CS472 Module 2 Part C - Exhaustive Search

Athens State University

Outline

Contents

1 What is exhaustive search?

What is 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
  - evaluate potential solutions 1-by-1, eliminating infeasible solutions, if an optimization problem, keeping track of the best one found so far
  - Upon end of search, announce the solution(s) found

Example of why such algorithms can be problematic
One of the classic brain teasers assigned to novice computer programmers is the eight-queens problem:

- Place eight queens on a chessboard so that no queen attacks any other
- Each queen could, in principle, be placed in any of the 64 squares
  - Leads to $64^8 = 282,474,976,710,656$ possible combinations
- In reality, the queens are all alike and no two queens can be placed on the same square, so you actually have

  $$\binom{a}{b} = \frac{\frac{64!}{56!}}{8!} = 4,426,165,368$$

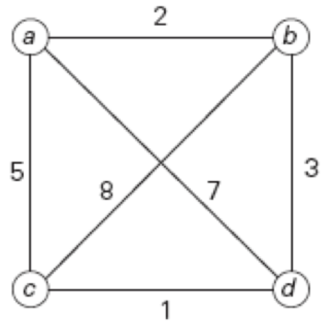
  possible solutions
- And no arrangement with two queens on the same row or column can be solution

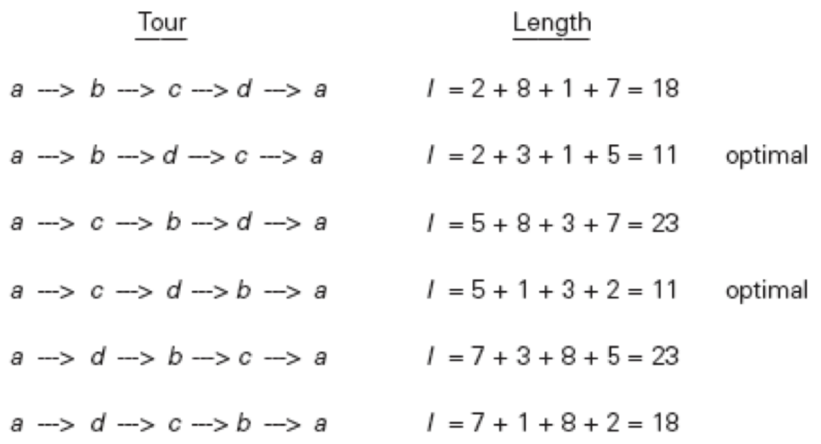
2 A few examples

Example: The Traveling Salesman Problem

- Given n cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once returning to the starting city
- Can be rephrased into the problem of finding the shortest Hamiltonian circuit in a weighted connected graph
  - *Hamiltonian circuit*: A cycle in a graph that passes through all the vertices of the graph exactly once.
- Closely related to the problem of finding longest path or cycle in a graph (arises often in pattern recognition problems)

Example: The Traveling Salesman Problem





Tour	Length	
a ---> b ---> c ---> d ---> a	$l = 2 + 8 + 1 + 7 = 18$	
a ---> b ---> d ---> c ---> a	$l = 2 + 3 + 1 + 5 = 11$	optimal
a ---> c ---> b ---> d ---> a	$l = 5 + 8 + 3 + 7 = 23$	
a ---> c ---> d ---> b ---> a	$l = 5 + 1 + 3 + 2 = 11$	optimal
a ---> d ---> b ---> c ---> a	$l = 7 + 3 + 8 + 5 = 23$	
a ---> d ---> c ---> b ---> a	$l = 7 + 1 + 8 + 2 = 18$	

FIGURE 3.7 Solution to a small instance of the traveling salesman problem by exhaustive search.

Example: The Traveling Salesman Problem: A (Very) Bad Algorithm

- Observation: A Hamiltonian circuit has to begin and end at the same vertex.
- So, for a circuit of length n, we simply need to generate all the permutations of the $n-1$ intermediate notes and computes the tour lengths

- Then find the tour with the shortest length (Simple... the crowd celebrates)
- And we can make things easier by noting that half the tours will differ only in their direction (the celebration gets louder)
- But note that the rate of growth of this process is $O(n!)$ (a loud disappointed groan emanates from the crowd)

Example: Knapsack Problem

- Given n items of known weights $w_1, w_2, \ldots, w_n$ and a knapsack of capacity W, what is the most valuable subset of the items that fit into the knapsack
- Example of an optimization problem: find the best solution given some sort of optimization function
  - In the case of the knapsack problem, combination of items whose total weight does not exceed the capacity of the sack

Example: Knapsack Problem: A (Very) Bad Algorithm

- Consider every possible subset of items
- Compute the total weight and discard if more than W
- Choose the remaining subset with maximum total value
- This is a $O(2^n)$ process!

Example: Assignment Problem

- You have exactly n people who need to be assigned to execute n jobs, one person per job
- The cost that would be paid if person i is assigned to job j is a known quantity $C[i, j]$ for each pair of person and job (up to n each)

Example: Assignment Problem: A Small Example

- Task is select one element from each row in the matrix so that all selected elements are in different columns and the total sum of costs is as small as possible

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

Example: Assignment Problem: Feasible Solutions and a (very) bad algorithm

- A feasible solution is denoted as $< j_1, \ldots, j_n >$ in which component i of the tuple indicates the column of the element in row i
- So, the brute-force solution is
  - generate all permutations of the integers from 1 to n,
  - compute the total cost of each assignment by summing the corresponding elements of the cost matrix
  - Then finding the permutation with the smallest sum

3 Some observations

Some observations

- Note that for the assignment problem, there are other techniques that can be used to find a solution
- For the traveling salesman and knapsack problems, exhaustive search leads to algorithms that are extremely inefficient on all inputs
  - Both problems are classic examples of a *NP-hard problem*
  - No polynomial-time algorithm is known for any NP-hard problem
  - It is thought but not proven that no such algorithm exists for any of these problems
    - The resulting conjecture: "the $P = NP$ problem" remains one of the fundamental unanswered questions in computer science