CSE 417: Algorithms and Computational Complexity

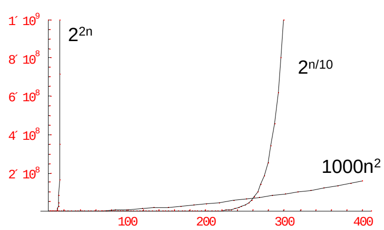
Complexity:
P, NP, and NP-completeness

Autumn 2002
Paul Beame

Some History

- 1930's
  – What is (is not) computable
- 1960/70's
  – What is (is not) *feasibly* computable
  - Goal – a (largely) technology independent theory of time required by algorithms
  - Key modeling assumptions/approximations
    - Asymptotic (Big-O), worst case is revealing
    - Polynomial, exponential time – qualitatively different

Polynomial vs Exponential Growth



Another view of Poly vs Exp

Next year's computer will be 2x faster. If I can solve problem of size n_0 today, how large a problem can I solve in the same time next year?

Complexity	Increase	E.g. $T=10^{12}$	
$O(n)$	$n_0 \rightarrow 2n_0$	10^{12}	2×10^{12}
$O(n^2)$	$n_0 \rightarrow \sqrt{2}\, n_0$	10^6	1.4×10^6
$O(n^3)$	$n_0 \rightarrow \sqrt[3]{2}\, n_0$	10^4	1.25×10^4
$2^{n/10}$	$n_0 \rightarrow n_0+10$	400	410
2^n	$n_0 \rightarrow n_0+1$	40	41

Polynomial versus exponential

- We'll say any algorithm whose run-time is
  - polynomial is good
  - bigger than polynomial is bad
- Note – of course there are exceptions:
  - n^{100} is bigger than $(1.001)^n$ for most practical values of n but usually such run-times don't show up
  - There are algorithms that have run-times like $O(2^{n^{1/22}})$ and these may be useful for small input sizes, but they're not too common either

Decision problems

- Computational complexity usually analyzed using decision problems
  - answer is just 1 or 0 (yes or no).
- Why?
  - much simpler to deal with
  - *deciding* whether **G** has a path from s to t, is certainly no harder than *finding* a path from s to t in G, so a *lower* bound on deciding is also a lower bound on finding
  - Less important, but if you have a good decider, you can often use it to get a good finder.

Computational Complexity

- Classify problems according to the amount of computational resources used by the best algorithms that solve them
- Recall:
  - worst-case running time of an algorithm
    - **max** # steps algorithm takes on any input of size **n**
- Define:
  - **TIME**(**f**(**n**)) to be the set of all decision problems solved by algorithms having worst-case running time **O**(**f**(**n**))

7

Polynomial time

- Define P (polynomial-time) to be
  - the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.
- $P = \bigcup_{k\geq 0} \mathrm{TIME}(n^k)$

8

Some Terminology

- "Problem"
  - The general case of a computational task
  - E.g. Given: a graph **G** and and nodes **s** and **t** in **G** does **G** contain a path from **s** to **t**?
- "Problem Instance"
  - A specific input for a problem, e.g.

- Decision Problems – Just **YES**/**NO** answers
  - Inputs requiring output **YES** are called **YES** instances, **NO** instances defined similarly

9

Beyond P?

- There are many natural, practical problems for which we don't know any polynomial-time algorithms
- e.g. decisionTSP:
  - Given a weighted graph **G** and an integer **k**, does there exist a tour that visits all vertices in **G** having total weight at most **k**?

10

Solving TSP given a solution to decisionTSP

- Use binary search and several calls to decisionTSP to figure out what the exact total weight of the shortest tour is.
  - Upper and lower bounds to start are **n** times largest and smallest weights of edges, respectively
  - Call **W** the weight of the shortest tour.
- Now figure out which edges are in the tour
  - For each edge **e** in the graph in turn, remove **e** and see if there is a tour of weight at most **W** using decisionTSP
    - if not then **e** must be in the tour so put it back

11

More examples

- Independent-Set:
  - Given a graph **G**=(**V**,**E**) and an integer **k**, is there a subset **U** of **V** with |**U**| ≥ **k** such that no two vertices in **U** are joined by an edge.
- Clique:
  - Given a graph **G**=(**V**,**E**) and an integer **k**, is there a subset **U** of **V** with |**U**| ≥ **k** such that every pair of vertices in **U** is joined by an edge.

