pollev.com/cse417

fill out whenever

Wrapping Up Reductions

CSE 417 Winter 21
Lecture 24

Announcements

We're (probably) going to get through the content I've planned a lecture-or-two before the end of the quarter.

I've listed a couple of possible topics on this Ed thread

"Heart" the ones you like.

Logistics for the final

Going up on the webpage soon (hopefully tonight)

Along with practice material

Big thing you need to know:

Planning to release final 11:59 PM Sunday of finals week, due 11:59 PM Wednesday of Finals week

You'd have the full 3 days to work if you want to (but goal when writing it will be something that could be done in a "normal" 2 hour exam)

If you have some reason that you really need to offset that by a day, let me know as soon as possible (email or private Ed post).

3-Coloring $\leq$ 3-SAT

Variables: is this vertex red? Blue? Green? (can't have just one variable, let's just have three).

Constraints?

If (u, v) is an edge, then u and v are different colors.

u gets exactly one color.

3-Coloring ≤ 3-SAT

Variables: is this vertex red? Blue? Green? (can't have just one variable, let's just have three).

$\underrightarrow{x_{u,r}}, x_{u,b}, x_{u,g}$

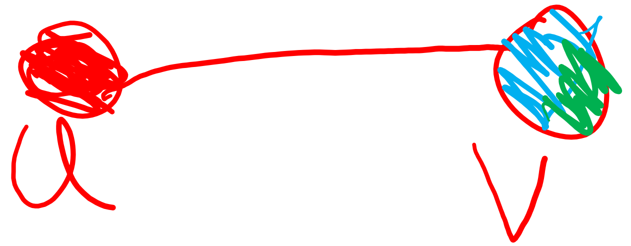
Constraints?

If (u, v) is an edge, then u and v are different colors.

u gets exactly one color.

These are going to take a bit of work:

Edge Requirements



We need to make sure the edges are different colors.

As an example

If u is red, and (u, v) is an edge, then v is blue OR v is green.

$x_{u,r} == False \,||\, x_{v,b} == True \,||\, x_{v,g} == True$

Law of implication: "if p then q" is equivalent to $!p||q$.

Edge Constraints

All combinations constraints:

English – for each edge (u, v)	SAT
If u is red, then v is blue or green	$x_{u,r} == False \mid\mid x_{v,b} == \text{True} \mid\mid x_{v,g} == True$
If u is blue, then v is red or green	$x_{u,b} == False \mid\mid x_{v,r} == \text{True} \mid\mid x_{v,g} == True$
If u is green, then v is red or blue	$x_{u,g} == False \mid\mid x_{v,r} == \text{True} \mid\mid x_{v,b} == True$
If v is red, then u is blue or green	$x_{v,r} == False \mid\mid x_{u,b} == \text{True} \mid\mid x_{u,g} == True$
If v is blue, then u is red or green	$x_{v,b} == False \mid\mid x_{u,r} == \text{True} \mid\mid x_{u,g} == True$
If v is green, then u is red or blue	$x_{v,g} == False \mid\mid x_{u,r} == \text{True} \mid\mid x_{u,b} == True$

Some of these aren't strictly necessary (are implied by the others) but better safe than sorry.

Are those constraints enough?

$x_{\_\_} = F$

Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

Suppose we used those constraints, ran the 3-SAT solver on these constraints, and just return what it says.

Are we done? If this reduction is correct, explain to each other why!
If it's not correct explain why not.

English – for each edge (u, v)	SAT
If u is red, then v is blue or green	$x_{u,r} == False \mid\mid x_{v,b} == \text{True} \mid\mid x_{v,g} == True$
If u is blue, then v is red or green	$x_{u,b} == False \mid\mid x_{v,r} == \text{True} \mid\mid x_{v,g} == True$
If u is green, then v is red or blue	$x_{u,g} == False \mid\mid x_{v,r} == \text{True} \mid\mid x_{v,b} == True$
If v is red, then u is blue or green	$x_{v,r} == False \mid\mid x_{u,b} == \text{True} \mid\mid x_{u,g} == True$
If v is blue, then u is red or green	$x_{v,b} == False \mid\mid x_{u,r} == \text{True} \mid\mid x_{u,g} == True$
If v is green, then u is red or blue	$x_{v,g} == False \mid\mid x_{u,r} == \text{True} \mid\mid x_{u,b} == True$

Are those constraints enough?

