CHAPTER 5
Dealing with NP-Complete Problems

Tackling Difficult Combinatorial Problems

There are three principal approaches to tackling difficult combinatorial problems (NP-hard problems):

- Use exhaustive search
- Use a strategy that guarantees solving the problem exactly but doesn't guarantee to find a solution in polynomial time
- Use an approximation algorithm that can find an approximate (sub-optimal) solution in polynomial time

Knapsack Problem

Given n items:

- weights: w_1 w_2 ... w_n
- values: v_1 v_2 ... v_n
- a knapsack of capacity W

Find most valuable subset of the items that fit into the knapsack

Example: Knapsack capacity W=16

item	weight	value
1	2	$2
2	5	$30
3	10	$50
4	5	$10

Knapsack Problem by Exhaustive Search

Subset	Total weight	Total value
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	not feasible
{1,2,4}	12	$60
{1,3,4}	17	not feasible
{2,3,4}	20	not feasible
{1,2,3,4}	22	not feasible

Efficiency:

The Assignment Problem

There are *n* people who need to be assigned to *n* jobs, one person per job. The cost of assigning person *i* to job *j* is C[*i*,*j*]. Find an assignment that minimizes the total cost.

	Job 0	Job 1	Job 2	Job 3
Person 0	9	2	7	8
Person 1	6	4	3	7
Person 2	5	8	1	8
Person 3	7	6	9	4

Algorithmic Plan: Generate all legitimate assignments, compute their costs, and select the cheapest one.

How many assignments are there?

Pose the problem as the one about a cost matrix:

Assignment Problem by Exhaustive Search

$$C = \begin{matrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{matrix}$$

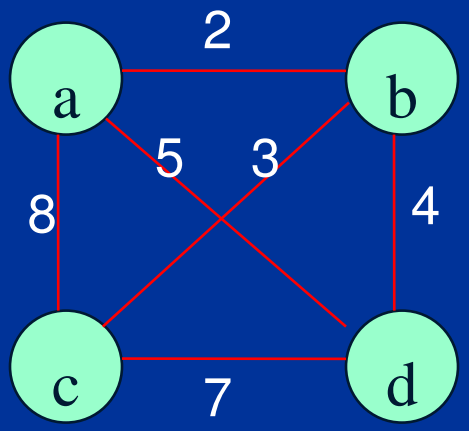
Assignment (col.#s)	Total Cost
1, 2, 3, 4	9+4+1+4=18
1, 2, 4, 3	9+4+8+9=30
1, 3, 2, 4	9+3+8+4=24
1, 3, 4, 2	9+3+8+6=26
1, 4, 2, 3	9+7+8+9=33
1, 4, 3, 2	9+7+1+6=23

etc.

(For this particular instance, the optimal assignment can be found by exploiting the specific features of the number given. It is:)

Travelling Salesman Problem

- Given *n* cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city
- Alternatively: Find shortest *Hamiltonian circuit* in a weighted connected graph
- Example:



TSP by Exhaustive Search

Tour	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17

More tours?
Less tours?
Efficiency:

Comments on Exhaustive Search

- Exhaustive-search algorithms run in a realistic amount of time only on very small instances

- In many cases, exhaustive search or its variation is the only known way to get exact solution

Branch-and-Bound

- For each node (partial solution) of a state-space tree, computes a bound on the value of the objective function for all descendants of the node (extensions of the partial solution)

- Uses the bound for:
  - ruling out certain nodes as “nonpromising” to prune the tree – if a node’s bound is not better than the best solution seen so far
  - guiding the search through state-space

Assignment Problem

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

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

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

Lower bound: Any solution to this problem will have total cost at least: 2 + 3 + 1 + 4 (or 5 + 2 + 1 + 4)

