COL 351 : Analysis & Design of Algorithms

LECTURE 38

NP-COMPLETENESS I : INTRODUCTION

NOV 05, 2024 | ROHIT VAISH

STORY SO FAR

STORY SO FAR

Problem	Algorithm	Running time
Sorting	Merge Sort	$O(n \log n)$
Strongly Connected Components	Kosaraju	$O(n+m)$
Shortest Paths	Dijkstra	$O((m+n) \log n)$
Minimum Spanning Tree	Kruskal	$O(m \log n)$
Sequence Alignment	Needleman-Wunsch	$O(mn)$
All Pairs Shortest Paths	Floyd-Warshall	$O(n^3)$

An Algorithmic Mystery

An Algorithmic Mystery

Minimum Spanning Tree

An Algorithmic Mystery

Minimum Spanning Tree

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 $\sum_{e \in T} c_e$.

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

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, $O(m \log n)$

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

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

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 $\sum_{e \in T} c_e$.

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 $\sum_{e \in T} c_e$.

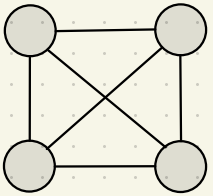
search space: n^{n-2} spanning trees in complete graphs

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 $\sum_{e \in T} c_e$.



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 $\sum_{e \in T} c_e$.

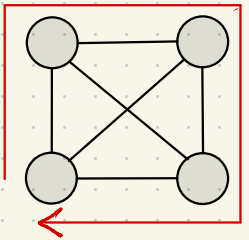
search space: n^{n-2} spanning trees in complete graphs

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 $\sum_{e \in T} c_e$.



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 $\sum_{e \in T} c_e$.

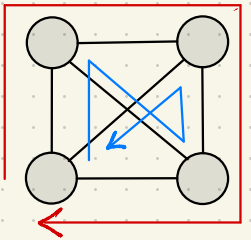
search space: n^{n-2} spanning trees in complete graphs

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 $\sum_{e \in T} c_e$.



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: ?

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

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: 

An Algorithmic Mystery

Applications

* sequenced processing of tasks
* genome reconstruction
* many more!

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: 

An Algorithmic Mystery

Applications

* sequenced processing of tasks
* genome reconstruction
* many more!

Fact

No known fast (i.e., poly time) algorithm for TSP (despite decades of effort)

