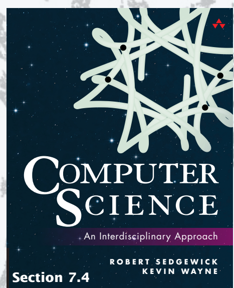
COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART II: ALGORITHMS, MACHINES, and THEORY



http://introcs.cs.princeton.edu

16. Intractability

COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART II: ALGORITHMS, MACHINES, and THEORY

16. Intractability

- **Reasonable questions**
- P and NP
- Poly-time reductions
- NP-completeness
- Living with intractability

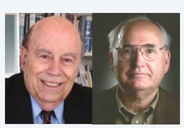
CS.16.A.Intractability.Questions

Context

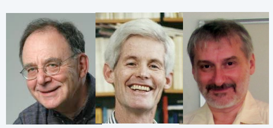
Fundamental questions
- What is a general-purpose computer? ✓
- Are there limits on what we can do with digital computers? ✓
- Are there limits on what we can do with the machines we can build? ← focus of today's lecture

Kurt Gödel 1909–1994
Asked the question in a "lost letter" to von Neumann

John Nash
Asked the question in a "lost letter" to the NSA



Michael Rabin Dana Scott
Introduced the critical concept of nondeterminism



Dick Karp Steve Cook Leonid Levin
Asked THE question

Answer still unknown

A difficult problem

Traveling salesperson problem (TSP)
- Given: A set of N cities, distances between each pair of cities, and a threshold M.
- Problem: Is there a tour through all the cities of length less than M?

Exhaustive search. Try all $N!$ orderings of the cities to look for a tour of length less than M.

How difficult can it be?

Excerpts from a recent blog...

If one took the 100 largest cities in the US and wanted to travel them all, what is the distance of the shortest route? I'm sure there's a simple answer. Anyone wanna help? A quick google revealed nothing.

I don't think there's any substitute for doing it manually. Google the cities, then pull out your map and get to work. It shouldn't take longer than an hour. Edit: I didn't realize this was a standardized problem.

Writing a program to solve the problem would take 5 or 10 minutes for an average programmer. However, the amount of time the program would need to run is, well, a LONG LONG LONG time.

My Garmin could probably solve this for you. Edit: probably not.

Someone needs to write a distributed computing program to solve this IMO.

How difficult can it be?

Imagine an UBERcomputer (a giant computing device)...
- With as many processors as electrons in the universe...
- Each processor having the power of today's supercomputers...
- Each processor working for the lifetime of the universe...

quantity	*value (conservative estimate)*
electrons in universe	10^{79}
supercomputer instructions per second	10^{13}
age of universe in seconds	10^{17}

Q. Could the UBERcomputer solve the TSP for 100 cities with the brute force algorithm?

A. Not even close. $100! > 10^{157} >> 10^{79} 10^{13} 10^{17} = 10^{109}$ ← Would need 10^{48} UBERcomputers

Lesson. Exponential growth dwarfs technological change.

Reasonable questions about algorithms

Q. Which algorithms are useful in practice?

Model of computation
- Running time: Number of steps as a function of input size N.
- Poly-time: Running time less than aN^b for some constants a and b.
- Definitely not poly-time: Running time $\sim c^N$ for any constant $c > 1$.
- Specific computer generally not relevant (simulation uses only a polynomial factor).

"Extended Church-Turing thesis" (stay tuned)

Def (in the context of this lecture). An algorithm is efficient if it is poly-time for *all* inputs.

outside this lecture: "guaranteed polynomial time" or just "poly-time"

Q. Can we find efficient algorithms for the practical problems that we face?

Reasonable questions about problems

Q. Which problems can we solve in practice?

A. Those for which we know efficient (guaranteed poly-time) algorithms.

Definition. A problem is intractable if no efficient algorithm exists to solve it.

Q. Is there an easy way to tell whether a problem is intractable?

A. Good question! Focus of today's lecture.

Existence of a faster algorithm like mergesort is not relevant to this discussion

Example 1: Sorting. Not intractable. (Insertion sort takes time proportional to N^2.)

Example 2: TSP. ??? (No efficient algorithm known, but no proof that none exists.)

