Approximation Algorithms

Announcements

- Problem Set 9 due right now.
- **Final exam** this Monday, Hewlett 200 from 12:15PM-3:15PM
  - Please let us know **immediately after lecture** if you want to take the final at an alternate time and haven't yet told us.
- **Final exam review sessions**:
  - **Saturday** from 3:00PM – 5:00PM in Gates 104
  - **Sunday** from 1:00PM – 3:00PM in Gates 104
- Friday Four Square today, 4:15, outside of Gates.

Decision vs. Function Problems

- All of the problems we have encountered this quarter are **decision problems** with yes/no answers.
- Many interesting questions do not have yes/no answers:
  - What is 1 + 1?
  - How many steps does it take this TM to halt on this string?
  - What is the shortest path from u to v?
- These questions are called **function problems** and require some object to be found.

NP-Hard Function Problems

- Some function problems are NP-hard: there is a polynomial-time reduction from any problem in NP to those problems.
- Examples:
  - What is the largest independent set in a graph?
  - What is the length of the longest path in a graph?
  - What is the minimum number of colors required to color a graph?
- A polynomial-time solver for any of these could be used to build a polynomial-time decider for any problem in NP.

NP-Hard Function Problems

- The **maximum independent set problem** (MAX-INDSET) is

  Given a graph G, find an independent set S in G of the largest possible size.

- MAX-INDSET is NP-hard by a reduction from independent set:
  - M = “On input ⟨G, k⟩:
    - Find a maximum independent set in G, call it S.
    - If |S| ≥ k, accept; otherwise, reject.”

NP-Hard Function Problems

- Because they can be used to solve any problem in NP, NP-hard function problems are believed to be computationally infeasible.
  - If **any** NP-hard function problem has a polynomial-time solution, then P = NP.
  - Since the P = NP question is still open, no NP-hard function problems are known to have polynomial-time solutions.

Approximation Algorithms

- An **approximation algorithm** is an algorithm for yielding a solution that is “close” to the correct solution to a problem.
- The definition of “close” depends on the problem:
  - Maximum independent set: Find an independent set that is not “much smaller” than the maximum independent set.
  - Longest path: Find a long path that is not “much smaller” than the longest path.
  - Graph coloring: Find a coloring of the graph that does not use “many more” colors than the optimal coloring.

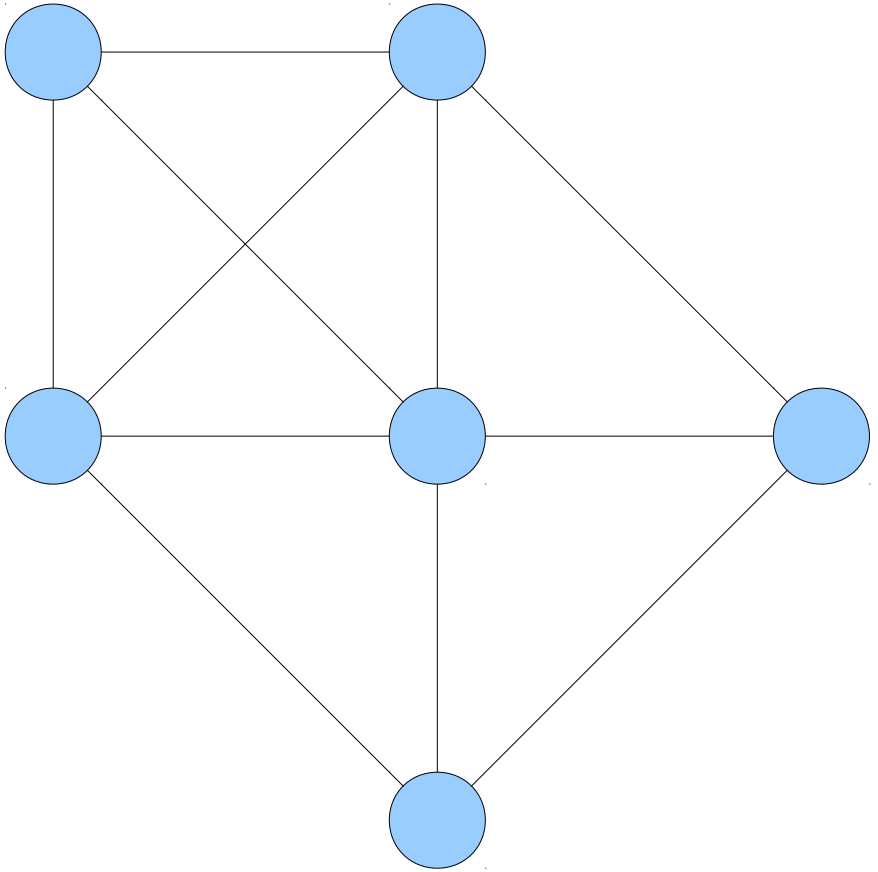
How Good is an Approximation?

- Approximation algorithms are only useful if there is some connection between the approximate answer and the real answer.
- We say that an approximation algorithm is a **k-approximation** if its answer is always within a factor of k of the optimal solution.
  - A **2-approximation** to the graph coloring problem always finds a coloring that uses at most twice the optimal number of colors.
  - A **2/3-approximation** to the longest path problem always finds a path that is at least 2/3 as long as the optimal path.

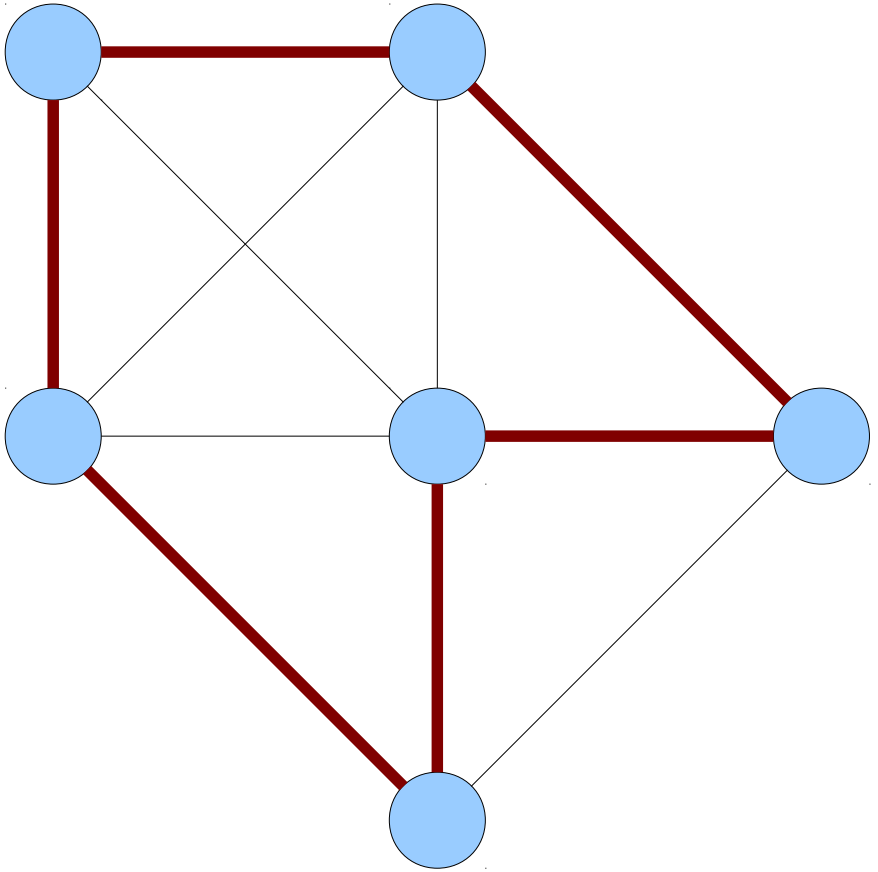
Why Approximations Matter

- A classic NP-hard problem is **job scheduling**.
- Given a set of tasks and a set of workers to perform the tasks, how do you distribute tasks to workers to minimize the total time required (assuming the workers work in parallel?)
  - Applications to manufacturing, operating systems, etc.
- Although finding an optimal solution is NP-hard, there are polynomial-time algorithms for finding **4/3-approximate** solutions.
- If we just need a “good enough” answer, the NP-hardness of job scheduling is not a deterrent to its use.

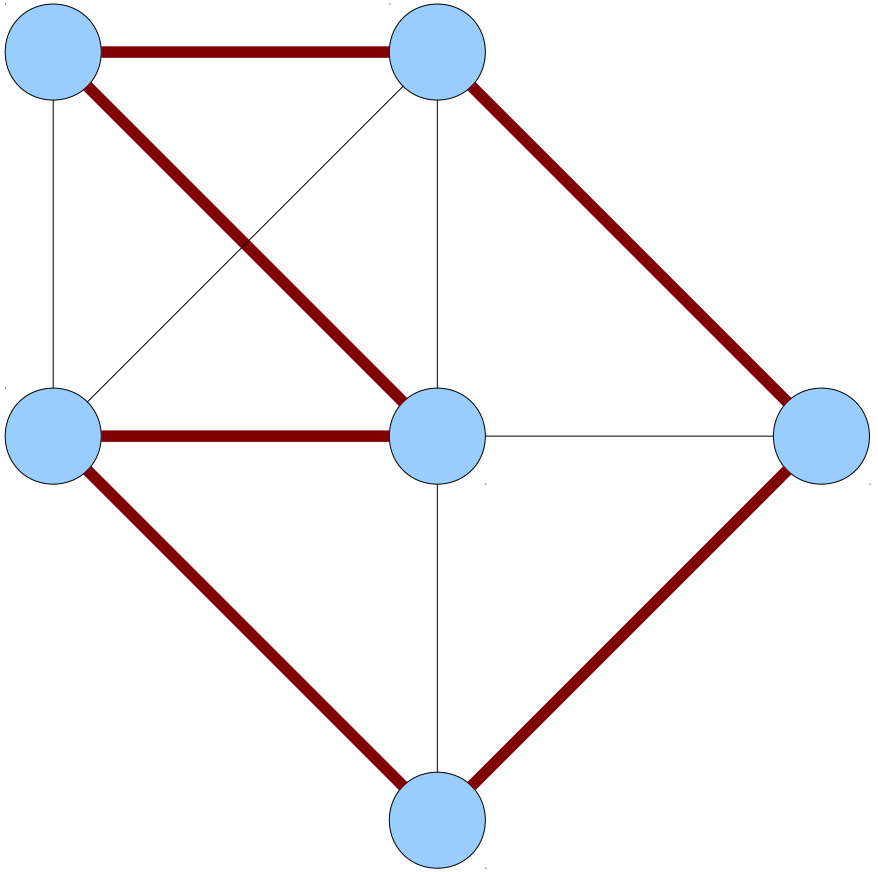
A **Hamiltonian cycle** in an undirected graph G is a simple cycle that visits every node in G.



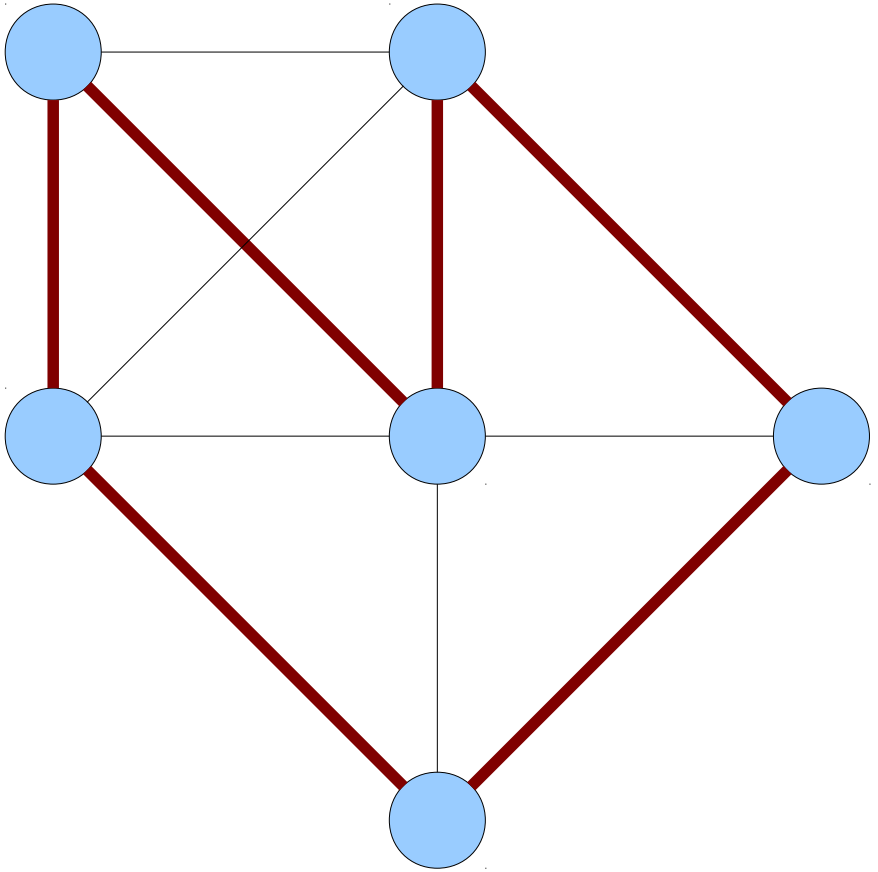
A **Hamiltonian cycle** in an undirected graph G is a simple cycle that visits every node in G.



A **Hamiltonian cycle** in an undirected graph G is a simple cycle that visits every node in G.



A **Hamiltonian cycle** in an undirected graph G is a simple cycle that visits every node in G.



A **Hamiltonian cycle** in an undirected graph G is a simple cycle that visits every node in G.

