CSE 421: Introduction to Algorithms

NP-completeness

Paul Beame

Relative Complexity of Problems

- Want a notion that allows us to compare the complexity of problems
- Want to be able to make statements of the form
  “If we could solve problem B in polynomial time then we can solve problem A in polynomial time”

- “Problem B is at least as hard as problem A”

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

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

Polynomial Time Reduction

- \( A \leq P B \) if there is an algorithm for A using a ‘black box’ (subroutine) that solves B that
  - Uses only a polynomial number of steps
  - Makes only a polynomial number of calls to a subroutine for B
- Thus, poly time algorithm for B implies poly time algorithm for A
  - Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!
- If you can prove there is no fast algorithm for A, then that proves there is no fast algorithm for B

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”.

-
A Special kind of Polynomial-Time Reduction

We will always use a restricted form of polynomial-time reduction often called Karp or many-one reduction.

\( A \preceq_P B \) if and only if there is an algorithm for \( A \) given a black box solving \( B \) that on input \( x \):
- Runs for polynomial time computing an input \( f(x) \)
- Makes one call to the black box for \( B \)
- Returns the answer that the black box gave

We say that the function \( f \) is the reduction.

Independent-Set \( \preceq_P \) 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 \( G \)
- \( \Leftrightarrow U \) is a clique in \( G' \)

Reduction Idea

- Claim: In a graph \( G = (V, E) \), \( S \) is an independent set iff \( V - S \) is a vertex cover
- \( \Rightarrow \) Let \( S \) be an independent set in \( G \)
  - Then \( S \) contains at most one endpoint of each edge of \( G \)
  - At least one endpoint must be in \( V - S \)
- \( \Leftarrow \) \( V - S \) is a vertex cover
  - Let \( W = V - S \) be a vertex cover of \( G \)
  - Then \( S \) does not contain both endpoints of any edge (else \( W \) would miss that edge)
  - \( S \) is an independent set

Reductions by Simple Equivalence

- Show: Independent-Set \( \preceq_P \) Clique
- Independent-Set:
  - Given a graph \( G = (V, E) \) and an integer \( k \), is there a subset \( U \) of \( V \) with \( |U| \geq k \) 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 \( |U| \geq k \) such that every pair of vertices in \( U \) is joined by an edge?

More Reductions

- Show: Independent Set \( \preceq_P \) Vertex-Cover
- Vertex-Cover:
  - Given an undirected graph \( G = (V, E) \) and an integer \( k \), is there a subset \( W \) of \( V \) with \( |W| \leq k \) such that every edge of \( G \) has at least one endpoint in \( W \) (i.e. \( W \) covers all edges of \( G \))?
- Independent-Set:
  - Given a graph \( G = (V, E) \) and an integer \( k \), is there a subset \( U \) of \( V \) with \( |U| \geq k \) such that no two vertices in \( U \) are joined by an edge?

Reduction

- Map \((G, k)\) to \((G, n - k)\)
  - Previous lemma proves correctness
- Clearly polynomial time
- We also get that Vertex-Cover \( \preceq_P \) Independent Set
Reductions from a Special Case to a General Case

- **Show:** Vertex-Cover \( \leq_p \) Set-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 edges of \( G \)?)

- **Set-Cover:**
  - Given a set \( U \) of \( n \) elements, a collection \( S_1, ..., S_m \) of subsets of \( U \), and an integer \( k \), does there exist a collection of at most \( k \) sets whose union is equal to \( U \)?

Proof of Correctness

- **Two directions:**
  - If the answer to Vertex-Cover on \( (G,k) \) is YES then the answer for Set-Cover on \( f(G,k) \) is YES
    - If a set \( W \) of \( k \) vertices covers all edges then the collection \( \{S_v \mid v \in W\} \) of \( k \) sets covers all of \( U \)
  - If the answer to Set-Cover on \( f(G,k) \) is YES then the answer for Vertex-Cover on \( (G,k) \) is YES
    - If a subcollection \( S_{v_1}, ..., S_{v_k} \) covers all of \( U \) then the set \( \{v_1, ..., v_k\} \) is a vertex cover in \( G \).

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.

