

6.5 Intractability



- ▶ reduction (for algorithm design)
- ▶ reduction (for lower bounds)
- ▶ P and NP
- ▶ intractability
- ▶ NP-completeness
- ▶ coping with intractability

A Reasonable Question about Algorithms

Q. Which algorithms are useful in practice?

A. [von Neumann 1953, Gödel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]

- Model of computation = deterministic Turing machine.
- Measure running time as a function of input size N .
- Useful in practice ("efficient") = polynomial time for all inputs
[worst-case running time is $O(N^b)$ for some b .]

Ex 1. Sorting N elements.

Takes N^2 compares with insertion sort [Useful.]

quicksort is even more useful

Ex 2. Finding best TSP tour on N points.

Takes $N!$ steps with exhaustive search [NOT useful.]

Theory. Definition is broad and robust.

Practice. Poly-time algorithms scale to large problems.

Exponential Growth

Exponential growth dwarfs technological change.

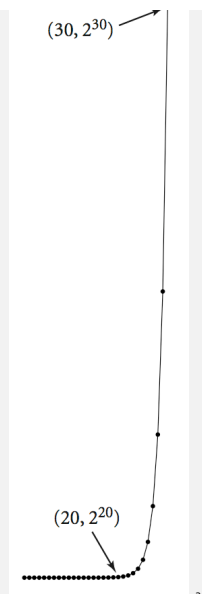
- Suppose you have a giant parallel computing device...
- With as many processors as electrons in the universe...
- And each processor has power of today's supercomputers...
- And each processor works for the life of the universe...

quantity	value
electrons in universe †	10^{79}
supercomputer instructions per second	10^{13}
age of universe in seconds †	10^{17}

† estimated

Will not help solve 1,000 city TSP problem via brute force.

$$1000! \gg 10^{1000} \gg 10^{79} \times 10^{13} \times 10^{17}$$



Reasonable Questions about Problems

Q. Which problems can we solve in practice?

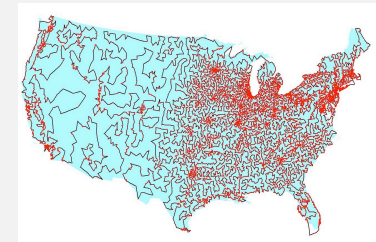
A. Those with easy-to-find answers or with guaranteed poly-time algorithms.

Q. Which problems have guaranteed poly-time algorithms?

A. Not so easy to know. Focus of today's lecture.



many known poly-time algorithms for sorting



no known poly-time algorithm for TSP

- ▶ reduction (for algorithm design)
- ▶ reduction (for lower bounds)
- ▶ P and NP
- ▶ intractability
- ▶ NP-completeness
- ▶ coping with intractability

5

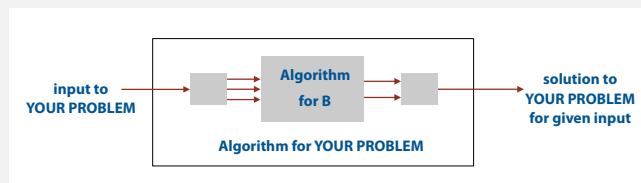


“ Give me a lever long enough and a fulcrum on which to place it, and I shall move the world. ”
— Archimedes

6

Reduction to design an algorithm

Def. YOUR PROBLEM **reduces to** problem B
if you can use an algorithm that solves B to help solve YOUR PROBLEM.



Cost of solving YOUR PROBLEM:
total cost of solving B + total cost of reduction.

↑
perhaps many calls to B
on problems of different sizes

↑
preprocessing
and postprocessing

We refer to B as a **model** for solving YOUR PROBLEM.

7

DUPEXIST reduces to SORTING

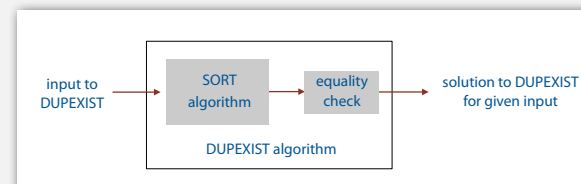
DUPEXIST (Duplicates existence).
Given a set of N elements, determine whether any two are equal.

Cost model: compares

Reduction. To solve DUPEXIST on N elements:

- Sort N elements ($\sim N \log N$ compares)
- Scan to check adjacent pairs for equality ($N-1$ compares)

Cost of solving DUPEXIST. $\sim N \log N + N-1 = \sim N \log N$ compares



8

3-COLLINEAR reduces to SORTING

3-COLLINEAR (cf. programming assignment 3).

Given a set of N points, determine whether any three are collinear.

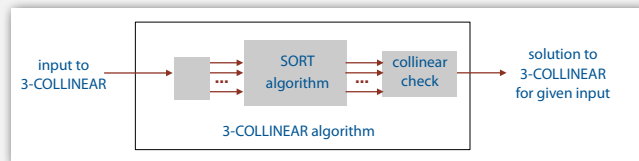
Cost model: compares and collinearity checks

Reduction. To solve 3-COLLINEAR on N elements:

For each point

- Sort other N - 1 points by polar angle ($\sim N^2 \log N$ compares)
- Scan to check adjacent pairs for equality (N^2 checks)

Cost of solving 3-COLLINEAR. $\sim N^2 \log N$ compares + N^2 checks



9

CONVEX HULL reduces to SORTING

CONVEX HULL (cf. lecture 6 slides 53-58).

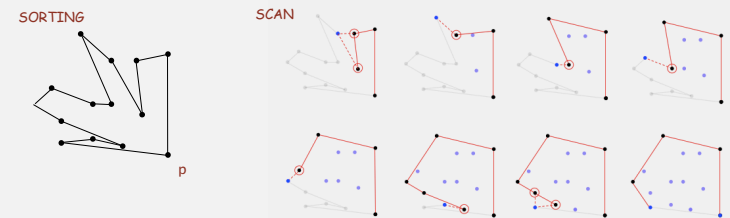
Given N points in the plane, identify the extreme points of the convex hull.

Cost model: compares and CCW checks

Reduction. To solve CONVEX HULL on N elements:

- Sort points by polar angle from point with min y-coord.
- Scan to eliminate CW turns (Graham scan).