Hamiltonian Cycles

- The **undirected Hamiltonian cycle problem** is

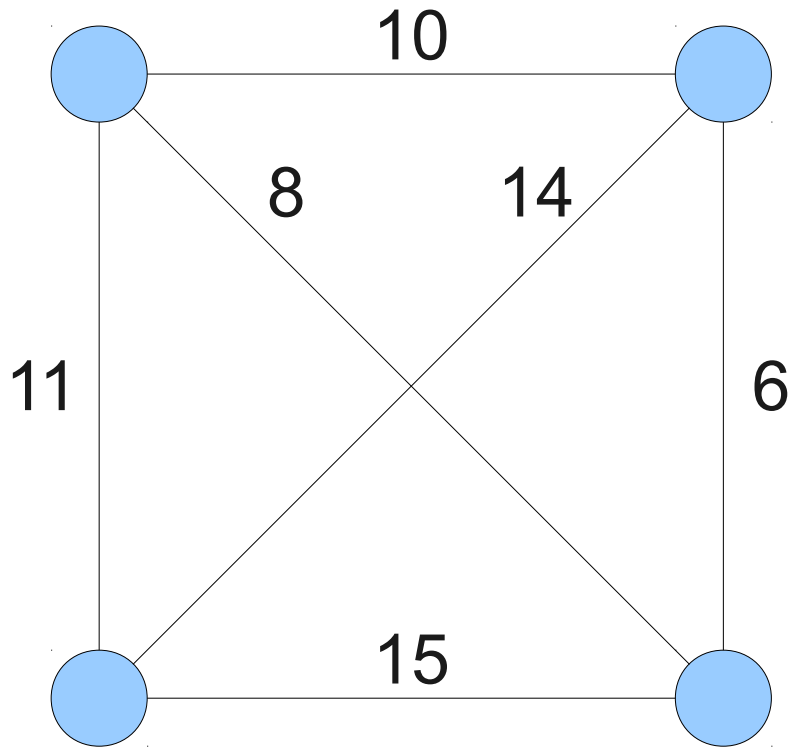
  Given an undirected graph G,
  does G contain a Hamiltonian cycle?

- As a formal language:

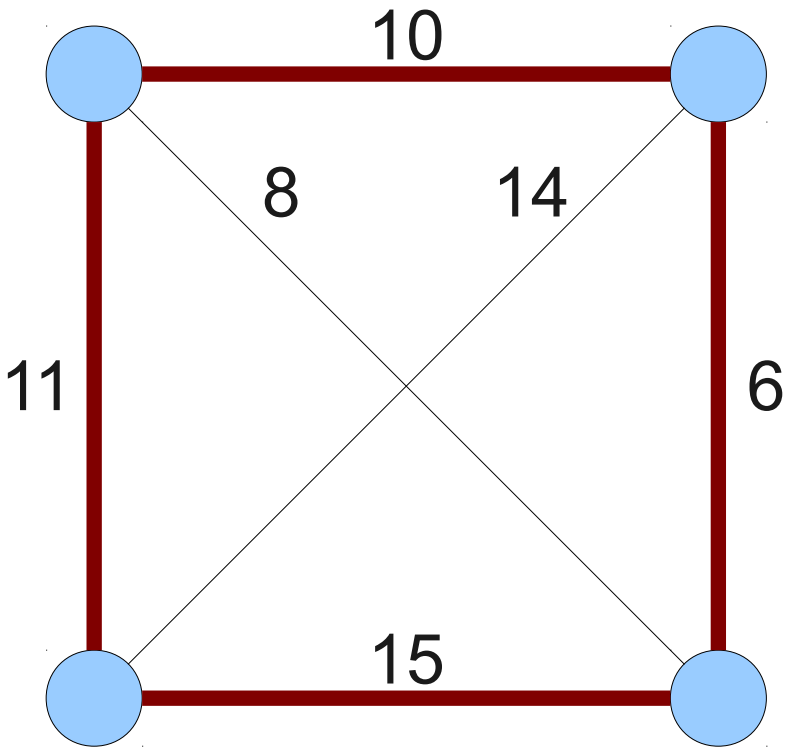
  UHAMCYCLE = { ⟨G⟩ | G is an undirected graph containing a Hamiltonian cycle }

- **Important fact**: *UHAMCYCLE* is NP-complete.
  - Reduction from *UHAMPATH*.

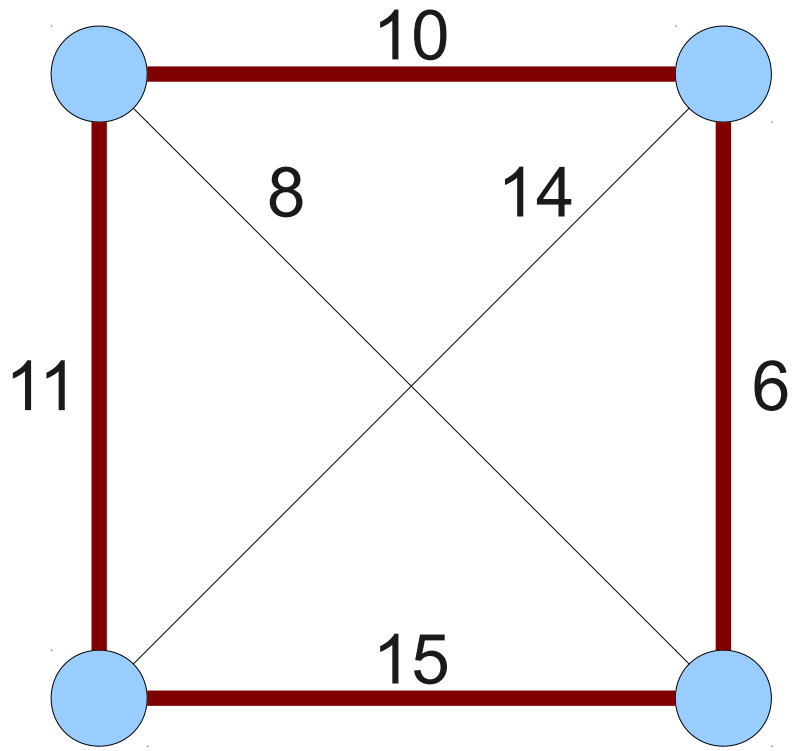
Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.



Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.

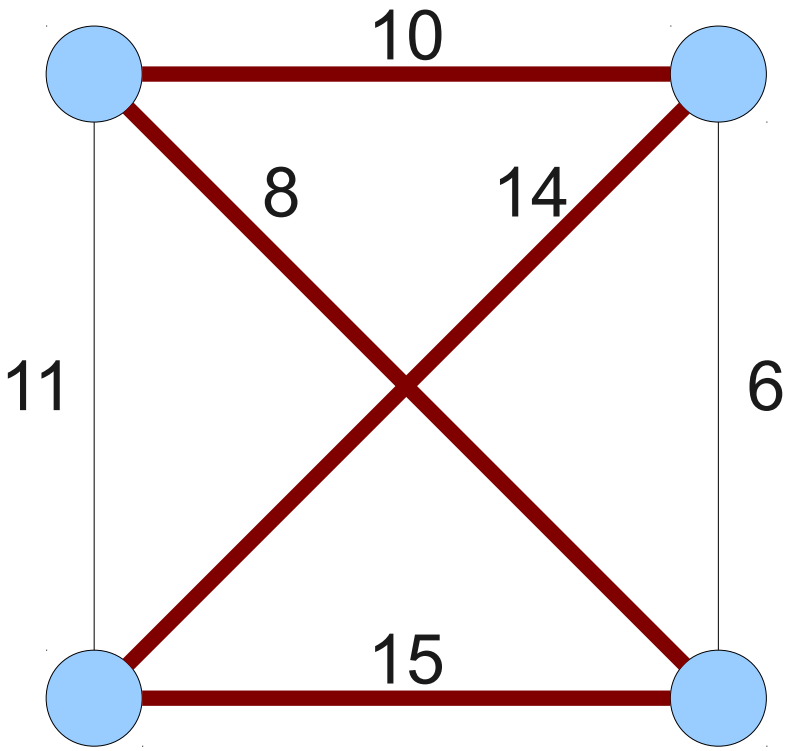


Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.

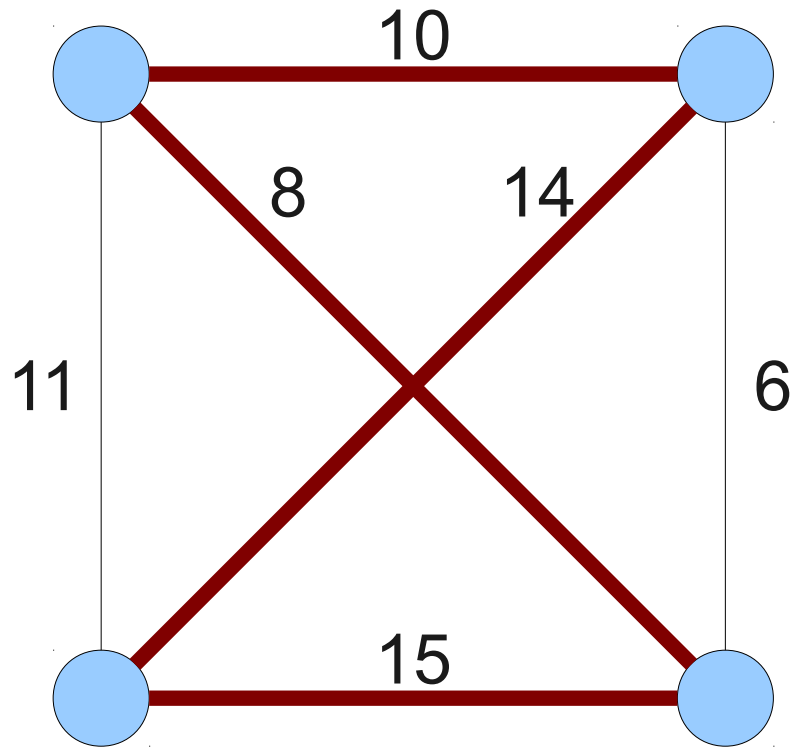


Cost: **42**

Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.

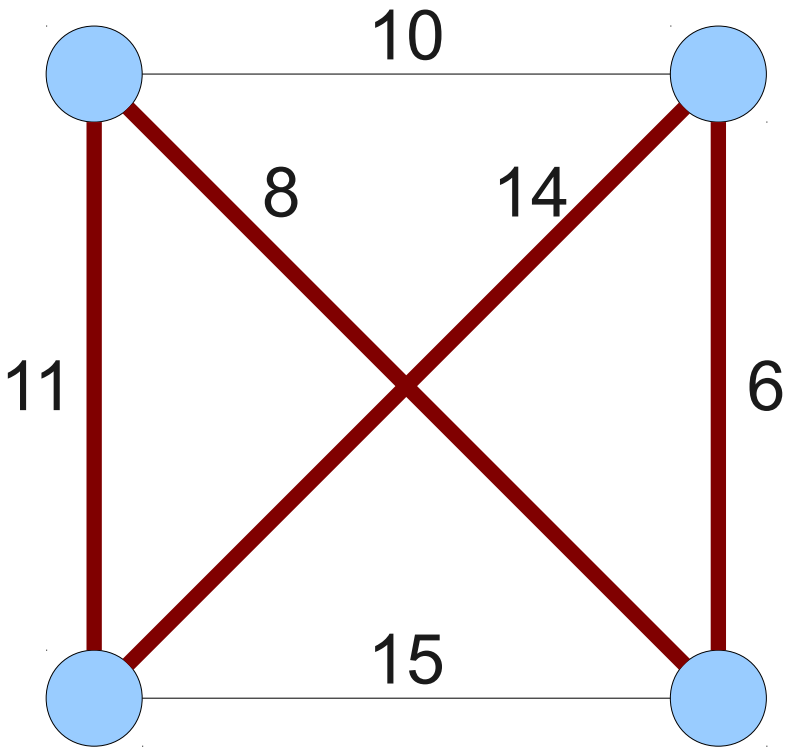


Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.

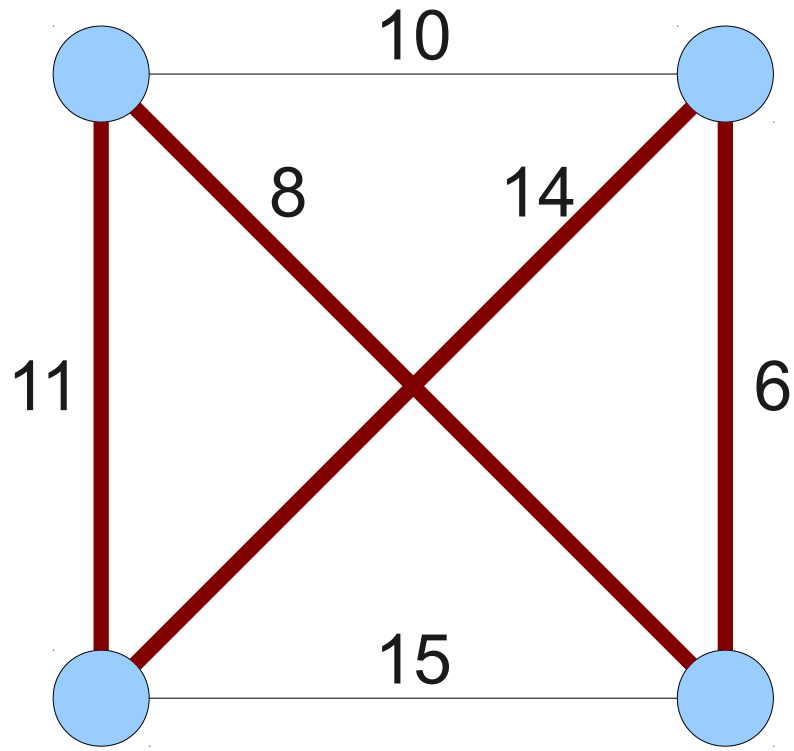


Cost: **47**

Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.

