بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ــ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

طراحی الگوریتم‌ها

حسین فلسفین

نشانهٔ اول برای هرس کردن همانند قبل است:

***An obvious sign** that a node is nonpromising is that there is no capacity left in the knapsack for more items. Therefore, if $weight$ is the sum of the weights of the items that have been included up to some node, the node is nonpromising if $weight \geq W$. It is nonpromising even if weight equals W because, in the case of optimization problems, "promising" means that we should expand to the children.*

یک روش قوی‌تر برای محاسبهٔ کران

We first order the items in nonincreasing order according to the values of $\frac{p_i}{w_i}$, where w_i and p_i are the weight and profit, respectively, of the ith item. Suppose we are trying to determine whether a particular node is promising. No matter how we choose the remaining items, we cannot obtain a higher profit than we would obtain if we were allowed to use the restrictions in the Fractional Knapsack problem from this node on. (Recall that in this problem the thief can steal any fraction of an item taken.)

☞ ***Therefore, we can obtain an upper bound on the profit that could be obtained by expanding beyond that node as follows. Let*** $profit$ ***be the sum of the profits of the items included up to and let*** $weight$ ***be the sum of the weights of those items. We initialize variables*** $bound$ ***and*** $totweight$ ***to*** $profit$ ***and*** $weight$***, respectively.***
☞ ***Next we greedily grab items, adding their profits to*** $bound$ ***and their weights to*** $totweight$***, until we get to an item that if grabbed would bring*** $totweight$ ***above*** W***. We grab the fraction of that item allowed by the remaining weight, and we add the value of that fraction to bound.*** *If we are able to get only a fraction of this last weight, this node cannot lead to a profit equal to bound, but bound is still an upper bound on the profit we could achieve by expanding beyond the node.*

Suppose the node is at level i, and the node at level k is the one that would bring the sum of the weights above W. Then

$$totweight = weight + \sum_{j=i+1}^{k-1} w_j, \quad \text{and}$$

$$bound = \underbrace{\left(profit + \sum_{j=i+1}^{k-1} p_j\right)}_{\text{Profit from first } k-1 \text{ items taken}} + \underbrace{(W - totweight)}_{\text{Capacity available for } k\text{th item}} \times \underbrace{\frac{p_k}{w_k}}_{\text{Profit per unit weight for } k\text{th item}}.$$

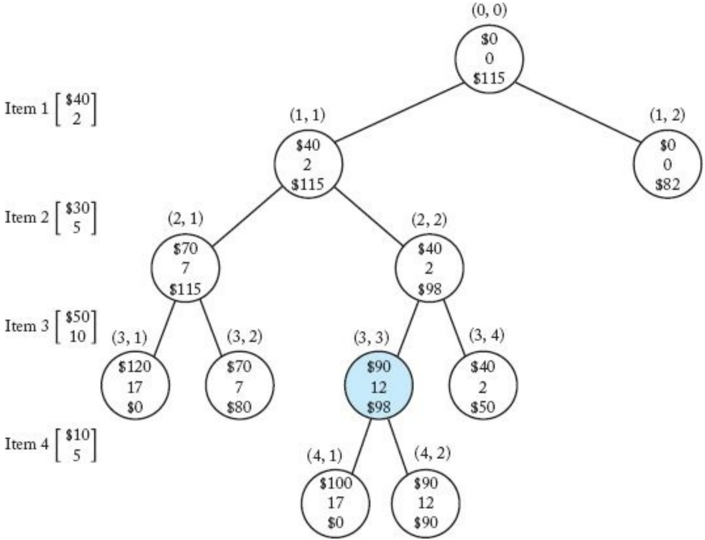
If $maxprofit$ is the value of the profit in the best solution found so far, then a node at level i is nonpromising if

$$bound \leq maxprofit.$$

We are using greedy considerations only to obtain a bound that tells us whether we should expand beyond a node. We are not using it to greedily grab items with no opportunity to reconsider later (as is done in the greedy approach).

