COL 351 : Analysis & Design of Algorithms

LECTURE 39

NP-COMPLETENESS II : THE CLASS NP

NOV 06, 2024 | ROHIT VAISH

An Algorithmic Mystery

Minimum Spanning Tree

Traveling Salisman Problem

An Algorithmic Mystery

Minimum Spanning Tree

input: an undirected graph $G = (V, E)$
with real-valued edge cost c_e
for each edge e

Traveling Salesman Problem

input: an undirected graph $G = (V, E)$
with real-valued edge cost c_e
for each edge e

An Algorithmic Mystery

Minimum Spanning Tree

input: an undirected graph $G = (V, E)$ with real-valued edge cost c_e for each edge e

output: a spanning tree T with minimum total cost $\sum_{e \in T} c_e$.

Traveling Salesman Problem

input: an undirected graph $G = (V, E)$ with real-valued edge cost c_e for each edge e

output: a tour T with minimum total cost $\sum_{e \in T} c_e$.

An Algorithmic Mystery

Minimum Spanning Tree

input: an undirected graph $G = (V, E)$ with real-valued edge cost c_e for each edge e

output: a spanning tree T with minimum total cost $\sum_{e \in T} c_e$.

search space: n^{n-2} spanning trees in complete graphs

Traveling Salesman Problem

input: an undirected graph $G = (V, E)$ with real-valued edge cost c_e for each edge e

output: a tour T with minimum total cost $\sum_{e \in T} c_e$.

search space: $\frac{1}{2}(n-1)!$ on complete graphs

An Algorithmic Mystery

Minimum Spanning Tree

input: an undirected graph $G=(V,E)$ with real-valued edge cost c_e for each edge e

output: a spanning tree T with minimum total cost $\sum_{e \in T} c_e$.

search space: n^{n-2} spanning trees in complete graphs

algorithms: Prim's, Kruskal's, $O(m \log n)$

Traveling Salesman Problem

input: an undirected graph $G=(V,E)$ with real-valued edge cost c_e for each edge e

output: a tour T with minimum total cost $\sum_{e \in T} c_e$.

search space: $\frac{1}{2}(n-1)!$ on complete graphs

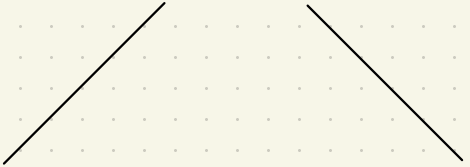
algorithms: 

Easy : Problems with poly-time algorithms

Hard : Problems without any polynomial-time algorithm

Easy : Problems with poly-time algorithms

Hard : Problems without any polynomial-time algorithm



because one doesn't exist?

because we have failed to find one?

Easy : Problems with poly-time algorithms

Hard : Problems without any polynomial-time algorithm

Physics

Chemistry

Economics

Biology

Games

because one doesn't exist?

because we have failed to find one?

Easy : Problems with poly-time algorithms

Hard : Problems without any polynomial-time algorithm

Physics
Chemistry
Economics
Biology
Games

because one doesn't exist?

because we have failed to find one?

Conjecture [Edmonds '67]

TSP doesn't have a poly-time algo.

Easy : Problems with poly-time algorithms

Hard : Problems without any polynomial-time algorithm

Physics
Chemistry
Economics
Biology
Games

because one doesn't exist?

because we have failed to find one?

Conjecture [Edmonds '67]

TSP doesn't have a poly-time algo.

Short of a mathematical proof, how to amass evidence of hardness?

Easy : Problems with poly-time algorithms

Hard : Problems without any polynomial-time algorithm

Physics
Chemistry
Economics
Biology
Games

because one doesn't exist?

because we have failed to find one?

Conjecture [Edmonds '67]

TSP doesn't have a poly-time algo.

Short of a mathematical proof, how to amass evidence of hardness?

Relative hardness and reductions!

"I can't find an efficient algorithm, but neither can all these famous people."

Source : Garey and Johnson (1979)

Building a case with reductions

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems

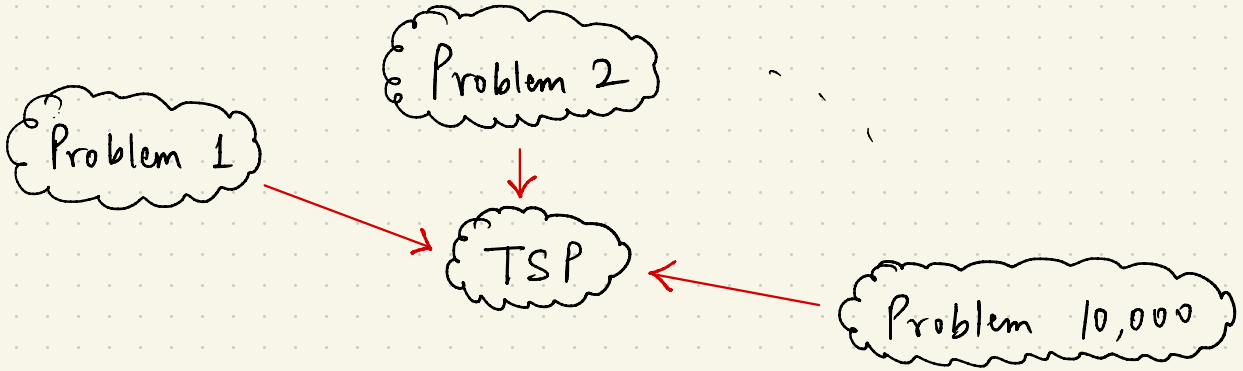
* Show that solving your problem (e.g., TSP) will solve all problems in C.