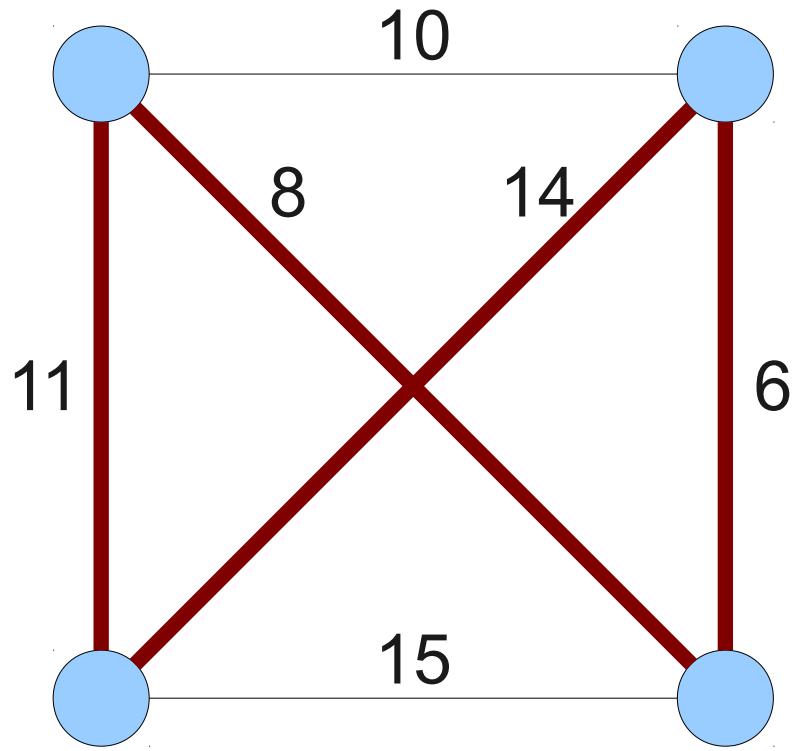


Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.



Cost: **39**

Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.



Cost: **39**

(This is the optimal solution)

Given a complete, undirected, weighted graph G, the **traveling salesman problem** (TSP) is to find a Hamiltonian cycle in G of least total cost.

TSP, Formally

- Given as input
  - A **complete**, **undirected** graph G, and
  - a set of **edge weights**, which are positive integers,

  the **TSP** is to find a Hamiltonian cycle in G with least total weight.
- Note that since G is complete, there has to be at least one Hamiltonian cycle. The challenge is finding the least-cost cycle.

TSP

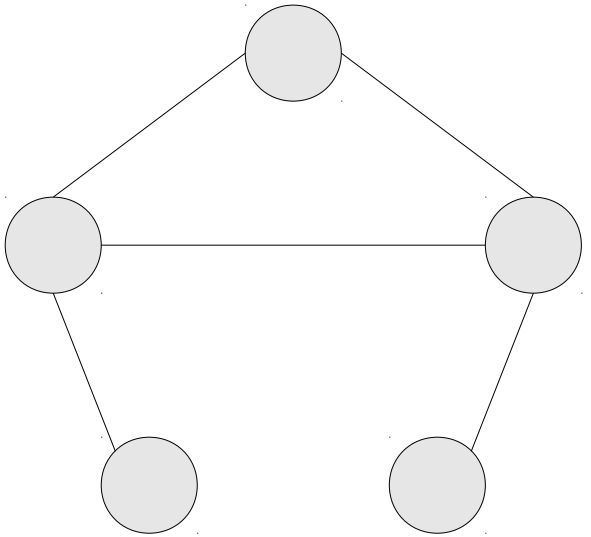
- There are **many** variations on TSP:
  - Directed versus undirected graphs.
  - Complete versus incomplete graphs.
  - Finding a path versus finding a cycle.
  - Nonnegative weights versus arbitrary weights.
- All of these problems are known to be NP-hard.
- The best-known algorithms have horrible runtimes: $O(n^2 2^n)$.

TSP is NP-Hard

- To show that TSP is NP-hard, we reduce the undirected Hamiltonian cycle problem to TSP.

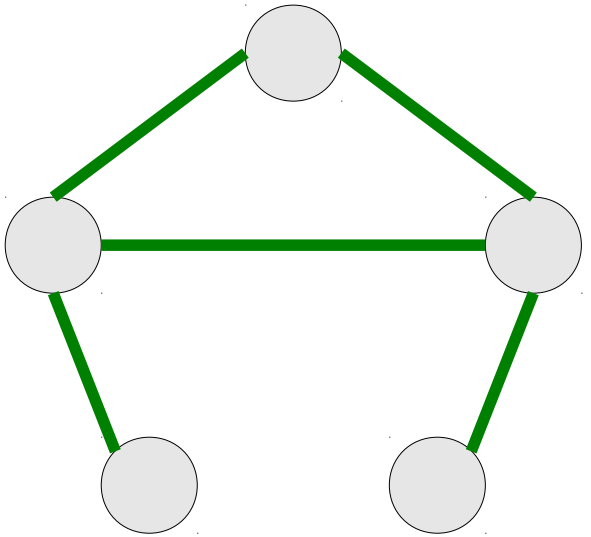
TSP is NP-Hard

- To show that TSP is NP-hard, we reduce the undirected Hamiltonian cycle problem to TSP.



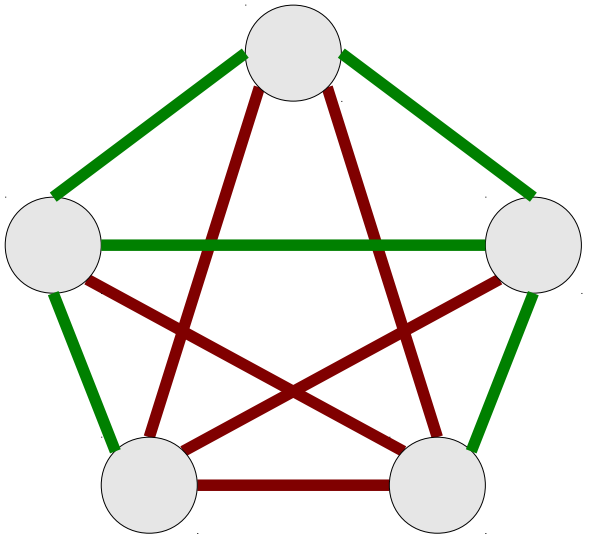
TSP is NP-Hard

- To show that TSP is NP-hard, we reduce the undirected Hamiltonian cycle problem to TSP.



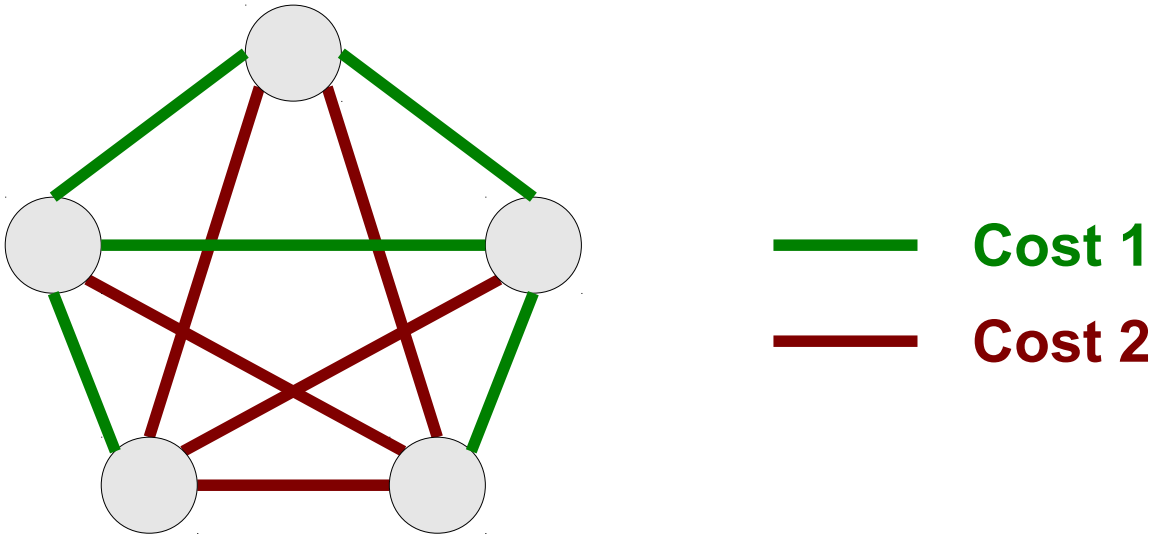
TSP is NP-Hard

- To show that TSP is NP-hard, we reduce the undirected Hamiltonian cycle problem to TSP.



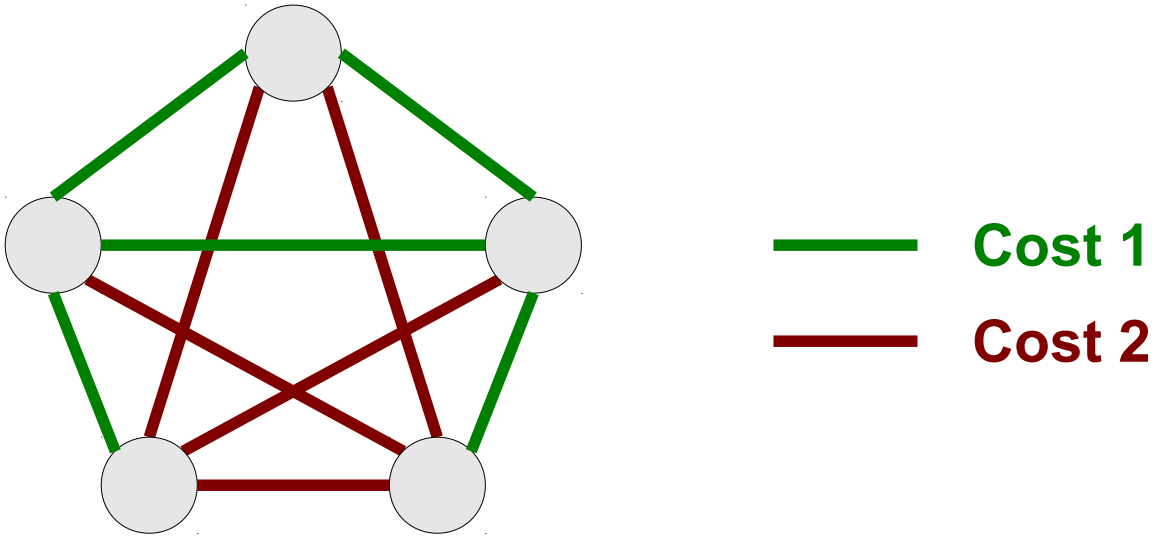
TSP is NP-Hard

- To show that TSP is NP-hard, we reduce the undirected Hamiltonian cycle problem to TSP.



TSP is NP-Hard

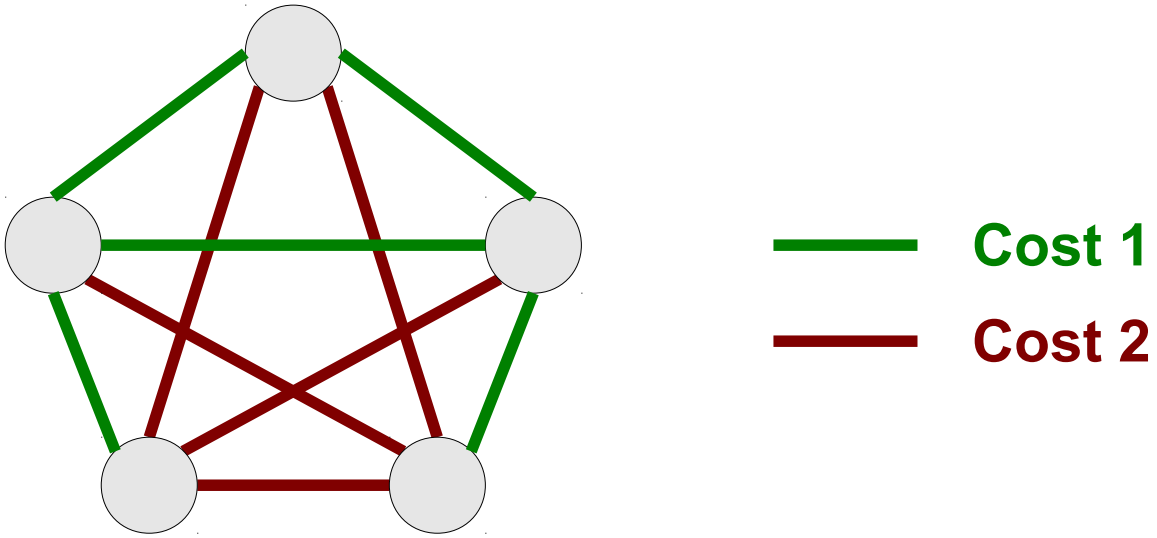
- To show that TSP is NP-hard, we reduce the undirected Hamiltonian cycle problem to TSP.



If there is a Hamiltonian cycle in the original graph, there is a TSP solution of cost n in the new graph (where n is the number of nodes)

TSP is NP-Hard

- To show that TSP is NP-hard, we reduce the undirected Hamiltonian cycle problem to TSP.



If there is a Hamiltonian cycle in the original graph, there is a TSP solution of cost n in the new graph (where n is the number of nodes)

If there is no Hamiltonian cycle, the TSP solution has cost at least n + 1.

Answering TSP

- TSP has great practical importance, and we want to be able to answer it.
  - Determining the shortest subway routes to link stations in a ring.
  - Determining the fastest way to move a mechanical drill so it can drill in the appropriate locations.
- Because TSP is NP-hard, obtaining an exact answer is impractically hard.
- Can we approximate it?

