Brute-Force Algorithms

Joaquim Madeira

Version 0.1 – September 2015

Overview

- Brute Force
- Array Searching and Sorting
- String Matching
- The Closest-Pair Problem
- The Convex-Hull Problem
- Exhaustive Search
- The Traveling Salesman Problem
- The Knapsack Problem
- The Assignment Problem
- The N-Queens Problem

Brute-Force

- The (most) straightforward approach to solving a problem

- Directly based on
  - The problem statement
  - The definitions involved

- Strengths
  - Simplicity
  - Applicable to different kinds of problems

- Weaknesses
  - (Very!) Low efficiency in some cases
  - Useful only for instances of (relatively) small size !!

Brute-Force

- Where to apply?

- Numerical problems, searching, sorting, etc.
  - Acceptable efficiency
  - Can be used for large problem instances

- Combinatorial problems
  - Exhaustive search
  - Set of candidate solutions grows very fast
  - Used only for reduced size instances

Brute-Force

- How many examples do you know?

- Add n numbers
- Direct matrix multiplication
- Sequential search
- Selection sort
- Bubble sort
- …

Finding the Divisors of n

- Given a natural number n ($n \geq 1$), find its divisors

- Exhaustive search:
  - Enumerate all integers from 1 to n
  - Check if each one is a divisor of n

```
d ← 1
while ( d < n + 1 ) do
        if ( n mod d == 0 )
        then output (d)
        d ← d + 1
```

Finding the Divisors of n

- How many candidates are there?
- How to reduce the search space?
  - But for n, which number is the largest possible divisor?
- How long does it take?
- Example: n has 16 decimal digits
  - It takes at least 10^{15} comparisons
  - How many days?
- What if n is a 64-bit natural number?

Greatest-Common Divisor

- Given natural numbers m and n ($m, n \geq 1$)
- Sequentially check possible divisors

```
t ← min { m, n }
while ( t  > 0 ) do
        if ( m mod t == 0 and n mod t == 0 )
        then return t
        t ← t - 1
```

- How does it compare to Euclid's gcd algorithm?

Efficiency

- How many comparisons are made?
- Complexity?
  - O(n)
  - BUT, O(2^b) regarding the number of bits in the number's binary representation

Sequential Search

- Unsorted array or linked list with n elements
- Search for
  - Given X item
  - Smallest / largest item
  - First occurrence / last occurrence

- Worst and average cases : $O(n)$

- How to improve?
  - Mandatory for large n !!

Selection Sort

- Recursive strategy !

- Given an array with n elements
  - Find the smallest / largest item : Seq. Search
  - Move it into its final position, if needed
  - Do the same for the remaining (n – 1) elements

- How many
  - Item comparisons ?
  - Item exchanges ?

- Is this always a stable algorithm?

Selection sort

```
i ← 0
while ( i < n – 1 ) do
        min ← i
        j ← i + 1
        while ( j < n ) do
                if ( A[ j ] < A[ min ] )
                then min ← j
                 j ←  j + 1
         if ( i ≠ min )
         then A[ i ] ↔ A[ min ]
         i ← i + 1
```

Selection Sort

- Number of comparisons ?
  - Fixed !
  - $(n - 1) + (n - 2) + \ldots + 2 + 1 = n (n - 1) / 2$
  - $O(n^2)$
- Number of exchanges
  - $B_{Ex}(n) = 0$
  - $W_{Ex}(n) = (n - 1)$ – When does this happen?
  - $A_{Ex}(n) = ?$
- Can we do better ?

String Matching

- Given a text t, with n chars, and a pattern p, with m chars
  - m ≤ n
  - Usually, m << n

- Find the first occurrence of the pattern

- Or find all occurrences of the pattern

- Successively align the pattern within the text

t [i] == p [0]
t [i + 1] == p[1]
…
t [i + m - 1] == p [m - 1]

String Matching

```
i ← 0
while ( i <= n – m ) do
    j ← 0
    while ( j < m and p[ j ] == t[ i + j ] ) do
            j ←  j + 1
    if ( j == m )
    then return i
     i ← i + 1
return -1
```

String Matching

- Best Case
  - m comparisons : O (m)
  - Quite rare…
- Worst Case
  - Always m comparisons before shifting
  - (n – m + 1) attempts : m (n – m + 1) comparisons
  - O (m n), if n >> m
  - Quite rare…
- Average Case
  - Shifting after a reduced number of comparisons
  - Random texts : O (m + n), O (n), if n >> m