Suppose we used those constraints, ran the 3-SAT solver on what we got.

If the graph is 3-colorable, then the 3-SAT instance has a solution (pick your favorite coloring and set the variables to match that coloring).

If the graph is not 3-colorable

The 3-SAT solver will still say there’s a solution for this instance. Just set every variable to false!

Consistency Constraints

Reductions often need extra constraints/structures.

When you say “I want this variable to mean X” you really need to force the variable to mean X.

So if you want a coloring, you need to make sure even “well, yeah of course that’s what I meant” requirements are explicit.

What are we missing? Every vertex needs exactly one color.

Consistency

More constraints:

English – for each vertex	SAT
If u is red, then u cannot be blue	$x_{u,r} == False \mid\mid x_{u,b} ==$ False
If u is red, then u cannot be green	$x_{u,r} == False \mid\mid x_{u,g} ==$ False
If u is blue, then u cannot be red	$x_{u,b} == False \mid\mid x_{u,r} ==$ False
If u is blue, then u cannot be green	$x_{u,b} == False \mid\mid x_{u,g} ==$ False
If u is green, then u cannot be red	$x_{u,g} == False \mid\mid x_{u,r} ==$ False
If u is green, then u cannot be blue	$x_{u,g} == False \mid\mid x_{u,b} ==$ False
u gets a color!	$x_{u,r} == True \mid\mid x_{u,g} == True \mid\mid x_{u,b} == True$

From 2 to 3.

Hang on! Is this allowed in 3-SAT?

$x_{u,r} == False \,||\, \underbrace{x_{u,b} == \text{False}}$ || $\underbrace{x_{u,b} == \text{False}}$

The definition said 3 items each…

A trick to fix it. Make two copies, or in a $\underbrace{\text{dummy variable } d}$ being True in one and false in the other.

$x_{u,r} == False \,||\, x_{u,b} == \text{False}$ ~~$|| d == True$~~ ✓

$\underbrace{x_{u,r} == False \,||\, x_{u,b} == \text{False}}$ ~~$|| d == False$~~

d will make one of the two true. The other copy is satisfied if and only if the original one was.

Reduction

Given a graph G, we make the following 3-SAT instance

Variables: $x_{u,r}, x_{u,g}, x_{u,b}$ for each vertex u

Constraints: As described on the last few slides.

Run a 3SATSolver.

Return whatever it returns.

Running Time?

We need n^{+1} variables and $\underbrace{6m + 13n}$ constraints.

Making them is mechanical, definitely polynomial time.

Correctness

Our correctness proofs are usually:

Certificate for 3-coloring becomes a certificate for 3-SAT

The only certificates for 3-SAT come from certificates for 3-coloring

Let's start with

If G is 3-colorable, then the reduction says YES.

Correctness

If G is 3-colorable, then the reduction says YES.

If G is 3-colorable, then there is a 3-coloring. From any 3-coloring, set

$x_{u,r}$ to be true if u is red and false otherwise.

$x_{u,g}$ to be true if u is green and false otherwise.

$x_{u,b}$ to be true if u is blue and false otherwise.

The constraints are satisfied (for the reasons listed on the prior slides)

So the 3-SAT algorithm must say the constraints are satisfiable, and the reduction returns true!

Correctness

If the reduction returns YES, then G was 3-colorable.

If the reduction returns YES, then the 3-SAT algorithm returned YES, so the 3-SAT instance had a satisfying assignment.

We can convert the variables to a coloring:

For every u, exactly one of $x_{u,r}, x_{u,g}, x_{u,b}$ is true. We have a constraint requiring at least one, and constraints preventing more than one variable for the same vertex being true.