Approximating TSP

- An approximation algorithm to TSP is a k-approximation if it finds a Hamiltonian cycle whose cost is at most k times the optimal solution.
  - If the optimal solution has cost 10, a 2-approximation would return a Hamiltonian cycle of cost between 10 and 20, inclusive.
  - If the optimal solution has cost 10, a 3-approximation would return a Hamiltonian cycle of cost between 10 and 30, inclusive.
- For what values of k is there an efficient k-approximation to TSP?

Theorem: If P ≠ NP, then there is no polynomial-time k-approximation to TSP for **any** natural number k.

Hardness of Approximation

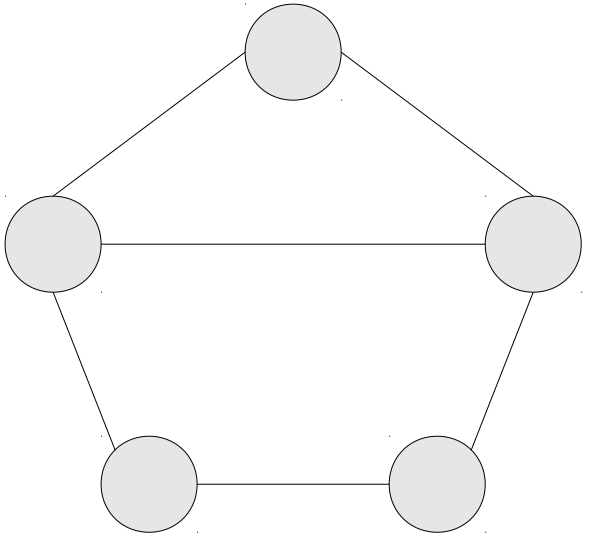
- The proof that TSP is hard to approximate is based on a beautiful construction:

If we could obtain a k-approximation to TSP in polynomial-time, we would have a polynomial-time algorithm for *UHAMCYCLE*.

- Since *UHAMCYCLE* is NP-complete, this is a contradiction if P ≠ NP.

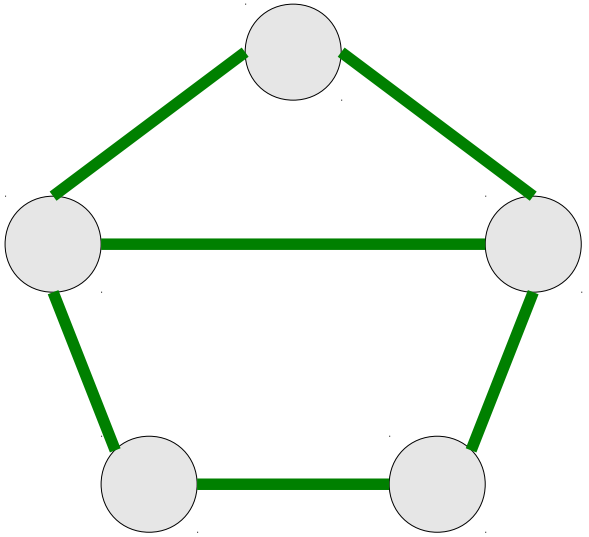
The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



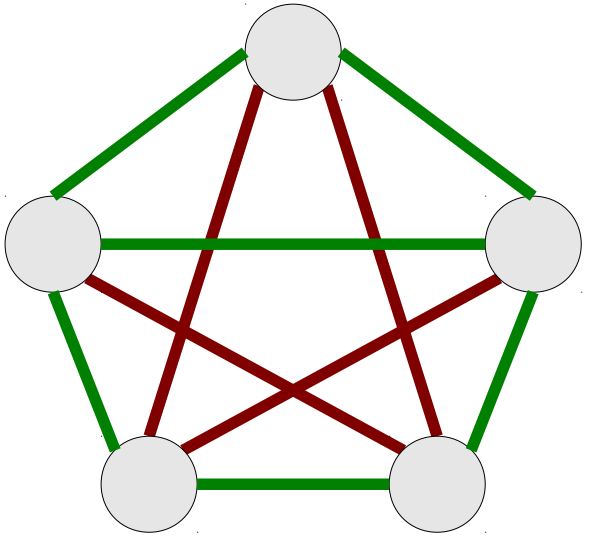
The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



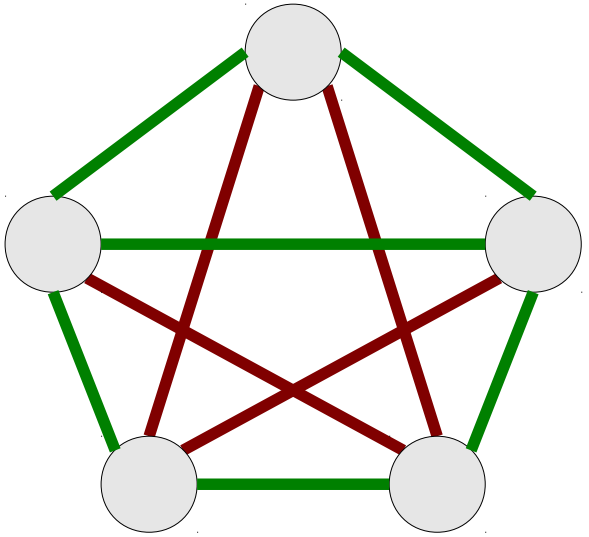
The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.

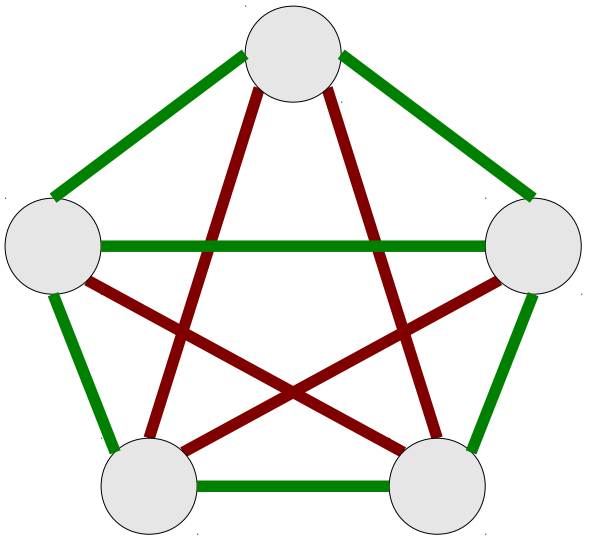


Cost 1

Cost ??

The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



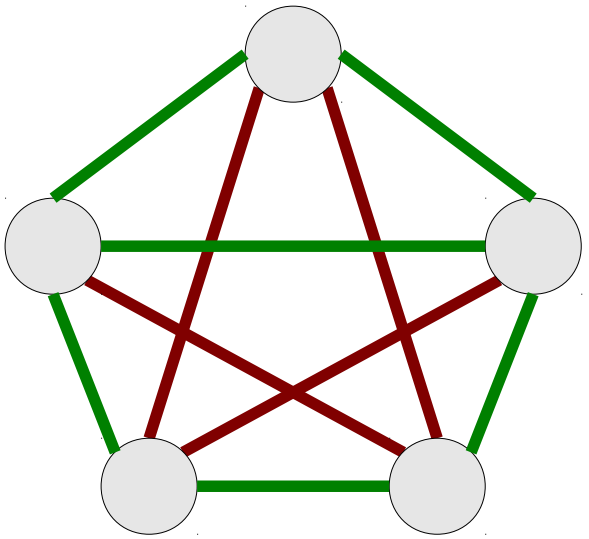
If the original graph of n nodes has a Hamiltonian cycle, the TSP solution has cost n.

Cost 1

Cost ??

The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



Cost 1

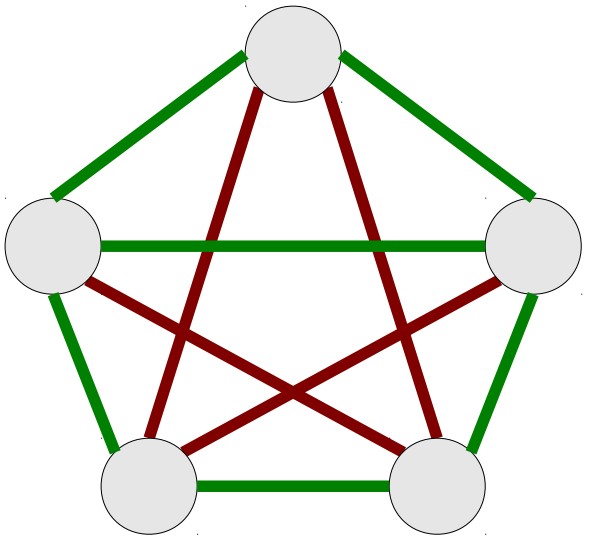
Cost ??

If the original graph of n nodes has a Hamiltonian cycle, the TSP solution has cost n.

The k-approximation algorithm thus must hand back a Hamiltonian cycle of cost at most kn.

The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



Cost 1

Cost ??

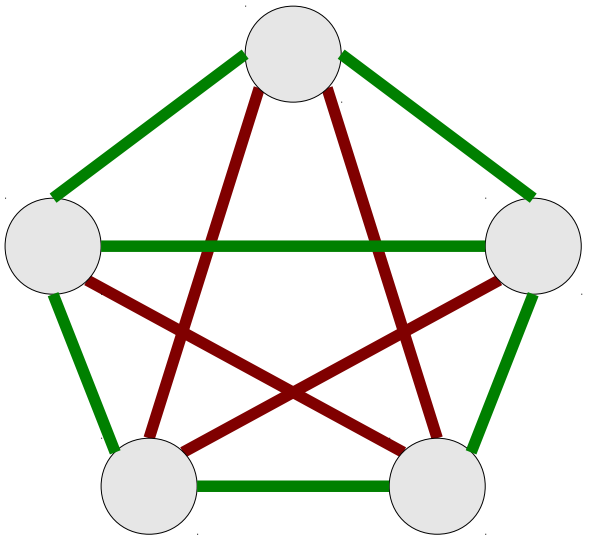
If the original graph of n nodes has a Hamiltonian cycle, the TSP solution has cost n.

The k-approximation algorithm thus must hand back a Hamiltonian cycle of cost at most kn.

What if we made the cost of the red edges so large that any solution using them must cost more than this?

The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



Cost 1

Cost kn + 1

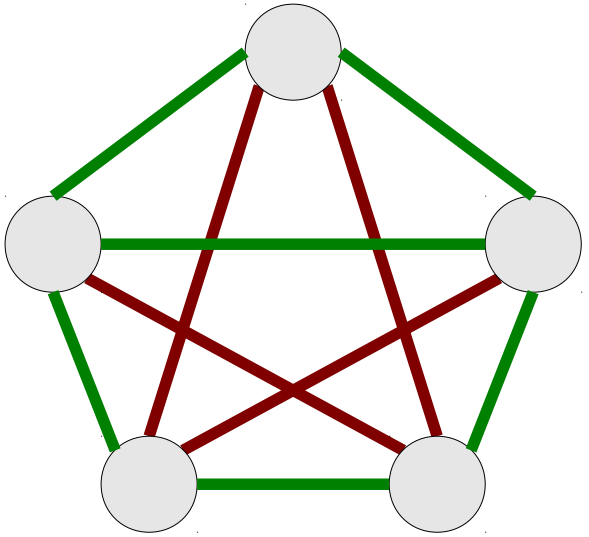
If the original graph of n nodes has a Hamiltonian cycle, the TSP solution has cost n.

The k-approximation algorithm thus must hand back a Hamiltonian cycle of cost at most kn.

What if we made the cost of the red edges so large that any solution using them must cost more than this?

The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.

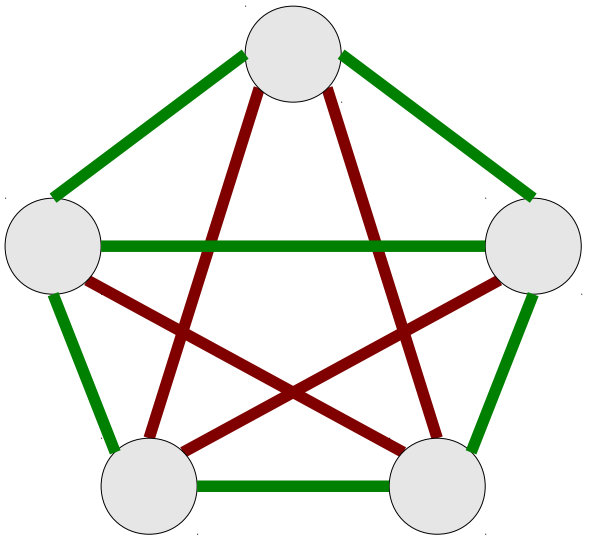


Cost 1

Cost kn + 1

The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



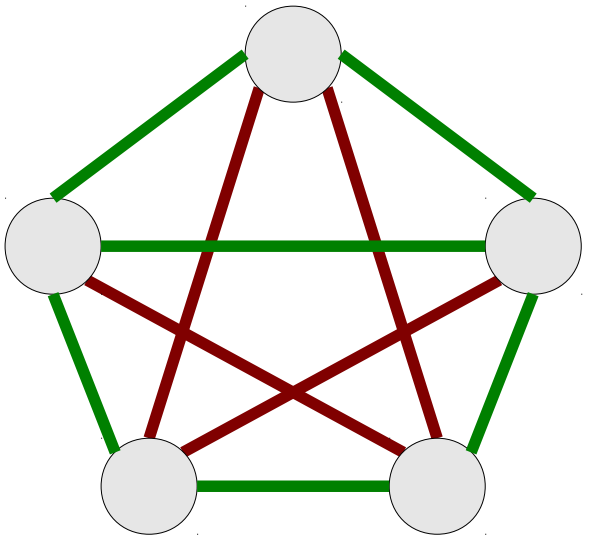
Cost 1

Cost kn + 1

If the k-approximation hands back a solution of cost at most kn, the original graph has a Hamiltonian cycle.