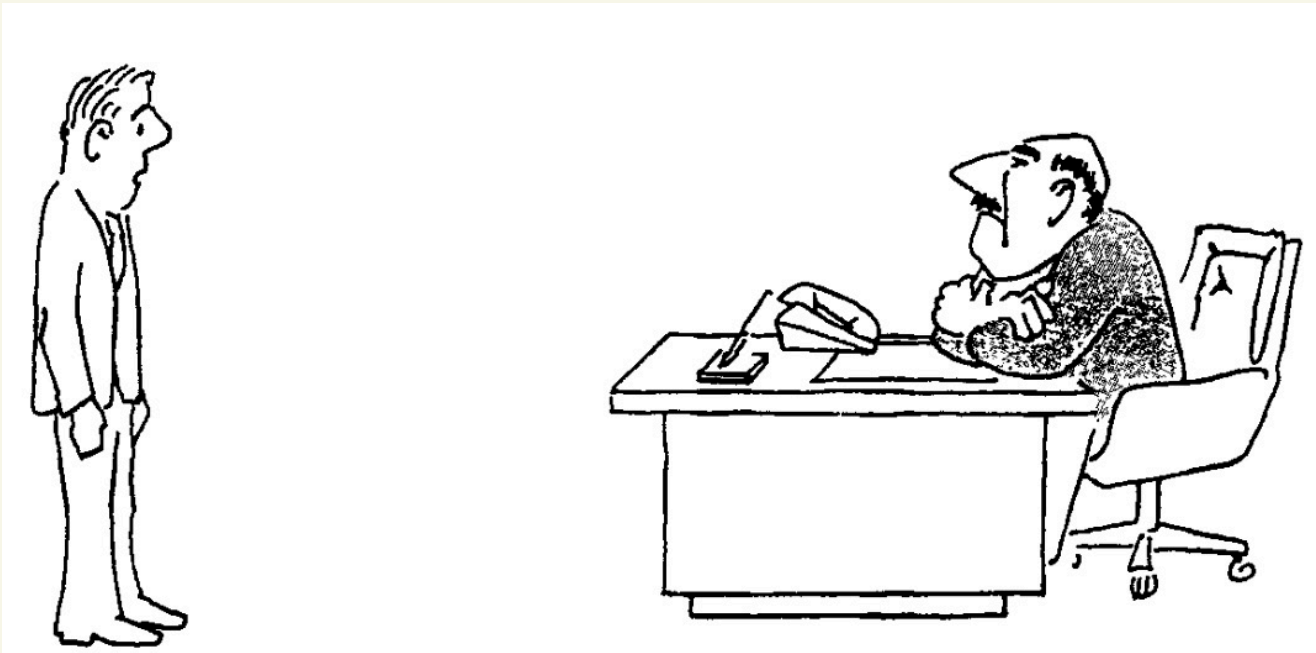
Traveling Salesman Problem

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 $\sum_{e \in T} c_e$.

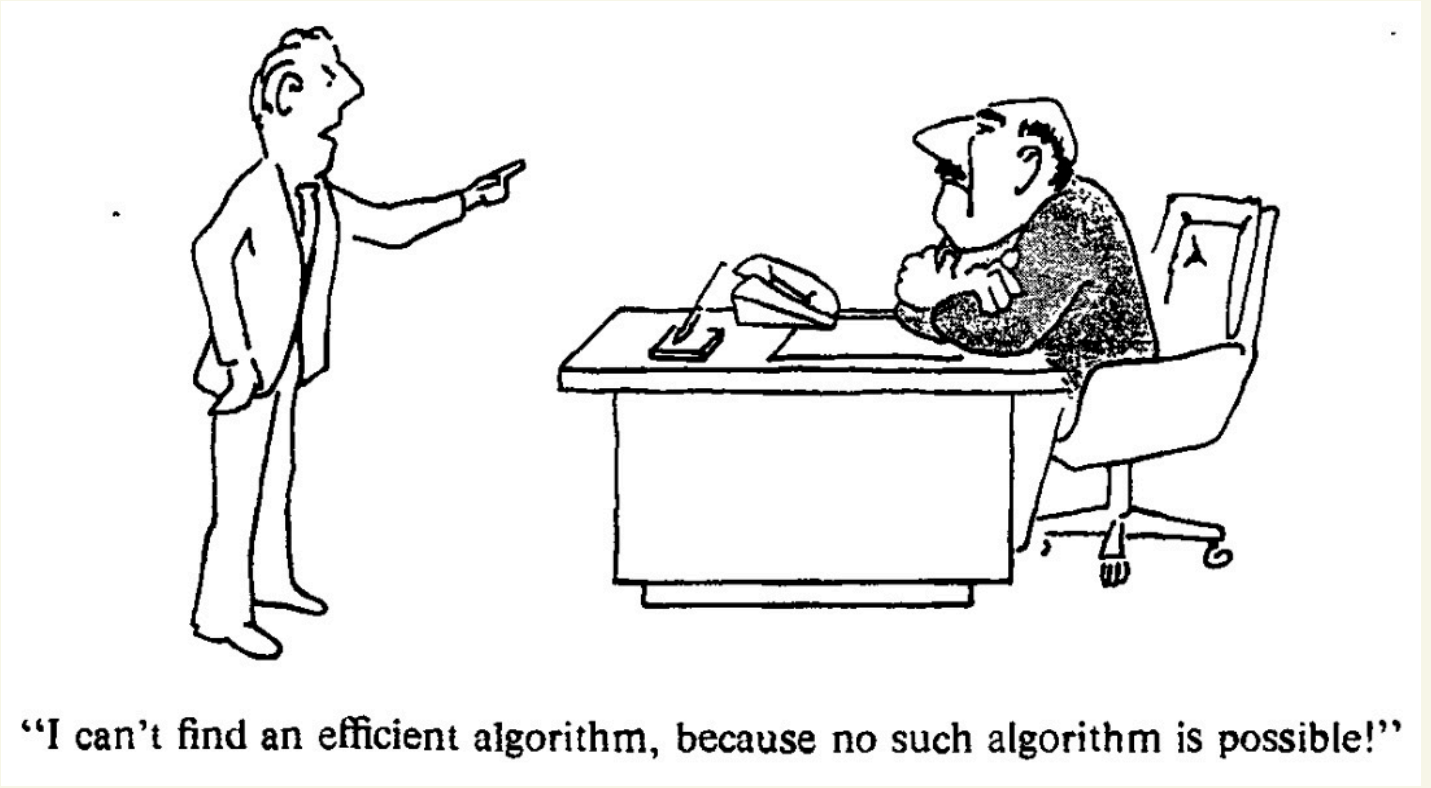
search space: $\frac{1}{2}(n-1)!$ on complete graphs