Color the vertices the associated colors. Since every vertex is colored, at least one of the constraints is active for each edge, so we have a valid coloring.

Two Last Thoughts

I didn't care at all about the efficiency of our reductions (other than them being polynomial time).

Why?

Well if it's worse than polynomial it doesn't count! (Goal is to get a polynomial-time algorithm for problem A, if we write one for problem B).

But…do we expect to ever run these reductions?

Not really!…If we ever find an efficient 3-SAT algorithm we can always go back and make them efficient.

Two Last Thoughts

Vertex Cover is NP-complete (you can do a reduction from independent set. It's good practice!)

But we wrote a polynomial time algorithm for vertex cover didn't we? We wrote two– a DP one and an LP one. What's going on?

The algorithms we saw only handled special cases – Vertex cover on trees or vertex cover on bipartite graphs. We didn't prove $P = NP$. We carved off part of the problem that was easy and solved that (solved only the "easy" instances).

Why are P and NP interesting?

Why do we care?

We’ve seen a few NP-complete problems.

3-SAT, 3-coloring, Hamiltonian Path/Cycle, Independent Set.

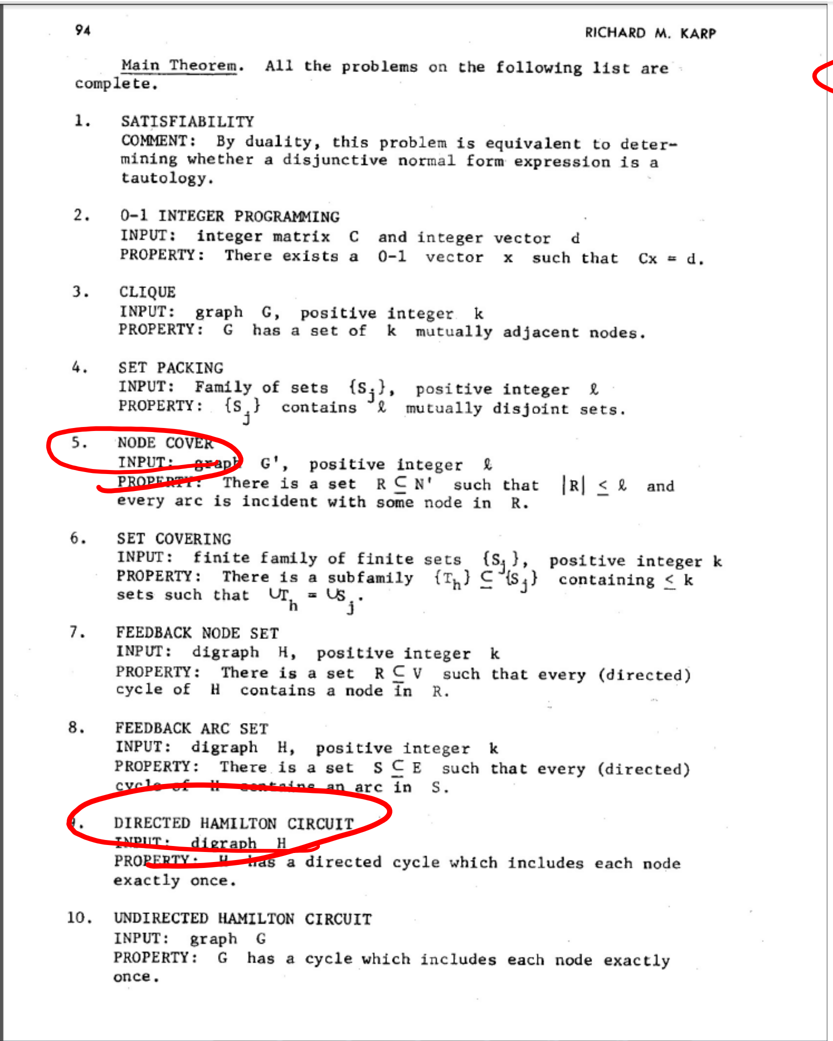
But why should we care about those few?

