Exact Algorithms for NP-Complete Problems

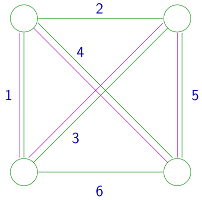
The Traveling Salesman Problem

Algorithms: Design and Analysis, Part II

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]

A Optimal Substructure Lemma?

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

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

D) Nothing!

A Optimal Substructure Lemma III?

Proposed subproblems: For every edge budget $i \in \{0, 1, \ldots, n\}$, destination $j \in \{1, 2, \ldots, n\}$, let
L_{ij} = length of shortest path from 1 to j with exactly i edges and no repeated vertices

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 original problem

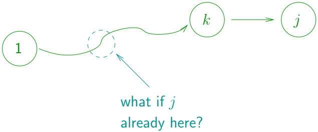
D) Nothing!

A Optimal Substructure Lemma III? (con'd)

Hope: Use the following recurrence: $L_{ij} = \min_{k \neq 1,j} \{ L_{i-1,k} + c_{kj} \}$

shortest path from 1 to k, $(i-1)$ edges no repeated vertices

cost of final hop



Problem: What if j already appears on the shortest $1 \rightarrow k$ path with $(i-1)$ edges and no repeated vertices?
$\Rightarrow$ Concatenating (k_{ij}) 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.