reduction

Building a case with reductions

* Choose a large class C of computational problems
* Show that solving your problem (e.g., TSP) will solve all problems in C.

reduction



BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems

* Show that solving your problem (e.g., TSP) will solve all problems in C.

reduction

$\Rightarrow$ If TSP is easy, then all problems in C are also easy.

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems

* Show that solving your problem (e.g., TSP) will solve all problems in C.

reduction

$\Rightarrow$ If TSP is easy, then all problems in C are also easy.

If you are willing to believe that problems in C are "hard", then you should also believe that TSP is hard.

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems

* Show that solving your problem (e.g., TSP) will solve all problems in C.

reduction

$\Rightarrow$ If TSP is easy, then all problems in C are also easy.

If you are willing to believe that problems in C are "hard", then you should also believe that TSP is hard.

Bigger the set C $\Rightarrow$ stronger evidence for TSP's intractability

CHOOSING A CLASS FOR TSP

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

Too ambitious ! Includes the HALTING problem.

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

Too ambitious ! Includes the HALTING problem.

"Given a program, determine whether it goes into an infinite loop or eventually halts."

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

Too ambitious ! Includes the HALTING problem.

"Given a program, determine whether it goes into an infinite loop or eventually halts."

(Turing, 1936) HALTING problem is undecidable.

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

Too ambitious ! Includes the HALTING problem.

"Given a program, determine whether it goes into an infinite loop or eventually halts."

(Turing, 1936) HALTING problem is undecidable.

Cannot be solved by a computer in any finite amount of time
(not even exponential, or factorial, ...)

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

* Birth of computer science
* Formulated Turing machines — formal model of general-purpose computers
* Showed undecidability of HALTING problem.

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

* Birth of computer science

* Formulated Turing machines — formal model of general-purpose computers

* Showed undecidability of HALTING problem.

Thus, computer scientists have been aware of computers' limitations literally from Day 1.

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

Some problems (e.g. HALTING) can't reduce to TSP.

because TSP is solvable in finite time via exhaustive search

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

Some problems (e.g. HALTING) can't reduce to TSP.

🤔 C = all problems that can be solved by exhaustive search

CHOOSING A CLASS FOR TSP

Why not choose C = ALL computational problems ?

Some problems (e.g. HALTING) can't reduce to TSP.

🤔 C = all problems that can be solved by exhaustive search

The class NP

THREE TYPES OF COMPUTATIONAL PROBLEMS

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

e.g., is there a TSP tour of cost ≤ 100?

no need to actually produce a tour if one exists

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

e.g., is there a TSP tour of cost ≤ 100?

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

e.g., is there a TSP tour of cost ≤ 100?

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

e.g., return any TSP tour of cost ≤ 100, if one exists.

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

e.g., is there a TSP tour of cost ≤ 100?

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

e.g., return any TSP tour of cost ≤ 100, if one exists.

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "yes" if there is a feasible solution and "No" otherwise.

e.g., is there a TSP tour of cost ≤ 100?

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

e.g., return any TSP tour of cost ≤ 100, if one exists.

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

e.g., return the min cost TSP tour

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

NOTE: every optimization has a natural search version.

e.g., TSP tour with cost ≤ 100,

Knapsack solution with value ≥ 100, etc.

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

Search reduces to optimization : easy! (Exercise)

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

Search reduces to optimization: easy! (Exercise)

Optimization reduces to search: via binary search (Exercise)

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem : Output "Yes" if true is a feasible solution and "No" otherwise.

Search Problem : Output a feasible solution if one exists, and "no solution" otherwise

Optimization Problem : Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if true is a feasible solution and "No" otherwise.

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

NP: Commonly defined for decision problems

We will define it for search problems

THREE TYPES OF COMPUTATIONAL PROBLEMS

Decision Problem: Output "Yes" if there is a feasible solution and "No" otherwise.

Search Problem: Output a feasible solution if one exists, and "no solution" otherwise

↕

Optimization Problem: Output a feasible solution with best-possible objective function value (or "no solution" if none exist)

NP: commonly defined for decision problems

We will define it for search problems

↳ technically, functional NP (FNP)

THE CLASS NP

THE CLASS NP

Recall our goal : To define the class of problems solvable via exhaustive search.

THE CLASS NP

Recall our goal : To define the class of problems solvable via exhaustive search.

