EXHAUSTIVE SEARCH ALGORITHMS

Brute Force Approach

Brute force is a **straightforward approach** to solving a problem, directly based on the **problem statement** and **definitions** of the concepts involved.

Exhaustive Search

Exhaustive search is a brute-force approach to a problem involving search for an **object with a special property**, usually among **combinatorial objects** such as **subsets** of a set, **permutations**, or **combinations** of sequences.

Exhaustive Search

Exhaustive search is a brute-force approach to a problem involving search for an **object with a special property**, usually among **combinatorial objects** such as **subsets** of a set, **permutations**, or **combinations** of sequences.

Method:

- **generate a list of all potential solutions** to the problem in a systematic manner,
- **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**.

Activity (in group)

- **Design your original algorithm**
  - TSP / Knapsack / Job Assignment Problem
  - Show how you can improve better than exhaustive approach
- Analyze: running time
- Write a pseudocode/program
- Share

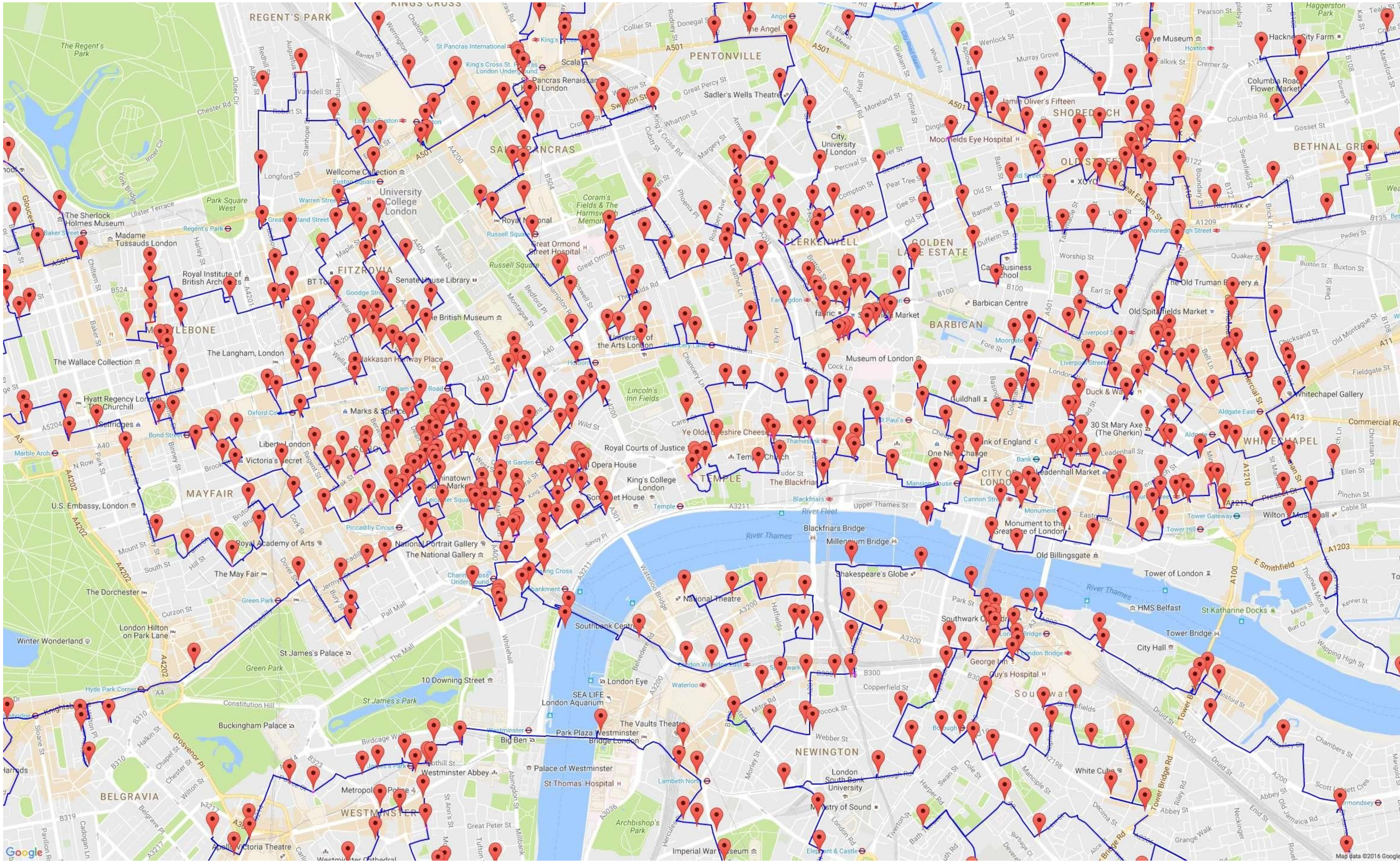
https://padlet.com/sem2_2018_2019/dsa2

Traveling Salesman Problem

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.

Traveling Salesman Problem

Traveling Salesman Problem



Traveling Salesman Problem

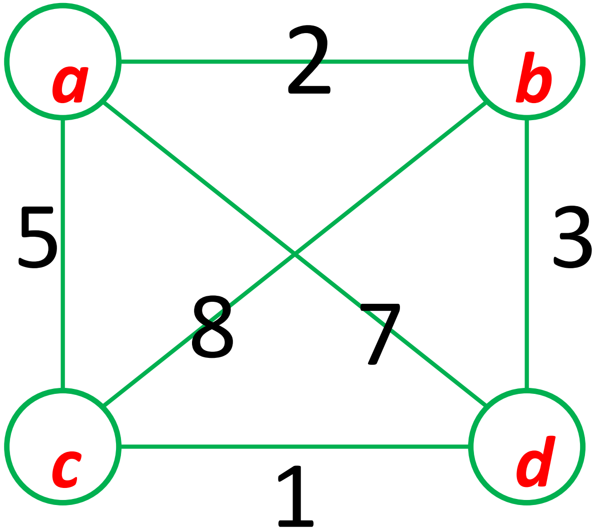
Problem: Given n cities with known distances between each pair, find the **shortest tour** that passes through all the cities exactly once before returning to the starting city.

Alternatively: Find the shortest **Hamiltonian cycle** (a cycle that passes through all vertices $(v_1, v_2, \dots, v_n)$ exactly once) in a weighted connected graph:

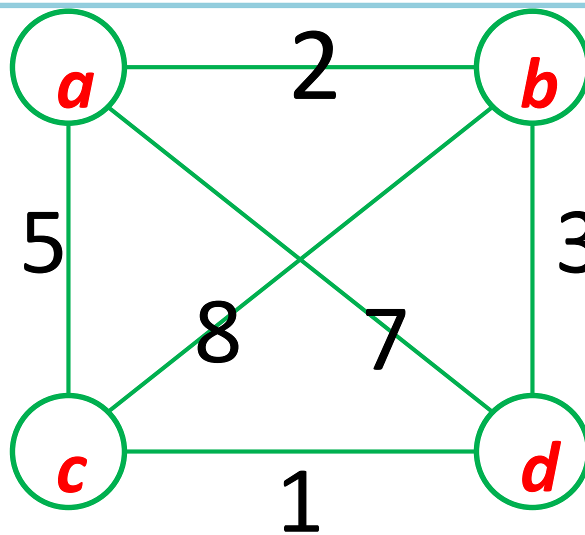
$$v'_1, \underbrace{v'_2, v'_3, \dots, v'_n}_{\text{distinct vertices}}, v'_1$$

Traveling Salesman Problem

Example:



Example:



Traveling Salesman Problem

1. $a \to b \to c \to d \to a$ $\quad d = 2 + 8 + 1 + 7 = 18$
2. $a \to b \to d \to c \to a$ $\quad d = 2 + 3 + 1 + 5 = 11$
3. $a \to c \to b \to d \to a$ $\quad d = 5 + 8 + 3 + 7 = 23$
4. $a \to c \to d \to b \to a$ $\quad d = 5 + 1 + 3 + 2 = 11$
5. $a \to d \to b \to c \to a$ $\quad d = 7 + 3 + 8 + 5 = 23$
6. $a \to d \to c \to b \to a$ $\quad d = 7 + 1 + 8 + 2 = 18$

Traveling Salesman Problem

The efficiency:

$$\mathrm{v'}_1, \underbrace{\mathrm{v'}_2, \mathrm{v'}_3, \ldots, \mathrm{v'}_n}_{\text{distinct vertices}}, \mathrm{v'}_1$$

- The number of all permutations: $(n-1)!$
- The half tour differs only by their directions: $\frac{1}{2}(n-1)!$
- $O(n!)$

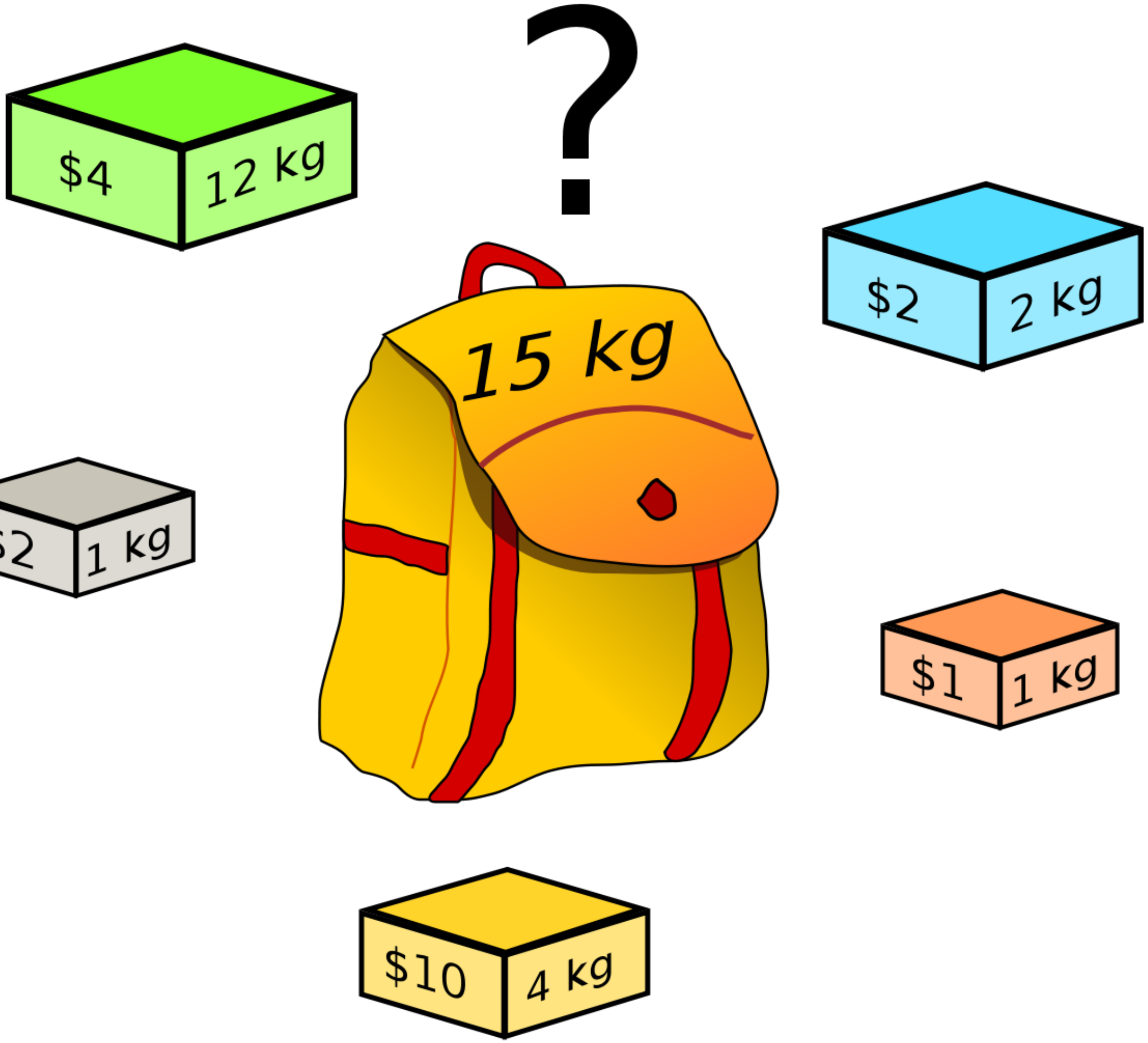
Knapsack Problem