Four fundamental problems

	Example of an instance	A solution
LSOLVE • Solve simultaneous linear equations. • Variables are real numbers.	$x_1 + x_2 = 1$ $2x_0 + 4x_1 - 2x_2 = 0.5$ $3x_1 + 15x_2 = 9$	$x_0 = -.25$ $x_1 = .5$ $x_2 = .5$
LP • Solve simultaneous linear *inequalities*. • Variables are real numbers.	$48x_0 + 16x_1 + 119x_2 \leq 88$ $5x_0 + 4x_1 + 35x_2 \geq 13$ $15x_0 \quad 4x_1 + 20x_2 \geq 23$ $x_0, x_1, x_2 \geq 0$	$x_0 = 1$ $x_1 = 1$ $x_2 = 0.2$
ILP • Solve simultaneous linear inequalities. • Variables are 0 or 1.	$x_1 + x_2 \geq 1$ $x_0 + x_2 \geq 1$ $x_0 + x_1 + x_2 \leq 2$	$x_0 = 0$ $x_1 = 1$ $x_2 = 1$
SAT • Solve simultaneous *boolean sums*. • Variables are *true* or *false*.	$\neg x_1 \vee x_2 = true$ $\neg x_0 \vee \neg x_1 \vee \neg x_2 = true$ $x_1 \vee \neg x_2 = true$	$x_0 = false$ $x_1 = true$ $x_2 = true$

Reasonable questions

LSOLVE, LP, ILP, and SAT are all important problem-solving models with countless practical applications.

Q. Do we have efficient algorithms for solving them?

A. Difficult to discern, despite similarities (!)

✓ LSOLVE. Yes. appropriate version of Gaussian elimination
✓ LP. Yes. Ellipsoid algorithm (*tour de force* solution)
? IP. No polynomial-time algorithm known.
? SAT. No polynomial-time algorithm known.

Q. Can we find efficient algorithms for IP and SAT?

Q. Can we prove that no such algorithms exist?

LSOLVE
• Solve simultaneous linear equations.
• Variables are real numbers.

LP
• Solve simultaneous linear *inequalities*.
• Variables are real numbers.

ILP
• Solve simultaneous linear inequalities.
• Variables are 0 or 1.

SAT
• Solve simultaneous *boolean sums*.
• Variables are *true* or *false*.

Intractability

Definition. An algorithm is efficient if it is poly-time for all inputs.

Definition. A problem is intractable if no efficient algorithm exists to solve it.

Definition. A problem is tractable if it solvable by an efficient algorithm.

Turing taught us something fundamental about computation by
• Identifying a problem that we might want to solve.
• Showing that it is not possible to solve it.
A reasonable question: Can we do something similar for intractability?

decidable : undecidable :: tractable : intractable

Q. We do not know efficient algorithms for a large class of important problems. Can we prove one of them to be intractable?

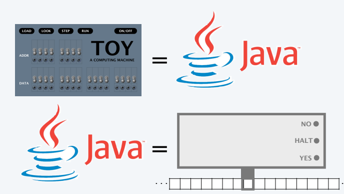
Another profound connection to the real world

Extended Church-Turing thesis.
Resources used by all reasonable machines are within a polynomial factor of one another.

Remarks
• A thesis, not a theorem.
• *Not* subject to proof.
• *Is* subject to falsification.

New model of computation or new physical process?
• Use *simulation* to prove polynomial bound.
• Example: TOY simulator in Java (see TOY lectures).
• Example: Java compiler in TOY.

Implications
• Validates exponential/polynomial divide.
• Enables rigorous study of efficiency.



COMPUTER SCIENCE
SEDGEWICK / WAYNE

Image sources

```
http://en.wikipedia.org/wiki/David_Hilbert#/media/File:Hilbert.jpg
http://www.nobelprize.org/nobel_prizes/economic-sciences/laureates/1994/nash-bio.html
http://en.wikipedia.org/wiki/Michael_O._Rabin#/media/File:Michael_O._Rabin.jpg
http://www.google.com/imgres?imgurl=http://www.cmu.edu/news/image-archive/Dana_Scott.jpg
http://en.wikipedia.org/wiki/Richard_M._Karp#/media/File:Karp_mg_7725-b.cr2.jpg
http://en.wikipedia.org/wiki/Stephen_Cook#/media/File:Prof.Cook.jpg
```


COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, MACHINES, and THEORY

16. Intractability

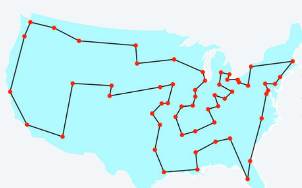
- Reasonable questions
- **P and NP**
- Poly-time reductions
- NP-completeness
- Living with intractability

Search problems

Search problem. Any problem for which an efficient algorithm exists to certify solutions.

