Branch and Bound Strategy

Compare to backtracking

- Very similar to backtrack in that a space tree is used to solve the problem.
- The complexity, in worst case, exponential
- The differences :
  - BnB doesn’t limit us to any particular way of traversing the tree
  - Used only for optimization problem

The idea behind the BnB

- BnB computes a number (bound) at a node to determine whether the node is promising.
- The number is a bound on the value of the solution that could be obtained beyond expanding the node
- If the bound is no better the the value of the best solution found so far, the node is nonpromising

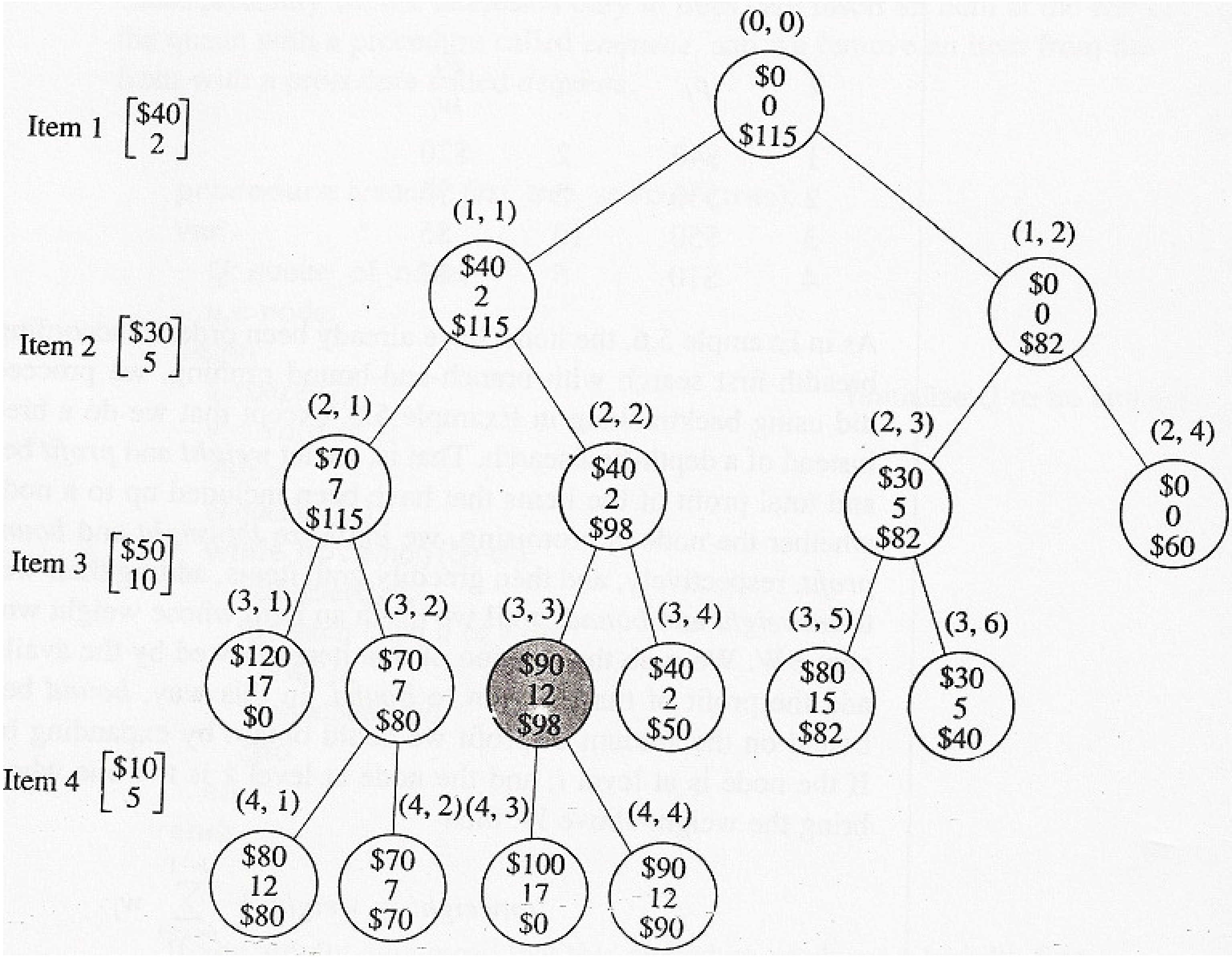
0/1 Knapsack : Case-1

i	Pi	wi	Pi/wi
1	$40	2	$20
2	$30	5	$6
3	$50	10	$5
4	$10	5	$2

Two strategies in traversing tree

- Breadth-first branch and bound
- Best-first branch and bound

Breadth First Search with BnB Pruning



1. Maxprofit = \$0
2. Visit (0,0)
   a. profit = \$0, weight = 0
   b. bound, i=0, and k=3 (because 2+5+7 = 17>16)

$$totweight = weight + \sum_{j=0+1}^{3-1} wj = 0 + 2 + 5 = 7$$

$$bound = profit + \sum_{j=0+1}^{3-1} Pj + (w - totweight) x (P3 / w3)$$

$$= \$0 + \$40 + \$30 + (16 - 7)x\$50/10 = \$115$$

   c. Promising ?

      weight < W and bound > maxprofit ➔promising

3. Visit (1,1)

a. profit = \$0 + \$40 = \$40, and weight = 0 + 2 = 2

b. maxprofit = \$40

c. bound, i=1, and k=3 (because 2+5+7 = 17>16)

$$totweight = weight + \sum_{j=1+1}^{3-1} wj = 2+5=7$$

$$bound = profit + \sum_{j=1+1}^{3-1} Pj + (w - totweight)x(P3/w3)$$