algorithms: 



"I can't find an efficient algorithm, I guess I'm just too dumb."

Source : Garey and Johnson (1979)



"I can't find an efficient algorithm, because no such algorithm is possible!"

Source : Garey and Johnson (1979)

JOURNAL OF RESEARCH of the National Bureau of Standards—B. Mathematics and Mathematical Physics
Vol. 71B, No. 4, October–December 1967

Optimum Branchings*

Jack Edmonds

traveling saleman problem [cf. 4]. I conjecture that there is no good algorithm for the traveling saleman problem. My reasons are the same as for any mathematical conjecture: (1) It is a legitimate mathematical possibility, and (2) I do not know.

good = polynomial-time

So far, we haven't managed to prove Edmonds' conjecture.

So far, we haven't managed to prove Edmonds' conjecture.

We have identified thousands of "TSP-like" problems:

* Graph coloring
* Integer Programming
* Ising Models
* Longest Paths
* Protein Folding
* Rubik's cube
* Number Partitioning
* Boolean Satisfiability
* ---

So far, we haven't managed to prove Edmonds' conjecture.

We have identified thousands of "TSP-like" problems:

* Graph coloring
* Integer Programming
* Ising Models
* Longest Paths
* Protein Folding
* Rubik's cube
* Number Partitioning
* Boolean Satisfiability
* ---

How can we build a compelling case that the conjecture is true?

So far, we haven't managed to prove Edmonds' conjecture.

We have identified thousands of "TSP-like" problems:

* Graph coloring
* Integer Programming
* Ising Models
* Longest Paths
* Protein Folding
* Rubik's cube
* Number Partitioning
* Boolean Satisfiability
* ---

How can we build a compelling case that the conjecture is true?

RELATIVE INTRACTABILITY

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

Source : Garey and Johnson (1979)

EASY PROBLEM

EASY PROBLEM

A problem is polynomial-time solvable if there is a polynomial-time algorithm that correctly solves it on all inputs.

EASY PROBLEM

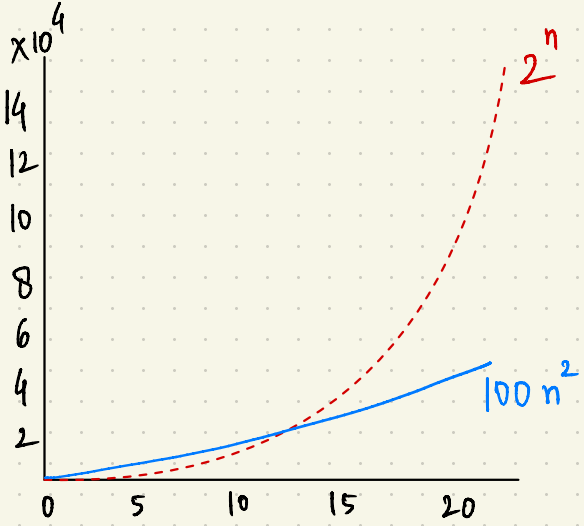
Equivalently:

A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.