Given n items $i_1, i_2, \dots, i_n$ of known:

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

Problem: Find the **most valuable subset** of the items $i_1, i_2, \dots, i_n$ that **fit into the knapsack**.

Knapsack Problem



Knapsack Problem

Example: Knapsack capacity $W = 16$

ITEM	WEIGHT	VALUE
1	2	\$20
2	5	\$30
3	10	\$50
4	5	\$10

The exhaustive search approach:

Knapsack Problem

Example: Knapsack capacity $W = 16$

ITEM	WEIGHT	VALUE
1	2	$20
2	5	$30
3	10	$50
4	5	$10

The exhaustive search approach: (1) generate **all subsets** of the set of n items; (2) compute the **total weight** of each subset; (3) find a **subset of largest value** among them.

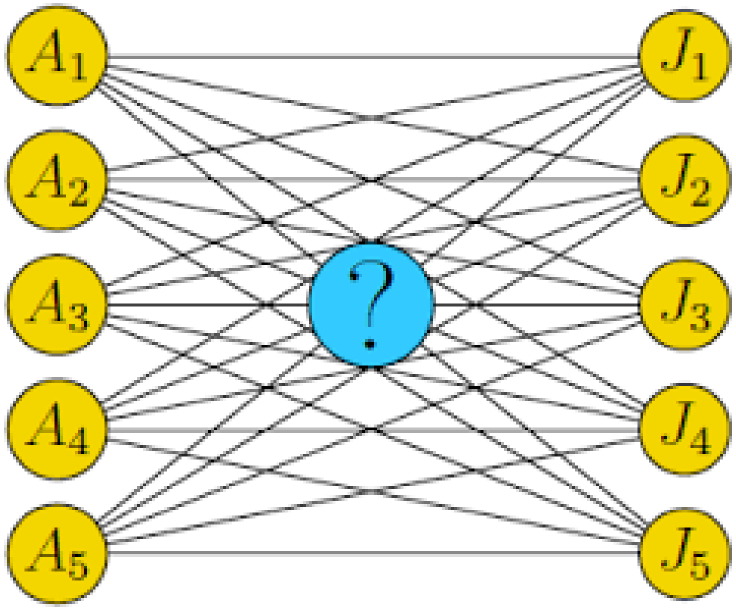
- The **total # of all subsets**: 2^n **Efficiency**: $\Omega(2^n)$