Just memorize them and avoid them, right?

It’s more than just a few…

NP-Complete Problems

But Wait! There's more!



94 RICHARD M. KARP

Main Theorem. All the problems on the following list are complete.

1. SATISFIABILITY
   COMMENT: By duality, this problem is equivalent to determining whether a disjunctive normal form expression is a tautology.

2. 0-1 INTEGER PROGRAMMING
   INPUT: integer matrix C and integer vector d
   PROPERTY: There exists a 0-1 vector x such that $Cx = d$.

3. CLIQUE
   INPUT: graph G, positive integer k
   PROPERTY: G has a set of k mutually adjacent nodes.

4. SET PACKING
   INPUT: Family of sets $\{S_j\}$, positive integer ℓ
   PROPERTY: $\{S_j\}$ contains ℓ mutually disjoint sets.

5. NODE COVER
   INPUT: graph G', positive integer ℓ
   PROPERTY: There is a set $R \subseteq N'$ such that $|R| \leq \ell$ and every arc is incident with some node in R.

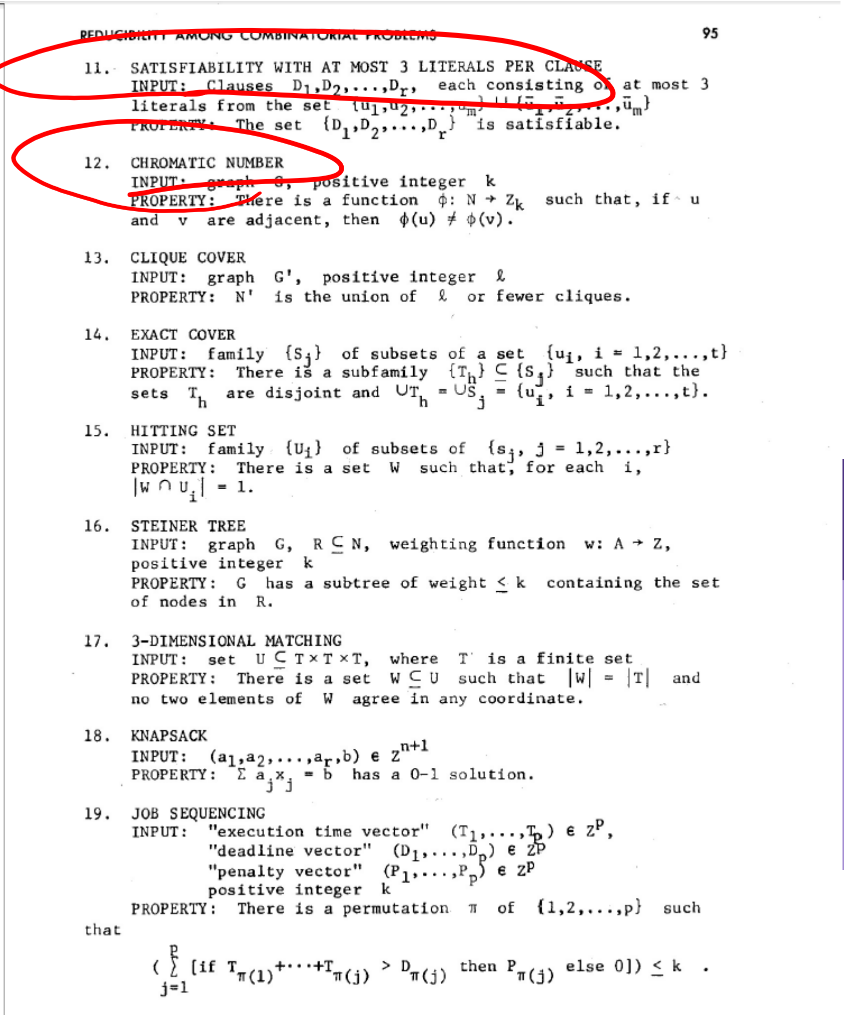
6. SET COVERING
   INPUT: finite family of finite sets $\{S_j\}$, positive integer k
   PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$.

