Chapter 12: Coping with the Limitations of Algorithm Power

There are two principal approaches to tackling NP-hard problems or other “intractable” problems:

• 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
Exact solutions

The exact solution approach includes the strategies:

- **Exhaustive search** (brute force)
  - useful only for small instances

- **Dynamic programming**
  - Applicable for some problems, e.g., knapsack problem, TSP

- **Backtracking**
  - eliminates some cases from consideration
  - yields solutions in reasonable time for many instances but worst case is still exponential

- **Branch-and-bound**
  - Only applicable for optimization problems
  - further cuts down on the search
  - fast solutions for most instances
  - worst case is still exponential

Need a state-space tree

Nodes: partial solutions
Edges: choices in completing solutions
Backtracking

Construct the **state-space tree**:  
• nodes: partial solutions  
• edges: choices in completing solutions

Explore the state space tree using **depth-first search (DFS)**

“Prune” non-promising subtrees  
• DFS stops exploring subtree rooted at nodes leading to no solutions and  
• “backtracks” to its parent node
Branch and Bound

An enhancement of backtracking.

Applicable to optimization problems

Uses a lower bound for the value of the objective function for each node (partial solution) to:

- no solution can beat the lower bound
- guide the search through state-space
- rule out certain branches as “unpromising” – do not explore these subtrees
- using a “best-first” rule
Example: The assignment problem

For example:

<table>
<thead>
<tr>
<th></th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
<th>Job 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person a</td>
<td>9</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Person b</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Person c</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Person d</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

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

If using exhaustive search, the permutation of $n$ persons $\Theta(n!)$
**Example: The assignment problem**

<table>
<thead>
<tr>
<th></th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
<th>Job 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person a</td>
<td>9</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Person b</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Person c</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Person d</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

**Lower bound:** Any solution to this problem will have total cost of **at least:** The summation of the smallest elements in each row

No solution can beat the lower bound!
Assignment problem: lower bounds

Most promising so far
State-space levels 0, 1, 2
Complete state-space
Traveling salesman example:

How to find the lower bound for each step?

\[
lb = \left\lfloor \sum_{i=1}^{N} \left( \min_{1} e_i + \min_{2} e_i \right) / 2 \right\rfloor
\]

for \( N \) nodes

Constraints:
- start from \( a \)
- \( b \) should be visited before \( c \)
- After visiting \( n-1 \) vertices, the last vertex must be visited and go back to \( a \)
Traveling salesman example:

\[
\left\lfloor \frac{(1+3)+(3+6)+(1+2)+(3+4)+(2+3)}{2} \right\rfloor
\]

\[a \ (lb=14)\]
Traveling salesman example:

\[ \frac{1}{2} \left[ (1+3) + (3+6) + (1+2) + (3+4) + (2+3) \right] / 2 \]
Traveling salesman example:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Does not satisfy the constraint \( b \) should be visited before \( c \)
Traveling salesman example:

\[
\begin{bmatrix}
    a & b & c & d & e \\
    0 & 3 & 1 & 5 & 8 \\
    3 & 0 & 6 & 7 & 9 \\
    1 & 6 & 0 & 4 & 2 \\
    5 & 7 & 4 & 0 & 3 \\
    8 & 9 & 2 & 3 & 0
\end{bmatrix}
\]

\[
\left( (1+5) + (3+6) + (1+2) + (3+5) + (2+3) \right) / 2
\]

Not promising!
Traveling salesman example:

\[
\frac{1}{2} \left( (1+8) + (3+6) + (1+2) + (3+4) + (2+8) \right)
\]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

a-e

Most promising! a-b (lb=14)
Not promising! a-c
Not promising! a-d (lb=16)
Not promising! a-e (lb=19)

a-b is the most promising, explore it
**Traveling salesman example:**

\[
\begin{bmatrix}
\begin{array}{ccccc}
0 & 3 & 1 & 5 & 8 \\
3 & 0 & 6 & 7 & 9 \\
1 & 6 & 0 & 4 & 2 \\
5 & 7 & 4 & 0 & 3 \\
8 & 9 & 2 & 3 & 0 \\
\end{array}
\end{bmatrix}
\]

\[
\frac{\left[ (1+3) + (3+6) + (1+6) + (3+4) + (2+3) \right]}{2}
\]

\[
a-b (lb=14) \quad b-a \quad b-c \quad c-b
\]

\[
a (lb=14) \quad a-b \quad (lb=14) \quad a-c \quad a-d (lb=16) \quad a-e (lb=19)
\]

\[
a-b-c (lb=16)
\]
Traveling salesman example:

```
  a  b  c  d  e
a  0  3  1  5  8
b  3  0  6  7  9
c  1  6  0  4  2
d  5  7  4  0  3
e  8  9  2  3  0
```

*a-b-c-d-e-a*

```
a-b (lb=14)  a-c  a-d (lb=16)  a-e (lb=19)
```

```
a-b-c (lb=16)
```

```
a-b-c-d-e-a
=3+6+4+3+8=24
```
Traveling salesman example:
Discussion on TSP using Branch-Bound

For every node except the n-1th vertex, we need to compute its corresponding lower bound.

For the n-1th vertex, we need to compute the total length

What operations we need?

- Find the minimum cost of each row
- Calculate the summations
- Compare with the best partial solution so far

Can we improve the efficiency?
Yes. Just update the cost involving the change.