SUBSETS	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

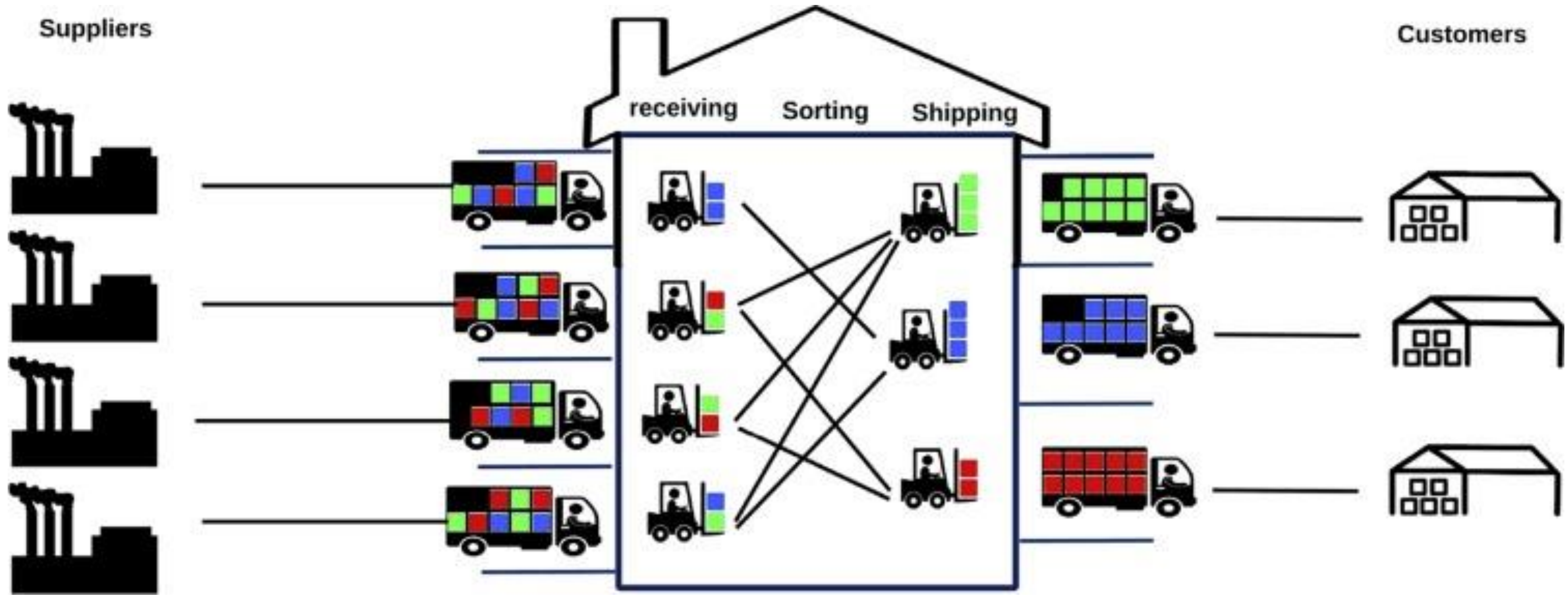
Assignment Problem

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**.

Assignment Problem



Assignment Problem



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

Assignment Problem

Solution: Select one element in each row so that all selected elements are in different columns and the total sum of the selected elements is the smallest possible.

$$\begin{array}{l} \quad\;\; \text{P1 P2 P3 P4} \end{array}$$

1. $\langle 1,2,3,4 \rangle$ $\quad$ $\text{cost} = 9 + 4 + 1 + 4 = 18$
2. $\langle 1,2,4,3 \rangle$ $\quad$ $\text{cost} = 9 + 4 + 8 + 9 = 30$
3. $\langle 1,3,2,4 \rangle$ $\quad$ $\text{cost} = 9 + 3 + 8 + 4 = 24$
4. $\langle 1,3,4,2 \rangle$ $\quad$ $\text{cost} = 9 + 3 + 8 + 6 = 26$
5. $\langle 1,4,3,2 \rangle$ $\quad$ $\text{cost} = 9 + 7 + 1 + 6 = 23$

Efficiency: $O(n!)$

Activity (in group)

- **Design your original algorithm**
  - TSP / Knapsack / Job Assignment Problem
  - Show how you can improve better than exhaustive approach
- Analyze: running time
- Write a pseudocode/program
- Share

https://padlet.com/sem2_2018_2019/dsa2