Example: TSP.

Problem instance:
Set of cities, pairwise distances, and threshold *M*.



Certify solution by adding distances and checking that the total is less than *M*

Solution: Permutation of the cities.

NP

Definition. **NP** is the class of all search problems.

problem	*description*	*instance I*	*solution S*	*algorithm*
TSP (S, M)	Find a tour of cities in S of length < M			Add up distances and check that the total is less than M
ILP (A, b)	Find a binary vector x that satisfies $Ax \le b$	$x_1 + x_2 \ge 1$ $x_0 + x_2 \ge 1$ $x_0 + x_1 + x_2 \le 2$	$x_0 = 0$ $x_1 = 1$ $x_2 = 1$	plug in values and check each equation
SAT (A, b)	Find a boolean vector x that satisfies $Ax = b$	$\neg x_1 \vee x_2 = \mathit{true}$ $\neg x_0 \vee \neg x_1 \vee \neg x_2 = \mathit{true}$ $x_1 \vee \neg x_2 = \mathit{true}$	$x_0 = \mathit{false}$ $x_1 = \mathit{true}$ $x_2 = \mathit{true}$	plug in values and check each equation
FACTOR (M)	Find a nontrivial factor of the integer M	147573952589676412927	193707721	long division

Significance. Problems that scientists, engineers, and applications programmers *aspire* to solve.

Brute force search

Brute-force search. Given a search problem, find a solution by *checking all possibilities.*

problem	*description*	*N (size)*	*number of possibilities*
TSP (S, M)	Find a tour of cities in S of length < M	number of cities	$N!$
ILP (A, b)	Find a binary vector x that satisfies $Ax \le b$	number of variables	2^N
SAT (Φ, b)	Find a boolean vector x that satisfies $Ax = b$	number of variables	2^N
FACTOR (x)	Find a nontrivial factor of the integer M	number of digits in M	$10^{\sqrt{N}}$

Challenge. Brute-force search is easy to implement, but *not* efficient.

P

Definition. **P** is the class of all tractable search problems. ← solvable by an efficient (guaranteed poly-time) algorithm

problem	*description*	*efficient algorithm*
SORT (S)	Find a permutation that puts the items in S in order	Insertion sort, Mergesort
3-SUM (S)	Find a triple in S that sums to 0	Triple loop
LSOLVE (A, b)	Find a vector x that satisfies $Ax = b$	Gaussian elimination*
LP (A, b)	Find a vector x that satisfies $Ax \le b$	Ellipsoid

Significance. Problems that scientists, engineers and applications programmers *do* solve.

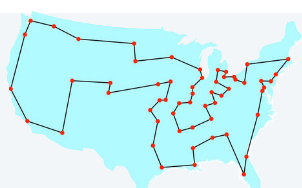
Note. All of these problems are also in **NP**.

Types of problems

Search problem. *Find* a solution.
Decision problem. Does there *exist* a solution?
Optimization problem. Find the *best* solution.

Some problems are more naturally formulated in one regime than another.

Example: TSP is usually formulated as an optimization problem.



"Find the shortest tour connecting all the cities."

The regimes are not technically equivalent, but conclusions that we draw apply to all three.

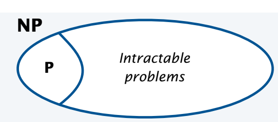
Note. Classic definitions of **P** and **NP** are in terms of decision problems.

The central question

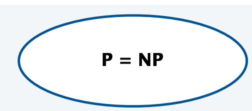
NP. Class of all search problems, some of which seem solvable only by brute force.
P. Class of search problems solvable in poly-time.

The question: Is **P = NP** ?

P ≠ NP
- Intractable search problems exist.
- Brute force search may be the best we can do for some problems.



P = NP
- *All* search problems are tractable.
- Efficient algorithms exist for IP, SAT, FACTOR ... *all* problems in **NP**.



Frustrating situation. Researchers believe that **P ≠ NP** but *no one has been able to prove it* (!!)

Nondeterminism: another way to view the situation

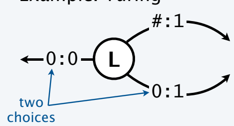
A nondeterministic machine can choose among multiple options at each step *and can guess the option that leads to the solution.*

Example: Java.

```
either x[0] = 0; or x[0] = 1;
either x[1] = 0; or x[1] = 1;
either x[2] = 0; or x[2] = 1;
```

← solves ILP

Example: Turing



Seems like a fantasy, but...

