MATH 409 LECTURE 2
PROBLEMS AND ALGORITHMS

REKHA THOMAS

In this lecture we consider two important combinatorial optimization problems that can both be stated in terms of an undirected graph with weights on its edges. If V is the set of vertices of a graph G and E the set of edges of G, we typically denote the graph as $G = (V, E)$. The graph is undirected if there are no directions on the edges of G.

The Traveling Salesman Problem (TSP).
Data: We are given n points $v_1, \ldots, v_n$ and $\binom{n}{2}$ numbers c_{ij} where c_{ij} is the cost of traveling between v_i and v_j.
Problem: Find the shortest **tour** through the n points. (A tour is a closed path that visits each point exactly once.)

We can model this problem using a graph. A graph on n vertices is called **complete** if every two vertices in the graph are connected by an edge. The abstract complete graph on n vertices is typically denoted as K_n. Consider the complete graph on the vertex set $v_1, \ldots, v_n$ and let e_{ij} be the undirected edge between v_i and v_j. The edge e_{ij} is assigned cost c_{ij}. Then the problem is to find a closed simple path in this complete graph of minimum cost. (A path in a graph $G = (V, E)$ is a sequence of edges $e_1, \ldots, e_k$ such that e_i is incident to e_{i+1}. It is closed if the path returns to the starting vertex. It is simple if each vertex in the path has exactly two edges incident to it.)

In this graph interpretation, we usually embed the points $v_1, \ldots, v_n$ in the plane $\mathbb{R}^2$ with coordinates (x_i, y_i) for the point v_i. This problem fits many applications. The usual one takes $v_1, \ldots, v_n$ to be n cities and the cost c_{ij} to be the Euclidean distance between $v_i = (x_i, y_i)$ and $v_j = (x_j, y_j)$. In this case,

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Alternately, $v_1, \ldots, v_n$ could be holes that need to be drilled on a circuit board and c_{ij} could be the time taken by the drill to travel from v_i to v_j. In this case, we are looking for the fastest way to drill all holes on the circuit board if the drill traces a tour through the locations that

Date: March 31, 2010.

need to be drilled. Yet another application might be that $v_1, \ldots, v_n$ are n celestial objects that need to be imaged by a satellite and c_{ij} is the amount of fuel needed to change the position of the satellite from v_i to v_j. In this case, the optimal tour allows the satellite to complete its job with the least amount of fuel. These and many other applications of the TSP as well as the state-of-the-art on this problem can be found on the web page `http://www.tsp.gatech.edu/`.

Algorithms to solve the TSP.
Later in this course we will see sophisticated methods to solve the TSP but for now we list two simple algorithms that attempt to solve the problem.
(a) **Complete enumeration**: In this method, we list all possible tours through the n cities, calculate the cost of each tour and output the cheapest tour. How many tours are there through n cities? Starting at a city, there are $n-1$ ways to pick the second city. Then there are $n-2$ ways to pick the third city, $n-3$ ways to pick the fourth city and so on giving us

$$(n-1)! = (n-1)(n-2)\cdots 2 \cdot 1$$

tours. Note that it did not matter where we started as every city has to be visited and so we can call any city the first city. However, we have double counted since the tour $1, 2, \ldots, n-1, n, 1$ is the same as the tour $1, n, n-1, \ldots, 2, 1$ as far as cost is concerned, and we have counted them as different tours in our analysis. Therefore, in reality there are $\frac{(n-1)!}{2}$ tours. Unfortunately, this number grows incredibly fast. For instance if $n = 50$, then $\frac{(n-1)!}{2} = 3.0414 \times 10^{62}$. If a computer could enumerate a tour in 10^{-9} seconds (a nanosecond), it would still take 9.64425×10^{43} centuries to enumerate all the tours of this TSP. If we wanted to find the shortest tour through the capitals of the 50 states in the U.S., complete enumeration would not be the way to go.

Exercise 1. Compare the functions $n!, n^n, 2^n, 2^{n/2}, n^{log\, n}, n^{2000}, n^2, n\, log\, n$ and $log\, n$ as n increases (assume n is a non-negative integer). Can you arrange the functions in increasing order of their values for large n and give an argument to justify your arrangement?

(b) **The nearest neighbor algorithm for the TSP:** This is a simple minded **heuristic** that does not guarantee the optimal tour but tries to get close. The idea is that at every point in a partial tour we pick as the next city, the one that is closest to the last city in our current partial tour. This method is *locally optimal* — i.e., at each intermediate

step we cannot do better but may not yield the globally optimal tour.

Optimal perfect matchings.
Data: We are given the complete graph $K_n = (V, E)$ where n is even and, costs c_e on the edges $e \in E$.
Problem: Find the cheapest **perfect matching** in K_n.

Definition 2. A **matching** in a graph $G = (V, E)$ is a collection of edges of G such that no two edges in the matching share a vertex. The matching is **perfect** if all vertices of the graph are contained in the matching. (Note that we need an even number of vertices in the graph for a perfect matching to exist.)

Algorithms.
We will see very efficient algorithms for this problem during the course but let us examine how complete enumeration might fare on this problem. Enumerating all possibilities we see that there are

$$\binom{n}{2}\binom{n-2}{2}\binom{n-4}{2}\cdots\binom{2}{2}$$

matchings in K_n when n is even. Unraveling this expression we get, $\binom{n}{2}\binom{n-2}{2}\binom{n-4}{2}\cdots\binom{2}{2} = \frac{n(n-1)}{1\cdot 2}\frac{(n-2)(n-3)}{1\cdot 2}\cdots\frac{(n-(n-2))(n-(n-1))}{1\cdot 2} = \frac{n!}{2^{n/2}}$. How fast does this function grow with n?

The matching problem also has many applications.

(1) A first example could be the problem of assigning teachers to class periods in a given quarter. Each teacher is assigned to a specific class period and each class period to a specific teacher. The cost c_{ij} of pairing teacher i to class j could be the willingness of teacher i to teach class j. We would like to find an assignment where the happiness factor is maximized. This is an example of an assignment problem which is a special case of matching.
(2) More to come ...

We will see that the matching problem has extremely efficient algorithms for solving it while the TSP is one of the hardest problems in combinatorial optimization. This will be made precise later.