The Construction

- **Proof Sketch**: Suppose, for the sake of contradiction, that there is a k-approximation to TSP for some k.



Cost 1

Cost kn + 1

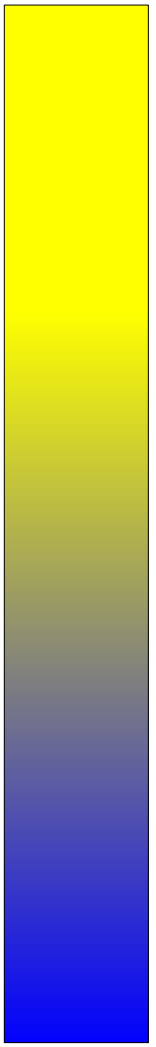
If the k-approximation hands back a solution of cost at most kn, the original graph has a Hamiltonian cycle.

If the k-approximation hands back a solution of cost at least kn + 1, the original graph has no Hamiltonian cycles.

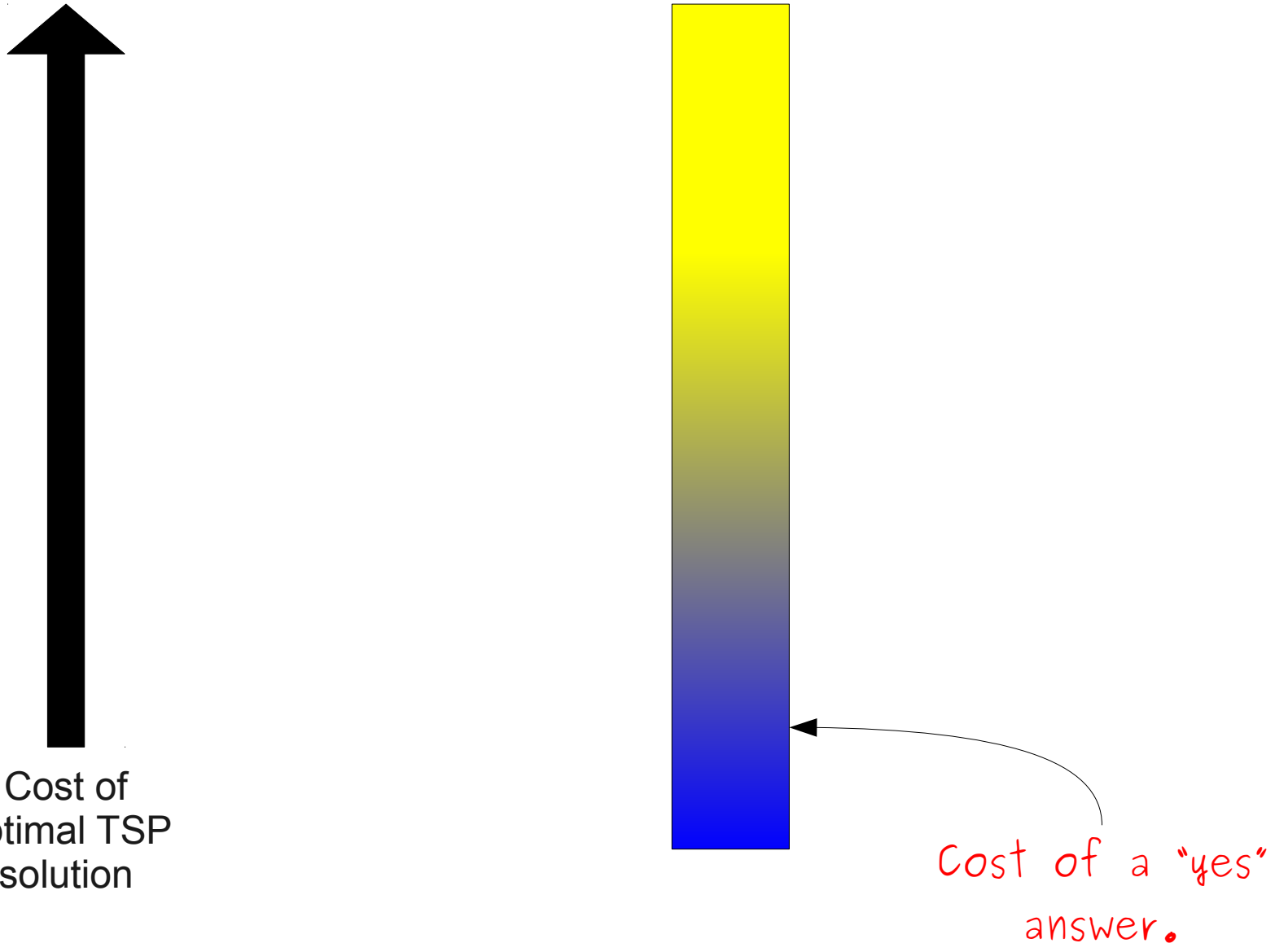
What Just Happened?

What Just Happened?

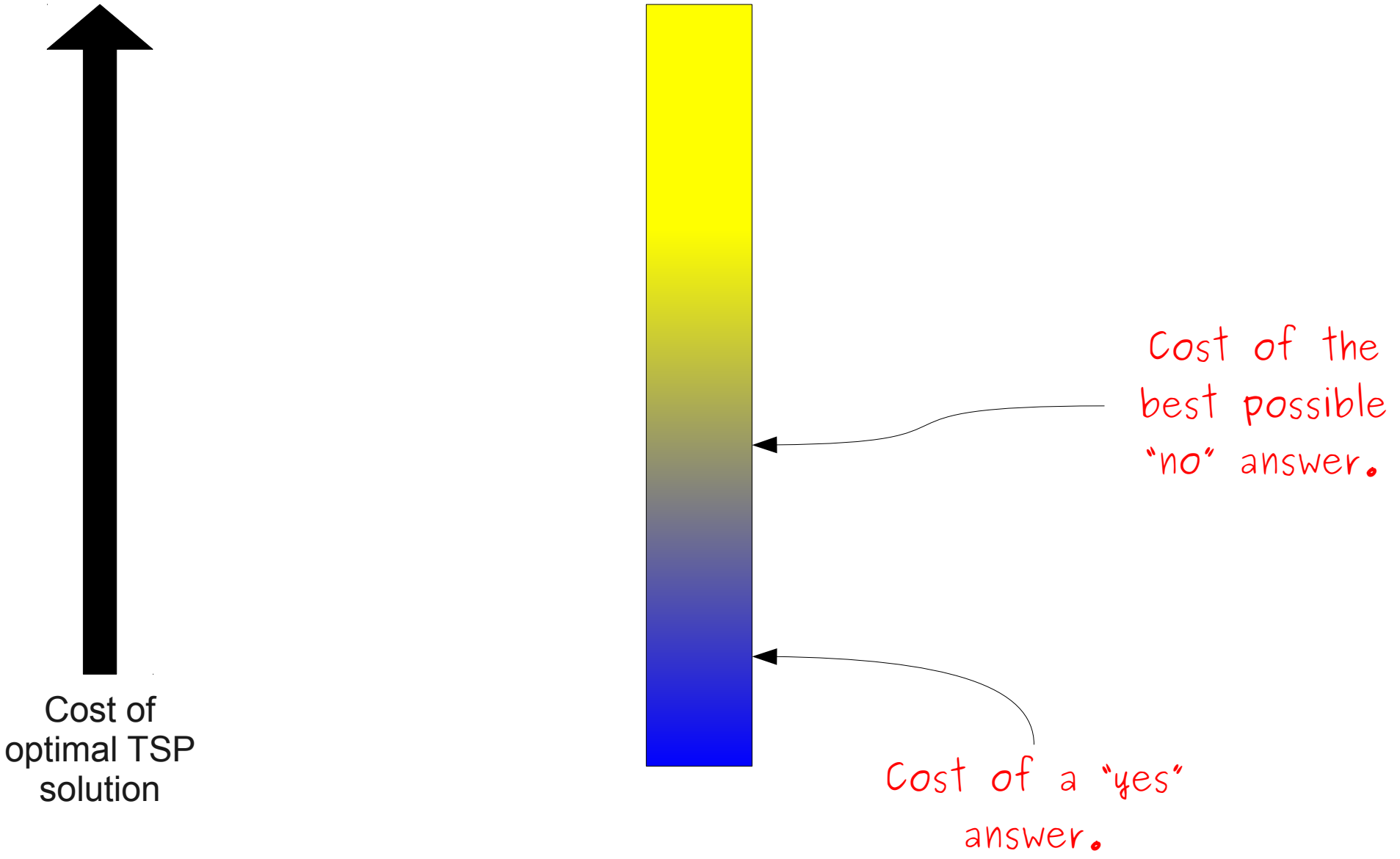
Cost of optimal TSP solution



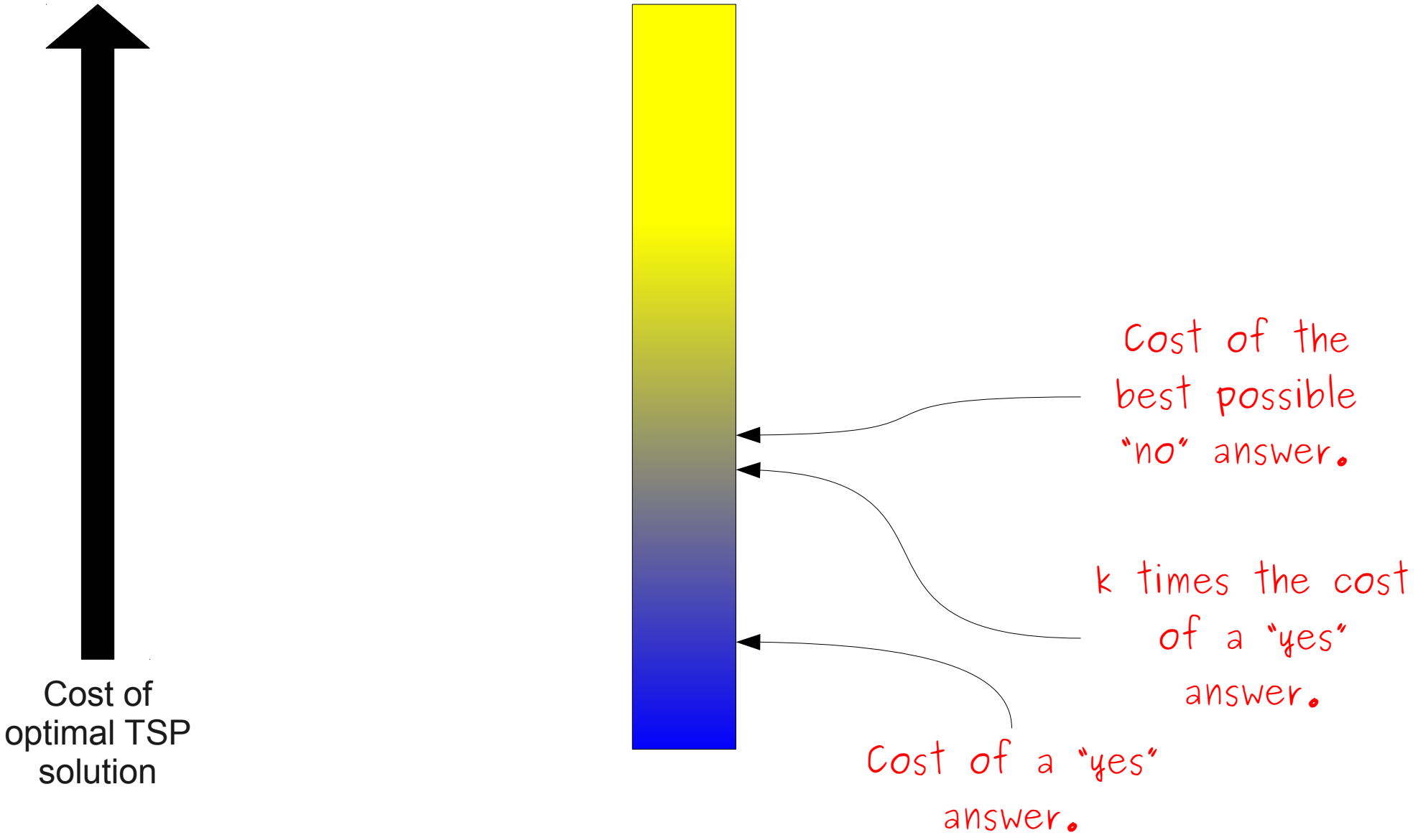
What Just Happened?