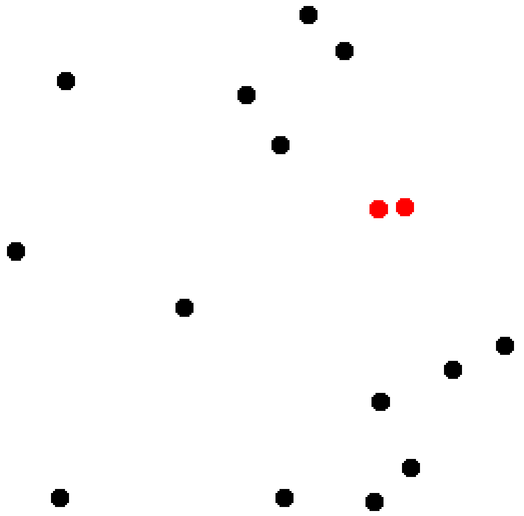
String Matching

- What if the pattern contains a less frequent char (e.g., Z)?
  - Search for that particular char
  - Search for the remaining pattern chars

- There are other, more efficient algorithms
  - Boyer-Moore
  - Knuth-Morris-Pratt

- Exercise
  - Find an example for Worst Case behavior

The 2D Closest-Pair Problem



[Wikipedia]

- Finite set of n points : P
- Find the two closest points in P
  - Shortest Euclidean distance
- How many pairs of points ?
  - (P_i, P_j), $i < j$

The 2D Closest-Pair Problem

- First, naïve version

```
dmin ← infinity
for each p in P
        for each q in P
                if ( p ≠ q and dist(p, q) < dmin )
                then   dmin ← dist(p, q)
                       closestPair ← (p, q)
return closestPair
```

The 2D Closest-Pair Problem

- Improved version

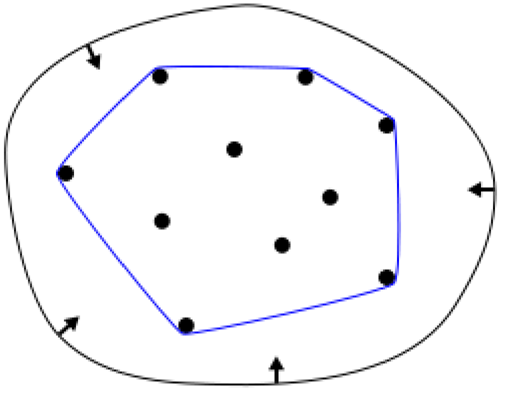
```
dmin ← infinity
for i ← 1 to n – 1 do
        for j ← i + 1 to n do
                d ← sqrt( (x_i – x_j)^2 + (y_i – y_j)^2 )
                if ( d < dmin )
                then   dmin ← d
                       closestPair ← (i, j)
return closestPair
```

The 2D Closest-Pair Problem

- Questions
  - Can we ignore the square root?
  - Can we use other distances? E.g., Manhattan
  - Will we get different results?

- Efficiency
  - Basic op. : squaring
  - $O(n^2)$

The 2D Convex-Hull Problem



[Wikipedia]

- Convex set S
  - Take any two points in S
  - Defined line segment also in S
- Convex hull of set T
  - The smallest convex set containing all points in T
- If T is convex, its convex hull is set T itself
  - What is the convex hull of a set of collinear points?

The 2D Convex-Hull Problem

- Theorem
  - The convex hull of set S (n > 2 points, not all collinear) is a convex polygon with the vertices (i.e., extreme points) at some of the points of S.
- Important property
  - A line segment connecting P_i and P_j is part of the boundary iff all other points of the set lie on the same side of the line through P_i and P_j.

The 2D Convex-Hull Problem

for i ← 1 to n – 1 do

  for j ← i + 1 to n do

    L ← line through P_i and P_j

    flag ← do all other points lie on the same side of L ?

    if (flag)

    then add P_i and P_j to the boundary

The 2D Convex-Hull Problem

- Questions
  - How to check relative position?
    - Use the equation of the straight line
  - What to do if some points are collinear?
  - How to store and output the boundary points?
    - Points should be ordered, e.g., CCW

- Efficiency
  - Basic op. : checking a point against line (P_i, P_j)
  - $O(n^3)$
  - Better alternatives ?

Exhaustive Search

- Brute-force approach to combinatorial problems
  - I.e., there is a discrete set of feasible solutions

- Strategy
  - Enumerate all possible solution candidates
  - Check if they satisfy the problem's statement
  - If needed, choose one solution from the set of feasible ones

- How to ensure that we check all candidates?

Exhaustive Search

- Basic algorithm

  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

Exhaustive Search

- 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 !!
- How to speed up the search?

