CS 395 – Analysis of Algorithms

Chapter 3 – Brute Force and Exhaustive Search

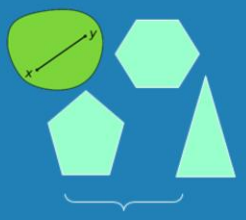
Read 3.4

- **Brute Force Algorithm Technique**
  - **Brute-Force Convex-Hull**
- **Exhaustive Search**
  - **Travelling Salesman Problem**
  - **Knapsack Problem**
  - **Assignment Problem**
- **Weekly Assignment 7**

Convex-Hull Problem

Definition - Convex

A set of points (finite or infinite) in the plane is called *convex* if for any two points p and q in the set, the entire line segment with the endpoints at p and q belong to the set.



Convex sets

Not Convex sets

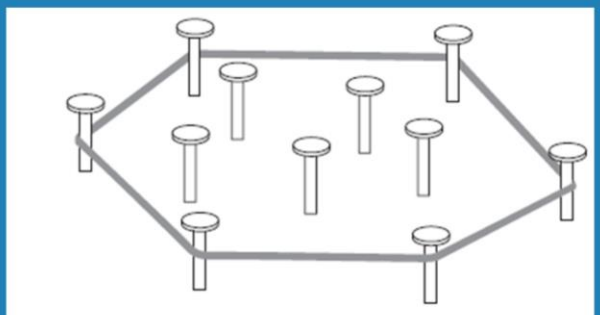
2

Draw example.

Convex-Hull

Formally: The *convex hull* of a set *S* of points is the smallest convex set containing *S*, *i.e.*, the convex hull of *S* is a subset of any convex set containing *S*.

Informally: It's a rubber-band wrapped around the "outside" points.



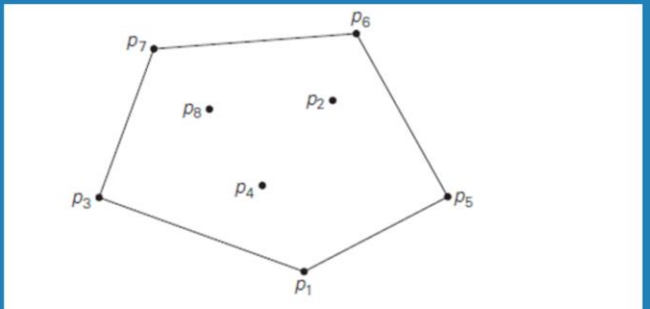
Rubber-band interpretation of the convex hull.

3

Draw example.

Convex-Hull

THEOREM The convex hull of any set S of $n>2$ points not all on the same line is a convex polygon with the vertices at some of the points of S. (If all the points do lie on the same line, the polygon degenerates to a line segment but still with the endpoints at two points of S.)



The convex hull for this set of eight points is the convex polygon with vertices at p_1, p_5, p_6, p_7, and p_3.

4

Draw example.

Convex-Hull Problem

- The *convex-hull problem* is the problem of constructing the convex hull for a given set *S* of *n* points.
- A fundamental problem in computational geometry
- Need to find extreme points.
- *Extreme point* of a convex set is a point of this set that is not a middle point of any line segment with endpoints in the set.

- To solve the Convex-Hull problem, need to know:
  - Which of *n* points in a given set are extreme points
  - Which pairs of points need to be connected to form the boundary
  - Or list the extreme points in a clockwise or counterclockwise order

5

Draw example.

Brute Force Algorithm

- **Observation**: a line segment connecting two points p_i and p_j of a set of n points is a part of the convex hull's boundary if and only if all the other points of the set lie on the same side of the straight line through these two points.