$$= \$40 + \$30 + (16 - 7)x\$50 / 10 = \$115$$

c. Promising ?

weight < W and bound > maxprofit ➔promising

4. Visit (1,2)

a. profit = \$0, and weight = 0

b. maxprofit = \$40

c. bound, i=1, and k=4 (because 5+10+5 = 20>16)

$$totweight = weight + \sum_{j=1+1}^{4-1} wj = 5+10=15$$

$$bound = profit + \sum_{j=1+1}^{4-1} Pj + (w - totweight)x(P4/w4)$$

$$= \$0 + \$30 + \$50 + (16 - 15)x\$10 / 5 = \$82$$

c. Promising ?

weight < W and bound > maxprofit ➔promising

General scheme of Breadth First

```
Procedure breadth_first_BnB(T: state_space_tree; var best : number)

Var
   Q : queue_of_node;
   u,v :node

Begin
   initialize(Q);
   v:=root of T;
   enqueue(Q,v);
   best := value(v);
    while not empty(Q) do
             dequeue(Q,v)
              for each u child of v do
                        if value(u) is greater than best than
                                  best := value(u)
                        if bound(u) is better than best than
                                  enqueue(Q,u)
end
```

Breadth First Algorithm

Problem: Let n items be given, where each item has a weight and a profit. The weights and profits are positive integers. Furthermore, let a positive integer W be given. Determine a set of items with maximum total profit, under the constraint that the sum of their weights cannot exceed W.

Inputs: positive integers n and W, arrays w and p, each containing n positive integers and sorted in nonincreasing order according to the value of $p[i]/w[i]$.