Example: Suppose we have the following instance of the 0-1 Knapsack problem: $n = 4$, $W = 16$,

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



Item 1 [$40 2]
Item 2 [$30 5]
Item 3 [$50 10]
Item 4 [$10 5]
(0, 0) $0 0 $115
(1, 1) $40 2 $115
(1, 2) $0 0 $82
(2, 1) $70 7 $115
(2, 2) $40 2 $98
(3, 1) $120 17 $0
(3, 2) $70 7 $80
(3, 3) $90 12 $98
(3, 4) $40 2 $50
(4, 1) $100 17 $0
(4, 2) $90 12 $90

1. *Visit node (0, 0) (the root).*

(a) *Set its $profit$ and $weight$ to $\$0$ and 0.*

(b) *Compute its $bound$ to be $\$115$: Because $2 + 5 + 10 = 17$, and $17 > 16$, the value of W, the third item would bring the sum of the weights above W. Therefore, $k = 3$, and we have*

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

$$bound = profit + \sum_{j=0+1}^{3-1} p_j + (W - totweight) \times \frac{p_3}{w_3}$$

$$= \$0 + \$40 + \$30 + (16 - 7) \times \frac{\$50}{10} = \$115.$$

(c) *Set $maxprofit$ to 0.*

2. Visit node (1, 1).

(a) Compute its $profit$ and $weight$ to be $\$40$ and 2.

(b) Because its $weight$ 2 is less than or equal to 16, the value of W, and its profit $\$40$ is greater than $\$0$, the value of $maxprofit$, set $maxprofit$ to $\$40$.

(c) Compute its bound to be $\$115$.

3. Visit node (1, 2).

(a) Compute its $profit$ and $weight$ to be $\$0$ and 0.

(b) Compute its $bound$ to be $\$82$. (چرا؟)

4. Determine promising, unexpanded node with the greatest $bound$.

(a) Because node (1, 1) has a $bound$ of \$115 and node (1, 2) has a $bound$ of \$82, node (1, 1) is the promising, unexpanded node with the greatest $bound$. We visit its children next.

5. Visit node (2, 1).

(a) Compute its $profit$ and $weight$ to be $\$70$ and 7.

(b) Because its $weight$ 7 is less than or equal to 16, the value of W, *and its profit $\$70$ is greater than $\$40$, the value of $maxprofit$, set $maxprofit$ to $\$70$.*

(c) Compute its bound to be $\$115$.

6. Visit node (2, 2).

(a) Compute its $profit$ and $weight$ to be $\$40$ and 2.

(b) Compute its bound to be $\$98$.

7. Determine promising, unexpanded node with the greatest bound.

(a) That node is node (2, 1). We visit its children next.

8. Visit node (3, 1).

(a) Compute its $profit$ and $weight$ to be $\$120$ and 17.

(b) Determine that it is nonpromising because its $weight$ 17 is greater than or equal to 16, the value of W. We make it nonpromising by setting its bound to \$0.

9. Visit node (3, 2).

(a) Compute its $profit$ and $weight$ to be $\$70$ and 7.

(b) Compute its $bound$ to be $\$80$.

10. Determine promising, unexpanded node with the greatest bound.
(a) That node is node (2, 2). We visit its children next.
11. Visit node (3, 3).
(a) Compute its $profit$ and $weight$ to be $\$90$ and 12.
(b) Because its $weight$ 12 is less than or equal to 16, the value of W, and its $profit$ $\$90$ is greater than $\$70$, the value of $maxprofit$, set $maxprofit$ to $\$90$.
(c) At this point, nodes (1, 2) and (3, 2) become nonpromising because their bounds, $\$82$ and $\$80$ respectively, are less than or equal to $\$90$, the new value of $maxprofit$.
(d) Compute its $bound$ to be $\$98$.
13. Determine promising, unexpanded node with the greatest bound.
(a) The only unexpanded, promising node is node (3, 3). We visit its children next.
14. Visit node (4, 1).
(a) Compute its $profit$ and $weight$ to be $\$100$ and 17.

(b) Determine that it is nonpromising because its $weight$ 17 is greater than or equal to 16, the value of W. We set its $bound$ to $\$0$.

15. Visit node (4, 2).

(a) Compute its $profit$ and $weight$ to be $\$90$ and 12.

(b) Compute its $bound$ to be $\$90$.

(c) Determine that it is nonpromising because its $bound$ $\$90$ is less than or equal to $\$90$, the value of $maxprofit$. Leaves in the state space tree are automatically nonpromising because their bounds cannot exceed $maxprofit$.

Because there are now no promising, unexpanded nodes, we are done.