Speeding up Brute-Force Searches

- Reduce the search space
  - Use analysis / heuristics to reduce the number of candidate solutions
  - E.g., the n-queens problem

- Reorder the search space
  - Useful whenever searching just for one solution
  - Expected running time depends on the order candidates are tested
  - Test the most promising candidates first !!
  - Question: should we search for a divisor of n in descending order?

The Traveling Salesman Problem

- Find the shortest tour through a given set of n cities
- BUT, visiting each city just once !
- AND returning to the starting city !

[Wikipedia]

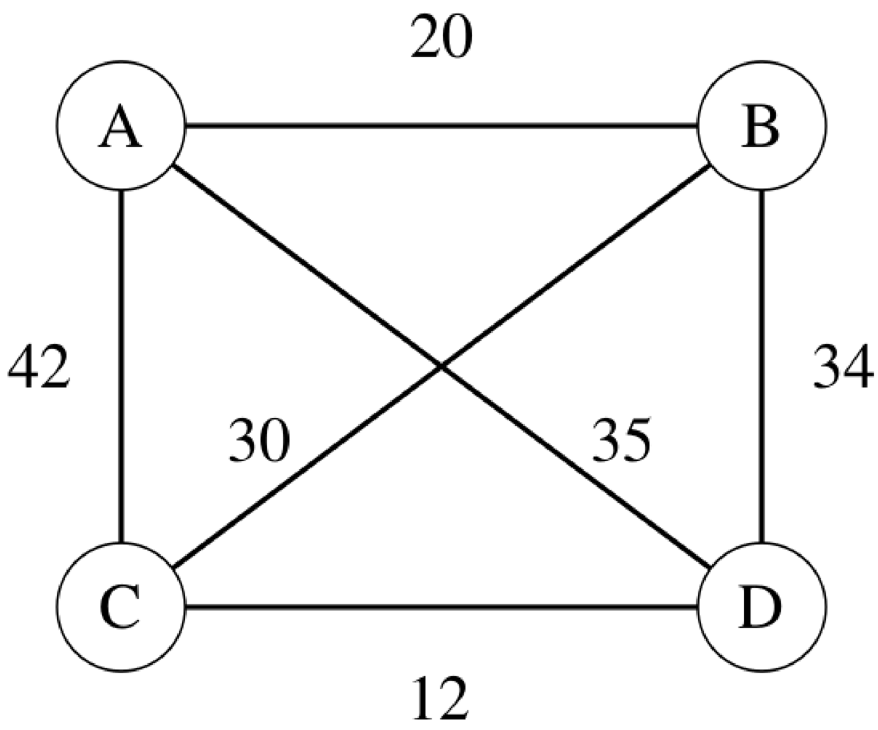
The Traveling Salesman Problem

- 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
- NP-complete problem !!

The Traveling Salesman Problem

- 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

The Traveling Salesman Problem



[Wikipedia]

- What is the solution?