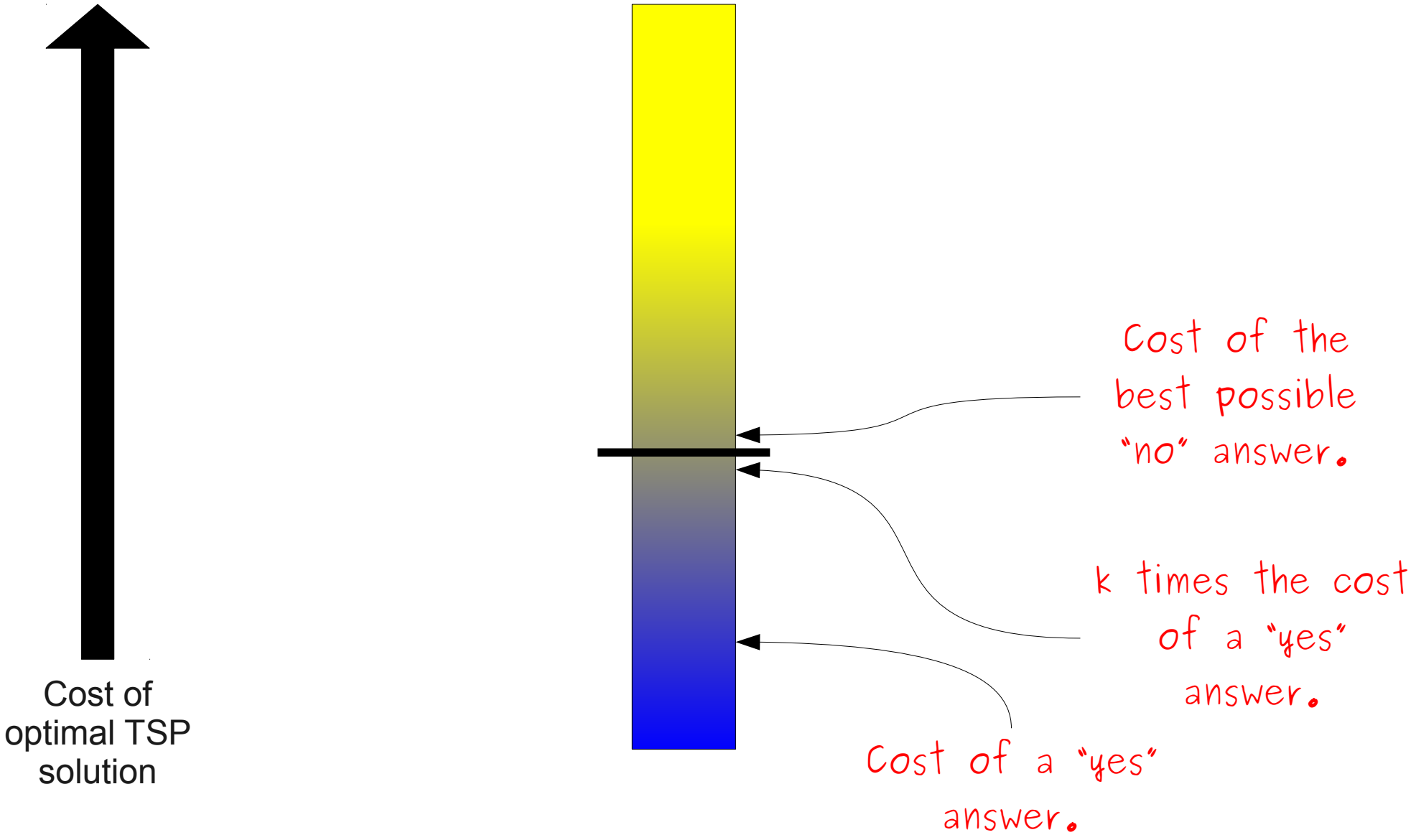
What Just Happened?



What Just Happened?

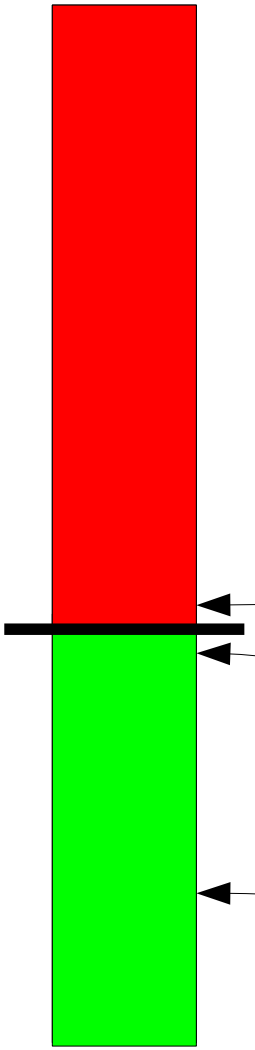


What Just Happened?



What Just Happened?

Cost of optimal TSP solution



Cost of the best possible "no" answer.

k times the cost of a "yes" answer.

Cost of a "yes" answer.

What Just Happened?

- Create an enormous **gap** between the TSP answer for “yes” and “no” instances of the Hamiltonian cycle problem.
- Make the gap so large that the worst possible k-approximate answer to a “yes” instance is distinguishable from the best possible “no” instance.
- Decide whether there is a Hamiltonian cycle by measuring this difference.

The PCP Theorem

Approximating 3SAT

- 3SAT is one of the canonical NP-complete problems.
- What would it mean to approximate a 3SAT answer?

Approximating 3SAT

- The **MAX-3SAT** problem is

  Given a 3CNF formula φ, find a satisfying assignment the maximizes the number of satisfied clauses.

- **Idea**: If we can satisfy the entire formula, then MAX-3SAT finds a satisfying assignment. If not, MAX-3SAT finds the "best" assignment that it can.
- There is a known randomized 7/8-approximation algorithm for MAX-3SAT.
- Is it possible to do better?

The 3-CNF Value

- If φ is a 3-CNF formula, the **value of φ** (denoted **val(φ)**) is the fraction of the clauses of φ that can be simultaneously satisfied by any assignment.
- If φ is satisfiable, val(φ) = 1.
  - All of the clauses are satisfiable.
- If φ is unsatisfiable, val(φ) < 1.
  - Some (but not all) clauses can be satisfied.

The PCP Theorem

For any language $L \in NP$,

There exists a poly-time reduction *f* from L to MAX-3SAT so

For any string *w*:

If $w \in L$, then val(f(*w*)) = 1.

If $w \notin L$, then val(f(*w*)) ≤ 7/8.

The PCP Theorem

For any language L ∈ NP,

There exists a poly-time reduction *f* from L to MAX-3SAT so

For any string *w*:

If $w \in L$, then val(f(w)) = 1.

If the original answer is "yes," the 3-CNF formula is satisfiable.

If $w \notin L$, then val(f(w)) ≤ 7/8.

The PCP Theorem

For any language L ∈ NP,

There exists a poly-time reduction *f* from L to MAX-3SAT so

For any string *w*:

If $w \in L$, then val(f(*w*)) = 1.

If the original answer is "yes," the 3-CNF formula is satisfiable.

If $w \notin L$, then val(f(*w*)) ≤ 7/8.

If the original answer is "no," at most 7/8 of the clauses can be satisfied.

What Does This Mean?

- Our proof that (unless P = NP) TSP cannot be efficiently approximated works by building up a gap between “yes” answers and “no” answers.
- The PCP theorem states that **any problem in NP** can be reduced to MAX-3SAT such that
  - All of the clauses are satisfiable if the original answer is “yes.”
  - At most 7/8 of the clauses are satisfiable if the answer is “no.”
- This gap of 1/8 of the clauses is large enough to preclude approximations beyond the known 7/8-approximation.

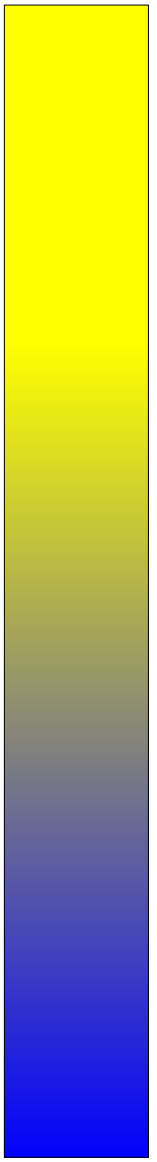
Theorem: If P ≠ NP, then for any r > 7/8, there is no polynomial-time r-approximation to MAX-3SAT.

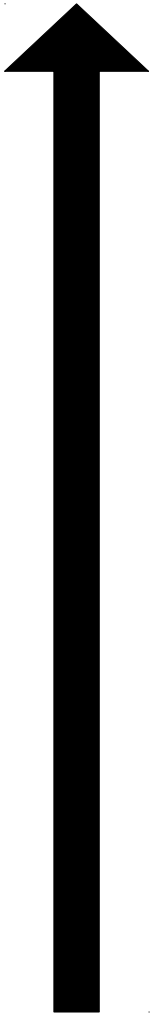
MAX-3SAT is Inapproximable

Theorem: If P ≠ NP, then for any r > 7/8, there is no polynomial-time r-approximation to MAX-3SAT.

MAX-3SAT is Inapproximable

Theorem: If P ≠ NP, then for any r > 7/8, there is no polynomial-time r-approximation to MAX-3SAT.