First two levels of the state-space tree

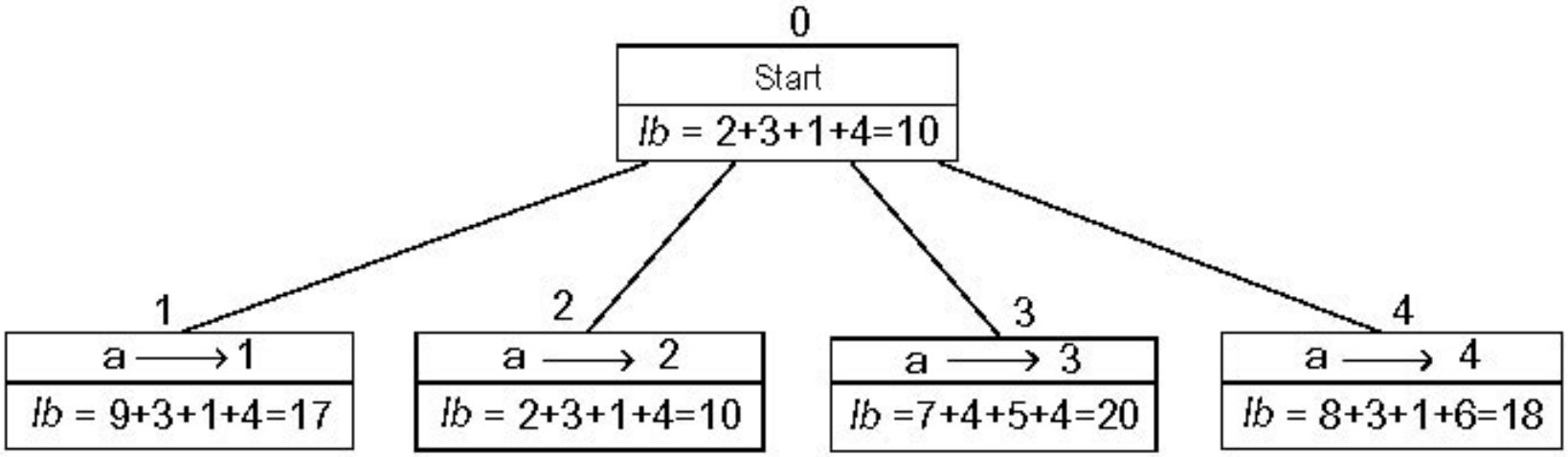


Figure 11.5 Levels 0 and 1 of the state-space tree for the instance of the assignment problem being solved with the best-first branch-and-bound algorithm. The number above a node shows the order in which the node was generated. A node's fields indicate the job number assigned to person a and the lower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

We assign a from col 1 to 4, but pick the minimum from the rest of the columns for b, c, d since b, c and d cannot from the same column as a.