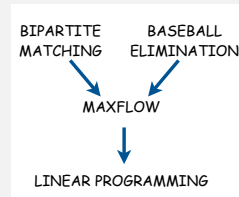
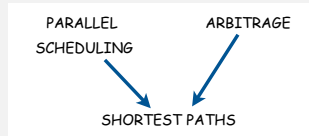
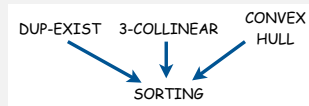
Cost of solving CONVEX HULL. $\sim N \log N$ compares + N checks



10

Some reductions in this course

problem	model	reference
duplicate existence	sorting	page 344
3-collinear	sorting	assignment 3
convex hull	sorting	lecture 6
parallel scheduling	shortest paths	page 663
arbitrage	shortest paths	page 679
bipartite matching	maxflow	page 906 lecture 21
maxflow	linear programming	page 908 lecture 22
baseball elimination	maxflow	assignment 8



11

Summary: a practical implication of reduction

Design an algorithm.

Show that YOUR PROBLEM reduces to problem B

[you can use an algorithm that solves B to help solve YOUR PROBLEM]

Mentality. You have code for B.

Can you use it to solve YOUR PROBLEM?

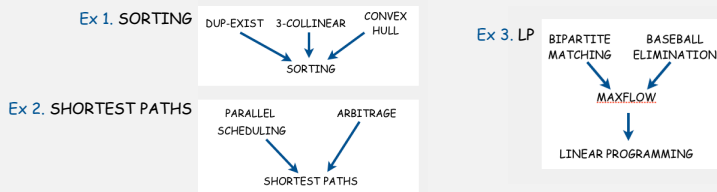
Result. Algorithm for YOUR PROBLEM.

12

Problem-solving models

Important problem-solving models

- We know how to reduce numerous important problems to them.
- We know how to efficiently solve them.



Profound question. Is there a **universal** problem solving model? **Could be (!).**

- **UNIVERSAL** would be a model for **every** problem that scientists, engineers, and applications programmers aspire to compute feasibly.

Requirements.

- We know how to reduce important problems to it. **YES (!)**
- We can efficiently solve all problems in the model. **No one knows (!!)**

13

Lower bounds

Goal. Prove that a problem requires a certain number of steps.

Ex. $\Omega(N \log N)$ lower bound for sorting.

```

1251432
2861534
3988818
4190745
13546464
89885444
43434213
    
```

Bad news. Very difficult to establish lower bounds from scratch.

- Complicated mathematical argument needed.
- Must apply to all conceivable algorithms.

Good news. Often easy to spread $\Omega(N \log N)$ lower bound to **YOUR PROBLEM**.

- Reduce **SORTING** to solve **YOUR PROBLEM**.
- [Only need a simple algorithmic argument.]

15

› reduction (for algorithm design)

› reduction (for lower bounds)

› P and NP

› intractability

› NP-completeness

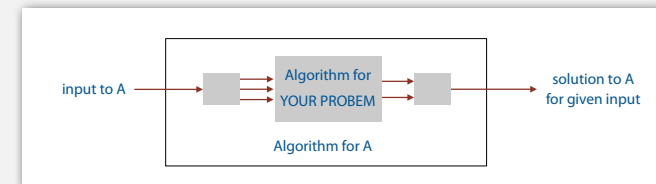
› coping with intractability

14

Reduction to prove lower bounds

Def. Problem **A** **reduces** to **YOUR PROBLEM**

if you can use an algorithm that solves **YOUR PROBLEM** to help solve **A**.



A lower bound for **A** gives a lower bound for **YOUR PROBLEM**.

Lower bound mentality.

- If you could easily solve **YOUR PROBLEM**, you could easily solve **A**.
- You can't easily solve **A**.
- Therefore, you can't easily solve **YOUR PROBLEM**.

16

Linear-time reductions

Def. Problem A **linear-time reduces** to problem B if A can be solved with:

- Linear number of standard computational steps.
- Constant number of calls to B.

Ex. Almost all of the reductions we've seen so far.

Q. Which one wasn't linear-time?

A. 3-COLLINEAR used N calls to SORTING

When a problem A linear-time reduces to a problem B that requires more than linear time, a lower bound is implied.

- Ex 1: If B takes $\Omega(N \log N)$ steps, then so does A.
- Ex 2: If B takes $\Omega(N^2)$ steps, then so does A

17

SORTING reduces to DUPEXIST

DUPEXIST (Duplicates existence).

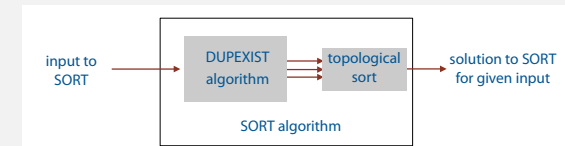
Given a set of N elements, determine whether any two are equal.

Cost model: compares

Reduction (sketch). To solve SORTING on N elements: ← must have $i \rightarrow (i+1)$ for all $i < N-1$

- Instrument DUPEXIST to output add edge $i \rightarrow j$ when it compares a_i and a_j .
- Topologically sort the digraph (linear time)

Cost of solving SORTING = cost of solving DUPEXIST (+ topological sort)



lower-bound mentality:
if I could solve DUP-EXIST with fewer,
I could solve SORT with fewer.

Implication. DUPEXIST requires $\Omega(N \log N)$ compares ←

18

Lower bound for convex hull

Proposition. In quadratic decision tree model, any algorithm for sorting N integers requires $\Omega(N \log N)$ steps.

allows linear or quadratic tests of the form:
 $x_i < x_j$ or $(x_j - x_i)(x_k - x_i) - (x_j - x_i) < 0$

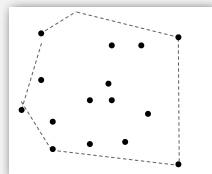
Proposition. Sorting linear-time reduces to convex hull.

Pf. [see next slide]

lower-bound mentality:
if I can solve convex hull
efficiently, I can sort efficiently

1251432
2861534
3988818
4190745
13546464
89885444
43434213

sorting



convex hull

linear or
quadratic tests

Implication. Any ccw-based convex hull algorithm requires $\Omega(N \log N)$ ops.

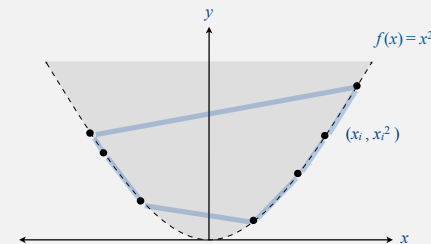
19

SORTING linear-time reduces to CONVEX HULL

Proposition. SORTING linear-time reduces to CONVEX HULL.

