Analysis of Algorithms

V. Adamchik CSCI 570 Fall 2016

Lecture 13 University of Southern California

NP Hardness

Based on Chapter 8

Algorithm Design by Kleinberg & Tardos

Outline

Intro to Turing Machines
Halting Problem
Graph Coloring
Hamiltonian Cycle
Traveling Salesman Problem

23 Problems of Hilbert

In 1900 Hilbert presented a list of 23 challenging (unsolved) problems in math

#1 The Continuum Hypothesis	impossible, 1963
#8 The Riemann Hypothesis	unproved yet
#10 On solving a Diophantine equations	impossible, 1970
#18 The Kepler Conjecture	proved,1998

Hilbert's 10th problem

Given a multivariate polynomial with integer coeffs, e.g. $4x^2y^3 - 2x^4z^5 + x^8$, "*devise a process according to which it can be determined in a finite number of operations*" whether it has an integer root.

Mathematicians: "we should try to formalize what counts as a 'process' ".

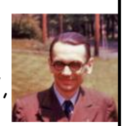
Hilbert's 10th problem

In 1928 Hilbert rephrased it as follows:

Given a statement in first-order logic, give an "*effectively calculable procedure*" for determining if it's provable.

Mathematicians: "we should try to formalize what counts as an 'algorithm' and an 'efficient algorithm'".

Gödel (1934):
Discusses some ideas for definitions of what functions/languages are "computable", but isn't confident what's a good definition.



Church (1936):
Invents lambda calculus, claims it should be the definition of "computable".

Gödel, Post (1936):
Argues that Church's definition isn't justified.

Meanwhile… a certain British grad. student in Princeton, unaware of all these debates…

Described a new model of computation, now known as the Turing Machine.

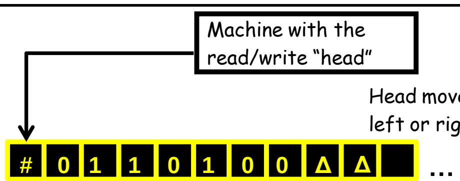
PH.D. student of A. Church

Alan Turing (1936, age 22)

Gödel, Kleene, and Church:
"Um, he nailed it. Game over, computation defined."

Turing's Inspiration

Human writes symbols on paper
WLOG, the paper is a sequence of squares
No upper bound on the number of squares
At most finitely many kinds of symbols
Human observes one square at a time
Human has only finitely many mental states
Human can change symbols and change focus to a neighboring square, but only based on its state and the symbol it observes
Human acts deterministically



Head moves to left or right

input (the "tape"), indefinitely extensible to the right.

input consists of symbols from an alphabet Σ

and Δ represent the start and end of the input

The machine never overwrites the leftmost symbol, but can overwrite the rightmost symbol.

When halt(accept/reject) state is reached, the machine halts. It might also never halt, in which case we say it loops.

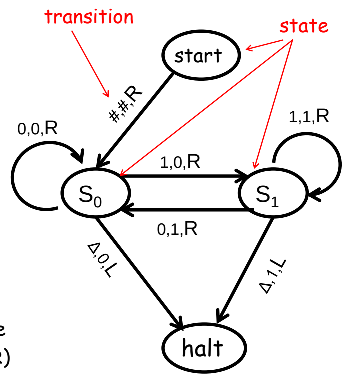
Example of a Turing machine

The machine that takes a binary string and appends 0 to the left side of the string.

Input: #10010Δ
Output: #010010Δ

- leftmost char
Δ – rightmost char

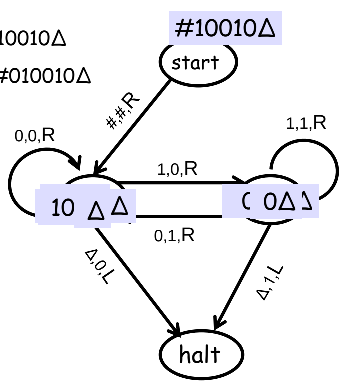
Transition on each edge
read,write,move (L or R)



Deterministic Turing Machine

Input: #10010Δ

Output: #010010Δ



Runtime Complexity

Let M be a Turing machine that halts on all inputs.

Assume we compute the running time purely as a function of the length of the input string.

Definition: The running complexity is the function $f: N \rightarrow N$ such that f(n) is the maximum number of steps that M uses on any input of length n.

Decidable Languages

The set Σ^* is the set of all finite sequences of elements of Σ.