Outputs: an integer *maxprofit* that is the sum of the profits in an optimal set.

```
type
  node = record
    level: integer;          {the node's level in the tree}
    profit: integer;
    weight: integer
  end;
```

```
procedure knapsack2(n: integer;
                    p,w: array[1..n] of integer;
                    W: integer;
                    var maxprofit: integer);
var
  Q: queue_of_node;
  u,v: node;
begin
  initialize(Q);                                    {Initialize Q to be empty.}
  v.level:= 0; v.profit:= 0; v.weight:= 0;          {Initialize v to the root.}
  maxprofit:= 0;
  enqueue(Q,v);
  while not empty(Q) do
    dequeue(Q,v);
    u.level:= v.level+1;                            {Set u to a child of v.}
    u.weight:= v.weight + w[u.level];               {Set u to the child that}
    u.profit:= v.profit + p[u.level];               {includes the next item.}
    if u.weight ≤ W and u.profit > maxprofit then
      maxprofit:= u.profit
    end;
    if bound(u) > maxprofit then
      enqueue(Q,u)
    end;
    u.weight:= v.weight;                            {Set u to the child that}
    u.profit:= v.profit;                            {does not include the}
    if bound(u) > maxprofit then                    {next item.}
      enqueue(Q,u)
    end
  end
end;
```

```
function bound(u: node): real;
var
  j,k: index; totweight: integer;
begin
  if u.weight ≥ W then
    bound:= 0
  else
    bound:= u.profit;
    j:= u.level+1;
    totweight:= u.weight;
    while j ≤ n and totweight + w[j] ≤ W do          {Grab as many items as}
      totweight:= totweight + w[j];                   {possible.}
      bound:= bound + p[j];
      j:= j+1
    end;
    k:= j;                                            {Use k for consistency}
    if k ≤ n then                                     {with formula in text.}
      bound:= bound + (W − maxweight)*p[k]/w[k]       {Grab fraction of kth}
    end;                                              {item.}
  end
end;
```

Compare with depth first search

- Node (1,2) found promising
- Decision to visit node’s children is made at the time the node visited