- Sorting instance: x_1, x_2, \dots, x_N .
- Convex hull instance: $(x_1, x_1^2), (x_2, x_2^2), \dots, (x_N, x_N^2)$.

lower-bound mentality:
if I can solve convex hull
efficiently, I can sort efficiently



Pf.

- Region $\{x : x^2 \geq x\}$ is convex \Rightarrow all points are on hull.
- Starting at point with most negative x , counterclockwise order of hull points yields integers in ascending order.

20

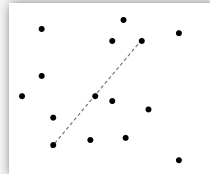
Lower bound for 3-COLLINEAR

3-SUM. Given N distinct integers, are there three that sum to 0?

3-COLLINEAR. Given N distinct points in the plane, ← recall Assignment 3 are there 3 that all lie on the same line?

```
1251432
-2861534
3988818
-4190745
13546464
89885444
-43434213
```

3-sum



3-collinear

Lower bound for 3-COLLINEAR

3-SUM. Given N distinct integers, are there three that sum to 0?

3-COLLINEAR. Given N distinct points in the plane, are there 3 that all lie on the same line?

Proposition. 3-SUM linear-time reduces to 3-COLLINEAR.

Pf. [next two slides]

Conjecture. Any algorithm for 3-SUM requires $\Omega(N^2)$ steps.

Implication. No sub-quadratic algorithm for 3-COLLINEAR likely.

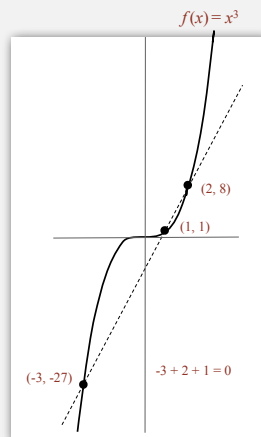
← your $N^2 \log N$ algorithm was pretty good

3-SUM linear-time reduces to 3-COLLINEAR

Proposition. 3-SUM linear-time reduces to 3-COLLINEAR.

- 3-SUM instance: x_1, x_2, \dots, x_N .
- 3-COLLINEAR instance: $(x_1, x_1^3), (x_2, x_2^3), \dots, (x_N, x_N^3)$.

Lemma. If $a, b,$ and c are distinct, then $a + b + c = 0$ if and only if $(a, a^3), (b, b^3),$ and (c, c^3) are collinear.



3-SUM linear-time reduces to 3-COLLINEAR

Proposition. 3-SUM linear-time reduces to 3-COLLINEAR.

- 3-SUM instance: x_1, x_2, \dots, x_N .
- 3-COLLINEAR instance: $(x_1, x_1^3), (x_2, x_2^3), \dots, (x_N, x_N^3)$.

Lemma. If $a, b,$ and c are distinct, then $a + b + c = 0$ if and only if $(a, a^3), (b, b^3),$ and (c, c^3) are collinear.

Pf. Three distinct points $(a, a^3), (b, b^3),$ and (c, c^3) are collinear iff:

$$\begin{aligned}
 0 &= \begin{vmatrix} a & a^3 & 1 \\ b & b^3 & 1 \\ c & c^3 & 1 \end{vmatrix} \quad \leftarrow \text{shortcut: setting slopes equal gives the same result} \\
 &= a(b^3 - c^3) - b(a^3 - c^3) + c(a^3 - b^3) \\
 &= (a - b)(b - c)(c - a)(a + b + c)
 \end{aligned}$$

Establishing lower bounds: summary

Establishing lower bounds through reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself no linear-time **CONVEX HULL** algorithm exists?

A1. [hard way] Long futile search for a linear-time algorithm.

A2. [easy way] Linear-time reduction from sorting.

Q. How to convince yourself no sub-quadratic **3-COLLINEAR** algorithm is likely?

A1. [hard way] Long futile search for a sub-quadratic algorithm.

A2. [easy way] Linear-time reduction from **3-SUM**.

Q. How to better understand the difficulty of a new problem?

A1. [hard way] Search in the dark for a fast algorithm.

A2. [easier way] Linear-time reduction **from** a known difficult problem (then you know it's not likely to be easier).

25

Summary: A second practical implication of reduction

Design an algorithm.

Show that **YOUR PROBLEM** reduces to problem B

[you can use an algorithm that solves B to help solve **YOUR PROBLEM**]

Mentality. You have code for B.

Can you use it to solve **YOUR PROBLEM**?

Result. Algorithm for **YOUR PROBLEM**.

Prove a lower bound.

Show that **A** reduces to **YOUR PROBLEM**

[you can use an algorithm that solves **YOUR PROBLEM** to help solve **A**]

Mentality. If you can solve **YOUR PROBLEM**, you can also solve **A**.

Would that be a better solution than possible (or known solutions)?

Result. Lower bound for **YOUR PROBLEM**.

26

- › reduction (for algorithm design)
- › reduction (for lower bounds)
- › **P and NP**
- › intractability
- › NP-completeness
- › coping with intractability

27

Four Fundamental Problems

LSOLVE. Given a system of **linear equations**, find a solution.

$$\begin{array}{l} 0x_0 + 1x_1 + 1x_2 = 4 \\ 2x_0 + 4x_1 - 2x_2 = 2 \\ 0x_0 + 3x_1 + 15x_2 = 36 \end{array} \quad \begin{array}{l} x_0 = -1 \\ x_1 = 2 \\ x_2 = 2 \end{array}$$

LP. Given a system of **linear inequalities**, find a solution.

$$\begin{array}{l} 48x_0 + 16x_1 + 119x_2 \leq 88 \\ 5x_0 + 4x_1 - 35x_2 \geq 13 \\ 15x_0 + 4x_1 + 20x_2 \geq 23 \\ x_0, x_1, x_2 \geq 0 \end{array} \quad \begin{array}{l} x_0 = 1 \\ x_1 = 1 \\ x_2 = 1/5 \end{array}$$

0/1 ILP. Given a system of **linear inequalities**, find a **binary** solution.

$$\begin{array}{l} 0x_0 + 1x_1 + 1x_2 \geq 1 \\ 1x_0 + 0x_1 + 1x_2 \geq 1 \\ 1x_0 + 1x_1 + 1x_2 \leq 2 \end{array} \quad \begin{array}{l} x_0 = 0 \\ x_1 = 1 \\ x_2 = 1 \end{array} \rightarrow \text{each } x_i \text{ is either 0 or 1}$$

SAT. Given a system of **boolean equations**, find a solution.

$$\begin{array}{l} x_0 \text{ or } x_1 \text{ or } \neg x_2 = T \\ x_0 \text{ or } \neg x_1 \text{ or } x_2 = T \\ \neg x_0 \text{ or } x_1 \text{ or } x_2 = T \end{array} \quad \begin{array}{l} x_0 = T \\ x_1 = F \\ x_2 = T \end{array} \rightarrow \text{each } x_i \text{ is either T or F}$$

standard form: use only or and \neg (not)

28

Four Fundamental Problems

LSOLVE. Given a system of linear equations, find a solution.