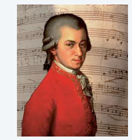
P ≠ NP
- Intractable search problems exist.
- Nondeterministic machines would admit efficient algorithms.

P = NP
- No intractable search problems exist.
- Nondeterministic machines would be of no help!

Frustrating situation. No one has been able to *prove* that nondeterminism would help (!!)

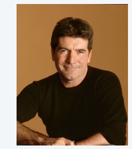
Creativity: another way to view the situation

Creative genius versus ordinary appreciation of creativity.



Creative genius

Examples
- Mozart composes a piece of music; the audience appreciates it.
- Wiles proves a deep theorem; a colleague checks it.
- Boeing designs an efficient airfoil; a simulator verifies it.
- Einstein proposes a theory; an experimentalist validates it.



Ordinary appreciation

Computational analog. **P** vs **NP**.

Frustrating situation. No one has been able to *prove* that creating a solution to a problem is more difficult than checking that it is correct.

COMPUTER SCIENCE
SEDGEWICK / WAYNE

Image sources

http://www.imdb.com/name/nm1101562/
http://en.wikipedia.org/wiki/Wolfgang_Amadeus_Mozart#/media/File:Wolfgang-amadeus-mozart_1.jpg

CS.16.B.Intractability.PandNP

COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART II: ALGORITHMS, MACHINES, and THEORY

19. Intractability

- Reasonable questions
- P and NP
- **Poly-time reductions**
- NP-completeness
- Living with intractability

CS.16.C.Intractability.Reductions

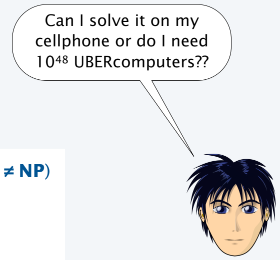
Classifying problems

Q. Which problems are in **P**?
A. The ones that we're solving with efficient algorithms.

Q. Which problems are intractable (in **NP** but not in **P**)?
A. Difficult to know (no one has found even *one* such problem).

Possible starting point: Assume that SAT is intractable (and hence **P ≠ NP**)
- Brute-force algorithm finds solution for any SAT instance.
- No known efficient algorithm does so.

A reasonable assumption.



Next. Proving relationships among problems.

Q. If **P ≠ NP** and SAT is intractable, which other problems are intractable?

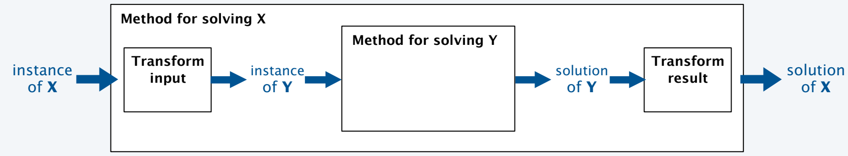
Poly-time reduction

Definition. Problem **X** *poly-time reduces to* problem **Y** if you can use an efficient solution to **Y** to develop an efficient solution to **X**.

X→Y

Typical reduction: Given an efficient solution to **Y**, solve **X** by
- Using an efficient method to transform the instance of **X** to an instance of **Y**.
- Calling the efficient method that solves **Y**.
- Using an efficient method to transform the solution of **Y** to an solution of **X**.

Similar to using a library method in modular programming.

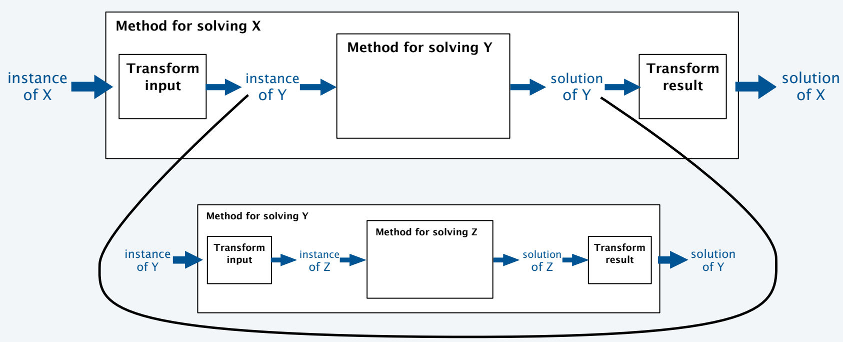


Note. Many ways to extend. (Example: Use a polynomial number of instances of **Y**.)