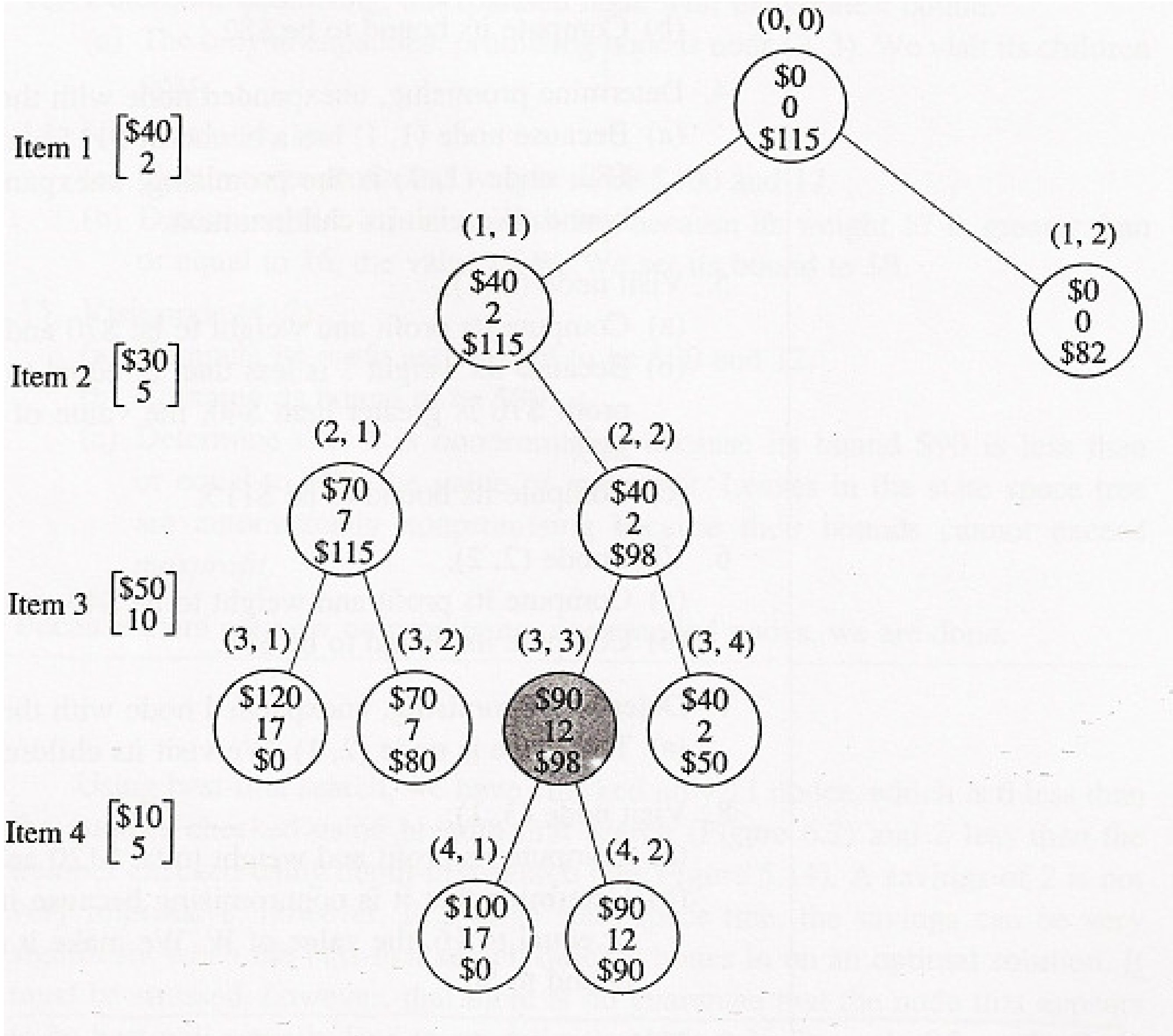
Exercise : 0/1 Knapsack

- Consider the following instance of the 0/1 Knapsack

  $n=4,\ W=19$

i	v_i	w_i	v_i/w_i
1	\$20	2	10
2	\$30	5	6
3	\$35	7	5
4	\$12	3	4
5	\$3	1	3

Best First with BnB Pruning



1. Visit (0,0)
   a. profit = \$0, weight = 0, maxprofit = \$0
   b. bound = \$115
2. Visit (1,1)
   a. profit = \$40, weight = 2, maxprofit = \$40
   b. bound = \$115
3. Visit (1,1)
   a. profit = \$0, weight = 0, maxprofit = \$40
   b. bound = \$82
4. Best bound = node (1,1)

5. Visit (2,1)
    a. profit = $70, weight = 7, maxprofit = $70
    b. bound = $115
6. Visit (2,2)
    a. profit = $40, weight = 2, maxprofit = $70
    b. bound = $98
7. Best bound = node (2,1)
8. Visit (3,1)
    a. profit = $120, weight = 17,maxprofit = $70
    b. bound = $0

9. Visit (3,2)
    a. profit = $70, weight = 7, maxprofit = $70
    b. bound = $80
10. Best bound = node (2,2)
11. Visit (3,3)
    a. profit = $90, weight = 12, maxprofit = $90
    b. bound = $98
    c. node (3,2) and node (1,2) nonpromising
12. Visit (3,4)
    a. profit = $40, weight = 2,maxprofit = $90
    b. bound = $50

13. Best bound = node(3,3)
14. Visit (4,1)
    a. profit = $100, weight = 17, maxprofit = $90
    b. bound = $0 → nonpromising

10. Visit (4,2)
    a. profit = $90, weight = 12, maxprofit = $90
    b. bound = $90 → nonpromising

General scheme of Best First

```
Procedure best_first_BnB(T: state_space_tree; var best : number)

Var
   PQ : Priority_queue_of_node;
   u,v :node

Begin
   initialize(Q);
   v:=root of T;
  best := value(v);
   insert(PQ,v)
   while not empty(PQ) do
            remove(PQ,v)
            if bound(v) is better than best then
                  for each u child of v do
                       if value(u) is greater than best than
                                  best := value(u)
                       if bound(u) is better than best than
                                  insert(PQ,u)
end
```