LP. Given a system of linear inequalities, find a solution.

ILP. Given a system of linear inequalities, find a binary solution.

SAT. Given a system of boolean equations, find a solution.

Q. Which of these problems have guaranteed poly-time solutions?

A. No easy answers.

✓ **LSOLVE.** Yes. Gaussian elimination solves n -by- n system in n^3 time.

✓ **LP.** Yes. Ellipsoid algorithm is poly-time. ← problem was open for decades

? **ILP and SAT** No poly-time algorithm known or believed to exist!

29

Search Problems

Search problem. Given an instance I of a problem, **find** a solution S . ← or report none exists

Requirement. Must be able to efficiently **check** that S is a solution. ← guaranteed poly-time in size of instance I



30

Search Problems

Search problem. Given an instance I of a problem, **find** a solution S .

Requirement. Must be able to efficiently **check** that S is a solution.

LSOLVE. Given a system of linear equations, find a solution.

$$\begin{array}{r} 0x_0 + 1x_1 + 1x_2 = 4 \\ 2x_0 + 4x_1 - 2x_2 = 2 \\ 0x_0 + 3x_1 + 15x_2 = 36 \end{array}$$

instance I

$$\begin{array}{r} x_0 = -1 \\ x_1 = 2 \\ x_2 = 2 \end{array}$$

solution S

- To check solution S , plug in values and verify each equation.

31

Search Problems

Search problem. Given an instance I of a problem, **find** a solution S .

Requirement. Must be able to efficiently **check** that S is a solution.

LP. Given a system of linear inequalities, find a solution.

$$\begin{array}{r} 48x_0 + 16x_1 + 119x_2 \leq 88 \\ 5x_0 + 4x_1 - 35x_2 \geq 13 \\ 15x_0 + 4x_1 + 20x_2 \geq 23 \\ x_0, x_1, x_2 \geq 0 \end{array}$$

instance I

$$\begin{array}{r} x_0 = 1 \\ x_1 = 1 \\ x_2 = 1/5 \end{array}$$

solution S

- To check solution S , plug in values and verify each inequality.

32

Search Problems

Search problem. Given an instance I of a problem, **find** a solution S .
Requirement. Must be able to efficiently **check** that S is a solution.

0/1 ILP. Given a system of linear inequalities, find a binary solution.

$\begin{aligned} 0x_0 + 1x_1 + 1x_2 &\geq 1 \\ 1x_0 + 0x_1 + 1x_2 &\geq 1 \\ 1x_0 + 1x_1 + 1x_2 &\leq 2 \end{aligned}$	$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 1 \end{aligned}$
instance I	solution S

- To check solution S , plug in values and verify each inequality (and check that solution is 0/1).

33

Search Problems

Search problem. Given an instance I of a problem, **find** a solution S .
Requirement. Must be able to efficiently **check** that S is a solution.

SAT. Given a system of boolean equations, find a solution.

$\begin{aligned} x_0 \text{ or } x_1 \text{ or } \neg x_2 &= T \\ x_0 \text{ or } \neg x_1 \text{ or } x_2 &= T \\ \neg x_0 \text{ or } x_1 \text{ or } x_2 &= T \end{aligned}$	$\begin{aligned} x_0 &= T \\ x_1 &= F \\ x_2 &= T \end{aligned}$
instance I	solution S

- To check solution S , plug in values and verify each equation

34

Search Problems

Search problem. Given an instance I of a problem, **find** a solution S .
Requirement. Must be able to efficiently **check** that S is a solution.

FACTOR. Find a nontrivial factor of the integer x .

147573952589676412927	193707721
instance I	solution S

- To check solution S , long divide 193707721 into 147573952589676412927.

35

NP

Def. NP is the class of all search problems
 [problems with poly-time checkable solutions].

problem	description	poly-time alg
LSOLVE	solve simultaneous linear equations	Gaussian elimination
LP	solve simultaneous linear inequalities	ellipsoid
ILP	solve simultaneous linear inequalities with 0-1 solution	??
SAT	solve simultaneous boolean equations (CNF formula)	??
FACTOR	find nontrivial factor	??

Significance. What scientists, engineers, and applications programmers **aspire to compute** feasibly.

36

Def. P is the class of all search problems that are solvable in poly-time.
 [Solved by some alg with worst-case running time $O(N^c)$ for some c .]

problem	description	poly-time alg
SORT	find a permutation that puts array in order	all algs in Chapter 2
st-CONNECTIVITY	find a path from s to t in a digraph	depth-first search
...
LSOLVE	solve simultaneous linear equations	Gaussian elimination
LP	solve simultaneous linear inequalities	ellipsoid

Significance. What scientists, engineers, and applications programmers do compute feasibly. [The problems we've studied before this lecture.]

Three types of problems:

A search problem: Find a solution. [our focus]

A decision problem: Does a solution exist? [standard focus]

An optimization problem: Find the best solution. [another possibility]

We focus on search problems

- need to make a choice to avoid confusion
- main ideas carry through to other types of problems
- [some natural problems are not search problems]
- Interested in distinctions? See COS 487.

Nondeterminism

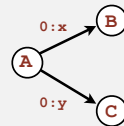
Nondeterministic machine can guess the desired solution

Ex 1. 0/1 ILP. Given a system of linear inequalities, guess a binary solution.