A language $L \subseteq \Sigma^*$ is decidable if there is a Turing Machine M which halts on every input $x \in L$.

A problem P is *decidable* if it can be solved by a Turing machine T that always halt.

We say that P has an algorithm.

Church-Turing Thesis:

"Any natural / reasonable notion of computation can be simulated by a TM."

This is not a theorem.

Is it… *…an observation?*

…a definition?

…a hypothesis?

…a law of nature?

…a philosophical statement?

Well, whatever. Everyone believes it.

Complexity Classes

A fundamental complexity class P (or PTIME) is a class of decision problems that can be solved by a deterministic Turing machine in polynomial time.

A fundamental complexity class EXPTIME is a class of decision problems that can be solved by a deterministic Turing machine in $O(2^{p(n)})$ time, where p(n) is a polynomial.

Nondeterministic Turing Machine

The deterministic Turing machine means that there is only one valid computation starting from any given input. A computation path is like a linked list.

Nondeterministic TM defined in the same way as deterministic, except that a computation is like a tree, where at any state, it's allowed to have a number of choices.

The big advantage: it is able to try out many possible computations in parallel and to accept its input if any one of these computations accepts it.

Complexity Class: NP

A fundamental complexity class NP is a class of decision problems that can be solved by a nondeterministic Turing machine in polynomial time.

This is the original NP definition, formulated by Karp in 1972.

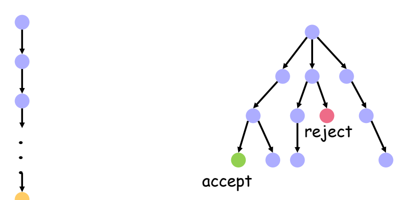
Equivalently, the NP decision problem has a certificate that can be checked by a polynomial time deterministic Turing machine.

These two definitions of NP is commonly accepted.

Computations

deterministic computation

nondeterministic computation



accept or reject

accepts if some branch reaches an accepting configuration

P =?= NP

It has been proven that Nondeterministic TM can be simulated by Deterministic TM.

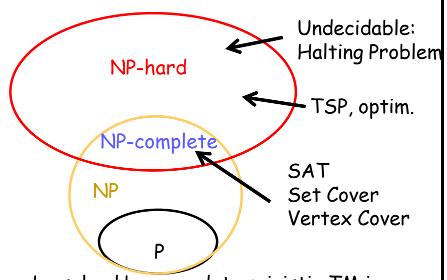
But how <u>fast</u> we can do that?

The famous P ≠ NP conjecture, would answer that we cannot hope to simulate nondeterministic Turing machines very fast (in polynomial time).

Venn Diagram (P ≠ NP)

NPC problems are the most difficult NP problems.

NPH problems do not have to be in NP.



NPC problems can be solved by a nondeterministic TM in polynomial time.

It's not known if NPC problems can be solved by a deterministic TM in polynomial time.

Is every language in decidable?
Is every function computable?

Answer: No

Write a program to output "HELLO WORLD" on the screen and then terminate (halt).

The TA grading script G must be able to take any program P and grade it.

What kind of program could a student hand in?

```
while (P == NP)
   print "HELLO WORLD";
```

Despite the simplicity of the HELLO assignment, there is no program to correctly grade it!

And we will <u>prove</u> this.

Undecidable Problems

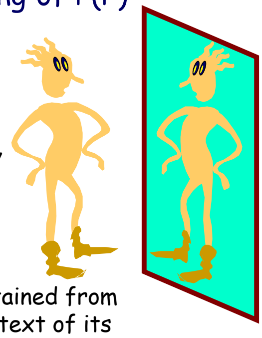
Undecidable means that there is no computer program that always gives the correct answer: it may give the wrong answer or run forever without giving any answer.

<u>The halting problem</u> is the problem of deciding whether a given Turing machine halts when presented with a given input.

Turing's Theorem:
The Halting Problem is not decidable.

The meaning of P(P)

P(x) means the output that arises from running program P on input x, assuming that P eventually halts.



P(P) means the output obtained from running program P on the text of its own source code.

The Halting Set K

Definition:

K is the set of all programs P such that P(P) halts.

K = { program P | P(P) halts }

Is there a program HALT such that:

HALT(P) = yes, if $P \in K$, so P(P) halts.
HALT(P) = no, if $P \notin K$, so P(P) doesn't halt.

The Halting Problem

Suppose a program HALT that solves the halting problem is indeed exist.