7. FEEDBACK NODE SET
   INPUT: digraph H, positive integer k
   PROPERTY: There is a set $R \subseteq V$ such that every (directed) cycle of H contains a node in R.

8. FEEDBACK ARC SET
   INPUT: digraph H, positive integer k
   PROPERTY: There is a set $S \subseteq E$ such that every (directed) cycle of H contains an arc in S.

9. DIRECTED HAMILTON CIRCUIT
   INPUT: digraph H
   PROPERTY: H has a directed cycle which includes each node exactly once.

10. UNDIRECTED HAMILTON CIRCUIT
    INPUT: graph G
    PROPERTY: G has a cycle which includes each node exactly once.



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS 95

11. SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
    INPUT: Clauses $D_1, D_2, \ldots, D_r$, each consisting of at most 3 literals from the set $\{u_1, u_2, \ldots, u_m\} \cup \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_m\}$
    PROPERTY: The set $\{D_1, D_2, \ldots, D_r\}$ is satisfiable.

12. CHROMATIC NUMBER
    INPUT: graph G, positive integer k
    PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

13. CLIQUE COVER
    INPUT: graph G', positive integer ℓ
    PROPERTY: N' is the union of ℓ or fewer cliques.

14. EXACT COVER
    INPUT: family $\{S_j\}$ of subsets of a set $\{u_i, i = 1,2,\ldots,t\}$
    PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i, i = 1,2,\ldots,t\}$.

15. HITTING SET
    INPUT: family $\{U_i\}$ of subsets of $\{s_j, j = 1,2,\ldots,r\}$
    PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

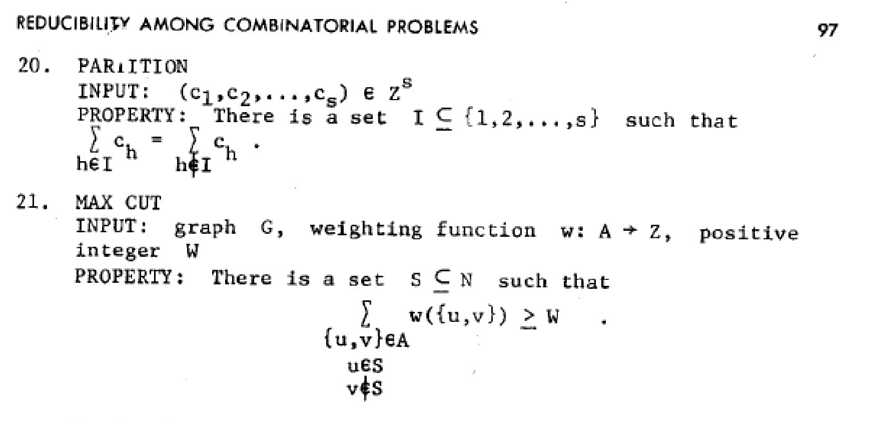
16. STEINER TREE
    INPUT: graph G, $R \subseteq N$, weighting function $w: A \to Z$, positive integer k
    PROPERTY: G has a subtree of weight $\leq k$ containing the set of nodes in R.

17. 3-DIMENSIONAL MATCHING
    INPUT: set $U \subseteq T \times T \times T$, where T is a finite set
    PROPERTY: There is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate.

18. KNAPSACK
    INPUT: $(a_1, a_2, \ldots, a_r, b) \in Z^{n+1}$
    PROPERTY: $\Sigma a_j x_j = b$ has a 0-1 solution.

19. JOB SEQUENCING
    INPUT: "execution time vector" $(T_1, \ldots, T_p) \in Z^p$,
    "deadline vector" $(D_1, \ldots, D_p) \in Z^p$
    "penalty vector" $(P_1, \ldots, P_p) \in Z^p$
    positive integer k
    PROPERTY: There is a permutation π of $\{1,2,\ldots,p\}$ such that

$$\left(\sum_{j=1}^{p} [\text{if } T_{\pi(1)} + \cdots + T_{\pi(j)} > D_{\pi(j)} \text{ then } P_{\pi(j)} \text{ else } 0]\right) \leq k \quad .$$



REDUCIBILITY AMONG COMBINATORIAL PROBLEMS 97

20. PARTITION
    INPUT: $(c_1, c_2, \ldots, c_s) \in Z^s$
    PROPERTY: There is a set $I \subseteq \{1,2,\ldots,s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$.

