Algorithm Analysis

Dr. Chengwei Lei
CEECS
California State University, Bakersfield
Brute-Force Polynomial Evaluation

Problem: Find the value of polynomial
\[ p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 \]
at a point \( x = x_0 \)

Brute-force algorithm

\[
\begin{align*}
p & \leftarrow 0.0 \\
\text{for } & \ i \leftarrow n \text{ downto } 0 \text{ do} \\
& \quad \text{power} \leftarrow 1 \\
& \quad \text{for } \ j \leftarrow 1 \text{ to } i \text{ do} \quad //\text{compute } x^i \\
& \quad & \quad \text{power} \leftarrow \text{power} \ast x \\
& \quad & \quad p \leftarrow p + a[i] \ast \text{power} \\
\text{return } & \ p
\end{align*}
\]

Efficiency:
Polynomial Evaluation: Improvement

We can do better by evaluating from right to left:

Better brute-force algorithm

```plaintext
p ← a[0]
power ← 1
for i ← 1 to n do
    power ← power * x
    p ← p + a[i] * power
return p
```

Efficiency:
Closest-Pair Problem

Find the two closest points in a set of $n$ points (in the two-dimensional Cartesian plane).

**Brute-force algorithm**

Compute the distance between every pair of distinct points and return the indexes of the points for which the distance is the smallest.
Closest-Pair Brute-Force Algorithm (cont.)

```
ALGORITHM BruteForceClosestPoints(P)

// Input: A list P of n (n ≥ 2) points P₁ = (x₁, y₁), ..., Pₙ = (xₙ, yₙ)
// Output: Indices index1 and index2 of the closest pair of points

dmin ← ∞
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        d ← sqrt((xᵢ - xⱼ)² + (yᵢ - yⱼ)²) // sqrt is the square root function
        if d < dmin
            dmin ← d; index1 ← i; index2 ← j

return index1, index2
```

**Efficiency:**

**How to make it faster?**
Brute-Force Strengths and Weaknesses

- **Strengths**
  - wide applicability
  - simplicity
  - yields reasonable algorithms for some important problems (e.g., matrix multiplication, sorting, searching, string matching)

- **Weaknesses**
  - rarely yields efficient algorithms
  - some brute-force algorithms are unacceptably slow
  - not as constructive as some other design techniques
Exhaustive Search

A brute force solution to a problem involving search for an element with a special property, usually among combinatorial objects such as permutations, combinations, or subsets of a set.

Method:

- generate a list of all potential solutions to the problem in a systematic manner (see algorithms in Sec. 5.4)

- evaluate potential solutions one by one, disqualifying infeasible ones and, for an optimization problem, keeping track of the best one found so far

- when search ends, announce the solution(s) found
Example 1: Traveling 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

<table>
<thead>
<tr>
<th>Tour</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>a→b→c→d→a</td>
<td>2+3+7+5 = 17</td>
</tr>
<tr>
<td>a→b→d→c→a</td>
<td>2+4+7+8 = 21</td>
</tr>
<tr>
<td>a→c→b→d→a</td>
<td>8+3+4+5 = 20</td>
</tr>
<tr>
<td>a→c→d→b→a</td>
<td>8+7+4+2 = 21</td>
</tr>
<tr>
<td>a→d→b→c→a</td>
<td>5+4+3+8 = 20</td>
</tr>
<tr>
<td>a→d→c→b→a</td>
<td>5+7+3+2 = 17</td>
</tr>
</tbody>
</table>

More tours?

Less tours?

Efficiency:

Example 2: Knapsack Problem

Given $n$ items:

- weights: $w_1 \ w_2 \ldots \ w_n$
- values: $v_1 \ v_2 \ldots \ v_n$
- a knapsack of capacity $W$

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

Example: Knapsack capacity $W=16$

<table>
<thead>
<tr>
<th>item</th>
<th>weight</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>$50</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>$10</td>
</tr>
</tbody>
</table>

## Knapsack Problem by Exhaustive Search

<table>
<thead>
<tr>
<th>Subset</th>
<th>Total weight</th>
<th>Total value</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td>{2}</td>
<td>5</td>
<td>$30</td>
</tr>
<tr>
<td>{3}</td>
<td>10</td>
<td>$50</td>
</tr>
<tr>
<td>{4}</td>
<td>5</td>
<td>$10</td>
</tr>
<tr>
<td>{1,2}</td>
<td>7</td>
<td>$50</td>
</tr>
<tr>
<td>{1,3}</td>
<td>12</td>
<td>$70</td>
</tr>
<tr>
<td>{1,4}</td>
<td>7</td>
<td>$30</td>
</tr>
<tr>
<td>{2,3}</td>
<td>15</td>
<td>$80</td>
</tr>
<tr>
<td>{2,4}</td>
<td>10</td>
<td>$40</td>
</tr>
<tr>
<td>{3,4}</td>
<td>15</td>
<td>$60</td>
</tr>
<tr>
<td>{1,2,3}</td>
<td>17</td>
<td>not feasible</td>
</tr>
<tr>
<td>{1,2,4}</td>
<td>12</td>
<td>$60</td>
</tr>
<tr>
<td>{1,3,4}</td>
<td>17</td>
<td>not feasible</td>
</tr>
<tr>
<td>{2,3,4}</td>
<td>20</td>
<td>not feasible</td>
</tr>
<tr>
<td>{1,2,3,4}</td>
<td>22</td>
<td>not feasible</td>
</tr>
</tbody>
</table>

**Efficiency:**
Example 3: 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.

<table>
<thead>
<tr>
<th></th>
<th>Job 0</th>
<th>Job 1</th>
<th>Job 2</th>
<th>Job 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person 0</td>
<td>9</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Person 1</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Person 2</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Person 3</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

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

\[
\begin{pmatrix}
9 & 2 & 7 & 8 \\
6 & 4 & 3 & 7 \\
5 & 8 & 1 & 8 \\
7 & 6 & 9 & 4
\end{pmatrix}
\]

<table>
<thead>
<tr>
<th>Assignment (col.#s)</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3, 4</td>
<td>9+4+1+4=18</td>
</tr>
<tr>
<td>1, 2, 4, 3</td>
<td>9+4+8+9=30</td>
</tr>
<tr>
<td>1, 3, 2, 4</td>
<td>9+3+8+4=24</td>
</tr>
<tr>
<td>1, 3, 4, 2</td>
<td>9+3+8+6=26</td>
</tr>
<tr>
<td>1, 4, 2, 3</td>
<td>9+7+8+9=33</td>
</tr>
<tr>
<td>1, 4, 3, 2</td>
<td>9+7+1+6=23</td>
</tr>
</tbody>
</table>

etc.

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

Final Comments on Exhaustive Search

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

- In some cases, there are much better alternatives!
  - Euler circuits
  - Shortest paths
  - Minimum spanning tree
  - Assignment problem

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