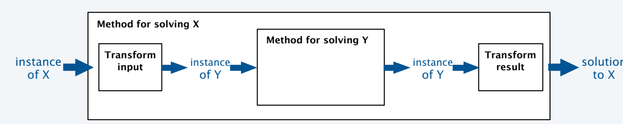
Key point: poly-time reduction is transitive

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

If **X→Y** and **Y→Z** then **X→Z**



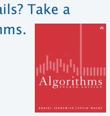
Two ways to exploit reduction



To design an algorithm to solve a new problem **X**
- Find a problem **Y** with a known efficient algorithm that solves it.
- Poly-time reduce **X** to **Y**.

The efficient algorithm for **Y** gives an efficient algorithm for **X**.

Not emphasized in this lecture. Interested in details? Take a course in algorithms.



To establish intractability of a new problem **Y** (assuming SAT is intractable)
- Find a problem **X** with a known poly-time reduction from SAT.
- Poly-time reduce **X** to **Y**.

An efficient algorithm for **Y** would imply an efficient algorithm for **X** (*and* SAT).

Critical tool for this lecture.

Easiest option: Just reduce SAT to Y (next)

Example: SAT poly-time reduces to ILP

SAT
- Solve simultaneous boolean sums.
- Variables are *true* or *false*

ILP
- Solve simultaneous linear inequalities.
- Variables are 0 or 1.

$$\begin{aligned} \neg x_0 \vee x_1 \vee x_2 &= \textit{true} \\ x_0 \vee \neg x_1 \vee x_2 &= \textit{true} \\ \neg x_0 \vee \neg x_1 \vee \neg x_2 &= \textit{true} \\ \neg x_0 \vee \neg x_1 \vee x_3 &= \textit{true} \end{aligned}$$

An instance of SAT

$x_0 = \textit{false}$
$x_1 = \textit{true}$
$x_2 = \textit{true}$
$x_3 = \textit{false}$

A solution

$$\begin{aligned} (1 - t_0) + t_1 + t_2 &\geq 1 \\ t_0 + (1 - t_1) + t_2 &\geq 1 \\ (1 - t_0) + (1 - t_1) + (1 - t_2) &\geq 1 \\ (1 - t_0) + (1 - t_1) + t_3 &\geq 1 \end{aligned}$$

$t_i = 0$ iff $x_i = \textit{false}$
$t_i = 1$ iff $x_i = \textit{true}$

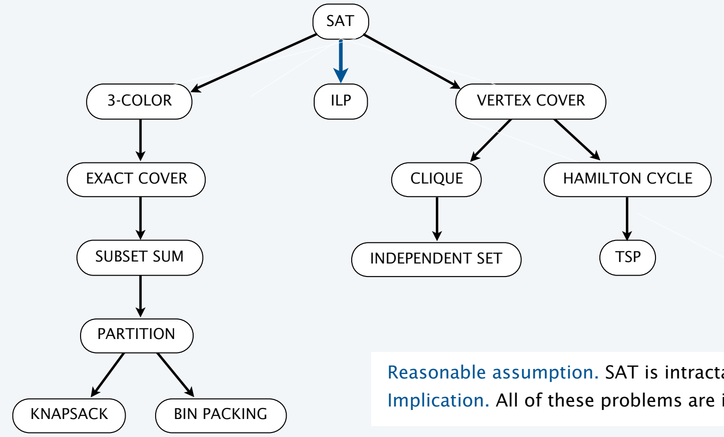
Poly-time reduction to an instance of ILP

$t_0 = 0$
$t_1 = 1$
$t_2 = 1$
$t_3 = 0$

A solution

Implication. If SAT is intractable, so is ILP.

More poly-time reductions from SAT



Dick Karp
1985 Turing Award

Reasonable assumption. SAT is intractable.
Implication. All of these problems are intractable.

Still more poly-time reductions from SAT

field of study	*typical problem known to be intractable if SAT is intractable*
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
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.

Reasonable assumption. SAT is intractable.
Implication. All of these problems are intractable.

