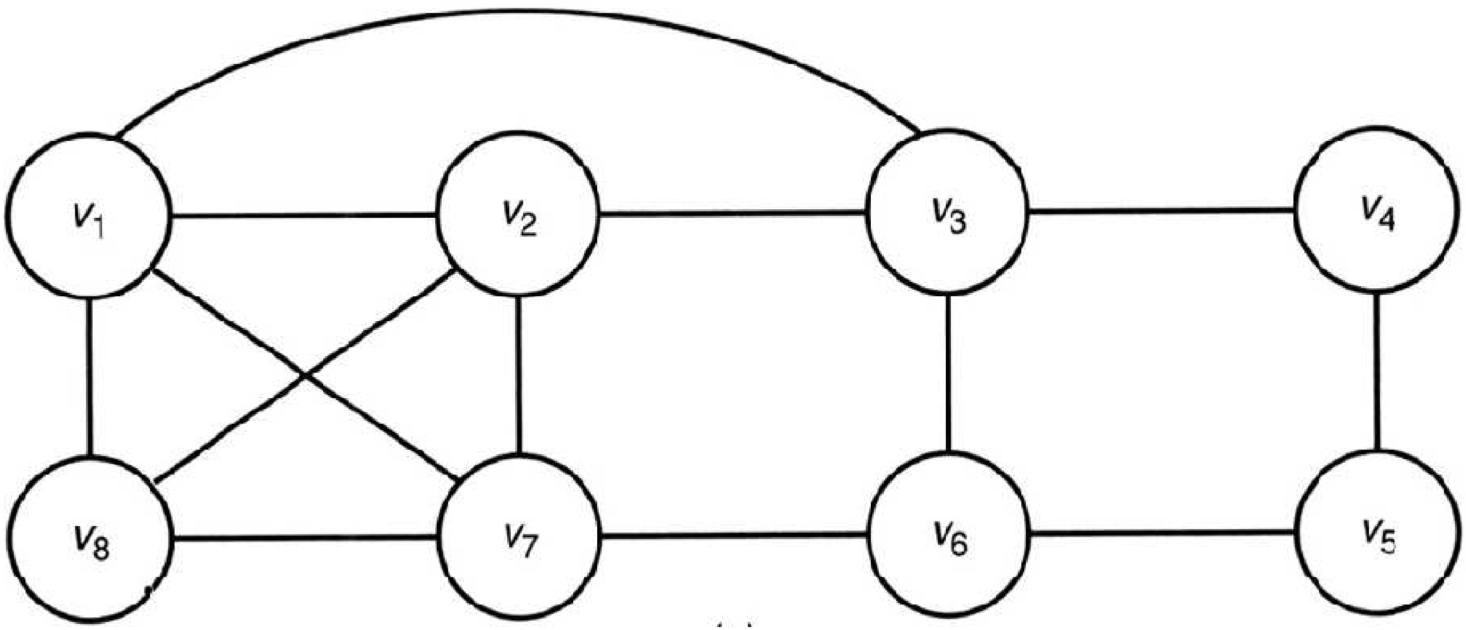
graph



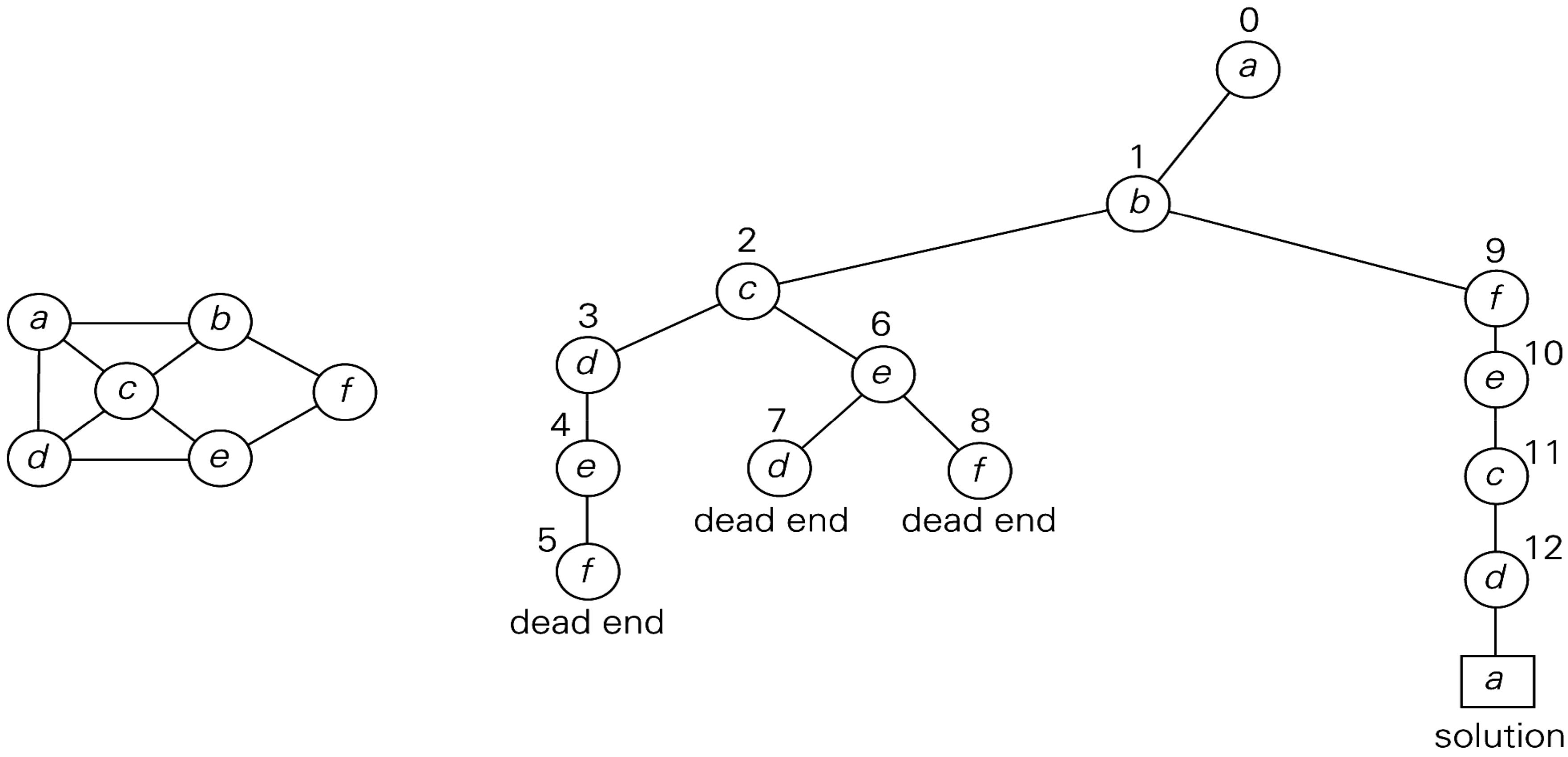
state space tree

Example

- Hamiltonian Circuit
  - [v1, v2, v8, v7, v6, v5, v4, v3, v1]



Example



(a) (b)

FIGURE 12.3 (a) Graph. (b) State-space tree for finding a Hamiltonian circuit. The numbers above the nodes of the tree indicate the order in which the nodes are generated.

Questions?