EASY PROBLEM

Equivalently:

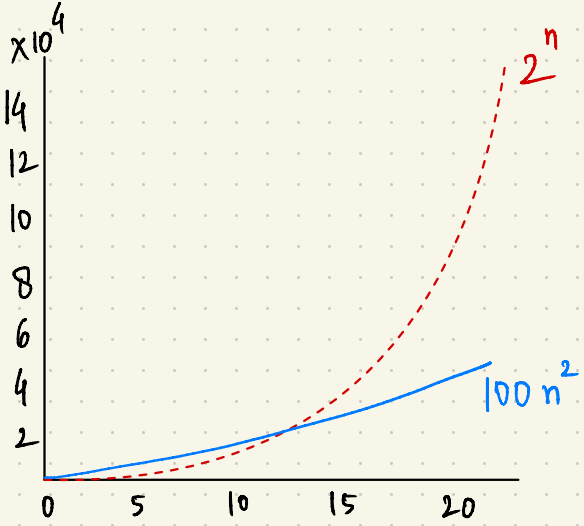
A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.



EASY PROBLEM

Equivalently:

A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.

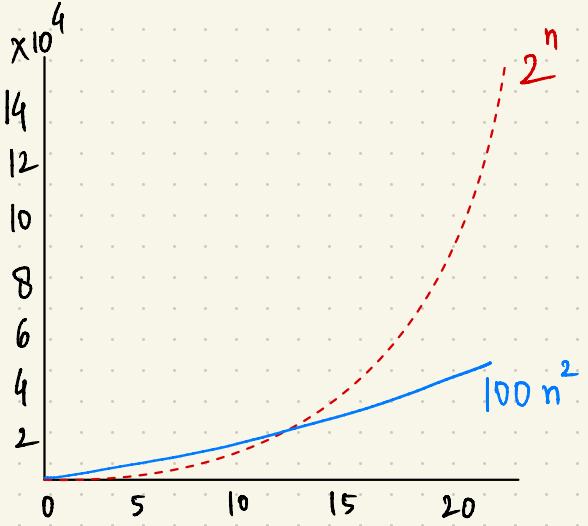


Fix time budget. Double compute power.

EASY PROBLEM

Equivalently:

A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.



Fix time budget. Double compute power.

Before After

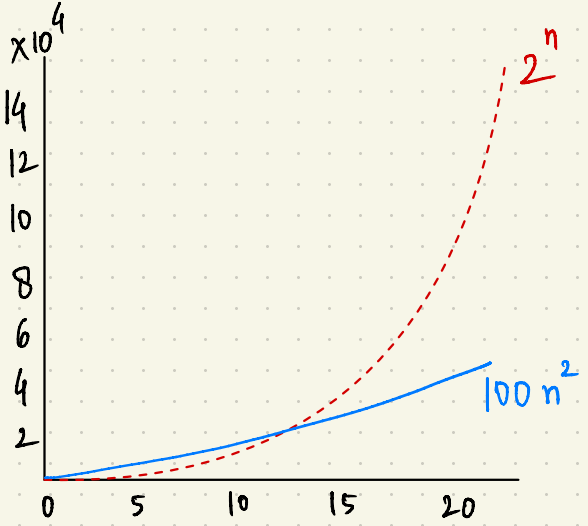
$100n^2$

2^n

EASY PROBLEM

Equivalently:

A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.



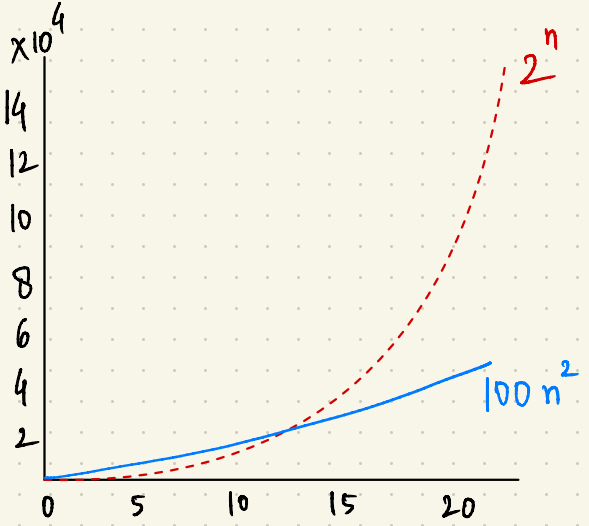
Fix time budget. Double compute power.

	Before	After
$100n^2$	1,000,000	?
2^n		

EASY PROBLEM

Equivalently:

A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.