Best First Algorithm

```
type
  node = record
    level: integer;
    profit: integer;
    weight: integer;
    bound: real
  end;
```

```
procedure knopsack3(n: integer;
                    p,w: array[1..n] of integer;
                    W: integer;
                    var maxprofit: integer);
var
  PQ: priority_queue_of_node; u,v: node;
begin
  initialized(PQ);
  v.level:= 0; v.weight:= 0; v.profit:= 0;          {Initialize v to be the}
  maxprofit:= 0;                                     {root.}
  v.bound:= bound(v);
  insert(PQ,v);
  while not empty(PQ) do                             {Remove node with}
    remove(PQ,v);                                    {best bound.}
    if v.bound > maxprofit then                      {Check if node is still}
      u.level:= v.level+1;                           {promising.}
      u.weight:= v.weight + w[u.level];              {Set u to the child}
      u.profit:= v.profit + p[u.level];              {that includes the}
      if u.weight ≤ W and u.profit > maxprofit then  {next item.}
        maxprofit:= u.profit
      end;
      u.bound:= bound(u);
      if u.bound > maxprofit then
        insert(PQ,u)
      end;
      u.weight:= v.weight; u.profit:= v.profit;      {Set u to the child}
      u.bound:= bound(u);                            {that does not}
      if u.bound > maxprofit then                    {include the next}
        insert(PQ,u)                                 {item.}
      end
    end
  end
end;
```

notes

- Best first : 11 node, breadth first = 17 node, deepth first : 13 node
- No guarantee that node appears to be best will lead to an optimal solution

Traveling Salesperson Problem

- Goal : find shortest path in a directed graph that start at a given vertex, visit each vertex exactly once, and end up back at starting vertex
- Need to determine the lower bound on the length of any tour that can be obtained by expanding beyond a given node
- Promising, if bound is less than current minimum tour length
- Initially, set minimum tour length to ∞

Traveling Salesman Problem

Adjacency matrix

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

Traveling Salesman Problem

- Lower bound on the cost of leaving vertex v_1 is given by the minimum of all nonzero entries in row 1 of the adjacency matrix,
- Lower bound on the cost of leaving vertex v_2 is given by the minimum of all nonzero entries in row 2 of the adjacency matrix,
- And so on..

Traveling Salesman Problem

- Lower bound on the cost of leaving the five vertices are:

  v_1 *minimum* (14, 4, 10, 20) = 4

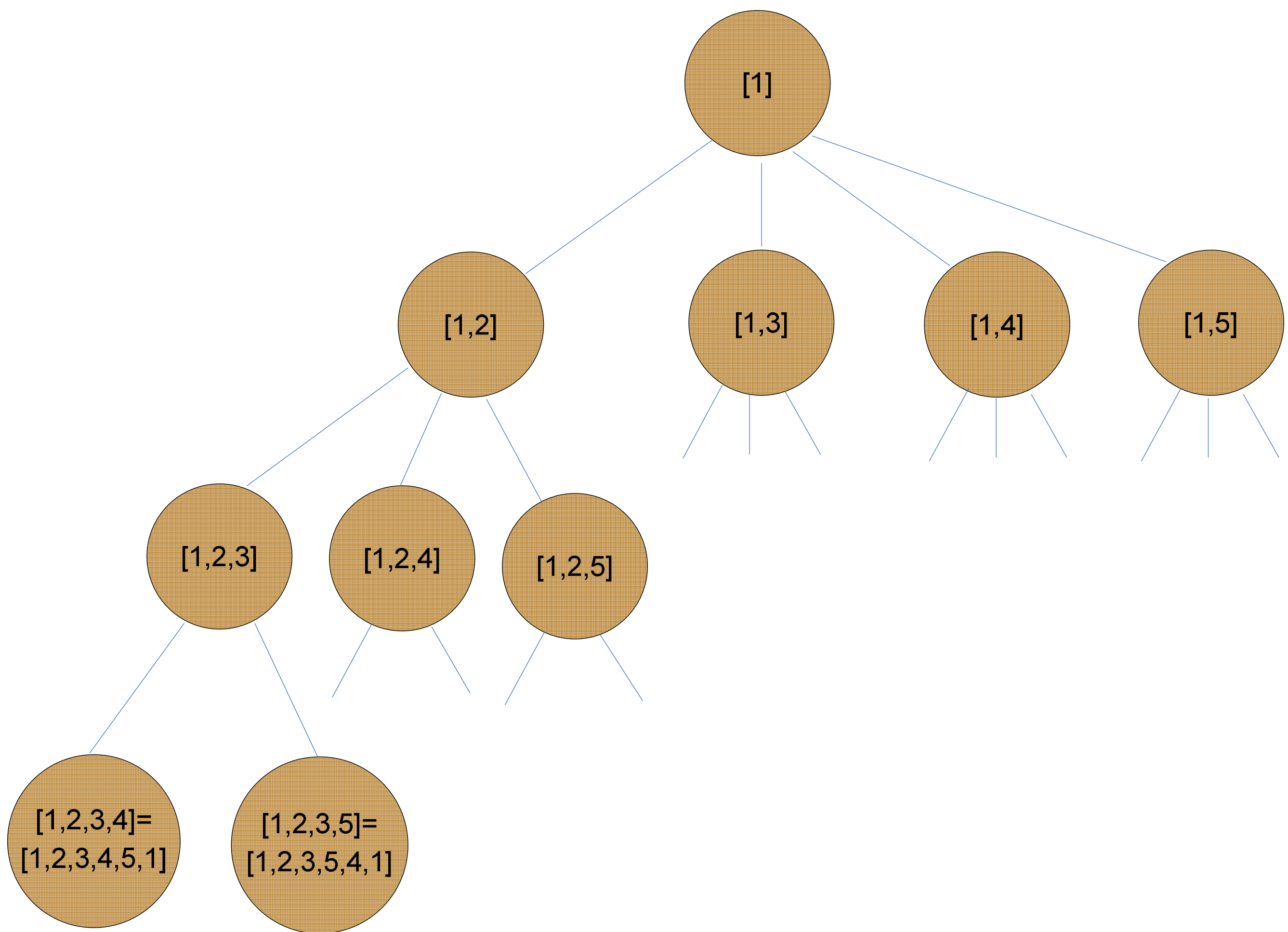
  v_2 *minimum* (14, 7, 8, 7) = 7

  v_3 *minimum* (4, 5, 7, 16) = 4

  v_4 *minimum* (11, 7, 9, 2) = 2

  v_5 *minimum* (18, 7, 17, 4) = 4
- The sum of these minimums is 21

Lower bound



Lower bound

- Lower bound on the node containing [1,2] :
  - The cost of getting to v_2 is 14
  - Obtain the minimum for v_2, it doesn't include the edge to v_1
  - Obtain the minimums for the other vertices it doesn't include v_2 because it's already been at v_2.

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

- Lower bound obtained by expanding beyond the node containing [1,2] is 14+7+4+2+4=31