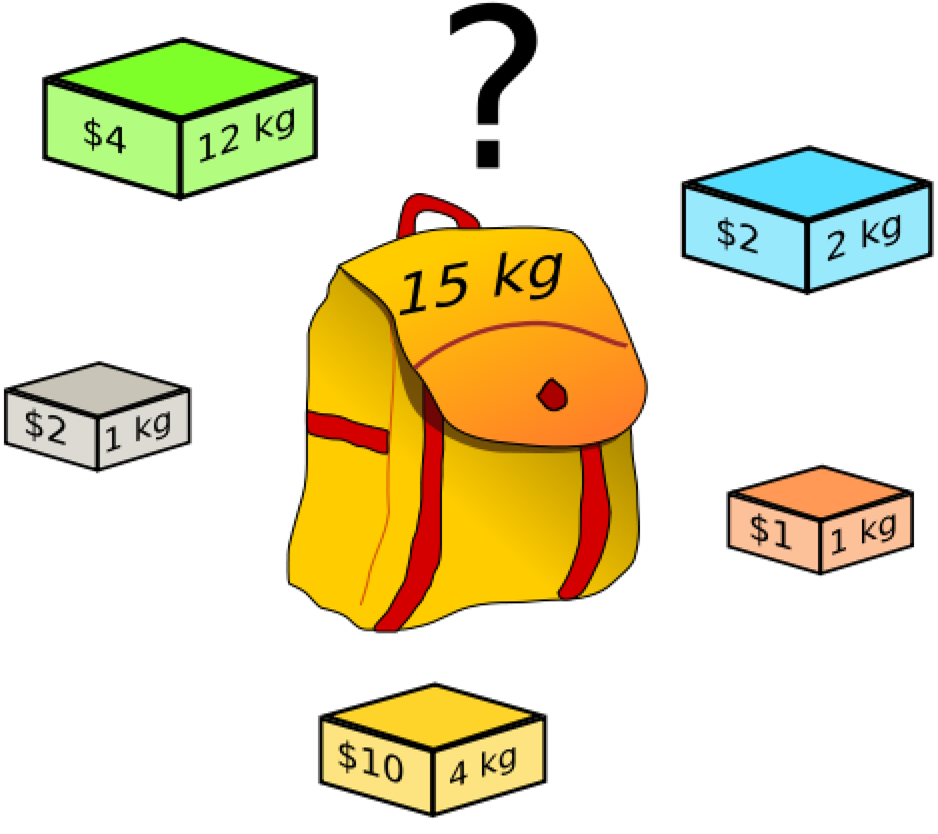
The Traveling Salesman Problem

- Questions
  - How do we store the graph?
  - Is it complete?
  - How to generate all permutations?

- Efficiency
  - O(n!)
  - Exhaustive search can only be applied to very small instances !! Alternatives ?
  - Slight improvements are still possible

The 0-1 Knapsack Problem

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



[Wikipedia]

The 0-1 Knapsack Problem

- Given n items
  - Known weight $w_1, w_2, \ldots, w_n$
  - Known value $v_1, v_2, \ldots, v_n$
- A knapsack of capacity W
- Which one is the most valuable subset of items that fit into the knapsack?
- NP-complete problem !!

The 0-1 Knapsack Problem

- How to formulate ?

$$\max \sum x_i v_i$$

subject to $\sum x_i w_i \leq W$

with x_i in $\{0, 1\}$

The 0-1 Knapsack Problem

- How to proceed?

  - 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

The 0-1 Knapsack Problem

- Knapsack of capacity W = 10

- 4 items
  - Item 1 : w = 7 ; v = $42
  - Item 2 : w = 3 ; v = $12
  - Item 3 : w = 4 ; v = $40
  - Item 4 : w = 5 ; v = $25

- Optimal solution ?

The 0-1 Knapsack Problem

- Question
  - How to generate all subsets?
  - Does order matter?

- Efficiency
  - $O(2^n)$
  - Exhaustive search can only be applied to small problem instances !!
  - Alternatives ?
    - Exact vs. approximate solutions

The Assignment Problem

- There are
  - n jobs to be done
  - n people, who need to be assigned

- BUT, only one person per job !!

- Cost matrix : C[i,j]
  - Cost from assigning person i to job j

- Find the (an) assignment with smallest cost !

The Assignment Problem

- How to formulate ?

$$\min \sum \sum x[i,j]\, c[i,j]$$

subject to $\sum x[i,j] = 1$, for every i

$\sum x[i,j] = 1$, for every j

with $x[i,j]$ in $\{0, 1\}$

The Assignment Problem

- Cost matrix
  - Select one element in each row and one element in each column !!

- Naïve solutions might not work
  - E.g., selecting the smallest element in each row

- Feasible solutions ?
  - n-tuples : permutations of the first n integers
  - The column selected for each matrix row
  - I.e., the job assigned to each person

The Assignment Problem

- How to proceed?
  - Generate the n! possible job assignments
  - For each such assignment, compute its total cost
    - Sum up the corresponding cost matrix elements
  - And keep the less expensive assignment

The Assignment Problem

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

- Optimal solution?

The Assignment Problem

- Efficiency
  - O(n!)
  - Exhaustive search can only be applied to very small instances !!
  - Alternatives ?
    - The Hungarian Method

- Note that
  - Not a NP-complete problem !!
  - Although the set of feasible solutions is O(n!)
  - Good news !!