Complete state-space tree

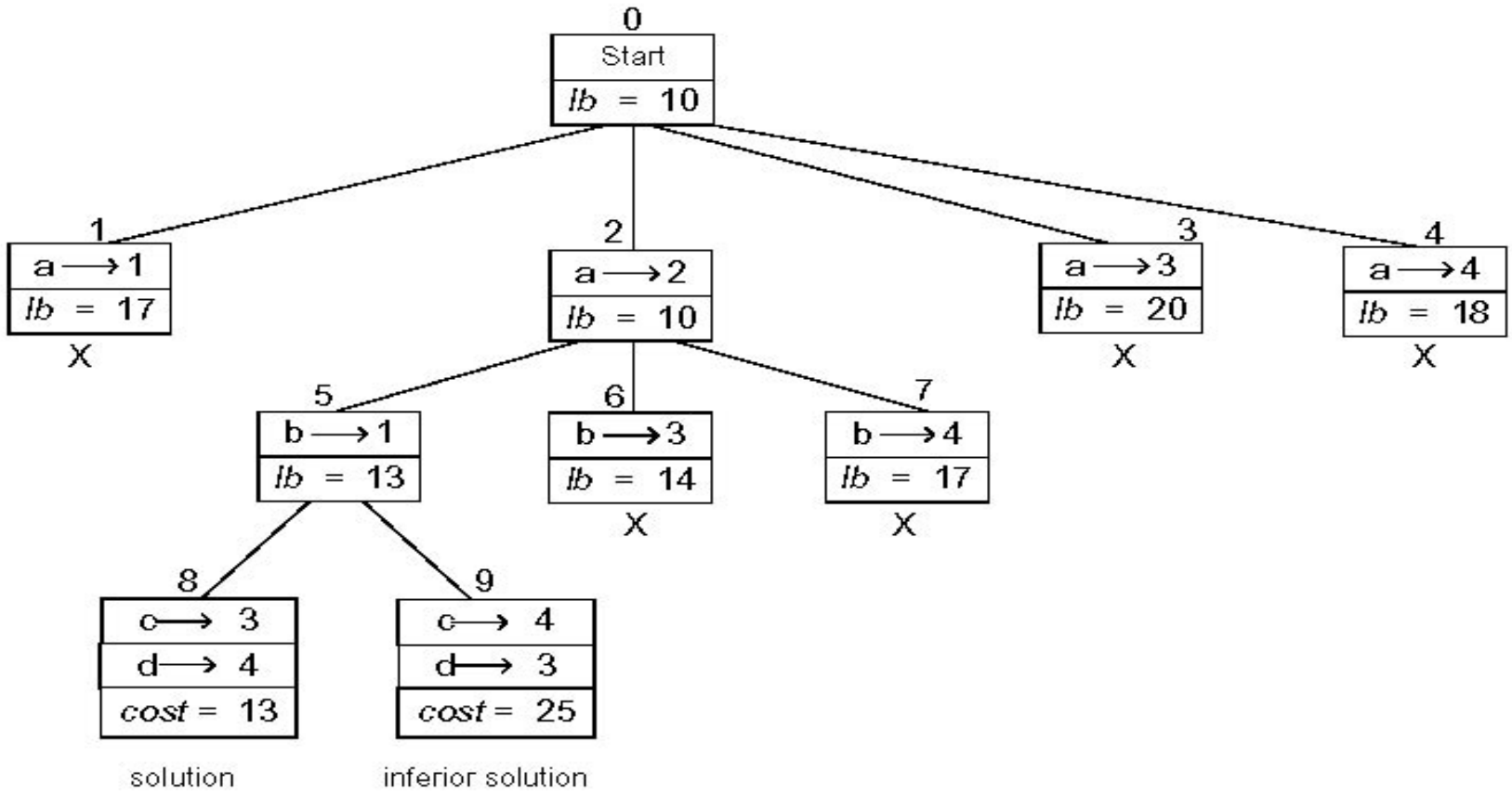
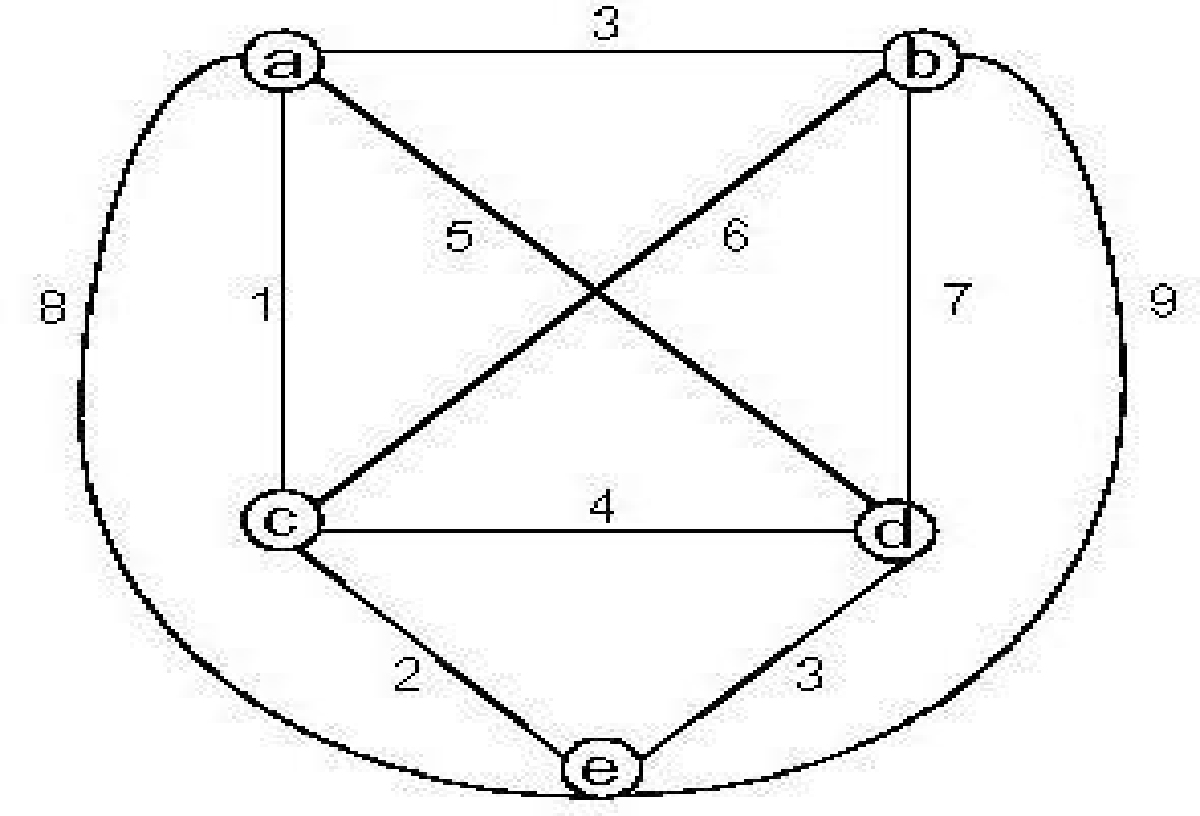