12

Satisfiability

- Boolean variables $x_1,\ldots,x_n$
  - taking values in {0,1}. 0=false, 1=true
- Literals
  - x_i or $\O x_i$ for $i=1,\ldots,n$
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \lor \O x_3 \lor x_7 \lor x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses

13

Satisfiability

- CNF formula example
  - $(x_1 \lor \O x_3 \lor x_7 \lor x_{12}) \land (x_2 \lor \O x_4 \lor x_7 \lor x_5)$
- If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable
  - the one above is, the following isn't
  - $x_1 \land (\O x_1 \lor x_2) \land (\O x_2 \lor x_3) \land \O x_3$
- Satisfiability: Given a CNF formula F, is it satisfiable?

14

More History – As of 1970

- Many of the above problems had been studied for decades
- All had real, practical applications
- *None* had polynomial time algorithms; exponential was best known
- But, it turns out they all have a very deep similarity under the skin

15

Common property of these problems

- There is a special piece of information, a **short hint** or proof, that allows you to **efficiently verify** (in polynomial-time) that the YES answer is correct. This hint might be very hard to find
- e.g.
  - **DecisionTSP**: the tour itself,
  - **Independent-Set**, **Clique**: the set **U**
  - **Satisfiability**: an assignment that makes **F** true.

16

The complexity class NP

NP consists of all decision problems where

- You can **verify** the **YES** answers efficiently (in polynomial time) given a short (polynomial-size) **hint**

And

- **No hint** can fool your polynomial time verifier into saying **YES** for a **NO** instance

17

More Precise Definition of NP

- A decision problem is in NP iff there is a polynomial time procedure verify(.,.), and an integer k such that
  - for every input **x** to the problem that is a **YES** instance there is a hint **h** with $|h| \le |x|^k$ such that **verify**(**x**,**h**) = **YES**

  and
  - for every input **x** to the problem that is a **NO** instance there does **not** exist a hint **h** with $|h| \le |x|^k$ such that **verify**(**x**,**h**) = **YES**

18

Example: CLIQUE is in NP

procedure **verify**(**x**,**h**)

if

x is a well-formed representation of a graph **G** = (**V**, **E**) and an integer **k**,

and

h is a well-formed representation of a vertex subset **U** of **V** of size **k**,

and

U is a clique in **G**,

then output "**YES**"

else output "**I'm unconvinced**"

Is it correct?

For every **x** = (**G**,**k**) such that **G** contains a **k**-clique, there is a hint **h** that will cause **verify**(**x**,**h**) to say **YES**,

- **h** = a list of the vertices in such a **k**-clique

And no hint can fool **verify**(**x**,ᵡ) into saying **YES** if either

- **x** isn't well-formed (the uninteresting case)
- **x** = (**G**,**k**) but **G** does not have any cliques of size **k** (the interesting case)

Keys to showing that a problem is in NP

- What's the output? (must be **YES**/**NO**)
- What must the input look like?
- Which inputs need a **YES** answer?
  - Call such inputs **YES** inputs/**YES** instances
- For every given **YES** input, is there a hint that would help?
  - OK if some inputs need no hint
- For any given **NO** input, is there a hint that would trick you?

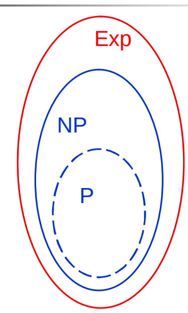
Solving NP problems without hints

- The only **obvious algorithm** for most of these problems is **brute force**:
  - try all possible hints and check each one to see if it works.
  - *Exponential* time:
    - 2^n truth assignments for **n** variables
    - **n!** possible TSP tours of **n** vertices
    - $\binom{n}{k}$ possible **k** element subsets of **n** vertices
    - etc.