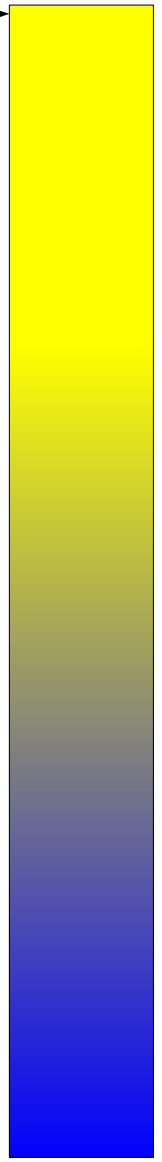


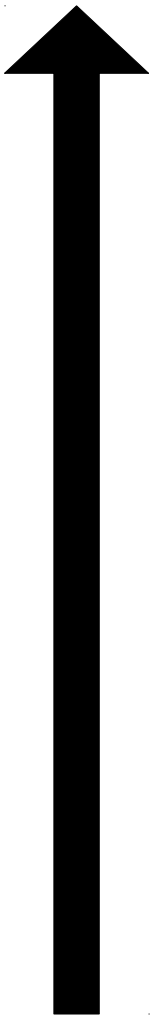


Fraction of Satisfiable Clauses

MAX-3SAT is Inapproximable

Fraction satisfiable in a "yes" answer

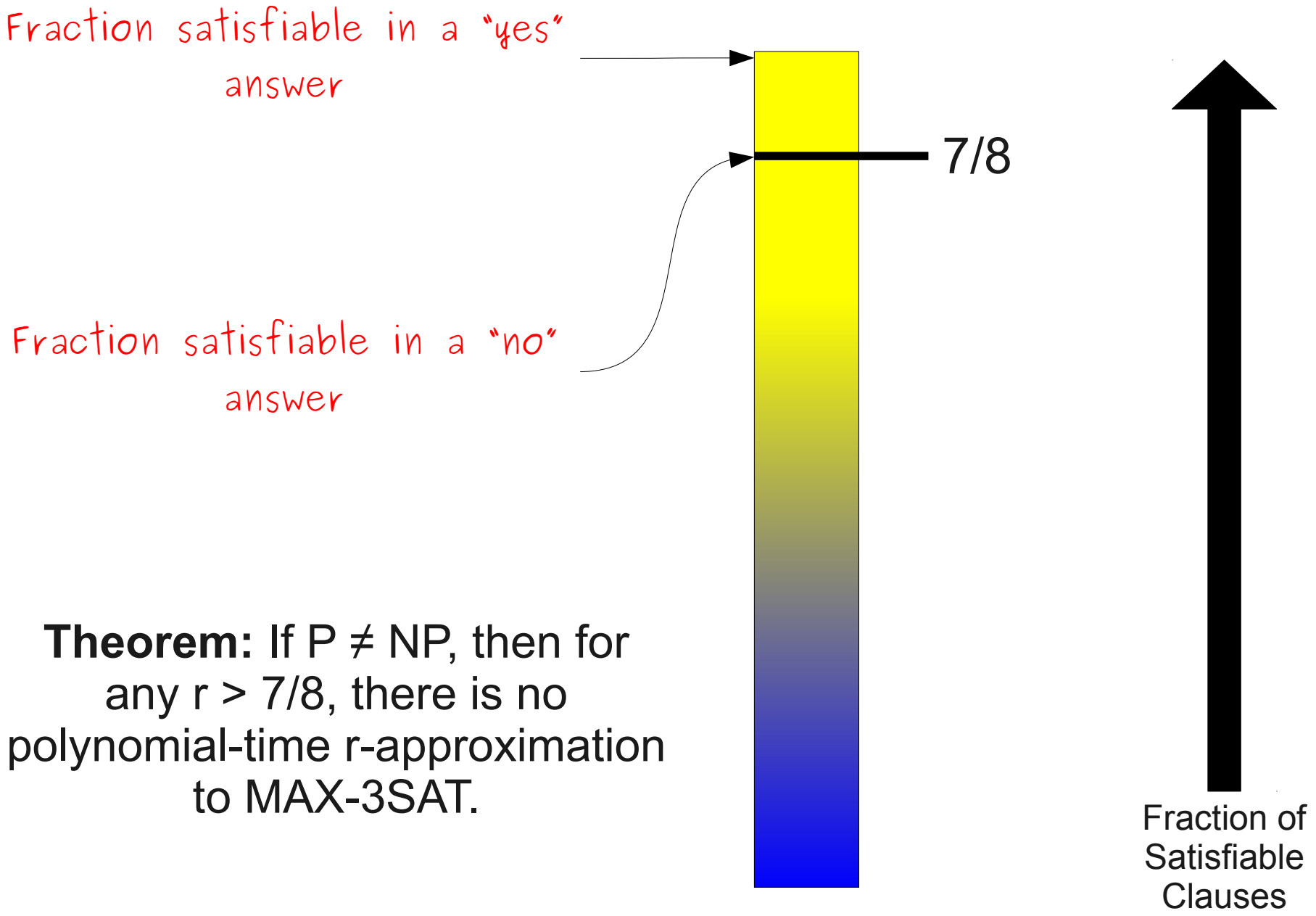




Fraction of Satisfiable Clauses

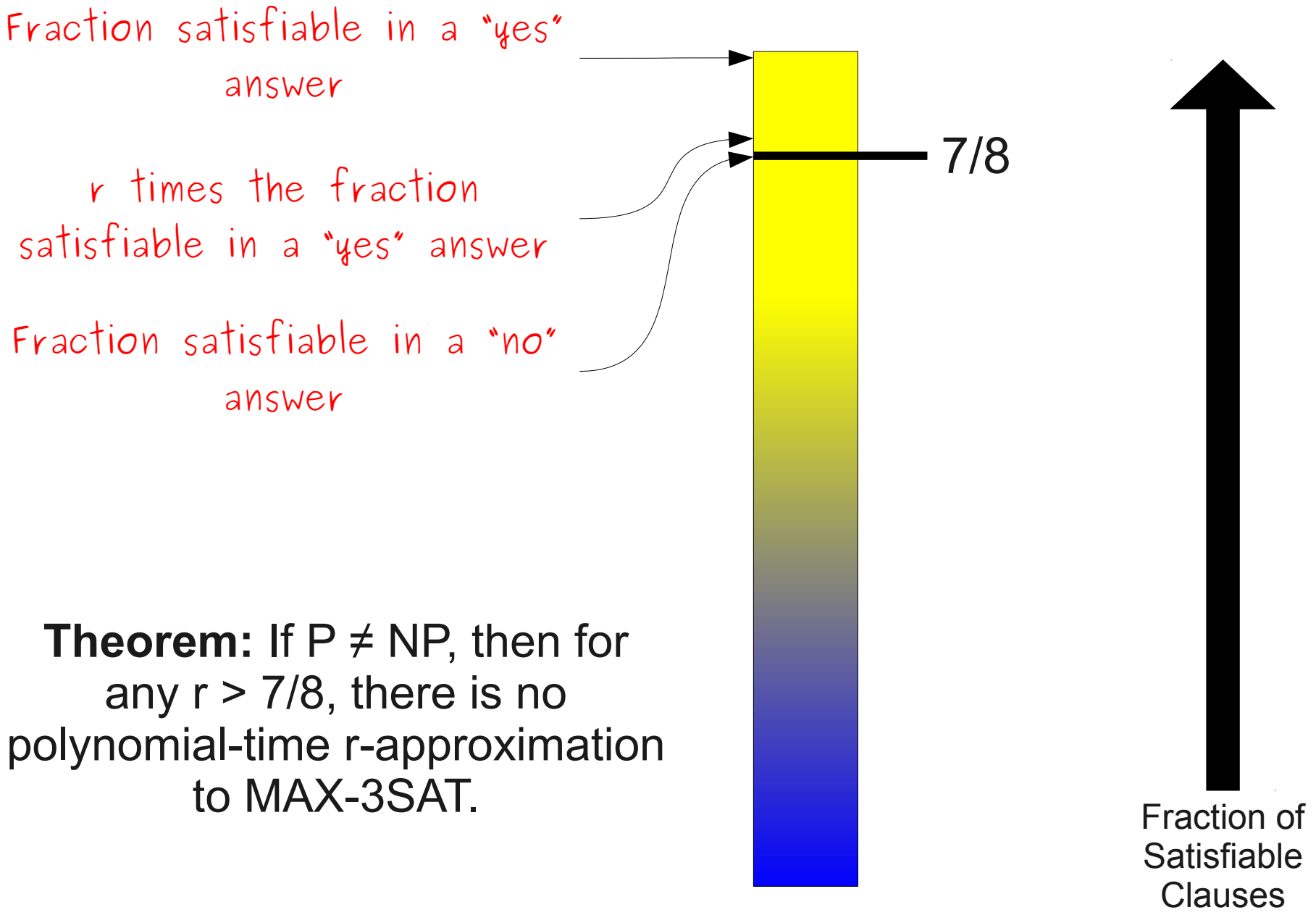
Theorem: If P ≠ NP, then for any r > 7/8, there is no polynomial-time r-approximation to MAX-3SAT.

MAX-3SAT is Inapproximable



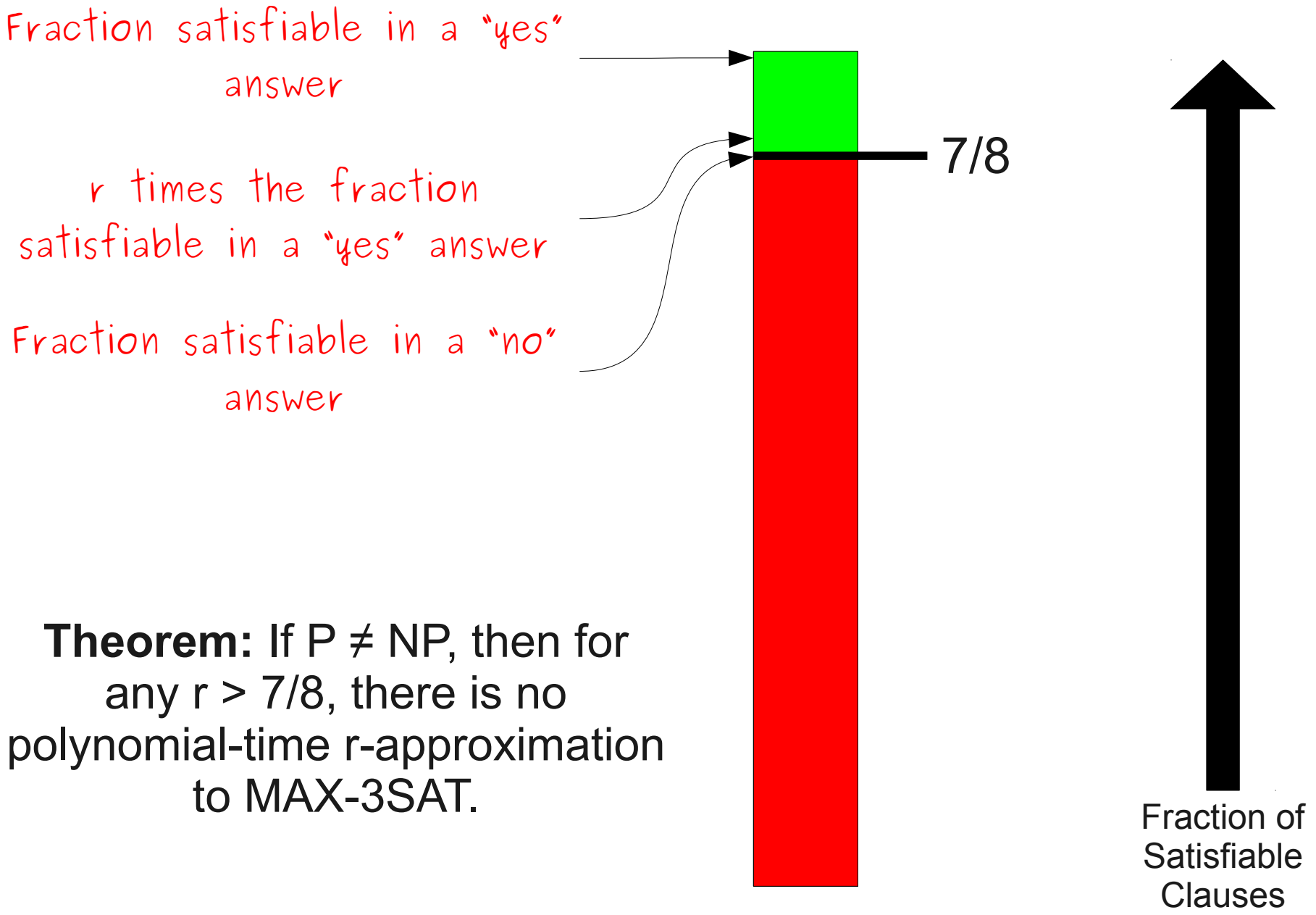
Theorem: If P ≠ NP, then for any r > 7/8, there is no polynomial-time r-approximation to MAX-3SAT.

MAX-3SAT is Inapproximable



Theorem: If P ≠ NP, then for any r > 7/8, there is no polynomial-time r-approximation to MAX-3SAT.

MAX-3SAT is Inapproximable



Theorem: If P ≠ NP, then for any r > 7/8, there is no polynomial-time r-approximation to MAX-3SAT.

Effects on Approximability

- Assuming P ≠ NP, there is a limit to how well we can approximate 3SAT.
- Look at our reduction from 3SAT to INDSET:

(x ∨ y ∨ ¬z) ∧ (¬x ∨ ¬y ∨ z) ∧ (¬x ∨ y ∨ ¬z)

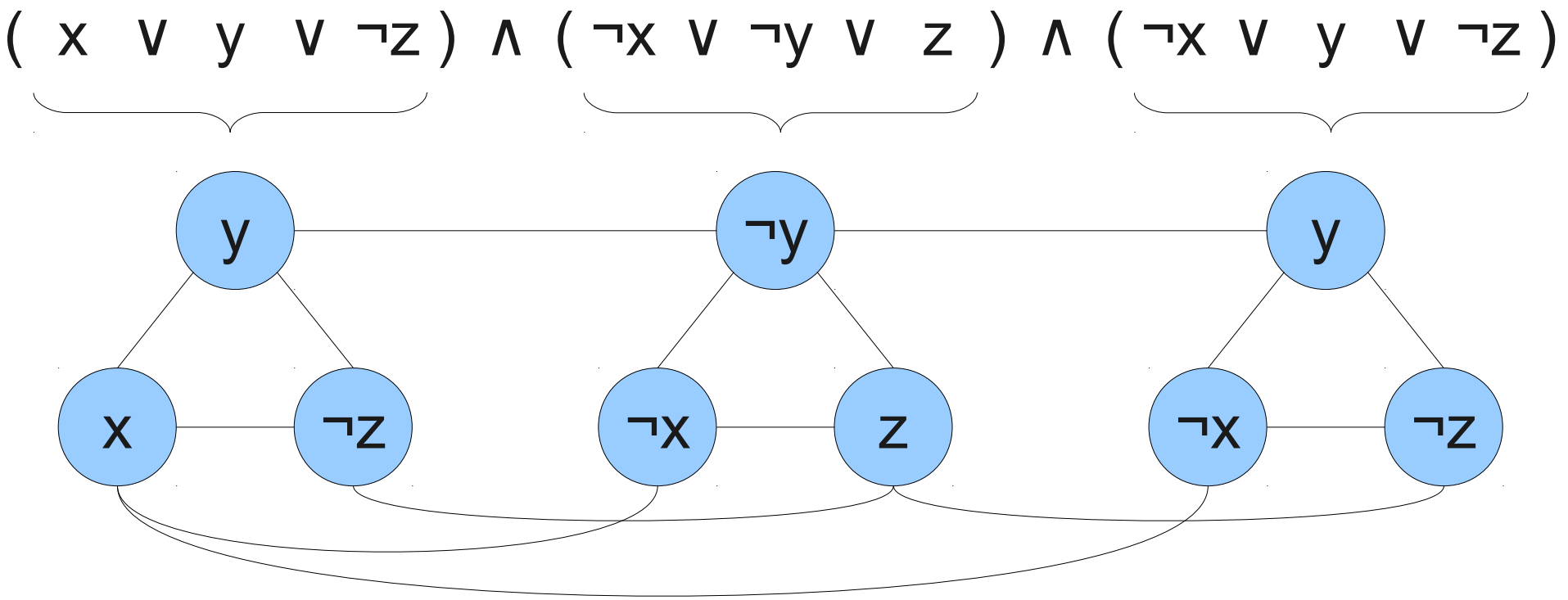
Effects on Approximability

- Assuming P ≠ NP, there is a limit to how well we can approximate 3SAT.
- Look at our reduction from 3SAT to INDSET:

$$(x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

Effects on Approximability

- Assuming P ≠ NP, there is a limit to how well we can approximate 3SAT.
- Look at our reduction from 3SAT to INDSET:

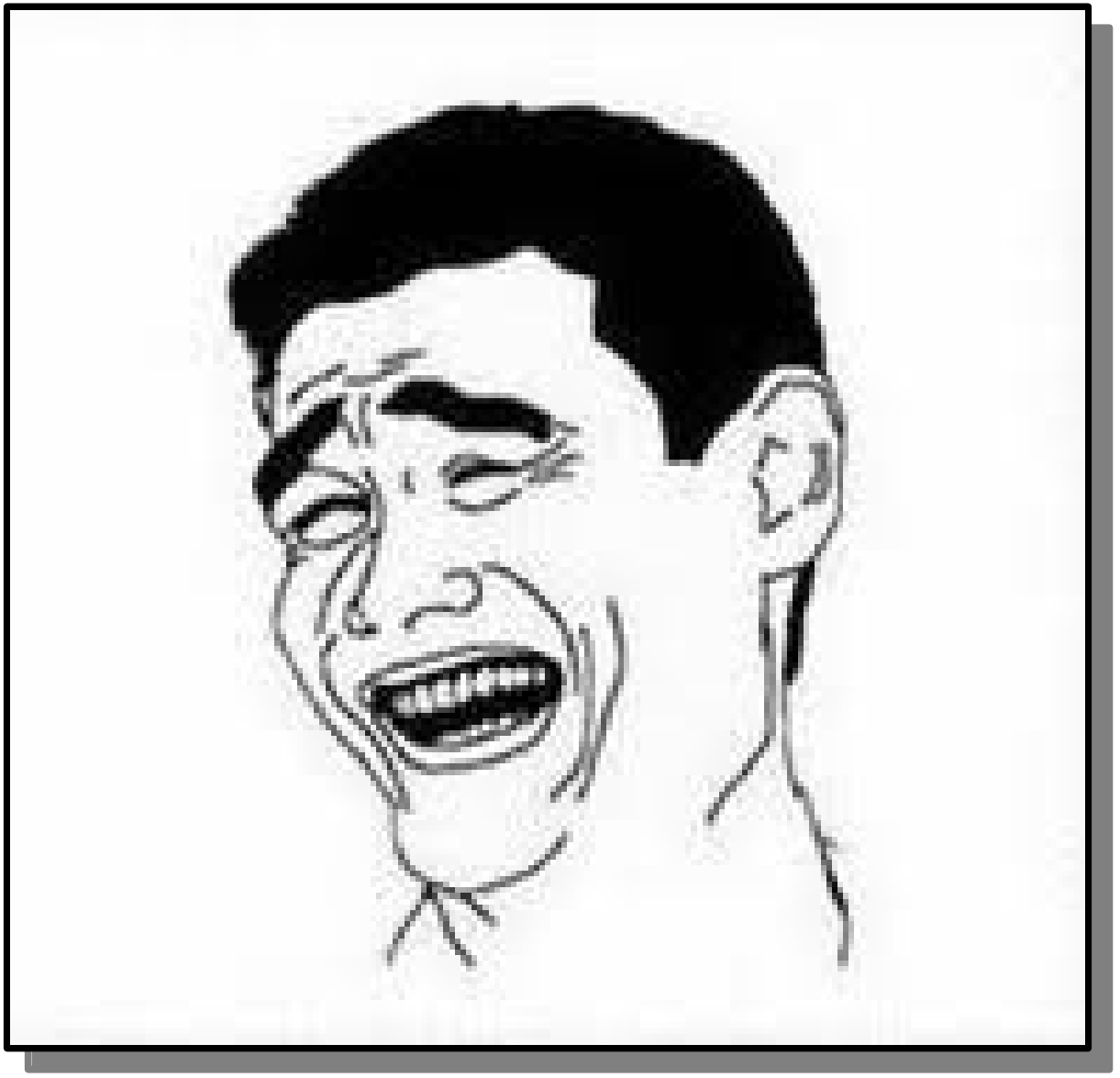


Effects on Approximability

- Many reductions preserve the size of some difficult-to-approximate quantity.
- Assuming if P ≠ NP:
  - Because MAX-3SAT is not efficiently 7/8-approximable, MAX-INDSET is not efficiently 7/8-approximable either.
  - Because MAX-INDSET is not efficiently 7/8-approximable, MAX-CLIQUE is not efficiently 7/8-approximable.
  - Because MAX-INDSET is not efficiently ρ-approximable, MAX-SETPACK is not efficiently 7/8-approximable.
- Not all reductions have this property; some NP-hard problems can be efficiently approximated to very high precision.