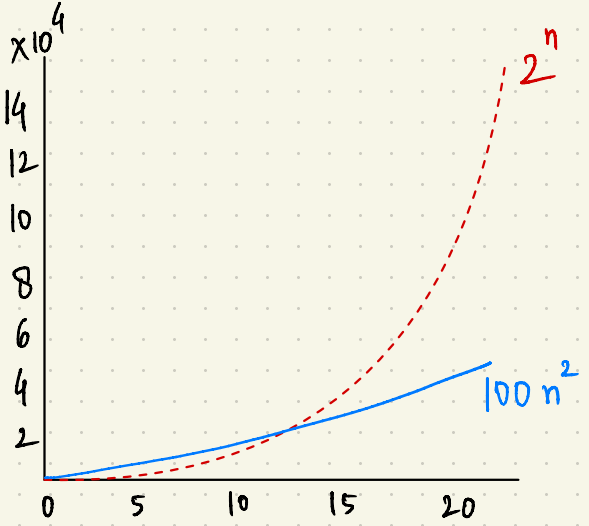
Fix time budget. Double compute power.

	Before	After
$100n^2$	1,000,000	1,414,213
2^n		

EASY PROBLEM

Equivalently:

A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.



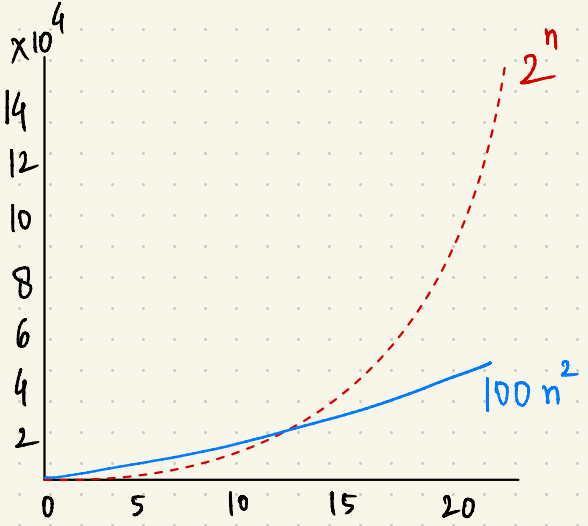
Fix time budget. Double compute power.

	Before	After
$100n^2$	1,000,000	1,414,213
2^n	1,000,000	?

EASY PROBLEM

Equivalently:

A problem is polynomial-time solvable if there is an algorithm for which the solvable input size (for a fixed time budget) scales multiplicatively with increasing computational power.



Fix time budget. Double compute power.

	Before	After
$100n^2$	1,000,000	1,414,213
2^n	1,000,000	1,000,001

HARD PROBLEM

HARD PROBLEM

A problem with no polynomial-time algorithm ?

HARD PROBLEM

A problem with no polynomial-time algorithm ?

because there isn't one ?

or because we haven't found one yet ?

HARD PROBLEM

A problem with no polynomial-time algorithm ?

- because there isn't one ?
- or because we haven't found one yet ?
  - we'll never admit to this !

HARD PROBLEM

A problem with no polynomial-time algorithm ?