What We Know

- Nobody knows if all problems in **NP** can be done in polynomial time, i.e. does **P=NP**?
  - one of the most important open questions in all of science.
  - huge practical implications
- Every problem in **P** is in **NP**
  - one doesn't even need a hint for problems in P so just ignore any hint you are given
- Every problem in **NP** is in exponential time

P and NP



P vs NP

- Theory
  - **P = NP?**
  - Open Problem!
  - I bet against it
- Practice
  - Many interesting, useful, natural, well-studied problems known to be **NP**-complete
  - With rare exceptions, no one routinely succeeds in finding exact solutions to large, arbitrary instances

25

More Connections

- Some Examples in NP
  - Satisfiability
  - Independent-Set
  - Clique
  - Vertex Cover
- All hard to solve; hints seem to help on all
- Very surprising fact:
  - Fast solution to *any* gives fast solution to *all!*

26

NP-hardness & NP-completeness

- Some problems in **NP** seem hard
  - people have looked for efficient algorithms for them for hundreds of years without success
- However
  - nobody knows how to **prove** that they are really hard to solve, i.e. **P** ≠ **NP**

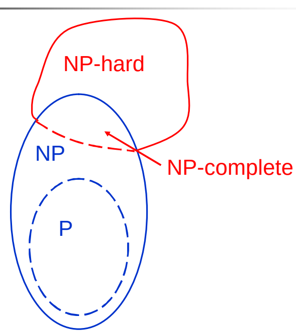
27

NP-hardness & NP-completeness

- Alternative approach
  - show that they are at least as hard as any problem in **NP**
- Rough definition:
  - A problem is NP-hard iff it is at least as hard as any problem in **NP**
  - A problem is NP-complete iff it is both
    - **NP**-hard
    - in **NP**

28

P and NP



29

How do we show that one problem is ‘at least as hard as’ another?

- We’ve done this before in a different context
  - We used the undecidability of the halting problem to show that other problems were undecidable
  - This really amounted to showing that those other problems were ‘at least as hard as’ the halting problem in some sense

30

To show that problem A is at least as hard as the Halting Problem

- We created a program **H** that solved the Halting Problem using a program for **A** as a subroutine
- This involved creating some transformation code **T** that took the input <**P**,**x**> for the Halting Problem and converted it to an input **y** for **A**
- For historical reasons this transformation **T** is called a **reduction**

31

Reductions: What we did

- We write: **Halting Problem** ≤ **A**
- We transformed an instance of **Halting Problem** into an instance of **A** such that **A**'s answer is **Halting Problem**'s.
  - Function **H**(**z**)
    - Run program **T** to translate input **z** for **H** into an input **y** for **A**
    - Call a subroutine for problem **A** on input **y**
    - Output the answer produced by **A**(**y**)
  - (**z** was of the form <**P**,**x**>.)

32

Reductions: general case

- We write: **L** ≤ **R**
- We transform an instance of **L** into an instance of **R** such that **L**'s answer is **R**'s.
  - Function **L**(**x**)
    - Run program **T** to translate input **x** for **L** into an input **y** for **R**
    - Call a subroutine for problem **R** on input **y**
    - Output the answer produced by **R**(**y**)

33

This isn't enough

- We care about time bounds now so this isn't enough
  - In the case of **P** and **NP** all our problems are decidable
    - Exponential time at worst
  - A "cheating" reduction **T** could simply solve the problem **L** directly and create some stupid input for **R** for which it already knows the answer
    - **T** would be doing all the work and it wouldn't say anything about how the hardnesses of **L** and **R** compare
- Solution: **L** £^{p} **R**
  - Require that **T** work in polynomial time

34

Polynomial Time Reduction

- **L** £^{p} **R** if there is a poly time algorithm for **L** *assuming* a poly time subroutine for **R**
- Thus, fast algorithm for **R** implies fast algorithm for **L**
- If you can prove there is no fast algorithm for **L**, then that proves there is no fast algorithm for **R**

35

Why the name reduction?

- Weird: it maps an easier problem into a harder one
- Same sense as saying Maxwell reduced the problem of analyzing electricity & magnetism to solving partial differential equations
  - solving partial differential equations in general is a much harder problem than solving E&M problems

36

A geek joke

- An engineer
  - is placed in a kitchen with an empty kettle on the table and told to boil water; she fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - she is next confronted with a kettle full of water sitting on the counter and told to boil water; she puts it on the stove, turns on the gas and boils water.
- A mathematician
  - is placed in a kitchen with an empty kettle on the table and told to boil water; he fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - he is next confronted with a kettle full of water sitting on the counter and told to boil water: he empties the kettle in the sink, places the empty kettle on the table and says, "I've reduced this to an already solved problem".