$\begin{aligned} 0x_0 + 1x_1 + 1x_2 &\geq 1 \\ 1x_0 + 0x_1 + 1x_2 &\geq 1 \\ 1x_0 + 1x_1 + 1x_2 &\leq 2 \end{aligned}$	$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 1 \end{aligned}$
instance I	solution S

Ex 2. Turing machines.

- deterministic: state, input determines next state
- nondeterministic: more than one possible next state



NP: Set of problems solvable in poly time on a nondeterministic machine.
 [Another way to define search problems.]

Extended Church-Turing Thesis

Extended Church-Turing thesis.

P = search problems solvable in poly-time in this universe.

Evidence supporting thesis.

- Seems to be true for all physical computers.
- Simulating one computer on another adds poly-time cost factor.
- Nondeterministic machine seems to be a fantasy.

Implication. To make future computers more efficient, suffices to focus on improving implementation of existing designs.

A new law of physics? A constraint on what is possible.
 Possible counterexample? Quantum computer

The Central Question

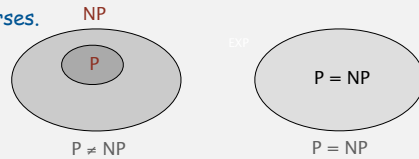
P. Class of search problems solvable in poly-time.

NP. Class of all search problems.

Does $P = NP$?

- can you always avoid brute-force search and do better??
- does nondeterminism make a computer more efficient??
- are there **any** intractable search problems??

Two possible universes.



If **yes**... Poly-time algorithms for SAT, ILP, TSP, FACTOR, ...

If **no**... Would learn something fundamental about our universe.

Overwhelming consensus. $P \neq NP$.

41

Not in P

Q. How to solve an instance of SAT with N variables?

A. Exhaustive search: try all 2^N truth assignments.



Q. Can we do anything substantially more clever?

A. No one knows!

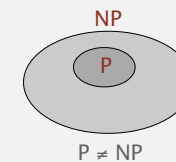
Conjecture (widely accepted). SAT is not in P. [no poly time alg for SAT].

A search problem that is not in P is said to be **intractable**.

42

Classifying Problems

Suppose that we live in the $P \neq NP$ universe (the overwhelming consensus).



Q. Which search problems are in P?

A. Those solved by algs we've studied: SORTING, MAXFLOW, LP, SP...

Q. Which search problems are not in P (intractable)?

A. No easy answers (we don't even know for sure that $P \neq NP$).

Only tool available. Use reduction to prove relationships among problems.

44

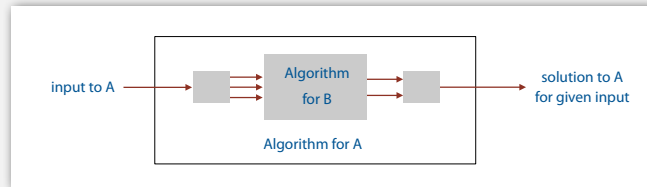
- › reduction (for algorithm design)
- › reduction (for lower bounds)
- › P and NP
- › **intractability**
- › NP-completeness
- › coping with intractability

43

"Cook" reduction

Def. Problem A **poly-time reduces** to problem B if A can be solved with:

- Number of computational steps for reduction **bounded by a polynomial**
- Number of calls to B **bounded by a polynomial**



Very general (but not the only) notion of reduction.

Ex. All of the reductions we've seen so far.

Two applications

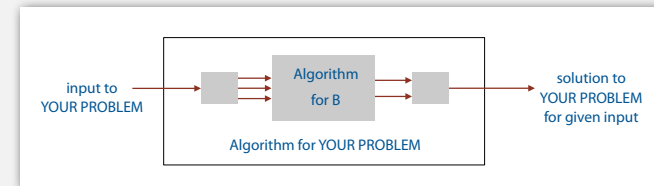
- prove that a problem is in P
- prove that a problem is not in P (intractable)

45

Reduction to prove that a problem is in P [design an algorithm]

Def. YOUR PROBLEM **poly-time reduces** to problem B

if you can use an algorithm that solves B to help solve YOUR PROBLEM.



A poly-time algorithm for B gives a poly-time algorithm for YOUR PROBLEM.

To prove that YOUR PROBLEM is in P:

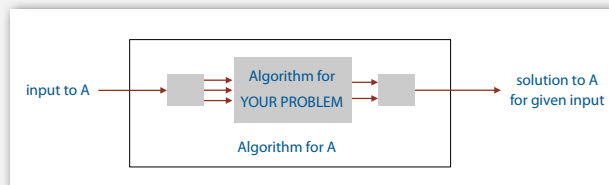
Poly-time reduce it to a problem known to be in P.

46

Reduction to establish intractability [prove a lower bound]

Def. Problem A **poly-time reduces** to YOUR PROBLEM

if you can use an algorithm that solves YOUR PROBLEM to help solve A.



If A is intractable, then YOUR PROBLEM is intractable.

Lower bound mentality.

- If you could solve YOUR PROBLEM in poly time, you could solve A in poly time.
- You can't solve A in poly time (it is intractable).
- Therefore, you can't solve YOUR PROBLEM in poly time (it is intractable).

To prove that YOUR PROBLEM is intractable (not in P):

Poly-time reduce an intractable problem to YOUR PROBLEM.

47

Using reduction to classify problems

To prove that YOUR PROBLEM is in P:

- Find a problem B known to be in P.
[MAXFLOW, LP, SORTING,]
- Poly-time reduce YOUR PROBLEM to B.

To prove that YOUR PROBLEM is intractable (not in P):

- Find an intractable problem A.
[Starting assumption: SAT is intractable.]
- Poly-time reduce A to YOUR PROBLEM.

Next: Several examples.

48

SAT reduces to 3-SAT

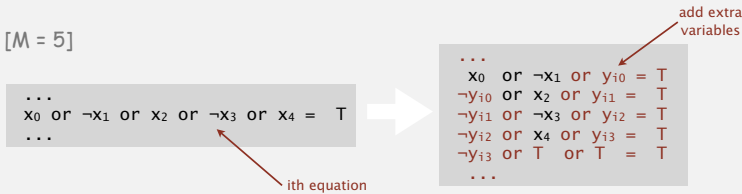
3-SAT. Given a system of boolean equations with **three literals per equation**, find a solution.

Reduction. To solve SAT:

- Convert each M -literal equation to M 3-literal equations
- Solve 3-SAT instance.