The Simple Reduction

- **Transformation** \( f \) maps \( (G=(V,E),k) \) to \( (U,S_1, ..., S_m,k') \)
  - \( U \leftarrow E \)
  - For each vertex \( v \in V \) create a set \( S_v \) containing all edges that touch \( v \)
    - \( k' \leftarrow k \)
- Reduction \( f \) is clearly polynomial-time to compute
- We need to prove that the resulting algorithm gives the right answer!

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.

Beyond \( P \)?

- There are many other 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 \( k \)?
Satisfiability

- Boolean variables $x_1, \ldots, x_n$
  - taking values in $\{0,1\}$; 0=false, 1=true
- Literals
  - $x_i$ or $\neg x_i$ for $i=1,\ldots,n$
- Clause
  - a logical OR of one or more literals
    - e.g. $(x_1 \lor \neg x_2 \lor x_7)$
- CNF formula
  - a logical AND of a bunch of clauses
- $k$-CNF formula
  - All clauses have exactly $k$ variables

Common property of these problems

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

More Precise Definition of NP

- A decision problem is in NP iff there is a polynomial time procedure $\text{verify}(\cdot, \cdot)$, and an integer $k$ such that
  - for every input $x$ to the problem that is a YES instance there is a certificate $t$ with $|t| \leq |x|^k$ such that $\text{verify}(x,t) = \text{YES}$
  - and for every input $x$ to the problem that is a NO instance there does not exist a certificate $t$ with $|t| \leq |x|^k$ such that $\text{verify}(x,t) = \text{YES}$

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) certificate
  - And
  - No certificate can fool your polynomial time verifier into saying YES for a NO instance

Example: CLIQUE is in NP

```
procedure verify(x,t)
    if $x$ is a well-formed representation of a graph $G = (V, E)$ and an integer $k$, and
    $t$ 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"
```

Satisfiability

- CNF formula example
  - $(x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_4 \lor x_5) \land (x_2 \lor \neg x_4 \lor x_5)$
- 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
  - $x_1 \land (\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land \neg x_3$

3-SAT: Given a CNF formula $F$ with 3 variables per clause, is it satisfiable?
Is it correct?

For every $x = (G,k)$ such that $G$ contains a $k$-clique, there is a certificate $t$ that will cause $\text{verify}(x,t)$ to say YES,
- $t$ is a list of the vertices in such a $k$-clique

And no certificate can foil $\text{verify}(x,:)$ 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)

Solving NP problems without hints

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

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. $P \neq NP$

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 certificate for problems in $P$ so just ignore any hint you are given
- Every problem in NP is in exponential time

Problems in NP that seem hard

- Some Examples in NP
  - 3-SAT
  - Independent-Set
  - Clique
  - Vertex Cover
- All hard to solve; certificates seem to help on all
- Fast solution to any gives fast solution to all!
NP-hardness & NP-completeness

- Alternative approach to proving problems not in $P$
  - 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 $NP$

Definition: A problem $B$ is NP-hard iff every problem $A \in NP$ satisfies $A \leq_P B$

Definition: A problem $B$ is NP-complete iff $A$ is NP-hard and $A \in NP$

Even though we seem to have lots of hard problems in $NP$ it is not obvious that such super-hard problems even exist!

Implications of Cook-Levin 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 3-SAT
    - Many of these problems are exactly as hard as 3-SAT

Cook-Levin Theorem

Theorem (Cook 1971, Levin 1973):

- 3-SAT is NP-complete

Recall

- CNF formula
  - $(x_1 \lor \neg x_1 \lor x_2) \land (x_2 \lor \neg x_3 \lor x_3) \land (x_1 \lor \neg x_1 \lor x_2)$
- 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
- 3-SAT: Given a 3-CNF formula $F$, is it satisfiable?

A useful property of polynomial-time reductions

- Theorem: If $A \leq_P B$ and $B \leq_P C$ then $A \leq_P C$
- Proof idea: (Using $\leq_P^*$)
  - Compose the reduction $f$ from $A$ to $B$ with the reduction $g$ from $B$ to $C$ to get a new reduction $h(x) = g(f(x))$ from $A$ to $C$.
  - The general case is similar and uses the fact that the composition of two polynomials is also a polynomial.
Cook-Levin Theorem & Implications

- Theorem (Cook 1971, Levin 1973): 3-SAT is NP-complete
  
  For proof see CSE 431

- Corollary: B is NP-hard \( \Rightarrow \) 3-SAT \( \leq \) B
  
  (or A \( \leq \) B for any NP-complete problem A)

- Proof:
  
  - If B is NP-hard then every problem in NP polynomial-time reduces to B, in particular 3-SAT does since it is in NP
  
  - For any problem A in NP, A \( \leq \) 3-SAT and so if 3-SAT \( \leq \) B we have A \( \leq \) B.
  