COMPUTER SCIENCE
SEDGEWICK / WAYNE

CS.16.C.Intractability.Reductions

COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART II: ALGORITHMS, MACHINES, and THEORY

19. Intractability

- Reasonable questions
- P and NP
- Poly-time reductions
- **NP-completeness**
- Living with intractability

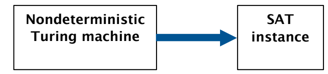
CS.16.D.Intractability.NPcomplete

NP-completeness

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

Theorem (Cook; Levin, 1971). *SAT is NP-complete.*

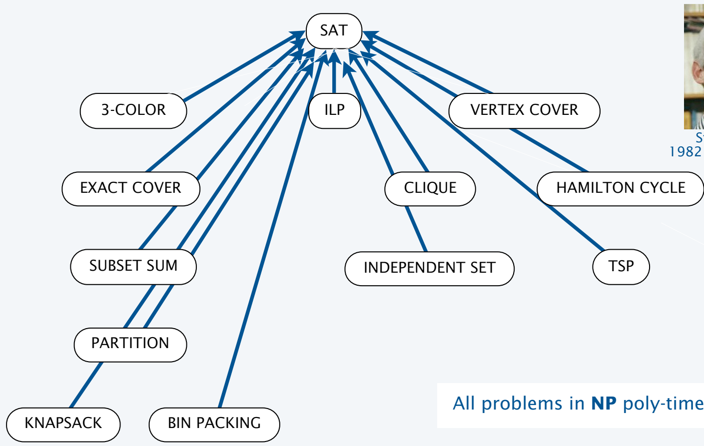
Extremely brief proof sketch
- Convert non-deterministic TM notation to SAT notation.
- An efficient solution to SAT gives an efficient solution to any problem in NP.

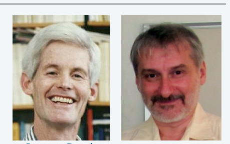


Corollary. SAT is tractable if and only if **P** = **NP**.

Equivalent. Assuming that SAT is intractable is the same as assuming that **P** ≠ **NP**.

Cook-Levin theorem



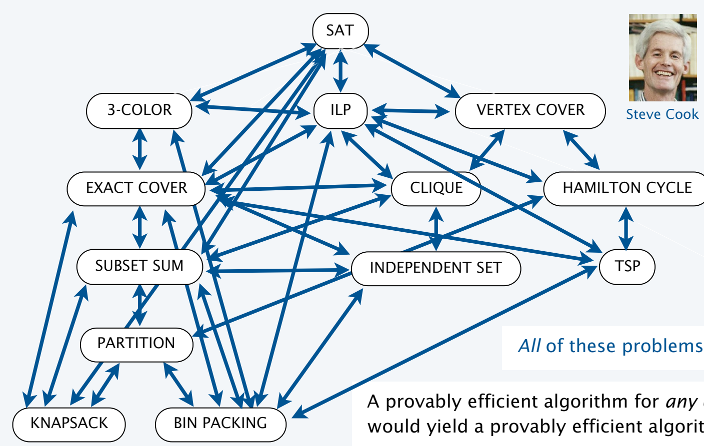


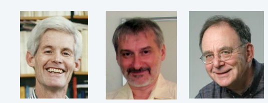
Steve Cook
1982 Turing Award

Leonid Levin

All problems in **NP** poly-time reduce to SAT.

Karp + Cook-Levin





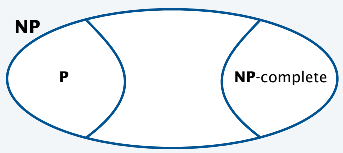
Steve Cook Leonid Levin Dick Karp

All of these problems are NP-complete.

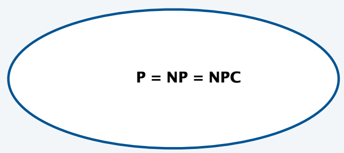
A provably efficient algorithm for *any one* of them would yield a provably efficient algorithm for *all* of them

Two possible universes

P ≠ NP
- Intractable search problems exist.
- Nondeterminism would help.
- Computing an answer is more difficult than correctly guessing it.
- Can prove a problem to be intractable by poly-time reduction from an **NP**-complete problem.



P = NP
- No intractable search problems exist.
- Nondeterminism is no help.
- Finding an answer is just as easy as correctly guessing an answer.
- Guaranteed poly-time algorithms exist for all problems in **NP**.



Frustrating situation. No progress on resolving the question despite 40+ years of research.

Summary

NP. Class of all search problems, some of which seem solvable only by brute force.

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

NP-complete. "Hardest" problems in NP.