37

Reductions

- Show: Independent-Set £^{p} Clique
- Independent-Set:
  - Given a graph **G**=(**V**,**E**) and an integer **k**, is there a subset **U** of **V** with |**U**| ≥ **k** such that no two vertices in **U** are joined by an edge.
- Clique:
  - Given a graph **G**=(**V**,**E**) and an integer **k**, is there a subset **U** of **V** with |**U**| ≥ **k** such that every pair of vertices in **U** is joined by an edge.

38

Independent-Set ≤^{p} Clique

- Given (**G**,**k**) as input to Independent-Set where **G**=(**V**,**E**)
- Transform to (**G'**,**k**) where **G'**=(**V**,**E'**) has the same vertices as **G** but **E'** consists of **precisely** those edges that are not edges of **G**
- **U** is an independent set in **G**

⇔ **U** is a clique in **G'**

39

Reductions Exercise

- Show: Independent Set £^{p} Vertex-Cover
- Vertex-Cover:
  - Given an undirected graph **G**=(**V**,**E**) and an integer **k** is there a subset **W** of **V** of size at most **k** such that every edge of **G** has at least one endpoint in **W**? (i.e. **W** covers all vertices of **G**).
- Independent-Set:
  - Given a graph **G**=(**V**,**E**) and an integer **k**, is there a subset **U** of **V** with |**U**| ≥ **k** such that no two vertices in **U** are joined by an edge.

40

NP-hardness & NP-completeness

- Definition: A problem **R** is NP-hard iff every problem **L** Î **NP** satisfies **L** £^{p}**R**
- Definition: A problem **R** is NP-complete iff **R** is NP-hard and **R** Î **NP**
- Even though we seem to have lots of hard problems in **NP** it is not obvious that such super-hard problems even exist!

41

Cook's Theorem

- Theorem (Cook 1971): **Satisfiability** is **NP**-complete
- Recall
  - CNF formula
    - e.g. $(x_1 \lor Øx_3 \lor x_7 \lor x_{12}) \land (x_2 \lor Øx_4 \lor x_7 \lor x_5)$
  - If there is some assignment of **0**'s and **1**'s to the variables that makes it true then we say the formula is satisfiable
  - **Satisfiability:** Given a CNF formula F, is it satisfiable?

42

Implications of Cook's Theorem?

- There is at least one interesting super-hard problem in **NP**
- Is that such a big deal?
- YES!
  - There are lots of other problems that can be solved if we had a polynomial-time algorithm for **Satisfiability**
  - Many of these problems are exactly as hard as **Satisfiability**

43

A useful property of polynomial-time reductions

- Theorem: If **L** £^{p}**R** and **R** £^{p}**S** then **L** £^{p}**S**
- Proof idea:
  - Compose the reduction **T** from **L** to **R** with the reduction **T'** from **R** to **S** to get a new reduction **T''(x)=T'(T(x))** from **L** to **S**.

44

Cook's Theorem & Implications

- Theorem (Cook 1971): **Satisfiability** is **NP**-complete
- Corollary: **R** is **NP**-hard ⇔ **Satisfiability** £^{p}**R**
  - (or **Q** £^{p}**R** for any **NP**-complete problem **Q**)
- Proof:
  - If **R** is **NP**-hard then every problem in **NP** polynomial-time reduces to **R**, in particular **Satisfiability** does since it is in **NP**
  - For any problem **L** in **NP**, **L** £^{p}**Satisfiability** and so if **Satisfiability** £^{p}**R** we have **L** £^{p} **R**.
    - therefore **R** is **NP**-hard if **Satisfiability** £^{p}**R**

45

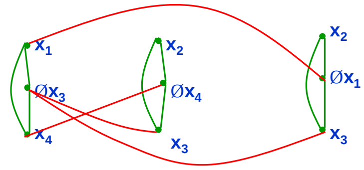
Another NP-complete problem: Satisfiability £^{p}Independent-Set