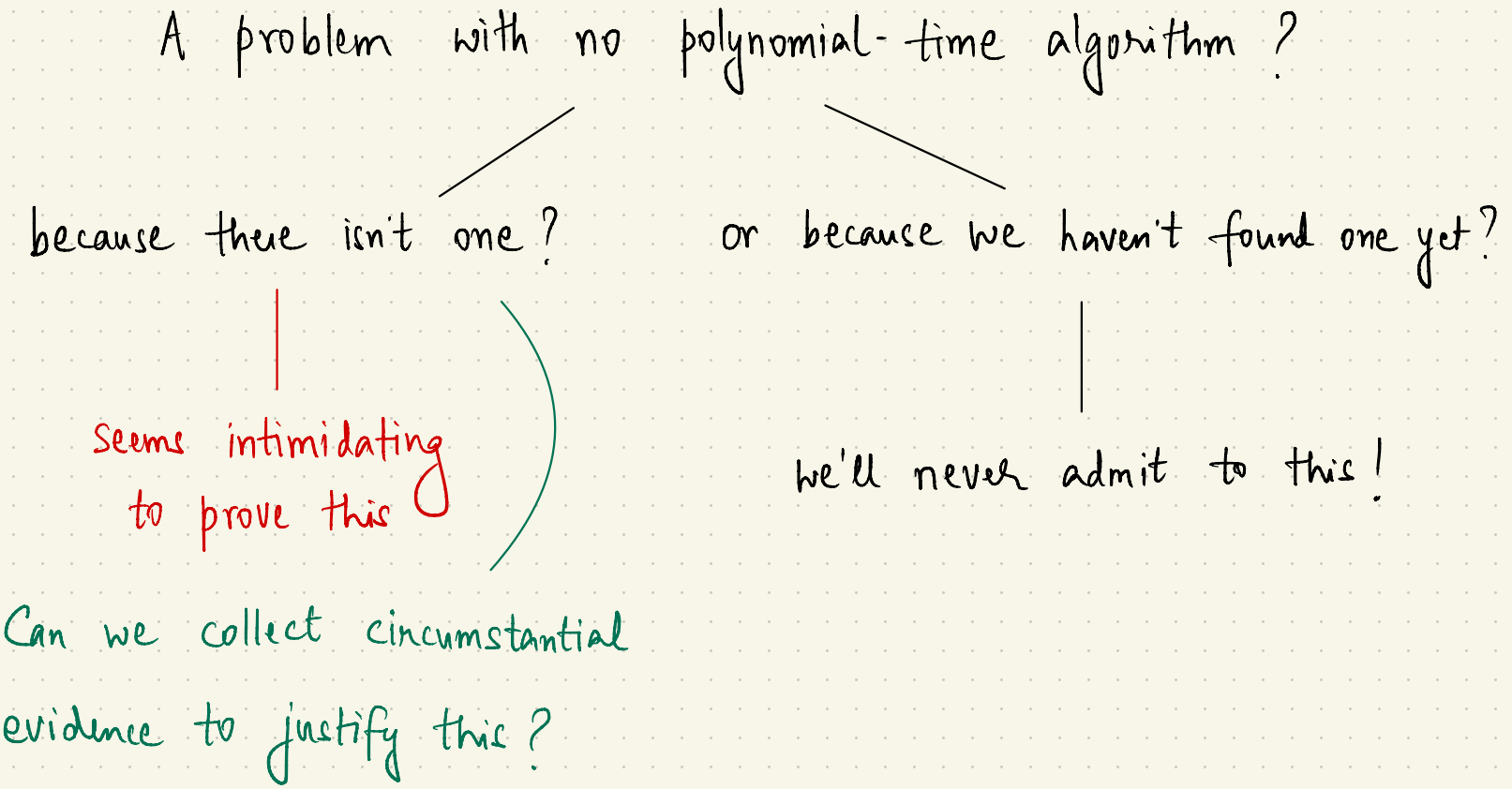
because there isn't one ?

Seems intimidating to prove this

or because we haven't found one yet ?

we'll never admit to this !

HARD PROBLEM



HARD PROBLEM

Weak evidence of hardness

Strong evidence of hardness

HARD PROBLEM

Weak evidence of hardness

A poly-time algorithm for Traveling Salesman Problem would solve a problem that has resisted the efforts of thousands of brilliant minds over many decades.

Strong evidence of hardness

HARD PROBLEM

Weak evidence of hardness

A poly-time algorithm for Traveling Salesman Problem would solve a problem that has resisted the efforts of thousands of brilliant minds over many decades.

Strong evidence of hardness

A poly-time algorithm for Traveling Salesman Problem would solve thousands of problems that have resisted the efforts of tens of thousands of brilliant minds over many decades.

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.

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems

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

* Therefore, your problem must also be out of reach.

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems

NP-hard

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

Reduction

* Therefore, your problem must also be out of reach.

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems
* Show that solving your problem (eg., TSP) will solve all problems in C.
* Therefore, your problem must also be out of reach.

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems
* Show that solving your problem (eg., TSP) will solve all problems in C.
* Therefore, your problem must also be out of reach.

If any one problem in C is proven to be not poly-time solvable, then so is TSP.

BUILDING A CASE WITH REDUCTIONS

* Choose a large class C of computational problems
* Show that solving your problem (eg., TSP) will solve all problems in C.
* Therefore, your problem must also be out of reach.

If any one problem in C is proven to be not poly-time solvable, then so is TSP.

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