Oh, and a proof of the PCP theorem?

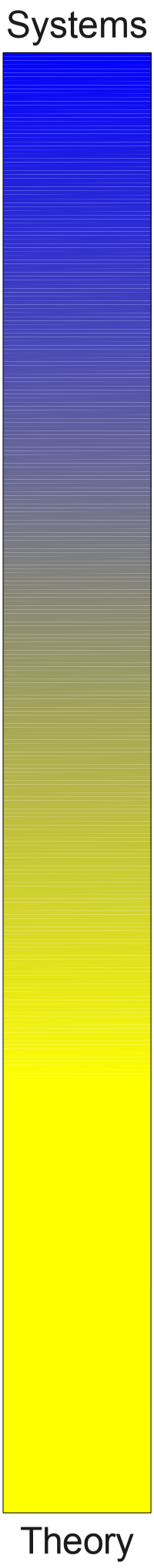
Oh, and a proof of the PCP theorem?

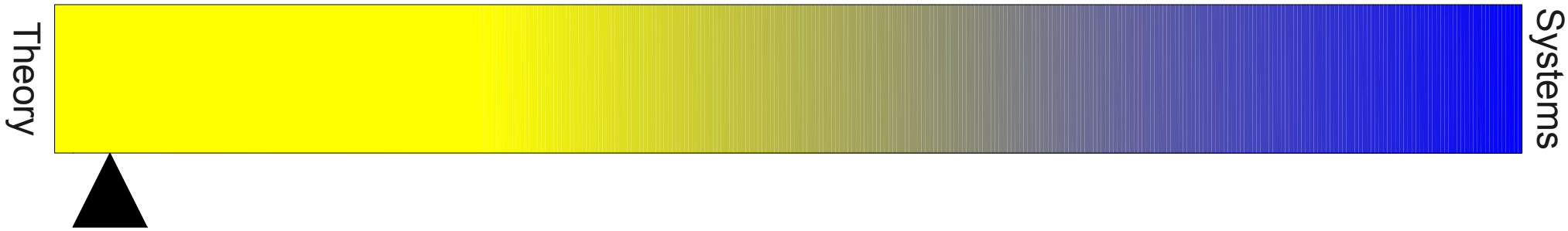


Summary of Approximability

- While many NP-hard problems can be efficiently approximated, certain NP-hard problems cannot unless P = NP.
- Efficiently approximating some NP-hard problems too well would solve NP-complete problems.
- The PCP theorem states that the gap between yes and no answers can be so large that approximating the solution would essentially solve the problem.

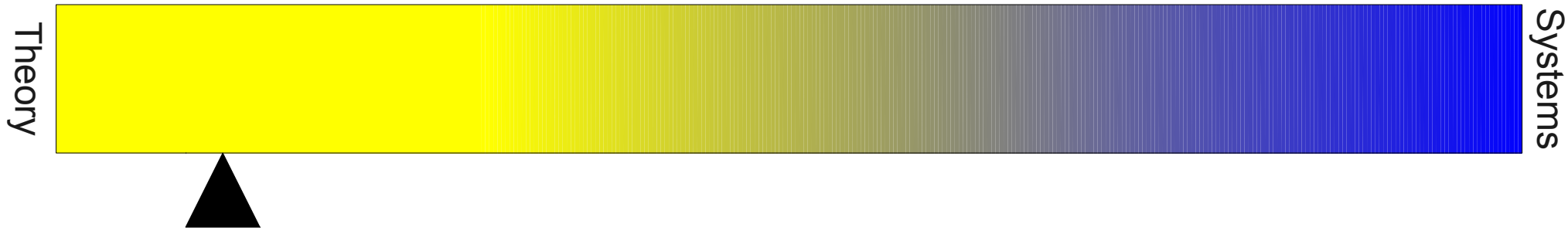
Where to go from here?


Systems
Theory



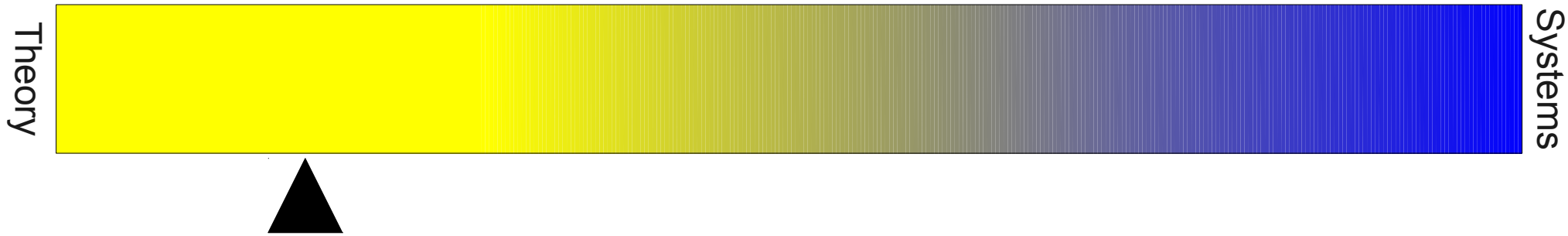
CS154

- **Intro to Automata and Complexity Theory**
- An in-depth treatment of automata, computability, and complexity.
- Emphasis on theoretical results in automata theory and complexity.
- Launching point for more advanced courses (CS254, CS354)



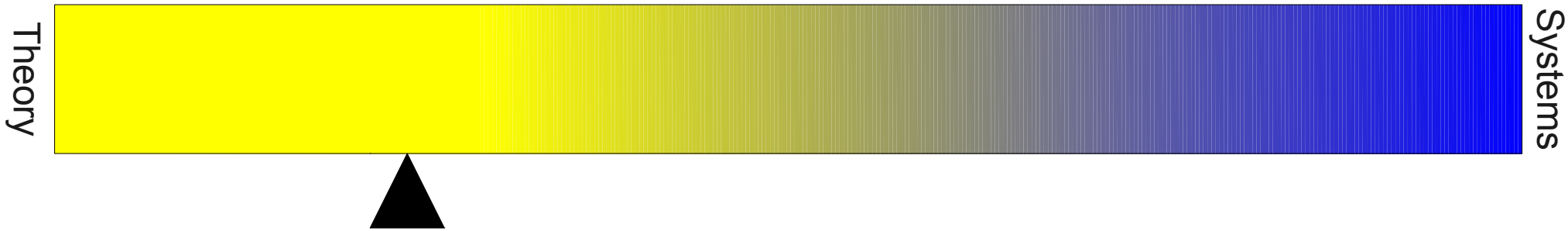
CS258

- **Intro to Programming Language Theory**
- Explore questions of computability in terms of recursion and recursive functions.
- Excellent complement to the material in this course; highly recommended.
- Offered every other year but available next quarter; consider checking it out!



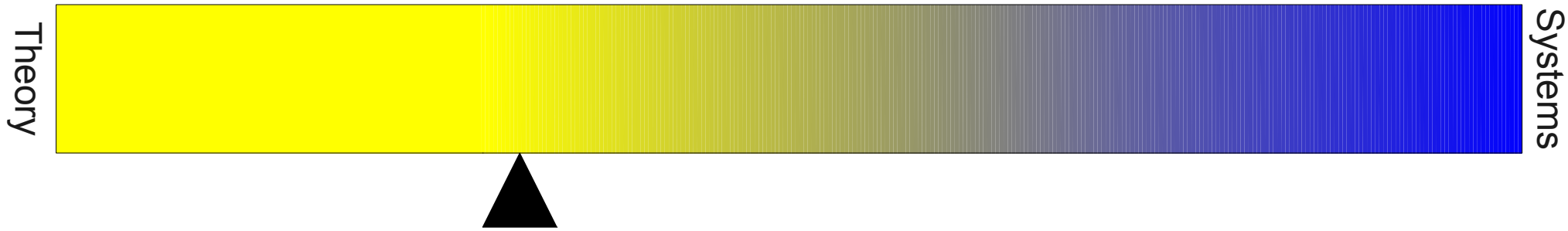
CS109

- **Intro to Probability for Computer Scientists**
- Learn to embrace randomness.
- Use your newly acquired proof skills in an entirely different domain.
- Learn why Mehran Sahami is so awesome.



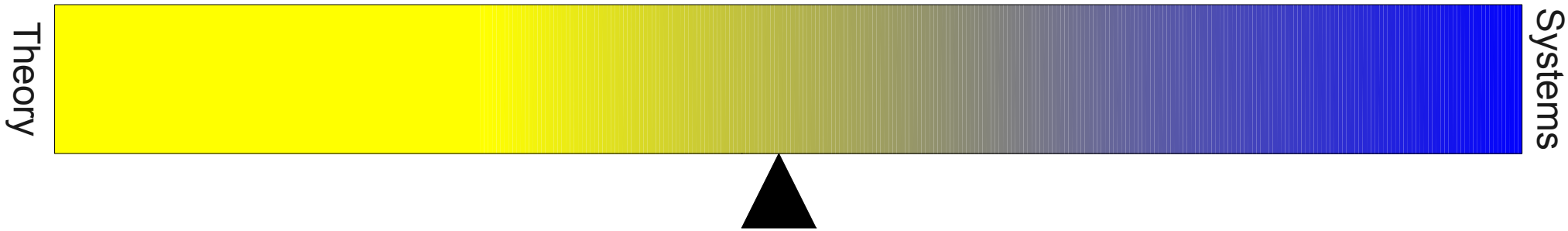
CS255

- **Intro to Cryptography**
- How can you use hard problems to your advantage?
- How can NP-completeness help you keep secrets?



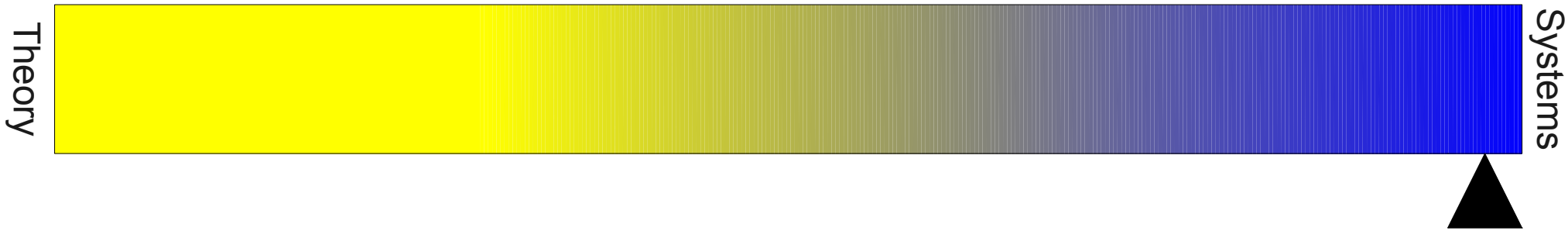
CS161

- **Design and Analysis of Algorithms**
- Learn how to approach new problems and solve them efficiently.
- Learn how to deal with NP-completeness.
- Learn how to ace job interviews



CS143

- **Compilers**
- Watch automata, grammars, undecidability, and NP-completeness come to life by building a complete working compiler from scratch.
- See just how much firepower you can get from all this material.



CS107

- **Computer Organization and Systems**
- You don't need to be a theoretician to love computer science!
- If you want to learn how the machine works under the hood, look no further.

There are more problems to solve than there are programs to solve them.

Where We've Been

- Given this hard theoretical limit, what **can** we compute?
- How powerful of a computer do we need to reach this limit?
- Of what we can compute, what can we compute *efficiently*?
- What techniques from mathematics can we use to reason about this?

What We've Covered

- Sets
- Graphs
- Proof Techniques
- Relations
- Induction
- Logic
- Pigeonhole Principle
- Trees
- DFAs
- NFAs
- Regular Expressions
- CFGs
- PDAs
- Pumping Lemmas
- Turing Machines
- Undecidability
- Unrecognizability
- Reductions
- Rice's Theorem
- Time Complexity
- P
- NP

Final Thoughts