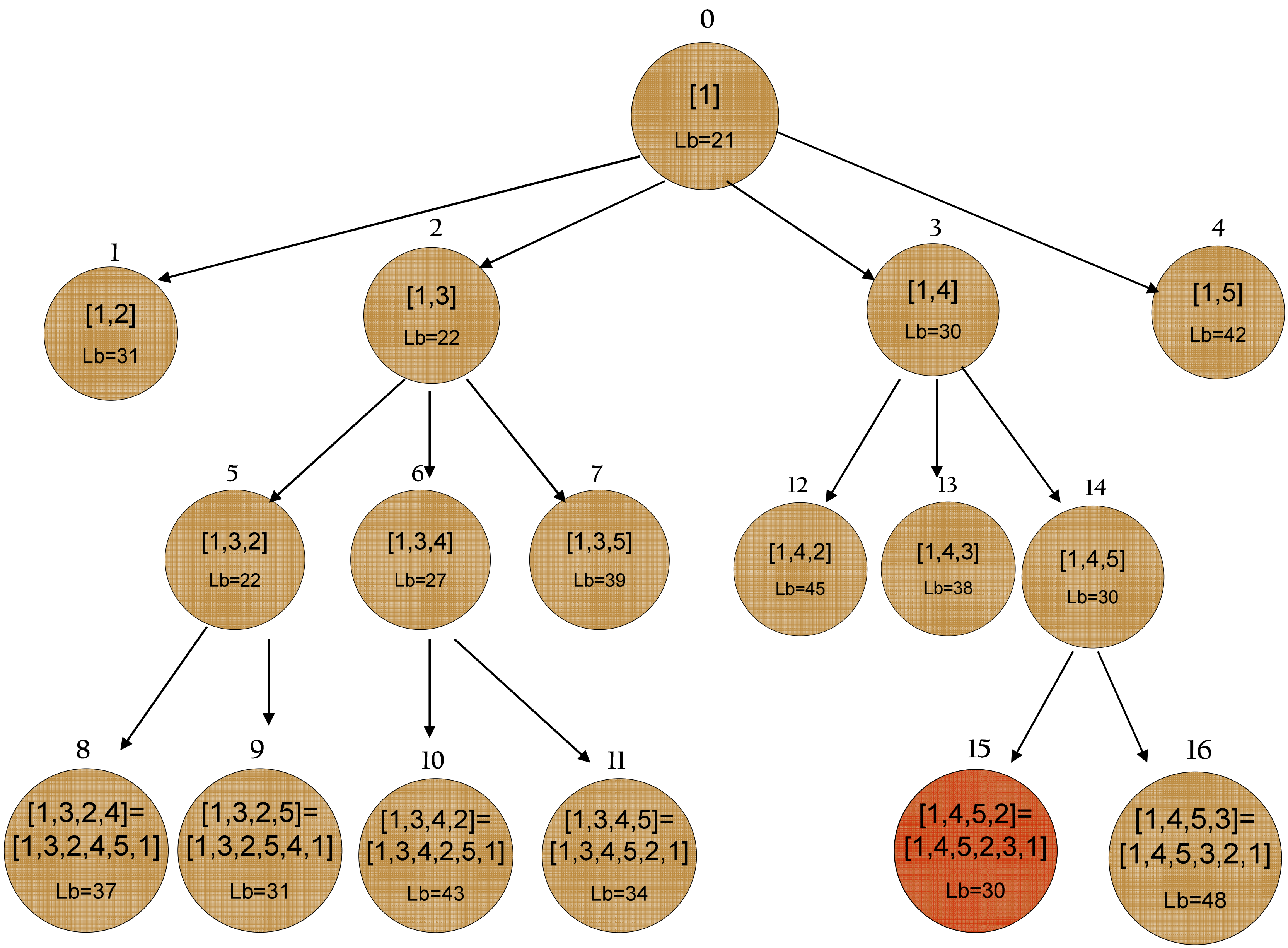
Lower bound

- Lower bound on the node containing [1,2,3]. Any tour obtained by expanding beyond this node has the following lower bound on the cost of leaving the vertices:

v_1	=	14
v_2	=	7
v_3 minimum(7, 16)	=	7
v_4 minimum(11, 2)	=	2
v_5 minimum(18, 4)	=	4

- The lower bound on the node [1,2,3] is 14+7+7+2+4=34

Best-first search with branch-and-bound pruning


0
[1]
Lb=21
1
[1,2]
Lb=31
2
[1,3]
Lb=22
3
[1,4]
Lb=30
4
[1,5]
Lb=42
5
[1,3,2]
Lb=22
6
[1,3,4]
Lb=27
7
[1,3,5]
Lb=39
12
[1,4,2]
Lb=45
13
[1,4,3]
Lb=38
14
[1,4,5]
Lb=30
8
[1,3,2,4]=
[1,3,2,4,5,1]
Lb=37
9
[1,3,2,5]=
[1,3,2,5,4,1]
Lb=31
10
[1,3,4,2]=
[1,3,4,2,5,1]
Lb=43
11
[1,3,4,5]=
[1,3,4,5,2,1]
Lb=34
15
[1,4,5,2]=
[1,4,5,2,3,1]
Lb=30
16
[1,4,5,3]=
[1,4,5,3,2,1]
Lb=48

TSP: an optimal tour