یک نکته دربارهٔ الگوریتم مبتنی بر راهبرد شاخه و کران برای مسئلهٔ کوله‌پشتی ۰ ـ ۱
(فرض کنید که از بین دو روش محاسبهٔ کران معرفی‌شده، از روش محاسبهٔ کران بهتر استفاده می‌کنیم):

The state space tree in the 0-1 Knapsack problem is the same as that in the Sum-of-Subsets problem. The number of nodes in that tree is

$$1 + 2 + 2^2 + 2^3 + \cdots + 2^n = 2^{n+1} - 1.$$

Our branch-and-bound algorithm checks all nodes in the state space tree for the following instance. For a given n*, let* $W = n$*, and*

$$\begin{cases} p_i = 1 \qquad w_i = 1, \qquad \textit{for } 1 \le i \le n-1, \\ p_n = n \qquad w_n = n. \end{cases}$$

The optimal solution is to take only the nth item, and this solution will not be found until we go all the way to the right to a depth of $n-1$ and then go left. Before the optimal solution is found, however, every non-leaf will be found to be promising, which means that all nodes in the state space tree will be checked.

Comparing the Dynamic Programming Algorithm and the Branch-and-Bound Algorithm for the 0-1 Knapsack Problem

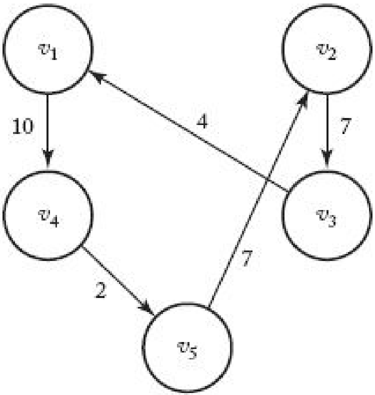
Recall that the worst-case number of entries that is computed by the dynamic programming algorithm for the 0-1 Knapsack problem is in $O(\text{minimum}(2^n, nW))$. In the worst case, the branch-and-bound algorithm checks $\Theta(2^n)$ nodes. Owing to the additional bound of nW, it may appear that the dynamic programming algorithm is superior. However, in branch-and-bound algorithms the worst case gives little insight into how much checking is usually saved by branch-and-bound. With so many considerations, it is difficult to analyze theoretically the relative efficiencies of the two algorithms. In cases such as this, the algorithms can be compared by running them on many sample instances and seeing which algorithm usually performs better. Horowitz and Sahni (1978) did this and found that the branch-and-bound algorithm is usually more efficient than the dynamic programming algorithm.

The Traveling Salesperson Problem (Again!)

Recall that the goal in this problem 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. Because it does not matter where we start, the starting vertex can simply be the first vertex (v_1).

Adjacency matrix representation of a graph that has an edge from every vertex to every other vertex (left), and the nodes in the graph and the edges in an optimal tour (right):

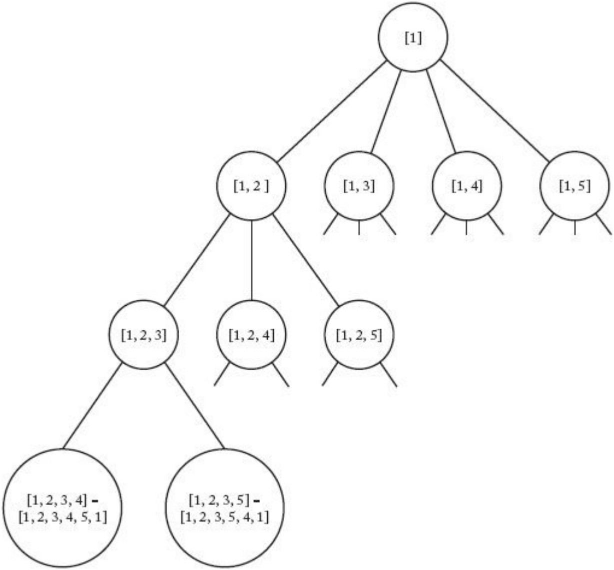
$$\begin{bmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{bmatrix}$$



State space tree

An obvious state space tree for this problem is one in which each vertex other than the starting one is tried as the first vertex (after the starting one) at level 1, each vertex other than the starting one and the one chosen at level 1 is tried as the second vertex at level 2, and so on.

State space tree



⁎ *In what follows, the term "node" means a node in the state space tree, and the term "vertex" means a vertex in the graph.*

⁎ *A node that is not a leaf represents all those tours that start with the path stored at that node. For example, the node containing [1, 2, 3] represents all those tours that start with the path [1, 2, 3]. That is, it represents the tours [1, 2, 3, 4, 5, 1] and [1, 2, 3, 5, 4, 1].*

⁎ *Each leaf represents a tour.*

⁎ *We need to find a leaf that contains an optimal tour.*

⁎ *We stop expanding the tree when there are four vertices in the path stored at a node because, at that time, the fifth one is uniquely determined. For example, the far-left leaf represents the tour [1, 2, 3, 4, 5, 1] because once we have specified the path [1, 2, 3, 4], the next vertex must be the fifth one.*

BOUND?

