# NP Complete Problems

COMP 215 Lecture 20

# Complexity Theory

- Complexity theory is a research area unto itself.
- The central project is classifying problems as either tractable or intractable.
  - Tractable – Worst case polynomial time. $W(n) \in O(p(n))$
  - Intractable – Not worst case polynomial time. $W(n) \notin O(p(n))$
- Another area is computability theory.
  - The central project there is classifying problems as either computable or non-computable.
- Turing machines are important tools in both areas.
- We'll manage without Turing machines.

# Complexity Theory

- A problem can fall into one of three categories:
  - Tractable – we have a polynomial time algorithm.
    - Examples?
  - Intractable – it has been proved that there can be no polynomial time algorithm.
    - Examples?
  - Unknown – there is no polynomial time algorithm. But it has never been proved that such an algorithm cannot exist.
    - Examples?
- As a matter of convenience we will restrict ourselves to decision problems.

# The Sets P and NP

- P is the set of all problems that can be solved with polynomial time algorithms.
- NP is the set of all problems that can be solved with non-deterministic polynomial time algorithms. (?)
- A non-deterministic algorithm is allowed to guess a solution in one step.
  - Actually it is allowed to guess *all* solutions simultaneously.
- It must then verify the solution in polynomial time.
- In other words, for a problem to be in NP.
  - It need not be possible to *find* a solution in polynomial time.
  - But it must be possible to *check* a solution in polynomial time.

# Problems That Aren't in NP

- It is hard to prove that problems are not in NP for basically the same reason it is hard to prove that problems are not in P.
- Of course non-computable problems are clearly not in NP.
  - Halting problem etc.
- Problems with non-polynomial size output don't count because they aren't decision problems.

# Relationship Between P and NP

- Every problem in P is definitely in NP.
  - non-deterministically do nothing.
  - then apply the polynomial time algorithm to find (and trivially verify) the solution.
- It is unknown whether every problem in NP is also in P.
  - If so P = NP.
- Easy enough to show that $P \neq NP$
  - Find one problem that is in NP, but not in P :)
- A little trickier to show that P = NP.
  - We would need to have a polynomial time algorithm for every problem in NP.
  - This is not as daunting as it sounds...

# CNF Satisfiability

- A logical expression in conjunctive normal form is a sequence of OR clauses separated by ANDs:

$$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3) \wedge (x_1 \vee x_4)$$

- CNF satisfiability is the problem of determining whether or not there is some variable assignment that makes a CNF expression true.
- This problem is in NP.
- Currently no known polynomial algorithm.

# Cook-Levin Theorem

- CNF Satisfiability is an NP-Complete problem.
- For a problem to be NP-complete
  - It must be in NP.
  - A polynomial time algorithm for the problem must allow the solution of *any* problem in NP in polynomial time.
- We won't go through the proof.
- This means that to prove P=NP, we only need to find a polynomial time algorithm for this one problem.

# NP-Completeness

- Are there other NP-Complete problems?
  - There are many.
- Do we need a Cook-Levin theorem for each one?
  - No.
- The key is reductions (or translations).
- We say that problem A is polynomial time many-one reducible reducible to problem B if there is a polynomial time algorithm that converts any instance of problem A to an instance of problem B.
  - If we could decide B in polynomial time, we could decide A in polynomial time.
  - Notation: $A \propto B$ .

# NP-Completeness

- We can show that some new problem is NP-Complete by showing that.

$$\text{CNF Satisfiability} \propto \text{New Problem}$$

- Or, since any sequence of polynomial time transformations is still in polynomial time:

$$\text{CNF Satisfiability} \propto \text{Any Number of Other Problems} \propto \text{New Problem}$$

# The Clique Decision Problem is NP Complete

- A clique is a subset of vertices in a graph such that every vertex is connected to every other vertex.
- The clique decision problem: given a graph $G$, and an integer $k$, determine if $G$ contains a clique of size $k$.
- First question: is this problem in NP?
- Second question: is it NP-complete?

# Reducing CNF SAT to CLIQUE

- We are given a boolean expression composed of k clauses:

$$B = C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

- We convert this to a graph G = (V,E) as follows:
  - A new vertex is created for every literal in every clause:

$$V = \{[y, i] \text{ such that } y \text{ is a literal in clause } C_i\}$$

  - All vertices are connected *unless* they were generated from the same clause, or they represent negations of the same literal:

$$E = \{([y, i], [z, j]) \text{ such that } i \neq j \text{ and } \bar{z} \neq y\}$$

- Can this transformation be performed in polynomial time?

# Reducing CNF SAT to CLIQUE

- Now we need to show that *B* is satisfiable if and only if *G* has a clique of size *k*.
- If *B* is satisfiable *G* has a clique of size *k*.
  - If B is satisfiable then there is some way to assign variables so that each clause has at least one true literal.
  - Select a set of vertices *V'* such that

    $$V' = \{[y, i] \text{ such that } y \text{ is a true literal from } C_i\}$$

  - Every member of V' must be connected to every other because
    - each is from a different clause.
    - if *y* is true in both clauses, then it is impossible for it to be negated in one and not the other.