21. MAX CUT
    INPUT: graph G, weighting function $w: A \to Z$, positive integer W
    PROPERTY: There is a set $S \subseteq N$ such that

$$\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w(\{u,v\}) \geq W \quad .$$

Karp's Theorem (1972)

A lot of problems are NP-complete

NP-Complete Problems

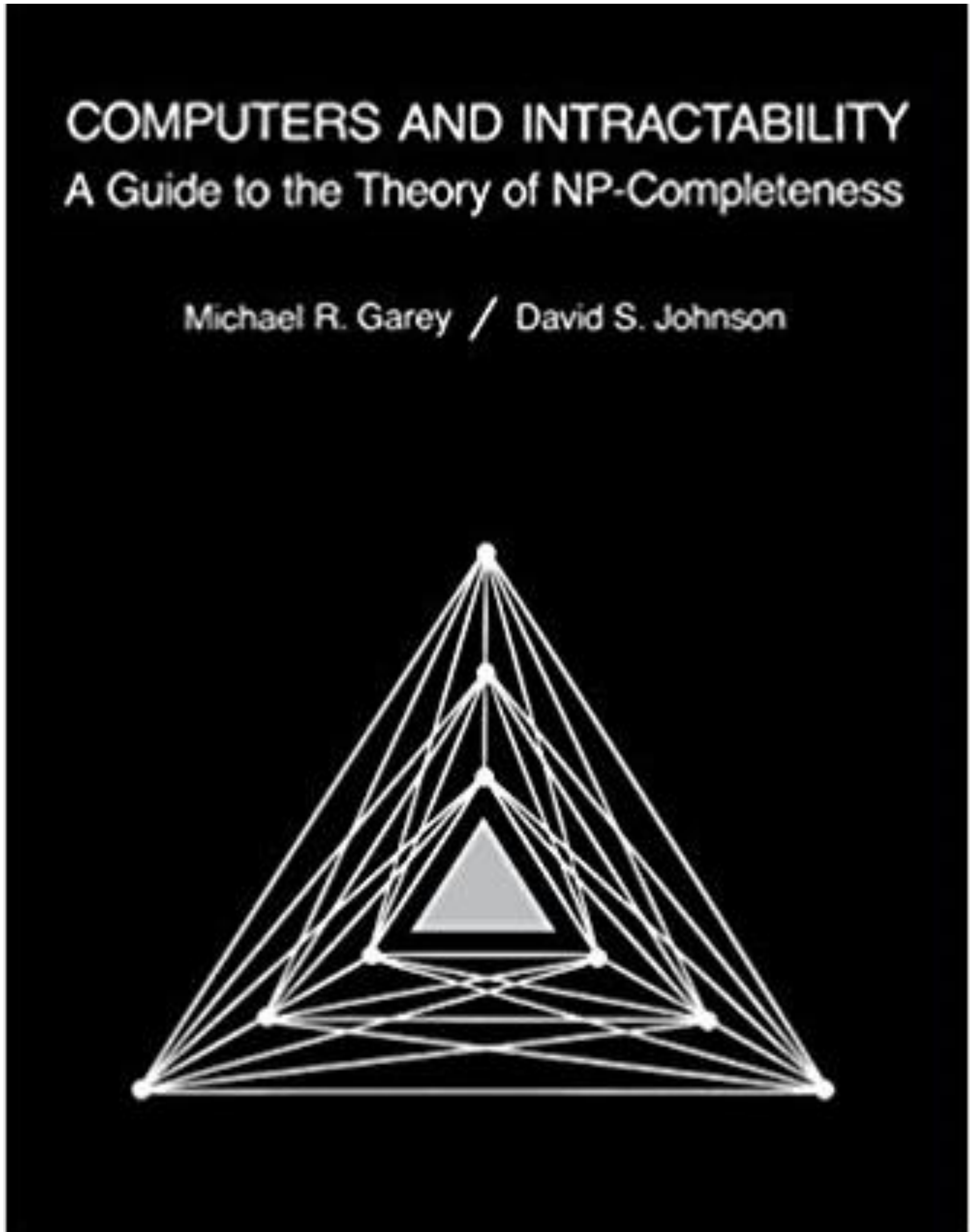
But Wait! There’s more!