ما با یک مسئلهٔ کمینه سازی مواجه هستیم. لذا در هر نود از درخت فضای حالت نیاز به یک کران پایین داریم.

In this problem, we need to determine a lower bound on the length of any tour that can be obtained by expanding beyond a given node.

We call the node promising only if its bound is less than the current minimum tour length (best-so-far or incumbent).

We can obtain a bound as follows:

In any tour, the length of the edge taken when leaving a vertex must be at least as great as the length of the shortest edge emanating from that vertex. Therefore, a lower bound on the cost (length of the edge taken) of leaving vertex v_1 is given by the minimum of all the nonzero entries in row 1 of the adjacency matrix, a lower bound on the cost of leaving vertex v_2 is given by the minimum of all the nonzero entries in row 2, and so on.

A lower bound on the length of a tour

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

$$\begin{array}{lll} v_1 & minimum\,(14, 4, 10, 20) & = 4 \\ v_2 & minimum\,(14, 7, 8, 7) & = 7 \\ v_3 & minimum\,(4, 5, 7, 16) & = 4 \\ v_4 & minimum\,(11, 7, 9, 2) & = 2 \\ v_5 & minimum\,(18, 7, 17, 4) & = 4 \end{array}$$

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

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

This is not to say that there is a tour with this length. Rather, it says that there can be no tour with a shorter length.

Suppose we have visited the node containing [1, 2]. In that case we have already committed to making v_2 the second vertex on the tour, and the cost of getting to v_2 is the weight on the edge from v_1 to v_2, which is 14. Any tour obtained by expanding beyond this node, therefore, has the following lower bounds on the costs of leaving the vertices:

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

$$\begin{array}{lll} v_1 & & 14 \\ v_2 & minimum\,(7,8,7) & = 7 \\ v_3 & minimum\,(4,7,16) & = 4 \\ v_4 & minimum\,(11,9,2) & = 2 \\ v_5 & minimum\,(18,17,4) & = 4 \end{array}$$

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

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

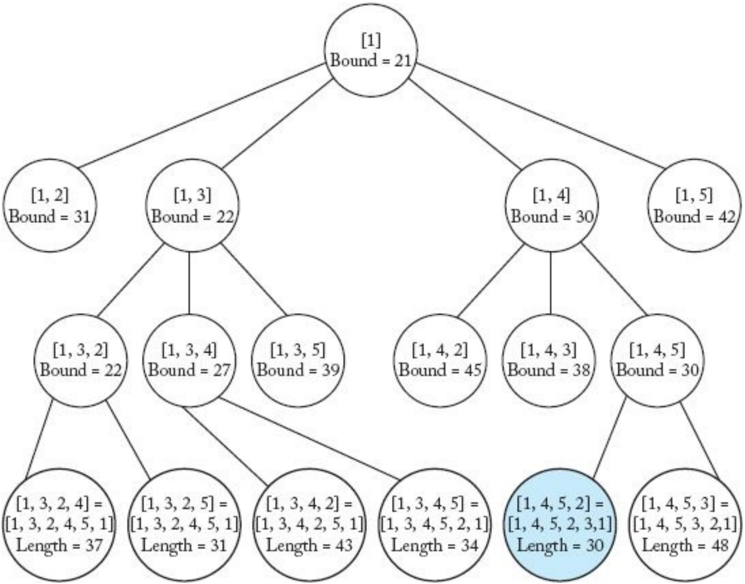
Suppose we have visited the node containing [1, 2, 3]. We have committed to making v_2 the second vertex and v_3 the third vertex. Any tour obtained by expanding beyond this node has the following lower bounds on the costs of leaving the vertices:

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

$$\begin{array}{lr} v_1 & 14 \\ v_2 & 7 \\ v_3 \quad minimum\,(7,16) = & 7 \\ v_4 \quad minimum\,(11,2) = & 2 \\ v_5 \quad minimum\,(18,4) = & 4 \end{array}$$

To obtain the minimums for v_4 and v_5 we do not consider the edges to v_2 and v_3, because we have already been to these vertices. The lower bound on the length of any tour we could obtain by expanding beyond the node containing [1, 2, 3] is

$$14 + 7 + 7 + 2 + 4 = 34.$$


[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

1. *Visit node containing [1] (the root).*

(a) Compute bound to be 21. (This is a lower bound on the length of a tour.)