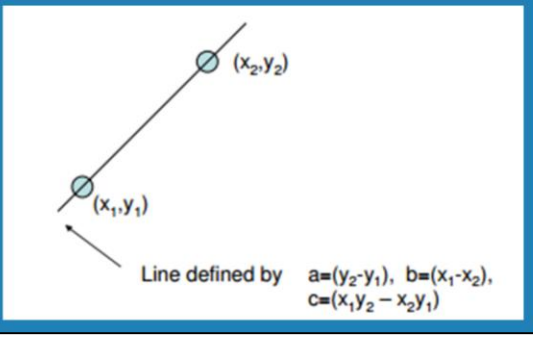
6

Draw example.

Brute Force Algorithm

- **Facts from analytic geometry:**
  - the straight line through two points (x_1, y_1), (x_2, y_2) in the coordinate plane can be defined by the equation

    $ax + by = c$, where $a = y_2 - y_1$, $b = x_1 - x_2$, $c = x_1 y_2 - y_1 x_2$.
  - such a line divides the plane into two half-planes: for all the points in one of them, $ax + by > c$, while for all the points in the other, $ax + by < c$.



7

Draw example.

Brute Force Algorithm

- Using these formulas, we can determine if two points are on the boundary to the convex hull.
- So, simply check whether the expression $ax + by - c$ has the same sign for each of these points to check whether certain points lie of the same side of the line.

8

Draw example.

Pseudocode

High level pseudocode for the algorithm then becomes:

```
for each point Pi
        for each point Pj where Pj ≠ Pi
                Compute the line segment for Pi and Pj
                for every other point Pk where Pk ≠ Pi and Pk ≠ Pj
                        If each Pk is on one side of the line segment, label Pi and Pj
                        in the convex hull
```

Time efficiency: $\Theta(n^3)$

9

Draw example.

Exhaustive Search

Exhaustive Search is an abstraction of brute force 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.

Features

- **Often simple to implement**
- **It will always find a solution, if there is one (?!?)**
- **BUT, cost proportional to the number of candidate solutions**
- **Combinatorial explosion!**
- **Practical only for very small problem instances !!**

10

Exhaustive Search

Method:

- **Enumerate all possible solution candidates: Construct a way of listing all potential solutions to the problem in a systematic manner** – This is not always trivial and we will look at some approaches next class.
  - **all solutions are eventually listed**
  - **no solution is repeated**
- **Check if they satisfy the problem's statement: Evaluate potential solutions one by one, (perhaps) disqualifying infeasible ones and, (for an optimization problem,) keeping track of the best one found so far.**
- **when search ends, return the solution(s) found**

Exhaustive Search

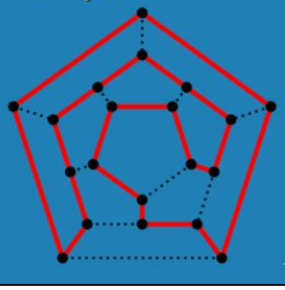
- **Basic algorithm (outline)**

```
c ← generate a first candidate solution
while ( c is a candidate ) do
    if ( c is a valid solution )
    then output (c)
    c ← generate the next candidate solution, if any
```

- **Might also stop after**
  - Finding the first valid solution
  - Finding a specified number of valid solutions
  - Testing a given number of candidates
  - Spending a given amount of CPU time

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
  - Hamiltonian path is a path in an undirected or directed graph that visits each vertex exactly once.
  - Hamiltonian circuit is a Hamiltonian path that is a cycle.



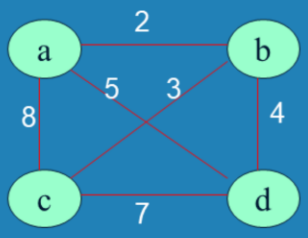
13

TSP by Exhaustive Search

- Use a weighted graph *G* to model the problem
- Find the shortest Hamiltonian circuit of *G*
  - Cycle of least cost /distance
  - Passes (just once) through all vertices
- Hamiltonian circuit
  - Sequence of $(n + 1)$ adjacent vertices
  - The first vertex is the same as the last !
- How to proceed?
  - Choose any one vertex as the starting point
  - Generate the $(n - 1)!$ possible permutations of the intermediate vertices
  - For each such cycle, compute its cost / distance
  - And keep the less expensive / shortest one