Cost of solving N -equation SAT = Cost of solving NM -equation 3-SAT

Ex. [$M = 5$]



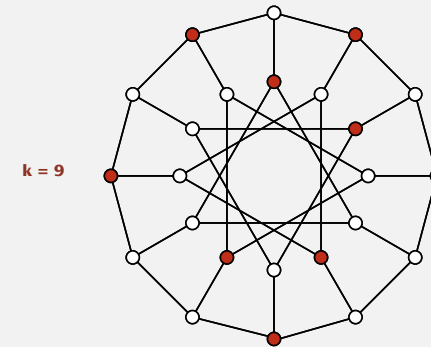
Implication. 3-SAT is intractable (assuming that SAT is intractable).
[poly-time solution to 3-SAT would give poly-time solution to SAT]

49

Independent set

An **independent set** in a graph is a set of vertices, no two of which are adjacent.

IND-SET. Given a graph G and an integer k , find an independent set of size k .



Applications. Scheduling, computer vision, clustering, ...

50

3-SAT reduces to IND-SET

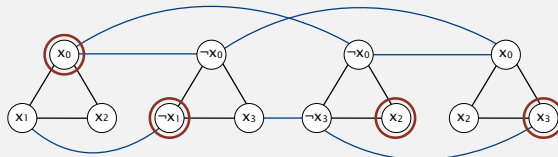
IND-SET. Given a graph G and an integer k , find an independent set of size k .

Reduction. To solve 3-SAT:

- Build a graph by creating 3 vertices in a triangle for each equation.
- Add edges between each literal and its negation.
- Solve IND-SET for that graph
- Set literals corresponding to independent set to T.

Ex. ($k = 4$)

x_0 or x_1 or $x_2 = T$
 $\neg x_0$ or $\neg x_1$ or $x_3 = T$
 $\neg x_0$ or x_2 or $\neg x_3 = T$
 x_0 or x_2 or $x_3 = T$



$x_0 = T$
 $x_1 = F$
 $x_2 = T$
 $x_3 = T$

51

SAT reduces to IND-SET

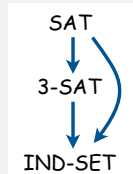
Reduction 1. SAT poly-time reduces to 3-SAT.

Reduction 2. 3-SAT poly-time reduces to IND-SET.

Transitivity. If X poly-time reduces to Y and Y poly-time reduces to Z , then X poly-time reduces to Z .

Therefore, SAT poly-time reduces to IND-SET.

Implication. Assuming SAT is intractable, so is IND-SET.



lower-bound mentality:
 if I could solve IND-SET efficiently,
 I could solve 3-SAT efficiently;
 if I could solve 3-SAT efficiently,
 I could solve SAT efficiently

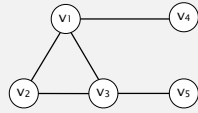
52

Independent set reduces to integer linear programming

Proposition. IND-SET poly-time reduces to ILP.

Pf. Given an instance G, k of IND-SET, create an instance of ILP as follows:

Intuition. $x_i = 1$ if and only if vertex v_i is in independent set.



is there an independent set of size 3?

$x_1 + x_2 + x_3 + x_4 + x_5 = 3$	← number of vertices selected
$x_1 + x_2 \leq 1$	← at most one vertex selected from each edge
$x_2 + x_3 \leq 1$	
$x_1 + x_3 \leq 1$	
$x_1 + x_4 \leq 1$	
$x_3 + x_5 \leq 1$	
all $x_i = \{0, 1\}$	← binary variables

is there a feasible solution?

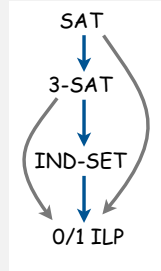
53

SAT reduces to 0/1 ILP

Reduction 1. SAT poly-time reduces to 3-SAT.

Reduction 2. 3-SAT poly-time reduces to IND-SET.

Reduction 3. IND-SET poly-time reduces to 0/1 ILP.



By transitivity, SAT poly-time reduces to 0/1 ILP.

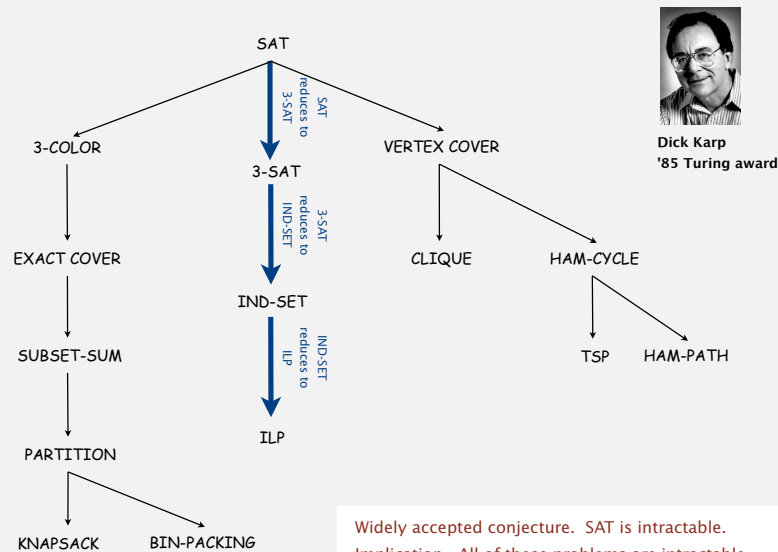
[also 3-SAT poly-time reduces to 0/1 ILP]

Implication. Assuming SAT is intractable, so is 0/1 ILP.

lower-bound mentality:
 if I could solve 0/1 ILP efficiently,
 I could solve IND-SET efficiently;
 if I could solve IND-SET efficiently,
 I could solve 3-SAT efficiently;
 if I could solve 3-SAT efficiently,
 I could solve SAT efficiently

54

More poly-time reductions from satisfiability



55

Still More Reductions from 3-SAT

- Aerospace engineering.** Optimal mesh partitioning for finite elements.
- Biology.** Phylogeny reconstruction.
- Chemical engineering.** Heat exchanger network synthesis.
- Chemistry.** Protein folding.
- Civil engineering.** Equilibrium of urban traffic flow.
- Economics.** Computation of arbitrage in financial markets with friction.
- Electrical engineering.** VLSI layout.
- Environmental engineering.** Optimal placement of contaminant sensors.
- Financial engineering.** Minimum risk portfolio of given return.
- Game theory.** Nash equilibrium that maximizes social welfare.
- Mathematics.** Given integer a_1, \dots, a_n , compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \dots \times \cos(a_n\theta) d\theta$
- Mechanical engineering.** Structure of turbulence in sheared flows.
- Medicine.** Reconstructing 3d shape from biplane angiocardiogram.
- Operations research.** Traveling salesperson problem, integer programming.
- Physics.** Partition function of 3d Ising model.
- Politics.** Shapley-Shubik voting power.
- Pop culture.** Versions of Sudoku, Checkers, Minesweeper, Tetris.
- Statistics.** Optimal experimental design.