(b) Set $minlength$ to $+\infty$. (best-so-far or incumbent)

REMARK: We initialize the value of the best solution to $+\infty$ (infinity) because there is no candidate solution at the root. (Candidate solutions exist only at leaves in the state space tree.)

2. *Visit node containing [1, 2].*

(a) Compute bound to be 31.

3. *Visit node containing [1, 3].*

(a) Compute bound to be 22.

4. *Visit node containing [1, 4].*

(a) Compute bound to be 30.

5. Visit node containing [1, 5].

(a) Compute bound to be 42.

6. Determine promising, unexpanded node with the smallest bound.

(a) That node is the node containing [1, 3]. We visit its children next.

7. Visit node containing [1, 3, 2].

(a) Compute bound to be 22.

8. Visit node containing [1, 3, 4].

(a) Compute bound to be 27.

9. Visit node containing [1, 3, 5].

(a) Compute bound to be 39.

10. Determine promising, unexpanded node with the smallest bound.

(a) That node is the node containing [1, 3, 2]. We visit its children next.

11. Visit node containing [1,3,2,4].

(a) Because this node is a leaf, compute tour length to be 37.
(b) Because its length 37 is less than $+\infty$, the value of $minlength$, set $minlength$ to 37.
(c) The nodes containing [1, 5] and [1, 3, 5] become nonpromising because their bounds 42 and 39 are greater than or equal to 37, the new value of $minlength$.

12. Visit node containing [1, 3, 2, 5].
(a) Because this node is a leaf, compute tour length to be 31.
(b) Because its length 31 is less than 37, the value of $minlength$, set $minlength$ to 31. (c) The node containing [1, 2] becomes nonpromising because its bound 31 is greater than or equal to 31, the new value of $minlength$.

13. Determine promising, unexpanded node with the smallest bound.
(a) That node is the node containing [1, 3, 4]. We visit its children next.

14. Visit node containing [1, 3, 4, 2].
(a) Because this node is a leaf, compute tour length to be 43.
15. Visit node containing [1, 3, 4, 5].
(a) Because this node is a leaf, compute tour length to be 34.
16. Determine promising, unexpanded node with the smallest bound.
(a) The only promising, unexpanded node is the node containing [1, 4]. We visit its children next.
17. Visit node containing [1, 4, 2].
(a) Compute bound to be 45.
(b) Determine that the node is nonpromising because its bound 45 is greater than or equal to 31, the value of $minlength$*.*
18. Visit node containing [1, 4, 3].
(a) Compute bound to be 38.
(b) Determine that the node is nonpromising because its bound 38 is greater than or equal to 31, the value of $minlength$*.*
19. Visit node containing [1, 4, 5].

(a) Compute bound to be 30.

20. Determine promising, unexpanded node with the smallest bound.

(a) The only promising, unexpanded node is the node containing [1, 4, 5]. We visit its children next.

21. Visit node containing [1, 4, 5, 2].

(a) Because this node is a leaf, compute tour length to be 30.

(b) Because its length 30 is less than 31, the value of $minlength$, set $minlength$ to 30.

22. Visit node containing [1, 4, 5, 3].

(a) Because this node is a leaf, compute tour length to be 48.

23. Determine promising, unexpanded node with the smallest bound.

(a) There are no more promising, unexpanded nodes. We are done.

We have determined that the node containing [1, 4, 5, 2], which represents the tour [1,4, 5, 2, 3, 1], contains an optimal tour, and that the length of an optimal tour is 30. There are 17 nodes in the tree, whereas the number of nodes in the entire state space tree is $1 + 4 + 4 \times 3 + 4 \times 3 \times 2 = 41$.

A problem does not necessarily have a unique bounding function. In the Traveling Salesperson problem, for example, we could observe that every vertex must be visited exactly once, and then use the minimums of the values in the columns in the adjacency matrix instead of the minimums of the values in the rows.

When two or more bounding functions are available, one bounding function may produce a better bound at one node whereas another produces a better bound at another node. When this is the case, the algorithm can compute bounds using all available bounding functions, and then use the best bound.

Generally, our goal is not to visit as few nodes as possible, but rather to maximize the overall efficiency of the algorithm. The extra computations done when using more than one bounding function may not be canceled out by the savings realized by visiting fewer nodes.

A branch-and-bound algorithm might solve one large instance efficiently but check an exponential (or worse) number of nodes for another large instance.