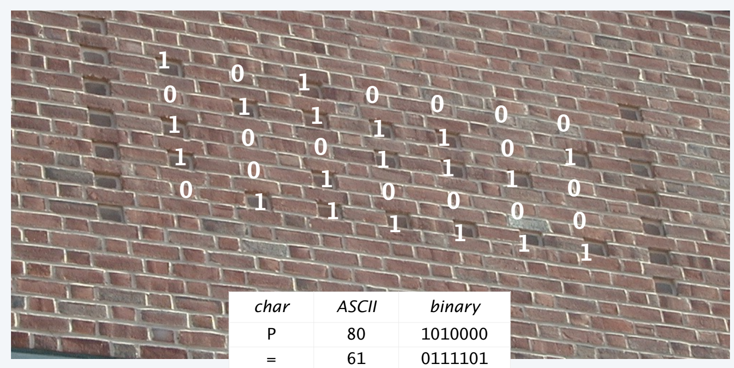
Intractable. Search problems not in P (if **P ≠ NP**).

TSP, SAT, ILP, and thousands of other problems are **NP**-complete.

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** ≠ **NP** and that such problems are intractable.
- Identify these situations and proceed accordingly.

Princeton CS building, west wall

Princeton CS building, west wall (closeup)



char	*ASCII*	*binary*
P	80	1010000
=	61	0111101
N	78	1001110
P	80	1010000
?	63	0111111

COMPUTER SCIENCE
SEDGEWICK / WAYNE

CS.16.D.Intractability.NPcomplete

COMPUTER SCIENCE
SEDGEWICK / WAYNE
PART II: ALGORITHMS, MACHINES, and THEORY

16. Intractability

- Reasonable questions
- P and NP
- Poly-time reductions
- NP-completeness
- **Living with intractability**

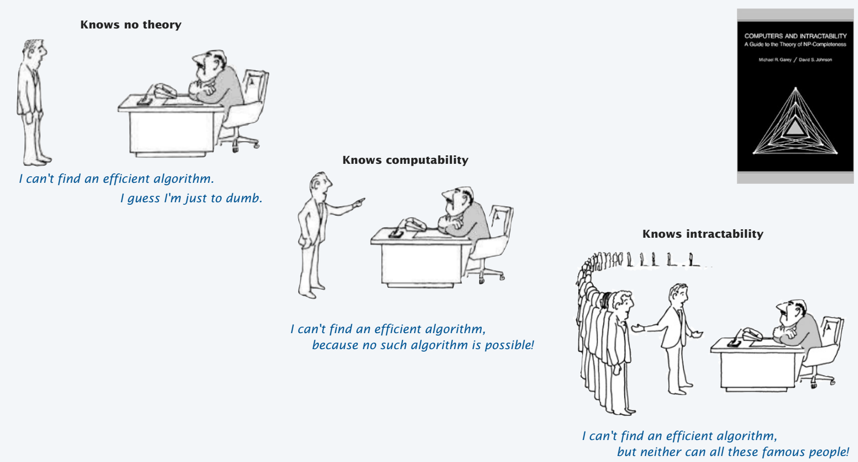
CS.16.E.Intractability.Living

Living with intractability

When you encounter an NP-complete problem
- It is safe to assume that it is intractable.
- What to do?

Four successful approaches
- Don't try to solve intractable problems.
- Try to solve real-world problem instances.
- Look for approximate solutions (not discussed in this lecture).
- Exploit intractability.

Living with intractability: Don't try to solve intractable problems



Understanding intractability: An example from statistical physics

1926: Ising introduces a mathematical model for ferromagnetism.

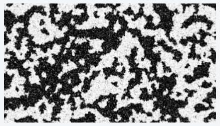
1930s: Closed form solution is a holy grail of statistical mechanics.

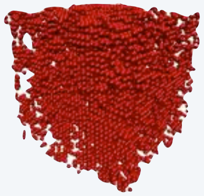
1944: Onsager finds closed form solution to 2D version in tour de force.

1950s: Feynman and others seek closed form solution to 3D version.

2000: Istrail shows that 3D-ISING is **NP**-complete.

Bottom line. Search for a closed formula seems futile.





Living with intractability: look for solutions to real-world problem instances

Observations
- Worst-case inputs may not occur for practical problems.
- Instances that do occur in practice may be easier to solve.