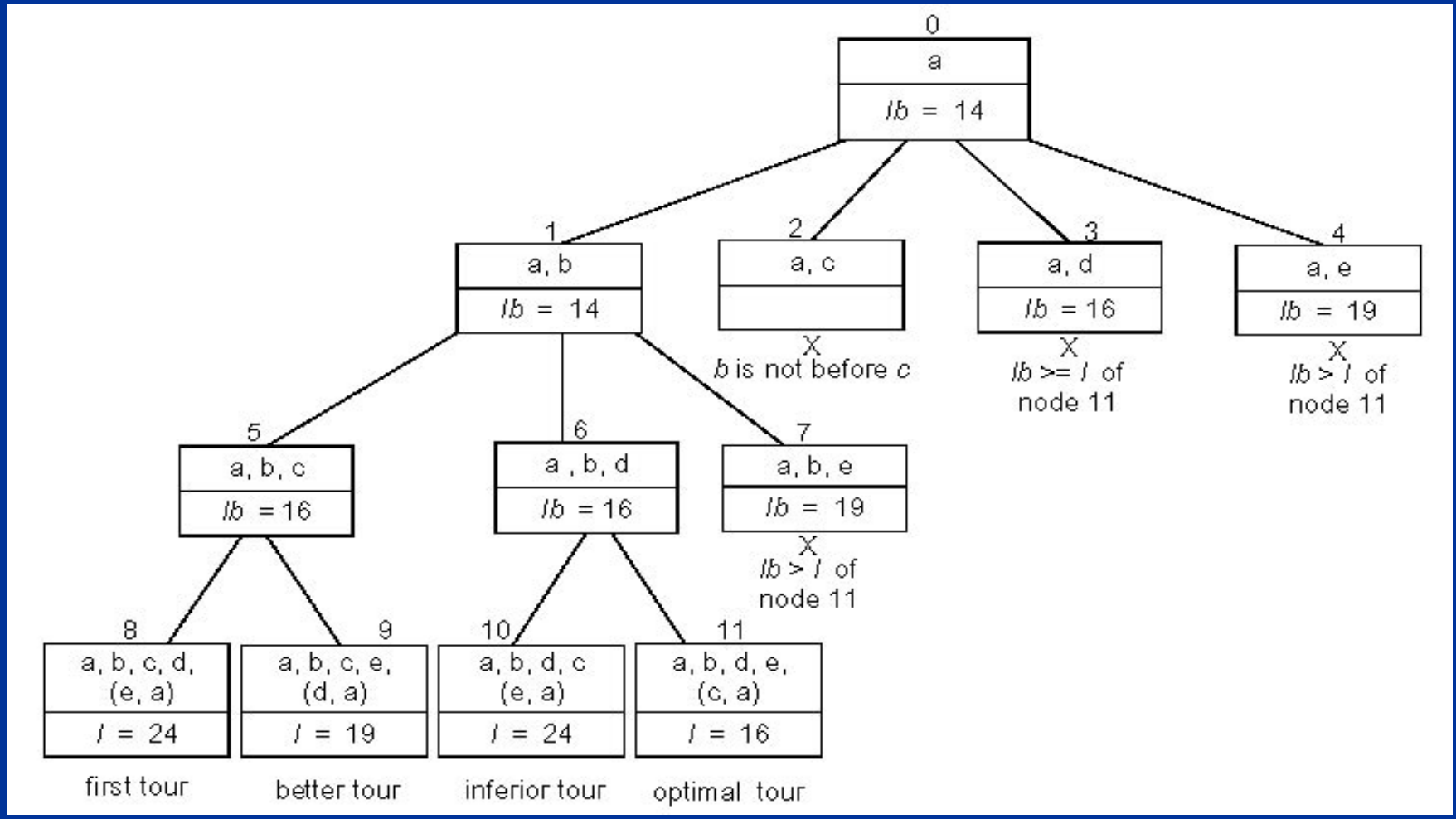
Figure 11.7 Complete state-space tree for the instance of the assignment problem solved with the best-first branch-and-bound algorithm