What are the minimal ingredients necessary to solve a problem via naive exhaustive search?

THE CLASS NP

Efficient recognition of alleged solutions.

THE CLASS NP

Efficient recognition of alleged solutions.

e.g.,

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

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

THE CLASS NP

Efficient recognition of alleged solutions.

e.g., checking whether $v_5 \to v_{17} \to v_{105} \to v_2 \to \cdots \to v_5$

is a valid TSP tour of cost ≤ 100.

THE CLASS NP

A search problem belongs to the complexity class NP if

THE CLASS NP

A search problem belongs to the complexity class NP if

(1) For every instance of the problem, every candidate solution has description length (in bits) bounded above by a polynomial function of the input size.

THE CLASS NP

A search problem belongs to the complexity class NP if

(1) For every instance of the problem, every candidate solution has description length (in bits) bounded above by a polynomial function of the input size.

e.g., any sequence of vertices is of polynomial size

THE CLASS NP

A search problem belongs to the complexity class NP if

(1) For every instance of the problem, every candidate solution has description length (in bits) bounded above by a polynomial function of the input size.

e.g., any sequence of vertices is of polynomial size

(2) For every instance and every candidate solution, the feasibility of the solution can be confirmed or denied in time polynomial in the input size.

THE CLASS NP

A search problem belongs to the complexity class NP if

(1) For every instance of the problem, every candidate solution has description length (in bits) bounded above by a polynomial function of the input size.

e.g., any sequence of vertices is of polynomial size

(2) For every instance and every candidate solution, the feasibility of the solution can be confirmed or denied in time polynomial in the input size.

e.g., checking validity of a sequence of vertex possible in poly time

THE CLASS NP

A search problem belongs to the complexity class NP if it can be solved by a Nondeterministic Turing machine in Polynomial time.

THE CLASS NP

A search problem belongs to the complexity class NP if it can be solved by a Nondeterministic Turing machine in Polynomial time.

NP ≠ Not Polynomial

Examples of Problems in NP

Examples of Problems in NP

1. Search version of TSP

EXAMPLES OF PROBLEMS IN NP

(1) Search version of TSP

input: An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a tour with cost $\leq t$ if one exists
report "no solution" otherwise

EXAMPLES OF PROBLEMS IN NP

① Search version of TSP

input: An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a tour with cost $\leq t$ if one exists
report "no solution" otherwise

Why in NP?

EXAMPLES OF PROBLEMS IN NP

① Search version of TSP

input: An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a tour with cost $\leq t$ if one exists
report "no solution" otherwise

Why in NP?

* Any sequence on n vertices can be described using $O(n \log n)$ bits.

EXAMPLES OF PROBLEMS IN NP

① Search version of TSP

input : An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a tour with cost $\leq t$ if one exists
report "no solution" otherwise

Why in NP ?

* Any sequence on n vertices can be described using $O(n \log n)$ bits.
* checking whether a given sequence of vertices is a valid tour and has cost $\leq t$ can be done in polynomial time

Examples of Problems in NP

(2) Search version of MST

EXAMPLES OF PROBLEMS IN NP

(2) Search version of MST

input : An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a spanning tree with cost $\leq t$ if one exists
report "no solution" otherwise

EXAMPLES OF PROBLEMS IN NP

(2) Search version of MST

input : An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a spanning tree with cost $\leq t$ if one exists
report "no solution" otherwise

Why in NP ?

EXAMPLES OF PROBLEMS IN NP

(2) Search version of MST

input: An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a spanning tree with cost $\leq t$ if one exists
report "no solution" otherwise

Why in NP?

* Any subgraph on n vertices can be described using $O(n \log n + m \log m)$ bits.

EXAMPLES OF PROBLEMS IN NP

(2) Search version of MST

input : An undirected graph $G = (V, E)$ with edge costs $\{c_e\}_{e \in E}$.

output: Return a spanning tree with cost $\leq t$ if one exists
report "no solution" otherwise

Why in NP ?

* Any subgraph on n vertices can be described using $O(n \log n + m \log m)$ bits.
* checking whether a given subgraph is a valid spanning tree and has cost $\leq t$ can be done in polynomial time

Examples of Problems in NP

(3) Search version of sequence alignment

EXAMPLES OF PROBLEMS IN NP

(3) Search version of sequence alignment

input: Two strings X and Y, gap penalty α_{gap}, mismatch penalty α_{xy}

output: Return an alignment with cost $\leq t$ if one exists
report "no solution" otherwise

EXAMPLES OF PROBLEMS IN NP

(3) Search version of sequence alignment

input : Two strings X and Y, gap penalty α_{gap}, mismatch penalty α_{xy}

output: Return an alignment with cost $\leq t$ if one exists
report "no solution" otherwise

Why in NP?

* Any alleged alignment can be described using $O(n+m)$ bits.
* checking whether a given pair of strings constitute a valid alignment and has cost $\leq t$ can be done in polynomial time