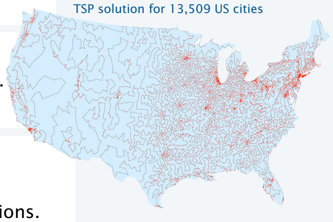
Reasonable approach: relax the condition of *guaranteed* poly-time algorithms.

SAT
- *Chaff* solves real-world instances with 10,000+ variables.
- Princeton senior independent work (!) in 2000.

TSP
- *Concorde* routinely solves large real-world instances.
- 85,900-city instance solved in 2006.

ILP
- *CPLEX* routinely solves large real-world instances.
- Routinely used in scientific and commercial applications.

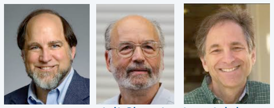
TSP solution for 13,509 US cities



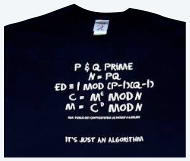
Exploiting intractability: RSA cryptosystem

Modern cryptography applications
- Electronic banking.
- Credit card transactions with online merchants.
- Secure communications.
- [very long list]

RSA cryptosystem exploits intractability
- To use: Multiply/divide two N-digit integers (easy).
- To break: Factor a $2N$-digit integer (intractable?).



Ron Rivest Adi Shamir Len Adelman

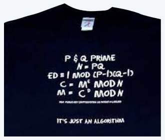


761838257287 * 193707721 $\xrightarrow{\text{Multiply (easy)}}$ 147573952589676412927

147573952589676412927 $\xrightarrow{\text{Factor (difficult)}}$ 761838257287 * 193707721

Exploiting intractability: RSA cryptosystem

RSA cryptosystem exploits intractability
- To use: Multiply/divide two N-digit integers (easy).
- To break: Solve FACTOR for a $2N$-digit integer (difficult).



Example: Factor this 212-digit integer

74037563479561712828046796097429573142593188889231289
08493623263897276503402826627689199641962511784399589
43305021275853701189680982867331732731089309005525051
16877063299072396380786710086096962537934650563796359

Q. Would an efficient algorithm for FACTOR prove that **P = NP** ?

A. Unknown. It is in **NP**, but no reduction from SAT is known. ← equivalent statement: "FACTOR is not known to be 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.

Fame and fortune through intractability

Factor this 212-digit integer

74037563479561712828046796097429573142593188889231289084936232638972765034028266276891996419625117843995894330502127585370118968098286733173273108930900552505116877063299072396380786710086096962537934650563796359

$30,000 prize claimed in July, 2012

Create an e-commerce company based on the difficulty of factoring

The Security Division of EMC

RSA sold to EMC for $2.1 billion in 2006

Resolve **P vs. NP**

$1 million prize unclaimed since 2000

probably another $1 million for Turing award

plus untold riches for breaking e-commerce if P=NP

or... sell T-shirts

A final thought

Example: Factor this 212-digit integer

74037563479561712828046796097429573142593188889231289084936232638972765034028266276891996419625117843995894330502127585370118968098286733173273108930900552505116877063299072396380786710086096962537934650563796359

Q. Would an efficient algorithm for FACTOR prove that **P = NP** ?
A. Unknown. It is in **NP**, but no reduction from SAT is known.

Q. Is it safe to assume that FACTOR is intractable?
A. Maybe, but not as safe an assumption as for an **NP**-complete problem.

Q. What else might go wrong?

Theorem (Shor, 1994). An N-bit integer can be factored in N^3 steps on a *quantum computer*.

Q. Do we still believe in the Extended Church-Turing thesis? ← Resources used by all reasonable machines are within a polynomial factor of one another.

A. Whether we do or not, intractability demands understanding.

Image sources

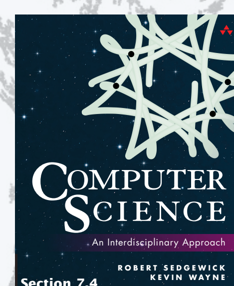
http://people.csail.mit.edu/rivest/

http://www.google.com/imgres?imgurl=http://upload.wikimedia.org/wikipedia/commons/1/1e/Adi_Shamir_at_TU_Darmstadt_(2013).jpg

http://www.google.com/imgres?imgurl=http://upload.wikimedia.org/wikipedia/commons/a/af/Len-mankin-pic.jpg

COMPUTER SCIENCE
SEDGEWICK / WAYNE

PART II: ALGORITHMS, MACHINES, and THEORY



http://introcs.cs.princeton.edu

16. Intractability