# Reducing CNF SAT to CLIQUE

- If $G$ has a clique of size $k$, $B$ is satisfiable.
  - Every member of the clique must have been generated from a different clause.
  - Select the set of variables represented by the k vertices in the clique:

$$S = \{ y \ such\ that\ [y, i] \in V' \}$$

  - Assign values to the variables as follows:

$$x_i = \begin{cases} true\ if\ x_i \in S \\ false\ if\ \bar{x}_i \in S \end{cases}$$

  - Assign all other literals arbitrarily.
  - This assignment guarantees that each clause has at least one true literal.

# More Reductions

- It is known that the Hamiltonian Circuits decision problem is NP-Complete.
  - Given a graph G is there a tour – a path that starts at one vertex, visits each vertex in the graph once, and ends up at the starting vertex.
- We can use this fact to prove that the undirected traveling salesperson decision problem is NP-Complete.
- We need a reduction from Hamiltonian Circuits to undirected TSP.

# Reducing from Hamiltonian Circuit to TSP

- Given an undirected graph $G$, create a completely connected graph $G'$ with the same vertices.
- Set the edge weights in $G'$ to 1 if the corresponding edge existed in $G$.
- Set them to 2 if not.
- $G$ contains a tour if and only if $G'$ has a tour of length $n$, where $n$ is the number of vertices.
- Therefore the undirected traveling salesperson decision problem is NP-Complete.

# Undirected TSP to TSP

- We now know that the undirected TSP is NP-Complete.
- What about the directed TSP?
- There is an easy reduction from undirected TSP to directed TSP.
- Given an undirected weighted graph *G*, create a directed weighted graph *G'* with two edges for every edge in *G*.
  - A forward edge and a backward edge, both with weight equal to the weight of the original edge in G.
- *G* has a tour of length *d* if and only if *G'* has a tour of length *d*.
- Therefore the directed TSP is NP-complete

# Facts About NP Complete Problems

- Many Many Problems are NP-Complete:
  - Graph 3-colorability, vertex cover, Hamiltonian path, subset-sum, 0-1 knapsack
- Obviously there is no known problem that is in P, and is also NP complete.
- There are problems that are in NP, are not known to be in P, and are not known to be NP complete:
  - Graph isomorphism.
  - Until 2002 the primes problem fell into this category.
- If an NP problem is found that is not in P and is not NP complete then, obviously, $P \neq NP$.

# The Class coNP

- A problem is in coNP if its complement is in NP.
- The complement of a problem is the problem that always has the opposite answer.
- The complement of the traveling salesman problem:
  - Does there *not* exist a path of length *k* in graph *G?*
- This problem is clearly in coNP – because we know that TSP is in NP.
- Is this problem in NP?
- Nobody knows.
- Not easy to see how to verify that graph does *not* have a length *k* path.

# Turing Reducibility

- Our previous notion of reducibility required converting an instance of one problem to an instance of another problem in polynomial time.
- Turing reducibility, denoted $A \propto_T B$ means that we can solve problem A using a hypothetical solution to problem B.
- Polynomial time Turing reducibility means that we can solve problem A in polynomial time using a hypothetical polynomial time algorithm for problem B.

# NP-Hard

- A problem is NP-Hard if every problem in NP is polynomial time Turing reducible to it.
  - A problem need not be in NP to be NP-Hard.
- The TSP decision problem is NP-complete.
- The TSP optimization problem is NP-Hard, but not NP-complete.
  - It cannot be in NP, because it is not a decision problem.
  - It is NP hard because any problem in NP can be reduced to the TSP decision problem, and
  - The TSP decision problem can be Turing reduced to the TSP optimization problem.

# NP-Easy, NP-Equivalent

- A problem *A* is NP-Easy if it is polynomial time Turing reducible to some problem *B* in NP.
  - $A \propto_T B$
- Problems in NP-Easy:
  - Every problem in P, every problem in NP.
  - Problems not in NP that have a polynomial time algorithm.
  - Some other problems not in NP – traveling salesperson optimization problem.
- A problem is NP-Equivalent if it is both NP-Hard and NP-Easy.
- P=NP iff there are polynomial time algorithms for all NP-Equivalent problems.

# More Complexity Classes

- PSPACE – is the class of decision problems that can be solved using polynomial space.
- P is contained in PSPACE.
  - Do you see why?

# More Complexity Classes

- PSPACE – is the class of decision problems that can be solved using polynomial space.
- P is contained in PSPACE.
  - Because an algorithm that finishes in polynomial time only has time to fill polynomial space.
- NP is contained in PSPACE.
  - This is harder to see.
- It is not known whether P = NP = PSPACE.
- EXP is the class of problems that can be solved in exponential time.
  - EXP contains PSPACE. (Why?)
  - It is known that $P \neq EXP$.

# EXPSPACE

- I'll bet you can guess...
- Obviously EXP is in EXPSPACE

# Example of a PSPACE Complete Problem

- TQBF is PSPACE-Complete.
- TQBF is the problem of determining whether a fully quantified boolean formula is true.
- A fully quantified boolean formula looks like:

$$\forall x_1 \exists x_2 [x_1 = x_2]$$

- $\forall$ *and* $\exists$ are quantifiers. A formula is fully qualified if every variable is quantified.
- A PSPACE-hard problem is finding an optimal policy in a finite horizon partially observable Markov decision problem.