We will call HALT as a subroutine in a new program called CONFUSE.

```
bool CONFUSE(P) {
  if (HALT(P) == True)
        then loop forever;
  else return True;
}
```

Does CONFUSE(CONFUSE) halt?

Does CONFUSE(CONFUSE) halt?

```
bool CONFUSE(P) {
  if (HALT(P) == True) then loop forever;
  else return True;
}
```

Consider two cases:
1. assume CONFUSE(CONFUSE) does halt.

   by definition of HALT, we have that HALT(CONFUSE) is True.

   then by definition of CONFUSE, we have that CONFUSE(CONFUSE) loops forever.

Does CONFUSE(CONFUSE) halt?

```
bool CONFUSE(P) {
  if (HALT(P) == True) then loop forever;
  else return True;
}
```

Second case:
2. CONFUSE(CONFUSE) does not halt.

   by definition of HALT, we have that HALT(CONFUSE) is False.

   Then by definition of CONFUSE, we have that CONFUSE (CONFUSE) returns True.

Thanksgiving Week

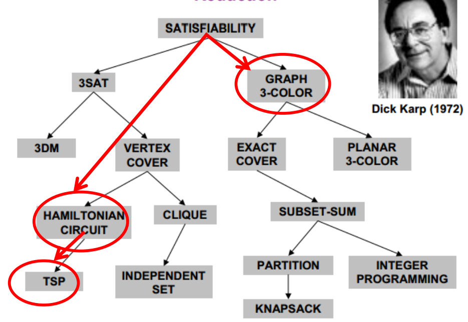
No classes on Wed, Thur and Fri.

We will hold discussions on Tuesday except the one at noon.

On Tuesday I will record a lecture (OHE 100B, at 12:30pm) on approximation algorithms and LP.

These are the last 570 topics.

Reduction



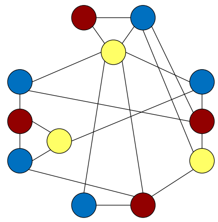
Karp introduced the now standard methodology for proving problems to be NP-Complete.
He received a Turing Award for his work (1985).

Graph Coloring

Given a graph, can you color the nodes with $\leq$ k colors such that the endpoints of every edge are colored differently?

Theorem.
3-Coloring is NP-complete.

Theorem. (k>2)
k-Coloring is NP-complete.



Graph Coloring: k = 2

How can we test if a graph has a 2-coloring?

We can do this by checking if the graph is bipartite.

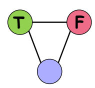
Alternatively, color G in the level order traversal.

3-SAT $\leq_p$ 3-colorable

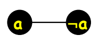
We construct a graph G that will be 3-colorable iff the 3-SAT instance is satisfiable.

Graph G consists of the following gadgets.

A truth gadget:

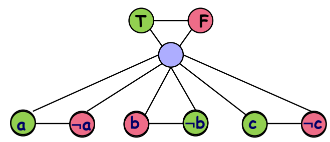


A gadget for each variable:



3-SAT $\leq_p$ 3-colorable

Combining those gadgets together:



Any 3-coloring of the above subgraph defines a valid truth assignment! And vise versa.

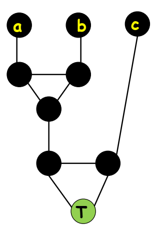
We have to make sure that the truth assignments satisfy the given clauses.

3-SAT $\leq_p$ 3-colorable

A special gadget for each clause in the 3-SAT

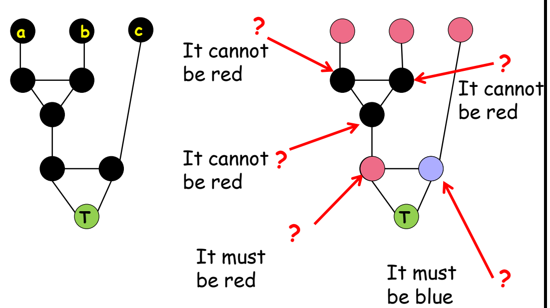
The bottom node is colorable as T (green) iff one of the inputs (a, b or c) is the same color as T.

Let us prove this.



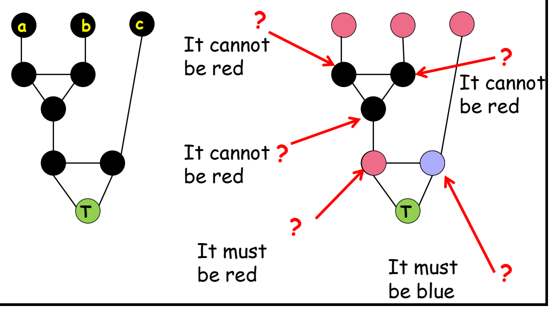
3-SAT $\leq_p$ 3-colorable

Suppose all a, b and c are all False (red).