14

TSP by Exhaustive Search



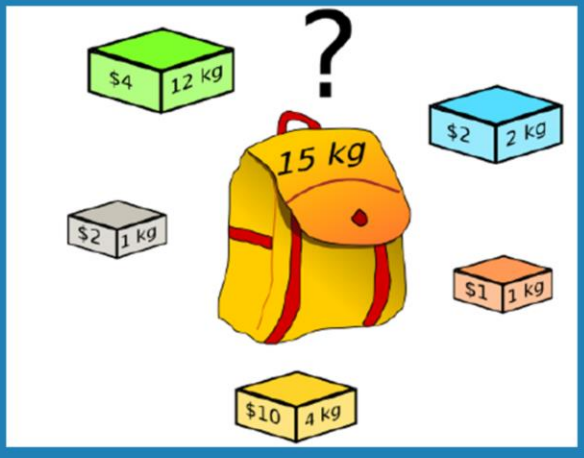
Tour	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17

Time Efficiency: $\Omega(n!)$

15

Example 2: Knapsack Problem

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



16

Example 2: Knapsack Problem

Given n items:

- weights: w_1 w_2 … w_n
- values: v_1 v_2 … v_n
- a knapsack of capacity W

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

Knapsack Problem by Exhaustive Search

Approach:

- Generate the 2^n subsets of a set of n items
  - For each such subset, compute its total weight
  - Feasible subset ?
  - And keep the most valuable feasible subset

Example: Knapsack capacity W=16

item	weight	value
1.	2	$20
2.	5	$30
3.	10	$50
4.	5	$10

18

Knapsack Problem by Exhaustive Search

Subset	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

Time Efficiency: $O(2^n)$ - Exponential

19

Dr. BC's preferred Exhaustive Search order

1234	Subset	Total weight	Total value
0000	{}	0	$0
0001	{4}	5	$10
0010	{3}	10	$50
0011	{3,4}	15	$60
0100	{2}	5	$30
0101	{2,4}	10	$40
0110	{2,3}	15	$80
0111	{2,3,4}	20	not feasible
1000	{1}	2	$20
1001	{1,4}	7	$30
1010	{1,3}	12	$70
1011	{1,3,4}	17	not feasible
1100	{1,2}	7	$50
1101	{1,2,4}	12	$60
1110	{1,2,3}	17	not feasible
1111	{1,2,3,4}	22	not feasible

Time Efficiency: $O(2^n)$ - Exponential

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.

	Job 0	Job 1	Job 2	Job 3
Person 0	9	2	7	8
Person 1	6	4	3	7
Person 2	5	8	1	8
Person 3	7	6	9	4

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:

21

Assignment Problem by Exhaustive Search

$$C = \begin{matrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{matrix}$$

Assignment (col.#s)	Total Cost
1, 2, 3, 4	9+4+1+4=18
1, 2, 4, 3	9+4+8+9=30
1, 3, 2, 4	9+3+8+4=24
1, 3, 4, 2	9+3+8+6=26
1, 4, 2, 3	9+7+8+9=33
1, 4, 3, 2	9+7+1+6=23
	etc.

22

Assignment Problem by Exhaustive Search

The column assignments in the example on the last slide can be viewed as n-tuples $<j_1 \ldots, j_n>$ where the i-th component, $i = 1, \ldots, n$, indicates the column of the element selected in the i-th row. Thus the exhaustive search assignment problem as formulated requires:

1. generating all permutations of integers 1, …, n
2. computing the total cost of each assignment
3. selecting the assignment with the smallest sum

Time efficiency: $O(n!)$ - factorial

23

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

24

W07: Cyclic Tower of Hanoi

- Like the Tower of Hanoi, but you can only move one to the right or from C back to A.