By 1979, at least 300 problems had been proven NP-complete.

Garey and Johnson put a list of all the NP-complete problems they could find in this textbook.

Took almost 100 pages to just list them all.

No one has made a comprehensive list since.


COMPUTERS AND INTRACTABILITY
A Guide to the Theory of NP-Completeness
Michael R. Garey / David S. Johnson

NP-Complete Problems

But Wait! There’s more!

In December 2018, mathematicians and computer scientists put papers on the arXiv claiming to show (at least) 25 more problems are NP-complete.

There are literally thousands of NP-complete problems known.

Examples

There are literally thousands of NP-complete problems.
And some of them look weirdly similar to problems we do know efficient algorithms for.

In P

Short Path

Given a directed graph, report if there is a path from s to t of length at most k.

NP-Complete

Long Path

Given a directed graph, report if there is a path from s to t of length at least k.

Examples

In P

Light Spanning Tree

Given a weighted graph, find a spanning tree (a set of edges that connect all vertices) of weight at most k.

NP-Complete

Traveling Salesperson

Given a weighted graph, find a tour (a walk that visits every vertex and returns to its start) of weight at most k.

The electric company just needs a greedy algorithm to lay its wires.
Amazon doesn’t know a way to optimally route its delivery trucks.

Examples

In P

2-Coloring

Given an undirected graph, can the vertices be labeled red and blue with no edge having the same colors on both endpoints?

NP-Complete

3-Coloring

Given an undirected graph, can the vertices be labeled red, blue, and green with no edge having the same colors on both endpoints?

Just changing a number by one takes us from one of the first problems we solved (and one of the fastest algorithms we've seen) to something we don't know how to solve efficiently at all.

Dealing with NP-hardness

Thousands of times someone has wanted to find an efficient algorithm for a problem…

…only to realize that the problem was NP-hard.

Takeaway 1:

Even though we haven't proven $P \neq NP$ (i.e. we haven't proven any of these problems **don't** have an efficient algorithm), this is good evidence that we shouldn't by trying to solve NP-hard problems.

It's probably not just a matter of finding the "right representation"/"right angle on the problem" we've tried a few thousand of them.

Why Should you care?

Why should you care about P vs. NP

Most computer scientists are convinced that P≠NP.

- A survey of experts (PhDs in CS) found 98% of them thought P≠NP.
- And the median guess was that we're at least 50 years from getting the answer.

Why should you care about this problem?

It's your chance for:

$1,000,000. The Clay Mathematics Institute will give $1,000,000 to whoever solves P vs. NP (or any of the 5 remaining mathematical conjectures they listed)

To get a Turing Award

Why should you care about P vs. NP

Most computer scientists are convinced that P≠NP.

A survey of experts (PhDs in CS) found 98% of them thought P≠NP.

And the median guess was that we're at least 50 years from getting the answer.

Why should you care about this problem?

It's your chance for:

$1,000,000. The Clay Mathematics Institute will give $1,000,000 to whoever solves P vs. NP (or any of the 5 remaining mathematical conjectures they listed)

To get ~~a Turing Award~~ the Turing Award renamed after you.

Why Should You Care if P=NP?

Suppose P=NP.

Specifically that we found a genuinely in-practice efficient algorithm for an NP-complete problem. What would you do?

- $1,000,000 from the Clay Math Institute obviously, but what's next?

Why Should You Care if P=NP?