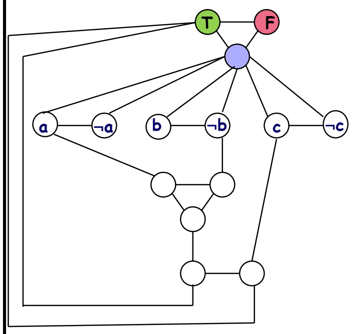


3-SAT $\leq_p$ 3-colorable

We have showed that if all the variables in a clause are false, the gadget cannot be 3-colored



Example: a ∨ ¬b ∨ c

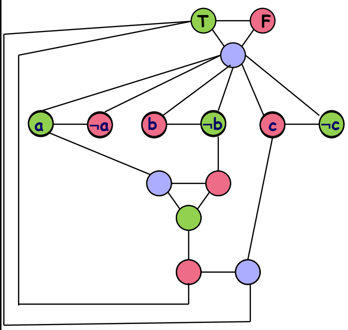


Connect a truth gadget with variables gadgets

Connect variables gadgets with a clause gadget.

Connect a clause gadget with a truth gadget.

Example: a ∨ ¬b ∨ c

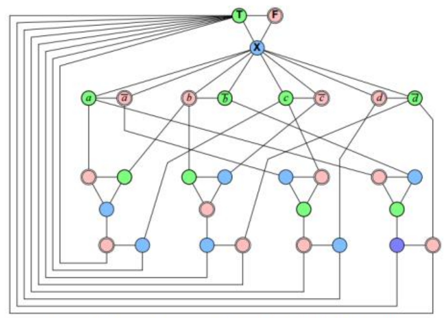


Let a = T, b = c = F
to make a ∨ ¬b ∨ c True

Color the variables with T or F colors.

Coloring for the rest vertices is forced.

Example with four clauses



A 3-colorable graph derived from a satisfiable 3CNF formula.
$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{d})$

a=c=T

b=d=F

3-SAT $\leq_p$ 3-colorable

<u>Claim</u>: *3-SAT instance is satisfiable if and only if G is 3-colorable.*

Proof: →)
Given a satisfying assignment for 3-SAT.

We color the truth gadget with T, F and blue.

We color the variables with T or F according to the assignment.

Coloring for the rest vertices is forced.

3-SAT $\leq_p$ 3-colorable

<u>Claim</u>: *3-SAT instance is satisfiable if and only if G is 3-colorable.*

Proof: ←)
Given a 3-coloring for that graph.

Choose green for T, red for F.

Sudoku

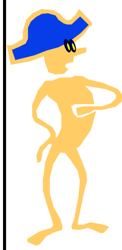
2			3		8		5	
		3		4	5	9	8	
		8			9	7	3	4
6		7		9				
9	8						1	7
				5		6		9
3	1	9	7			2		
	4	6	5	2		8		
	2		9		3			1

NP-?

NP-hard?

Sudoku graph: vertex is each cell, two vertices connected by an edge, if they are in the same row, column and small grid

Hamiltonian Cycle Problem



A Hamiltonian cycle (HC) in a graph is a cycle that visits each vertex exactly once.

Problem Statement:
Given a *directed* graph $G = (V,E)$
Find if the graph contains a Hamiltonian cycle.

A Hamiltonian cycle problem is NP-Complete

A Hamiltonian cycle problem is in NP. Easy!

A Hamiltonian cycle problem is in NP-Hard.

We will prove it by reduction 3-SAT $\leq_p$ HC.

Given a 3-CNF formula Φ, we want to construct a directed graph from Φ with the following properties:

1. a satisfying assignment to Φ translates into a Hamiltonian cycle
2. a Hamiltonian cycle can be translated into a satisfying assignment

3-SAT $\leq_p$ HC

We begin with an arbitrary instance of 3-SAT having variables $X_1,\dots\dots,X_n$ and clauses $C_1,\dots\dots,C_m$

$$(X_1 \vee \neg X_3 \vee \neg X_4) \wedge (X_1 \vee \neg X_2 \vee X_4) \wedge \dots$$

Since there are 2^n assignments, we create a graph containing 2^n different Hamiltonian cycles.