TSP

We will use the lower bound as
Let s = sum to two least weighted at the each vertex.
lb = s/2 is s is even, otherwise it is rounded up.



Travelling Salesman Problem



Note the a,c is not expanded because we consider tours in which b comes before c. If c comes before b, we can consider the reversed tour!

Knapsack problem

Use ub = v + (W – w) ($v_{(i+1)}/w_{(i+1)}$

where v is total value and w is the weight of items already selected.

Example: Knapsack capacity W=16

item	weight (w_i)	value(v_i)
1	2	$2
2	5	$30
3	10	$50
4	5	$10

Knapsack problem

Example: Knapsack capacity W=16

item	weight (w_i)	value(v_i)
1	2	$2
2	5	$30
3	10	$50
4	5	$10

Draw the space tree for the example above. start with v = 0 and w = 0 and at each level consider the branches either with or without item i.

Exercises

1. Assuming that each tour can be generated in constant time, what will be the time complexity class of the exhaustive search algorithm for TSP.

2. . Hamiltonian circuit is defined as a tour in an undirected graph that visits every vertex exactly once. Outline an exhaustive-search algorithm for finding Hamiltonian circuit.

3. Give an example of the best case input for the branch and bound algorithm for the assignment problem. In the best case, how many nodes will be in the state space tree of the branch and bound algorithm of the assignment problem.

Exercises

4. Solve the following instance of the knapsack problem by branch-and-bound algorithm.

item	Weight	value	
1	10	$100	
2	7	$63	W = 16
3	8	$56	
4	4	$12	

5. Suggest a more sophisticated bounding function for solving the knapsack problem.

Exercises

6. Apply branch and bound algorithm to solve the TSP problem on the following graph.