We found a genuinely in-practice efficient algorithm for an NP-complete problem. What would you do?

- Another $5,000,000 from the Clay Math Institute
- Put mathematicians out of work?
- Decrypt (essentially) all current internet communication. See your ethics project.

A world where P=NP is a very very different place from the world we live in now.

Why Should You Care if P≠NP?

We already expect P≠NP. Why should you care when we finally prove it?

P≠NP says something fundamental about the universe.

For some questions there is not a clever way to find the right answer

Even though you'll know it when you see it.

There is actually a way to obscure information, so it cannot be found quickly no matter how clever you are.

Why Should You Care if P≠NP?

To prove P≠NP we need to better understand the differences between problems.

- Why do some problems allow easy solutions and others don't?
- What is the structure of these problems?

We don't care about P vs NP just because it has a huge effect about what the world looks like.

We will learn a lot about computation along the way.

Dealing With NP-hardness

Dealing with NP-hardness

Thousands of times someone has wanted to find an efficient algorithm for a problem…

…only to realize that the problem was NP-hard.

Takeaway 2:

Sooner or later it will happen to one of you.

What do you do if you think your problem is NP-complete?

Dealing with NP-completeness

You just started your new job at Amazon. Your boss asks you to look into the following problem

You have a graph, each vertex is where a specific truck has to do a delivery. Starting from the warehouse, how do you make all the deliveries and return to the warehouse using the minimum amount of gas.

Traveling Salesperson

Given a weighted graph, find a tour (a walk that visits every vertex and returns to its start) of weight at most k.

Step 1: Make sure your problem is really NP-hard

Understand **exactly** what your inputs and outputs are.

2-coloring and 3-coloring are very different.

Finding a vertex cover of a general graph is NP-hard. Finding a vertex cover of a bipartite graph can be done in multiple very efficient ways.

Understand **exactly** what you're being asked to solve.
Realizing you're trying to solve a problem on a tree instead of a general graph almost always makes DP possible.
Are your constraints linear (can you use an LP)?
Are your constraints simple (are you solving 2SAT instead of 3SAT)?

Step 2: It still looks hard

Now that you know exactly what you’re trying to solve, and you still can’t solve it…

Next try to prove hardness (i.e. do a reduction).

Usually there’s a similar problem you can convert from!

It’s easier to do a reduction from 3-coloring to 5-coloring than from Hamiltonian Path to 5-coloring.

Both reductions **exist** but there’s no need to flex here, look up a list of NP-complete problems and see what’s similar looking.

Step 3: ???

So you go to your boss and say

“Sorry, problem’s NP-hard. I proved it.”

And your boss says:

“that’s a cool proof and all, but really. We need to tell the drivers where to go tomorrow…and we need to use less gas.

Step 3…wellllll, maybe not

Can you write your problem as a SAT instance?

Ok, you definitely can if it's in NP, that's what NP-hardness means…can you write it as a reasonably-sized SAT instance (n^3 instead of $1000000n^{100}$)?

There are SAT libraries that **often** run pretty fast. In the worst-case they're still exponential, but you don't always hit the worst case!

Can you write your problem as an integer program?

Run an integer programming library and see what happens!

Can you write your problem as a graph problem?

Many are very well studied for "simple" graphs (e.g. "planar" graphs, ones that can be drawn on a piece of paper without edges crossing).

Step 4 – Permanent Solutions

Those exponential time algorithms are great as band-aids.

If it’s a one-time thing, or just a “we’ll run this about once a week, if it takes too long once in a while no big deal” these are fine.

But what if you need a guarantee!

Your code is running every night, and you need an answer by 6 AM or the delivery trucks don’t go out.

Step 4 – Permanent Solutions

Two good options:

Exponential algorithms that aren’t-as-slow-as-others

Give you an exact answer; won’t take polynomial time but will be guaranteed take you less time than brute force.

Approximation algorithms

Don’t give you the best answer, but guarantees a reasonable amount of time, and a guaranteed-pretty-good-answer.

We’ll learn about these next time!