  - therefore B is NP-hard if 3-SAT \( \leq \) B

3-SAT \( \leq \) 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.
    
    \[ U \leq m \]

  - 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 \( \neg x \), \( U \) doesn’t contain vertices labeled both \( x \) and \( \neg x \) 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.

- 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 if
    
    \[ u \text{ and } v \text{ correspond to literals in the same clause of } F \text{, (green edges)} \]

    \[ u \text{ and } v \text{ correspond to literals } x \text{ and } \neg x \text{ (or vice versa) for some variable } x. \text{ (red edges)} \]

  - Set \( k=m \)

  - Clearly polynomial-time

3-SAT \( \leq \) Independent-Set

- Another NP-complete problem:
  
  - 3-SAT \( \leq \) Independent-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 \( \neg x \).

    - 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 3-SAT.
3-SAT ≤ Independent-Set

\[ F: (x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_4) \land (x_3 \lor x_4 \lor \neg x_1) \]

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

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 ≤ Clique and Clique is in NP.

Steps to Proving Problem B is NP-complete

- Show B is NP-hard:
  - State: “Reduction is from NP-hard Problem A”
  - Show what the map f is
  - Argue that f is polynomial time
  - Argue correctness: two directions Yes for A implies Yes for B and vice versa.
- Show B is in NP
  - State what hint/certificate is and why it works
  - Argue that it is polynomial-time to check.

Some other NP-complete examples you should know

- Hamiltonian-Cycle: Given a directed graph G is there a cycle in G that visits each vertex in G exactly once?
- Hamiltonian-Path: Given a directed graph G is there a path in G that visits each vertex in G exactly once?
  - Both are also NP-complete when G is an undirected graph
  - Note that deciding the similar questions for Eulerian-Cycle and Eulerian-Path (which require that each edge be visited exactly once rather than each vertex) can be done in polynomial time.
- How?

Travelling-Salesman Problem (TSP)

- Given a set of \( n \) cities \( v_1, \ldots, v_n \) and distances between each pair of cities \( d(v_i, v_j) \) what is the shortest tour that visits all the cities?
  - Not a decision problem
- DecisionTSP:
  - Given a set of distances given by d for each pair of cities in \( v_1, \ldots, v_n \) and an integer \( D \), does there exist a tour that visits all cities having total weight at most \( D \)?
Hamiltonian-Cycle \( \leq P \) Decision TSP

- Define the reduction
  - Vertices \( V \) of \( G=(V,E) \) become cities
  - Set \( d(v_i, v_j) = 1 \) if \( (v_i, v_j) \in E \)
    
  \[ d(v_i, v_j) = \begin{cases} 
    1 & \text{if } (v_i, v_j) \in E \\
    2 & \text{if not} 
  \end{cases} \]
  - Set \( D = |V| \)

- Claim: There is a Hamiltonian cycle in \( G \) iff there is a tour of length \(|V|\)

---

Graph Colorability

- Defn: Given a graph \( G=(V,E) \), and an integer \( k \), a \( k \)-coloring of \( G \) is an assignment of up to \( k \) different colors to the vertices of \( G \) so that the endpoints of each edge have different colors.
- 3-Color: Given a graph \( G=(V,E) \), does \( G \) have a 3-coloring?
- Claim: 3-Color is NP-complete
- Proof: 3-Color is in NP:
  - Hint is an assignment of red, green, blue to the vertices of \( G \)
  - Easy to check that each edge is colored correctly

---

3-SAT \( \leq P \) 3-Color

- Reduction:
  - We want to map a 3-CNF formula \( F \) to a graph \( G \) so that
    - \( G \) is 3-colorable iff \( F \) is satisfiable

---

3-SAT \( \leq P \) 3-Color

- Variable Part:
  - in 3-coloring, variable colors correspond to some truth assignment (same color as T or F)

- Clause Part:
  - Add one 6 vertex gadget per clause connecting its ‘outer vertices’ to the literals in the clause
Any truth assignment satisfying the formula can be extended to a 3-coloring of the graph.

