Algorithms: Design and Analysis, Part II

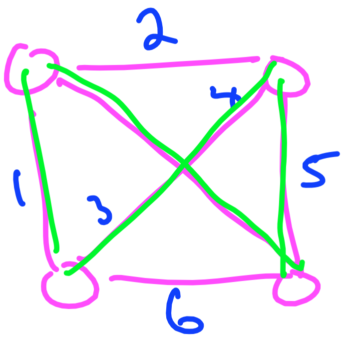
Exact Algorithms for NP-Complete Problems

The Traveling Salesman Problem

The Traveling Salesman Problem

Input: a complete undirected graph with nonnegative edge costs.

Output: a minimum-cost tour (i.e., a cycle that visits every vertex exactly once).



OPT = 13

Brute-force search: takes $\approx n!$ time

[tractable only for $n \approx 12, 13$]

Dynamic Programming: will obtain $O(n^2 2^n)$ running time

[tractable for n close to 30]

Tim Roughgarden

A Optimal Substructure Lemma?

Idea: Copy the format of the Bellman-Ford algorithm.

Proposed subproblems: For every edge budget $i \in \{0,1,2,\dots,n\}$, destination $j \in \{1,2,\dots,n\}$, let

L_{ij} = length of a shortest path from 1 to j that uses at most i edges.

Question: What prevents using these subproblems to obtain a polynomial-time algorithm for TSP?

(A) there is a super-polynomial number of subproblems

(B) can't efficiently compute solutions to bigger subproblems from smaller ones

(C) solving all subproblems doesn't solve original problem

(D) nothing!

A Optimal Substructure Lemma II?

Proposed subproblems: For every edge budget $i \in \{0,1,2,...,n\}$, destination $j \in \{1,2,...,n\}$, let

L_{ij} = length of shortest path from 1 to j that uses exactly i edges.

Question: What prevents using these subproblems to obtain a polynomial-time algorithm for TSP?

(A) there is a super-polynomial number of subproblems

(B) can't efficiently compute solutions to bigger subproblems from smaller ones

(C) solving these subproblems doesn't solve the original problem.

(D) nothing!

A Optimal Substructure Lemma III?

Proposed Subproblems: For every edge budget $i \in \{1, 2, ..., n\}$, destination $j \in \{1, 2, ..., n\}$, let

L_{ij} = length of a shortest path from 1 to j with exactly i edges and no repeated vertices

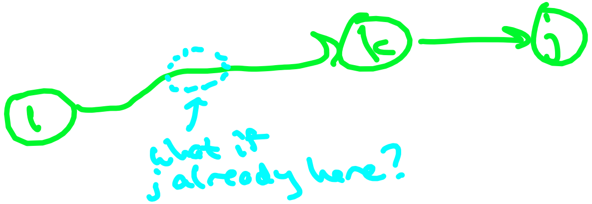
Question: What prevents using these subproblems to design a polynomial-time algorithm for TSP?

(A) there is a super-polynomial number of subproblems
(B) can't efficiently compute solutions to bigger subproblems from smaller ones
(C) solving all subproblems doesn't solve the original problem
(D) nothing!

Tim Roughgarden

A Optimal Substructure Lemma III?

Hope: use the following recurrence:



$$L_{ij} = \min_{k \neq 1, j} \{ \underbrace{L_{i-1,k}}_{\text{shortest path from 1 to k, (i-1) edges, no repeated vertices}} + \underbrace{c_{kj}}_{\text{cost of final hop}} \}$$

Problem: what if j already appears on the shortest $1 \to k$ path with $(i-1)$ edges and no repeated vertices?

$\Rightarrow$ concatenating (k,j) yields a second visit to j (not allowed)

Upshot: to enforce constraint that each vertex visited exactly once, need to remember the <u>identities</u> of vertices visited in subproblem.