Lecture #15 10/05/2016

- Reminder of what we did last time.

* TSP (travel salesman problems) all have the following 3 key elements:
  - a traveler
  - a set of sites
  - a set of costs

* A solution to a TSP is a trip starting & ending at a site and visiting all other sites exactly once.
  We call such a trip – a tour.

* Goal: Find a tour with minimal total cost (aka. an optimal solution)

* Hamilton paths and Hamilton circuits

Remarks: (1) In a circuit you can start from any vertex, walk along the arrows and return to the same vertex. Therefore, when asked to compute the number of Hamilton circuits, we will fix the starting (= ending) vertex.

(2) Reversing all the arrows, we get another Hamilton circuit (or a path if we started from a path) with the same cost of travel.

* In all TSP problems, we have a complete graph K_N [N = number of sites]

Last time: There are exactly $(N-1)! := 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (N-1)$ Hamilton circuits in K_N.
(e.g. 2 for K_3, 6 for K_4, 24 for K_5, 120 for K_6, ...)

Note: $N! = (N-1)! \cdot N \Rightarrow$ it is easy to evaluate $N!$ if you know $(N-1)!$

But, we have not answered the second q-n last time:

Q2: What is the number of Hamilton paths in K_N?
require ending vertex ≠ starting vertex

Answer to Q2: There are $N! := 1 \cdot 2 \cdot 3 \cdot \_\_ \cdot N$ Hamilton paths in K_N.

Starting from any of $(N-1)!$ of Hamilton circuits and erasing one of its N edges, we get a Hamilton path.

Reversely, starting from a Hamilton path and adding an edge b/w the starting and ending vertices (it exists as K_N-complete), we get a Hamilton circuit.

Hence, there are exactly $(N-1)! \cdot N = N!$ Hamilton paths in K_N.

Remark: For $N > 10$, the values of $N!$ are extremely large (the function $N \mapsto N!$ grows "very fast").

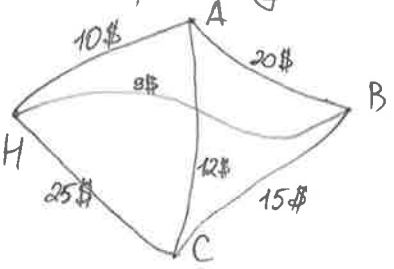
* To summarize, given a TSP problem, we remodel the problem by drawing a complete graph with costs on its edges. Then a tour corresponds to a Hamilton path/circuit.

Goal: Find the Hamilton path or circuit with the minimal total cost (In problem, it will be specified if the ending site must be the same as starting or not)

• The Brute-Force Algorithm (Section 6.3)

Recall example from Monday:

Ex1:



Start and end at H.

Goal: Find the optimal solution.

As we know, there are $3! = 6$ different circuits.

Since we can list all of them, we can also compute the total cost in each of them, and thus find the optimal one.

Tour	Total Cost	Tour
(1) H-A-B-C-H	10+20+15+25 = 70	(4) H-C-B-A-H
(2) H-A-C-B-H	10+12+15+9 = 46	(5) H-B-C-A-H
(3) H-B-A-C-H	9+20+12+25 = 66	(6) H-C-A-B-H.

Key Observation: We can split all Hamilton circuits into the pairs with opposite direction of travel, which obviously have the same total cost. This explains why we list (1-3) and (4-6) separately.

In our case, we see that 46 is the smallest value, hence, the optimal solution is $H \to A \to C \to B \to H$ or traveling opposite direction $H \to B \to C \to A \to H$.

* Check out Example 6.9 in your textbook, where similar approach is applied to the case of $N=5$ sites. In this case we have 24 Hamilton circuits, which split into 12 pairs as above.

* Thus, the "Brute-Force" algorithm consists in (1) listing all Hamiltonian paths in the associated complete graph K_N, (2) calculating the total cost (weight) of each of them, (3) choosing the Hamiltonian circuit with the least total cost.

Warning: In practice, the brute-force algorithm can be applied only for $N \leq 5$ if you don't have a computer. However, even with the fastest computer computation for $N > 23$ will take months.

Hence, we need some other ideas.

Lecture #15

10/05/2016

- The Nearest-Neighbor and Repetitive Nearest Neighbor Algorithms (Sect. 6.4)

Nearest-Neighbor Algorithm works as follows:

1. Start at the given starting vertex.
   If none is given, pick one!
2. Go to the nearest neighbor (i.e. the vertex with the smallest weight on the edge connecting it to the starting vertex)
3. Step-by-step go to the nearest neighbors which have not been visited yet
4. After all vertices have been visited, return to the starting vertex

! Warning: This is an approximate algorithm, i.e. it does not provide an optimal solution, but in many cases it provides a good approximation

To estimate how good our approximation is, we define the notion "relative error of a tour". Given a tour with cost C, we call the number $\varepsilon = \frac{C - Opt}{Opt}$ - the relative error of a tour, where Opt denotes the total cost of an optimal tour.

Q: Why do we care about approximate algorithms?

A: In the situations when $N > 20$, finding the best tour can take months even if using the Super-Computer. Therefore, we want to find a good approximation in reasonable amount of time.

Repetitive Nearest-Neighbor Algorithm

- Repeat the Nearest-Neighbor Algorithm, starting from every vertex.
- Of the N Nearest-Neighbor tours thus obtained, choose the one with the lowest cost. If required to start from a particular vertex, rewrite this tour accordingly.