The N-Queens Problem

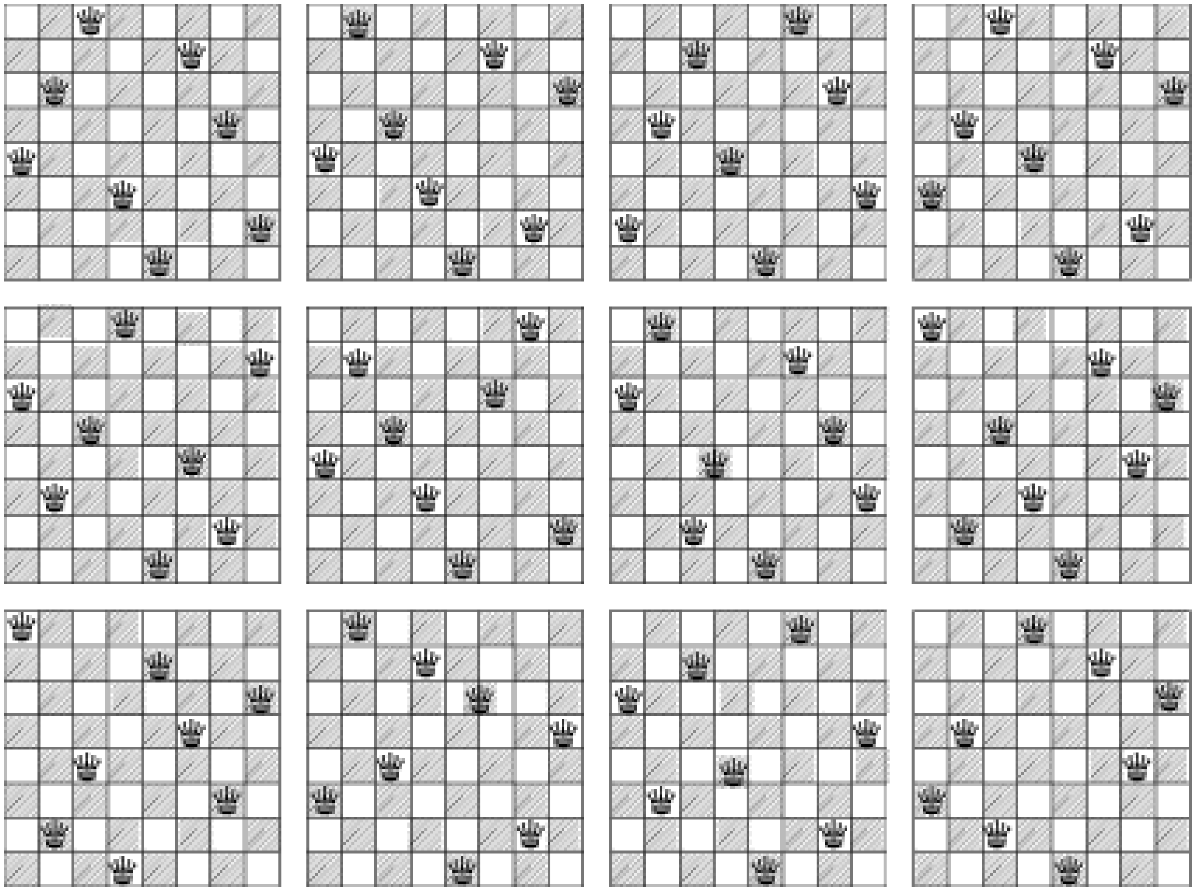
- How many ways are there, to place N queens on a (N x N) chessboard?
  - No queen can attack another !!

- Original puzzle : 8 x 8 chessboard
  - 1848, Max Bezzel

- Generalization : (N x N) chessboard

- Are there any boards, for which no solution exists?

The N-Queens Problem



[Mathworld]

- N = 8
  - 12 basic solutions
  - 92 solutions in all, through rotations and reflections

The N-Queens Problem

- How many queen configurations have to be tested?

- Example : N = 8

- No two queens on the same square !!
  - C(64,8) = 64! / (56! X 8!) ≈ 4.5 X 10^9

- No two queens on the same row / column
  - 8! = 40,320
  - Reduction in the number of candidate solutions
  - Check for diagonal attacks !!

The N-Queens Problem

- How to proceed?
  - Generate the n! row / column permutations
  - For each such queen configuration
    - Is it feasible?
    - Check diagonal attacks !!
  - Stop when a solution found
  - Or output all solutions found

The N-Queens Problem

- How many solutions are there for
  - A (2 x 2) board ?
  - A (3 x 3) board ?
  - A (4 x 4) board?
  - …
- Is the number of solutions always increasing?

The N-Queens Problem

- Efficiency
  - O(n!)
  - Exhaustive search can only be applied to small instances !!
- Alternatives ?
  - Backtracking
  - Local search / Heuristics
  - …
- Note that
  - Not a NP-complete problem !!
  - Although the set of candidate configurations is O(n!)
  - Good news !!

The N-Queens Problem

- Related problems
  - Use other chess pieces
    - How many can be placed on a (N x N) chessboard ?
  - Find the domination number of a board
    - Min number of pieces to occupy / attack every square ?
  - Use non-standard boards
  - …

References

- A. Levitin, *Introduction to the Design and Analysis of Algorithms,* 3rd Ed., Pearson, 2012
  - Chapter 3
- R. Johnsonbaugh and M. Schaefer, *Algorithms,* Pearson Prentice Hall, 2004
  - Chapter 9 + Chapter 11
- T. H. Cormen et al., *Introduction to Algorithms*, 3rd Ed., MIT Press, 2009
  - Chapter 32