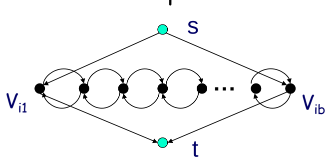
We will build the graph up from pieces called gadgets that "simulate" the clauses and variables.

The variable gadget (one for each X_i)

CNF

Graph

X_i

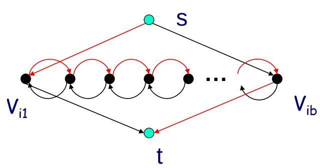


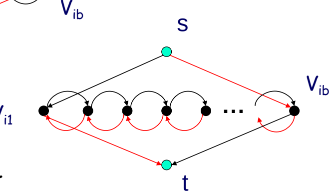
For each variable X_i (i,1,2,…,n) we create a gadget (a cross-bar) with $b = 2m$ (m is # of clauses) vertices $V_{i1}, V_{i2}, \dots, V_{ib}$ and with edges going in both directions.

We also added two special vertices (at the top and bottom)

The variable gadget (one for each X_i)

X_i true: we traverse from Left to Right





X_i false: from R to L

Example

$(X_1 \lor X_2 \lor \neg X_3) \land (\neg X_2 \lor X_3 \lor X_4) \land (\neg X_1 \lor X_2 \lor \neg X_4)$

n = 4, k = 3, b= 2*3 = 6

b = 2m (m is # of clauses)

Construct 4 gadgets:

X_1 consists of nodes $V_{1,1}, V_{1,2}, \ldots, V_{1,6}$

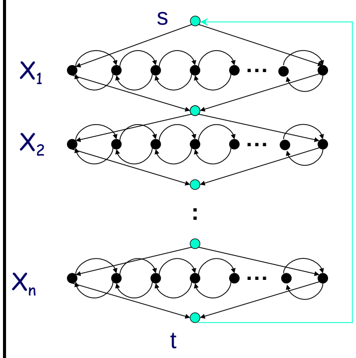
X_2 consists of nodes $V_{2,1}, V_{2,2}, \ldots, V_{2,6}$

X_3 consists of nodes $V_{3,1}, V_{3,2}, \ldots, V_{3,6}$

X_4 consists of nodes $V_{4,1}, V_{4,2}, \ldots, V_{4,6}$

Graph with 24+5 vertices.

The gadget for all variables



A path from s to t translates into a truth assignment to $X_1, \ldots, X_n$

Any Hamiltonian cycle must use the edge (t,s)

The clauses

We now add vertices to model the clauses. One vertex per clause.

We will connect each a variable gadget to a correspondent clause vertex:

For example, (note the direction of edges)

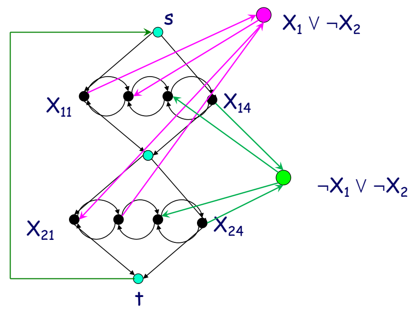
- If clause *C* contains literal X_1,

  we will add edges (X_{11}, C) and (C, X_{12})

- If *C* contains $\neg X_1$,

  we will add edges (X_{12}, C) and (C, X_{11})

Example: $(X_1 \lor \neg X_2) \land (\neg X_1 \lor \neg X_2)$



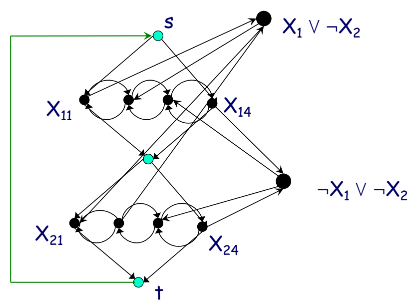
The clauses

In general

- We define a node c_j for each clause C_j.
- If C_j contains X_k, add edges $(X_{k,2j-1}, c_j)$ and $(c_j, X_{k,2j})$
- If C_j contains $\neg X_k$, add edges $(X_{k,2j}, c_j)$ and $(c_j, X_{k,2j-1})$

Graph is constructed !