- A Tricky Reduction:
  - mapping CNF formula **F** to a pair **<G,k>**
  - Let **m** be the number of clauses of **F**
  - Create a vertex in **G** for each literal in **F**
  - Join two vertices **u**, **v** in **G** by an edge iff
    - **u** and **v** correspond to literals in the same clause of **F**, (green edges) or
    - **u** and **v** correspond to literals **x** and Ø**x** (or vice versa) for some variable **x**. (red edges).
  - Set **k=m**
  - Clearly polynomial-time

46

Satisfiability £^{p}Independent-Set

F: $(x_1$ Ú Øx_3 Ú $x_4)$ Ù $(x_2$ Ú Øx_4 Ú $x_3)$ Ù $(x_2$ Ú Øx_1 Ú $x_3)$



47

Satisfiability £^{p}Independent-Set

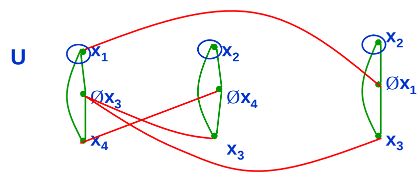
- Correctness:
  - If **F** is satisfiable then there is some assignment that satisfies at least one literal in each clause.
  - Consider the set **U** in **G** corresponding to the **first satisfied literal in each clause**.
    - **|U|=m**
    - Since **U** has only one vertex per clause, no two vertices in **U** are joined by green edges
    - Since a truth assignment never satisfies both **x** and Ø**x**, **U** doesn't contain vertices labeled both **x** and Ø**x** and so no vertices in **U** are joined by red edges
    - Therefore **G** has an independent set, **U**, of size at least **m**
  - Therefore **<G,m>** is a **YES** for independent set.

48

Satisfiability £PIndependent-Set

1 0 1 1 0 1 1 0 1

F: (x_1 Ú Øx_3 Ú x_4) Ù (x_2 Ú Øx_4 Ú x_3) Ù (x_2 Ú Øx_1 Ú x_3)



Given assignment x_1=x_2=x_3=x_4=1,
U is as circled

49

Satisfiability £PIndependent-Set

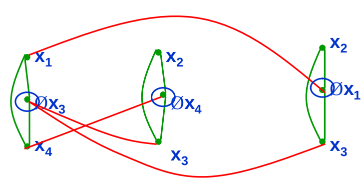
- Correctness continued:
  - If <G,m> is a YES for Independent-Set then there is a set U of m vertices in G containing no edge.
    - Therefore U has precisely one vertex per clause because of the green edges in G.
    - Because of the red edges in G, U does not contain vertices labeled both x and Øx
    - Build a truth assignment A that makes all literals labeling vertices in U true and for any variable not labeling a vertex in U, assigns its truth value arbitrarily.
    - By construction, A satisfies F
  - Therefore F is a YES for Satisfiability.

50

Satisfiability £PIndependent-Set

0 1 0 ? 1 0 ? 1 0

F: (x_1 Ú Øx_3 Ú x_4) Ù (x_2 Ú Øx_4 Ú x_3) Ù (x_2 Ú Øx_1 Ú x_3)



Given U, satisfying assignment
is x_1=x_3=x_4=0, x_2=0 or 1

51

Independent-Set is NP-complete

- We just showed that Independent-Set is NP-hard and we already knew Independent-Set is in NP.
- Corollary: Clique is NP-complete
  - We showed already that Independent-Set £P Clique and Clique is in NP.

52

Problems we already know are NP-complete

- Satisfiability
- Independent-Set
- Clique
- Vertex-Cover
- There are 1000's of practical problems that are NP-complete, e.g. scheduling, optimal VLSI layout etc.

53

Is NP as bad as it gets?

- NO! NP-complete problems are frequently encountered, but there's worse:
  - Some problems provably require exponential time.
    - Ex: Does P halt on x in $2^{|x|}$ steps?
  - Some require 2^n, 2^{2^n}, $2^{2^{2^n}}$, ... steps
  - And of course, some are just plain uncomputable

54

Summary

- Big-O(n^2) – good
- P – good
- Exp – bad
- Hints help? NP
- NP-hard, NP-complete – bad (I bet)

55