6,000+ scientific papers per year.

Widely accepted conjecture. SAT is intractable.
 Implication. All of these problems are intractable.

56

Implications of poly-time reductions from SAT

Establishing intractability through poly-time reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself that a new problem is (likely) intractable?

A1. [hard way] Long futile search for an efficient algorithm (as for SAT).

A2. [easy way] Reduction from some problem for which reduction from SAT is known (tens of thousands to choose from!).

Caveat. Intricate reductions are common.

57

- › reduction (for algorithm design)
- › reduction (for lower bounds)
- › P and NP
- › intractability
- › **NP-completeness**
- › coping with intractability

58

NP-Completeness

Q. Why do we believe SAT has no poly-time algorithm?

A. Because **Cook's theorem** tells us that would imply that $P = NP$

Def. An NP problem is **NP-complete** if all problems in NP reduce to it.

Cook's Theorem. [1971] SAT is NP-complete. ← every NP problem is a SAT problem in disguise

Extremely brief proof sketch:

- convert non-deterministic TM notation to SAT notation
- if you can solve SAT, you can solve any problem in NP



59

Cook's theorem implications

[An NP problem is NP-complete if all problems in NP poly-time to reduce to it.]

Cook's theorem. SAT is NP-complete.

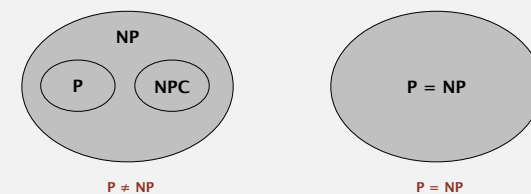
Corollary 1. SAT is tractable if and only if $P = NP$.

Pf (\leftarrow). If $P = NP$, all problems in NP are tractable (in P).

Pf (\rightarrow). Suppose SAT is tractable. Since any problem A in NP poly-time reduces to SAT, the reduction gives a poly-time algorithm for A.

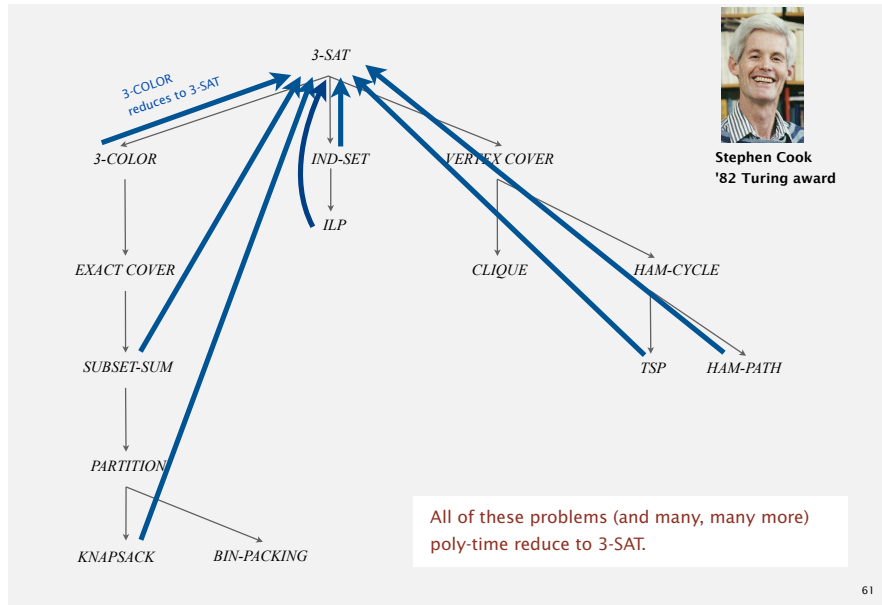
Corollary 2. Any NP-complete problem is tractable if and only if $P = NP$.

More detailed view of two universes:

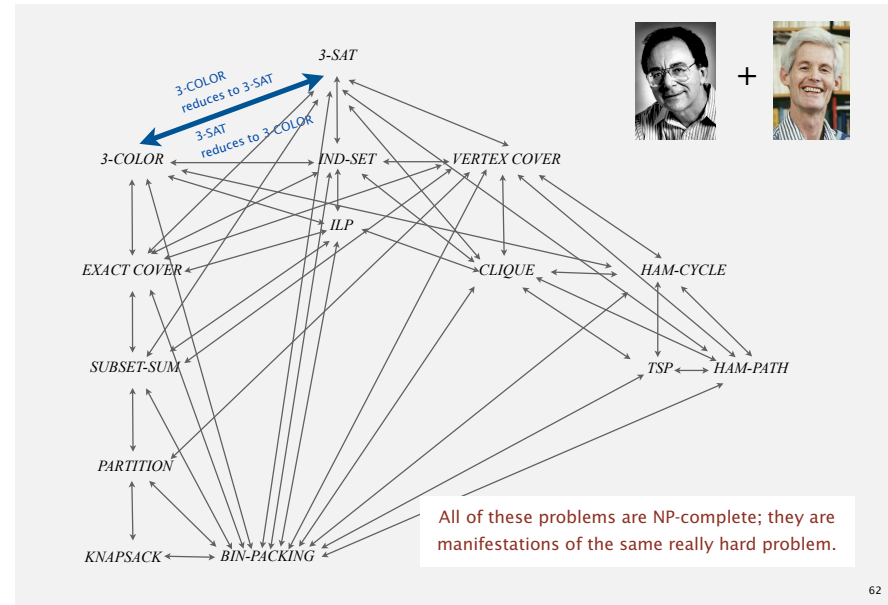


60

Implications of Cook's theorem



Implications of Karp + Cook



Implications of NP-completeness

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

Two possible universes

$P \neq NP$.