Example: $(X_1 \lor \neg X_2) \land (\neg X_1 \lor \neg X_2)$



Claim: *If G has a Hamiltonian cycle iff SAT is satisfiable*

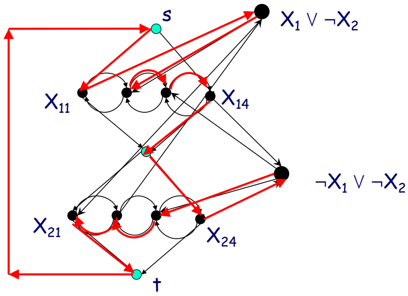
Example: $(X_1 \lor \neg X_2) \land (\neg X_1 \lor \neg X_2)$

$X_1 = T$

since path goes L to R in X_1 gadget

$X_2 = F$

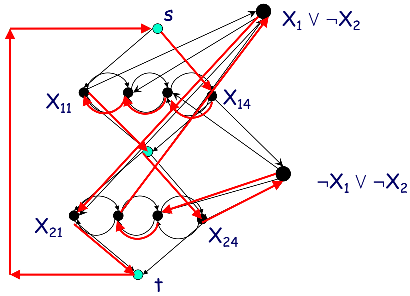
since path goes R to L in X_2 gadget



<u>Claim</u>: *If G has a Hamiltonian cycle then SAT is satisfiable*

Example: $(X_1 \lor \neg X_2) \land (\neg X_1 \lor \neg X_2)$

Let
$X_1 = F$
$X_2 = F$



<u>Claim</u>: *If SAT is satisfiable then G has a Hamiltonian cycle*

Hamiltonian Cycle Problem

<u>Claim</u>: *3-SAT instance is satisfiable if and only if G has a Hamiltonian cycle.*

Proof: $\rightarrow$)

Given a satisfying assignment for 3-SAT.

If X_i = T, traverse X_i gadget L to R, else R to L.
Since each clause C_j is satisfied by the assignment, there has to be at least one path that moves in the right direction to be able to cover node c_j.

Hamiltonian Cycle Problem

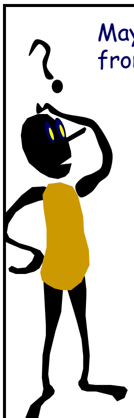
<u>Claim</u>: *3-SAT instance is satisfiable if and only if G has a Hamiltonian cycle.*

Proof: $\leftarrow$)

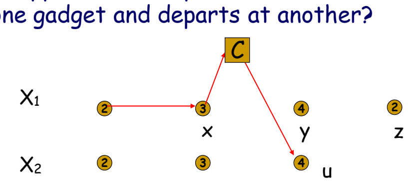
Given a Hamiltonian cycle.

Set each X_i true if path goes L to R through X_i's gadget, false if it goes R to L.
Do we satisfy all clauses?

Consider any clause. We visit a clause in either LR or RL direction. If it's LR - X_i is true, so Cj is satisfied , since it contains X_i. If it's RL - X_i is false, so Cj is satisfied, since it contains $\neg X_i$.



May it happen that the path enters a clause from one gadget and departs at another?



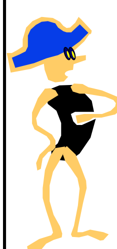
Assume that in a graph we can reach y either from x or z or clause C.

However, the cycle goes through x and C to a vertex u from another gadget.

If we continue traversing the cycle, we reach y from z, and then stuck, nowhere to go from y.

Thus, that is not a HC.

Traveling Sales~~man~~person Problem



Given a weighted graph G=(V,E) with positive edge costs, find the shortest Hamiltonian cycle.

Is it in NP? No, it's not.

We cannot check that there is no shorter cycle in polynomial time.

It follows, TSP is not NP-compete.

TSP: decision version

Given a weighted graph $G=(V,E)$ with positive edge costs, is there a Hamiltonian cycle that has total cost $\leq k$?

Is it in NP?

Yes, we can verify the solution in polynomial time.

Traveling Salesman Problem

Claim: *Decision TSP is NP-Complete.*

Proof by reduction from a HC.

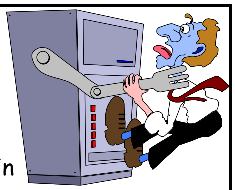
Given the input $G=(V,E)$ to HC, we modify it to construct a complete graph $G'=(V', E')$ and cost on each edge as follows:

$c(u,v) = 0$, if edge $(u,v) \in E$

$c(u,v) = 1$, otherwise.

G has a HC iff $|TSP(G')| = 0$

Don't be afraid of NP-hard problems.



Many reasonable instances (of practical interest) of problems in class NP can be solved!

The largest solved TSP an 85,900-vertex route calculated in 2006. The graph corresponds to the design of a customized computer chip created at Bell Laboratories, and the solution exhibits the shortest path for a laser to follow as it sculpts the chip.