Any 3-coloring of the graph colors each gadget triangle using each color.

Any 3-coloring of the graph has an $F$ opposite the $O$ color in the triangle of each gadget.

Any 3-coloring of the graph has $T$ at the other end of the blue edge connected to the $F$.

Any 3-coloring of the graph yields a satisfying assignment to the formula.

- **Subset-Sum problem**
  - Given $n$ integers $w_1, \ldots, w_n$ and integer $W$
  - Is there a subset of the $n$ input integers that adds up to exactly $W$?
  - $O(nW)$ solution from dynamic programming but if $W$ and each $w_i$ can be $n$ bits long then this is exponential time.
3-SAT $\leq_p$ Subset-Sum

- Given a 3-CNF formula with $m$ clauses and $n$ variables
- Will create $2m + 2n$ numbers that are $m+n$ digits long
  - Two numbers for each variable $x_i$
    - $t_i$ and $f_i$ (corresponding to $x_i$ being true or $x_i$ being false)
  - Two extra numbers for each clause
    - $u_j$ and $v_j$ (filler variables to handle number of false literals in clause $C_j$)

Matching Problems

- Perfect Bipartite Matching
  - Given a bipartite graph $G=(V,E)$ where $V=X \cup Y$ and $E \subseteq X \times Y$, is there a set $M$ in $E$ such that every vertex in $V$ is in precisely one edge of $M$?
  - In $P$
  - Network Flow gives $O(nm)$ algorithm where $n=|V|$, $m=|E|$.

3-Dimensional Matching

- Theorem: 3-Dimensional Matching is NP-complete
- Proof:
  - We’ve already seen that it is in NP
  - 3-Dimensional Matching is NP-hard:
    - Reduction from 3-SAT
    - Given a 3-CNF formula $F$ we create a tripartite hypergraph (“hyperedges” are triangles) $G$ based on $F$ as follows

3-SAT $\leq_p$ 3-Dimensional Matching

- Variable part:
  - If variable $x_i$ occurs $r_i$ times in $F$ create $r_i$ red and $r_i$ green triangles linked in a circle, one pair per occurrence
  - Perfect matching $M$ must either use all the green edges leaving red tips uncovered ($x_i$ is assigned false) or all the red edges leaving all green tips uncovered ($x_i$ is assigned true)
**3-SAT ≤ₚ 3-Dimensional Matching**

- **Clause part:** Two new nodes per clause joined to each of its literals:
  \[ C_i (x_1 \lor \neg x_2 \lor \neg x_3) \]

- **Well-formed:** Each triangle has one of each type of node:

- **Correctness:**
  - If \( F \) has a satisfying assignment then choose the following triangles which form a perfect 3-dimensional matching in \( G \):
    - Either the red or the green triangles in the cycle for \( x_i \) - the opposite of the assignment to \( x_i \)
    - The triangle containing the first true literal for each clause and the two clause nodes
    - 2m slack triangles one per new pair of nodes to cover all the remaining tips

**Slack:** If there are \( m \) clauses then there are 3m variable occurrences. That means 3m total tips are not covered by whichever of red or green triangles not chosen. Of these, \( m \) are covered if each clause is satisfied. Need to cover the remaining 2m tips.

**Solution:** Add 2m pairs of slack vertices. Add triangles joining each pair with every tip!

**Correctness continued:**

- Each blue node in the cycle for each \( x_i \) is contained in exactly two triangles, exactly one of which must be in \( M \). If one triangle in the cycle is in \( M \), the others must be the same color. We use the color not used to define the truth assignment to \( x_i \)
- The two nodes for any clause must be contained in an edge which must also contain a third node that corresponds to a literal made true by the truth assignment. Therefore the truth assignment satisfies \( F \) so it is satisfiable.

**Is NP as bad as it gets?**

- NO! NP-complete problems are frequently encountered, but there are worse:
  - Some problems provably require exponential time.
    - Ex: Does \( M \) halt on input \( x \) in \( 2^{|x|} \) steps?
    - Some require \( 2^2, 2^3, 2^4, \ldots \) steps
  - And some are just plain uncomputable

**P vs NP**

- **Theory**
  - \( P = NP? \)
  - Open Problem!
  - 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