- Intractable search problems exist.
- Nondeterminism makes machines more efficient.
- Can prove that a problem is intractable by reduction from an NP-complete problem [no other way is known!]
- Some search problems are neither NP-complete or in P [we don't know any useful ones].
- Some search problems are still not classified [ex. factoring, graph isomorphism].

$P \neq NP$

$P = NP$.

- No intractable search problems exist.
- Nondeterminism is no help.
- Poly-time solutions exist for NP-complete problems [and all other problems in NP, such as factoring and graph isomorphism].

$P = NP$

A pitfall

FACTOR. Given an n -bit integer x , find a nontrivial factor.

```
740375634795617128280467960974295731425931888892312890849362
326389727650340282662768919964196251178439958943305021275853
701189680982867331732731089309005525051168770632990723963807
86710086096962537934650563796359
```

Q. What is complexity of FACTOR?

A. In NP, but not known (or believed) to be in P or NP-complete.

Q. What if $P = NP$?

A. Poly-time algorithm for factoring; modern e-conomy collapses.

Quantum. [Shor 1994]

Can factor an N -bit integer in N^3 steps on a "quantum computer."

Pitfall. If the Extended Church-Turing thesis is not valid, P vs. NP is less relevant

if there exists a physical machine that cannot be simulated in poly time on standard machines, we might prefer that one!

65

Summary

P. Class of search problems solvable in poly-time.

NP. Class of all search problems, some of which seem wickedly hard.

NP-complete. Hardest problems in NP.

Intractable. Search problems not in P (if $P \neq NP$).

Many fundamental problems are NP-complete

- TSP, 3-SAT, 3-COLOR, ILP, (and thousands of others)
- 3D-ISING.

Use theory as a guide.

- An efficient algorithm for an NP-complete problem would be a stunning scientific breakthrough (a proof that $P = NP$)
- You will confront NP-complete problems in your career.
- It is safe to assume that $P \neq NP$ and that such problems are intractable.
- Identify these situations and proceed accordingly.

67

- › reduction (for algorithm design)
- › reduction (for lower bounds)
- › P and NP
- › intractability
- › NP-completeness
- › coping with intractability

66

Coping With Intractability

You have an NP-complete problem.

- It's safe to assume that it is intractable.
- What to do?

Relax one of desired features.

- Solve the problem in poly-time.
- Solve the problem to optimality.
- Solve arbitrary instances of the problem.

Complexity theory deals with worst case behavior.

- Instance(s) you want to solve may have easy-to-find answer.
- Ex: `chaff` solves real-world SAT instances with ~ 10K variables.

[Matthew Moskewicz '00, Conor Madigan '00, Sharad Malik]

PU senior independent work (!)

68

Coping With Intractability

You have an NP-complete problem.

- It's safe to assume that it is intractable.
- What to do?

Relax one of desired features.

- Solve the problem in poly-time.
- Solve the problem to optimality.
- Solve arbitrary instances of the problem.

Develop a heuristic, and hope it produces a good solution.

- No guarantees on quality of solution.
- Ex. TSP assignment heuristics.
- Ex. Metropolis algorithm, simulating annealing, genetic algorithms.

Approximation algorithm. Find solution of provably good quality.

- Ex. MAX-3SAT: provably satisfy 87.5% as many clauses as possible.
↑ but if you can guarantee to satisfy 87.51% as many clauses as possible in poly-time, then P = NP !

69

Coping With Intractability

You have an NP-complete problem.

- It's safe to assume that it is intractable.
- What to do?

Relax one of desired features.

- Solve the problem in poly-time.
- Solve the problem to optimality.
- Solve arbitrary instances of the problem.

Special cases may be tractable.

- Ex: Linear time algorithm for 2-SAT.
- Ex: Linear time algorithm for Horn-SAT.
↑ each clause has at most one un-negated literal

70

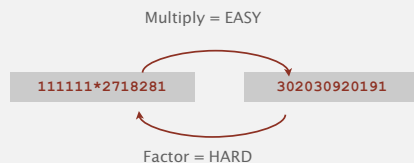
Exploiting Intractability: Cryptography

Modern cryptography.

- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA cryptosystem.

- To use: multiply two N-bit integers. [poly-time]
- To break: factor a 2N-bit integer. [unlikely poly-time]



71

Exploiting Intractability: Cryptography

FACTOR. Given an n -bit integer x , find a nontrivial factor.

```
740375634795617128280467960974295731425931888892312890849362
326389727650340282662768919964196251178439958943305021275853
701189680982867331732731089309005525051168770632990723963807
86710086096962537934650563796359
```

Q. What is complexity of FACTOR?

A. In NP, but not known (or believed) to be in P or NP-complete.

Q. Is it safe to assume that FACTOR is intractable?

A. Maybe, but not as safe an assumption as for an NP-complete problem.

72

Fame and Fortune through CS (revisited)

Factor this number:

740375634795617128280467960974295731425931888892312890849362
 326389727650340282662768919964196251178439958943305021275853
 701189680982867331732731089309005525051168770632990723963807
 86710086096962537934650563796359

RSA-704 (\$30,000 prize if you can factor)

Can't do it? Create a company based on the difficulty of factoring.

RSA algorithm

$P \ \& \ Q \ \text{PRIME}$
 $N = PQ$
 $ED = 1 \pmod{(P-1)(Q-1)}$
 $C = M^E \pmod N$
 $M = C^D \pmod N$

The RSA algorithm is the most widely used method of implementing public key cryptography and has been deployed in more than one billion applications worldwide.



RSA sold to EMC for
\$2.1 billion



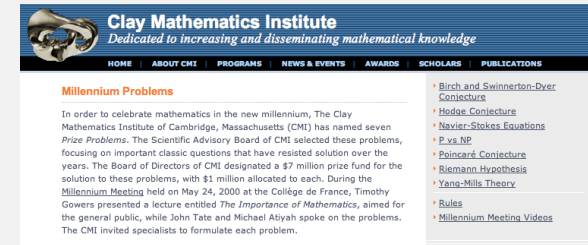
or, sell T-shirts

Fame and Fortune through CS (revisited)

Factor this number:

740375634795617128280467960974295731425931888892312890849362
 326389727650340282662768919964196251178439958943305021275853
 701189680982867331732731089309005525051168770632990723963807
 86710086096962537934650563796359

Too late? Try resolving $P = NP?$ question (might need a few math courses)
 or, try building a quantum computer.



Clay Institute (\$1 million prize)

plus untold riches for breaking e-commerce