NP-Completeness

Lecture for CS 302

Traveling Salesperson Problem

- You have to visit n cities
- You want to make the shortest trip
- How could you do this?
- What if you had a machine that could guess?

Non-deterministic polynomial time

- Deterministic Polynomial Time: The TM takes at most $O(n^c)$ steps to accept a string of length n
- Non-deterministic Polynomial Time: The TM takes at most $O(n^c)$ steps on each computation path to accept a string of length n

The Class P and the Class NP

- P = { L | L is accepted by a deterministic Turing Machine in polynomial time }
- NP = { L | L is accepted by a non-deterministic Turing Machine in polynomial time }
- They are sets of languages

P vs NP?

- Are non-deterministic Turing machines really more powerful (efficient) than deterministic ones?
- Essence of P vs NP problem

Does Non-Determinism matter?

Finite Automata?

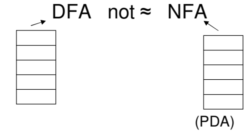
No!

DFA ≈ NFA

Push Down Automata?

Yes!

DFA not ≈ NFA



P = NP?

- No one knows if this is true
- How can we make progress on this problem?

Progress

- P = NP if every NP problem has a deterministic polynomial algorithm
- We could find an algorithm for every NP problem
- Seems… hard…
- We could use polynomial time reductions to find the “hardest” problems and just work on those

Reductions

- Real world examples:
  - Finding your way around the city reduces to reading a map
  - Traveling from Richmond to Cville reduces to driving a car
  - Other suggestions?

Polynomial time reductions

- PARTITION = { $n_1, n_2, \ldots n_k$ | we can split the integers into two sets which sum to half }
- SUBSET-SUM = { $<n_1, n_2, \ldots n_k, m>$ | there exists a subset which sums to m }
- 1) If I can solve SUBSET-SUM, how can I use that to solve an instance of PARTITION?
- 2) If I can solve PARTITION, how can I use that to solve an instance of SUBSET-SUM?

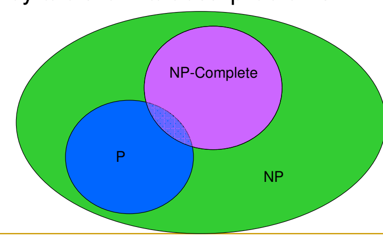
Polynomial Reductions

- 1) Partition REDUCES to Subset-Sum
  - Partition $<_p$ Subset-Sum
- 2) Subset-Sum REDUCES to Partition
  - Subset-Sum $<_p$ Partition
- Therefore they are equivalently hard

- How long does the reduction take?
- How could you take advantage of an exponential time reduction?

NP-Completeness

- How would you define NP-Complete?
- They are the “hardest” problems in NP

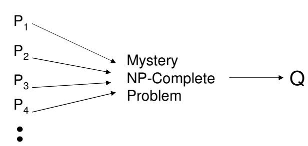


Definition of NP-Complete

- Q is an NP-Complete problem if:
- 1) Q is in NP
- 2) every other NP problem polynomial time reducible to Q

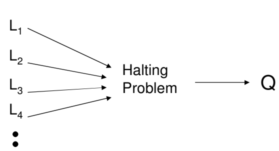
Getting Started

- How do you show that EVERY NP problem reduces to Q?
- One way would be to already have an NP-Complete problem and just reduce from that



Reminder: Undecidability

- How do you show a language is undecidable?
- One way would be to already have an undecidable problem and just reduce from that



SAT

- SAT = { f | f is a Boolean Formula with a satisfying assignment }

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

- Is SAT in NP?

Cook-Levin Theorem (1971)

- SAT is NP-Complete

If you want to see the proof it is Theorem 7.37 in Sipser (assigned reading!) or you can take CS 660 – Graduate Theory. You are not responsible for knowing the proof.

3-SAT

- 3-SAT = { *f* | *f* is in Conjunctive Normal Form, each clause has exactly 3 literals and *f* is satisfiable }
- 3-SAT is NP-Complete
- (2-SAT is in P)

NP-Complete

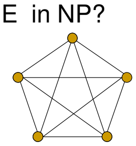
- To prove a problem is NP-Complete show a polynomial time reduction from 3-SAT
- Other NP-Complete Problems:
  - PARTITION
  - SUBSET-SUM
  - CLIQUE
  - HAMILTONIAN PATH (TSP)
  - GRAPH COLORING
  - MINESWEEPER (and many more)

NP-Completeness Proof Method

- To show that Q is NP-Complete:
- 1) Show that Q is in NP
- 2) Pick an instance, R, of your favorite NP-Complete problem (ex: Φ in 3-SAT)
- 3) Show a polynomial algorithm to transform R into an instance of Q

Example: Clique

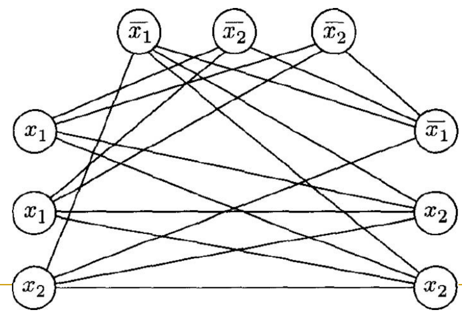
- CLIQUE = { <G,k> | G is a graph with a clique of size k }
- A clique is a subset of vertices that are all connected
- Why is CLIQUE in NP?



Reduce 3-SAT to Clique

- Pick an instance of 3-SAT, Φ, with *k* clauses
- Make a vertex for each literal
- Connect each vertex to the literals in other clauses that are not the negation
- Any k-clique in this graph corresponds to a satisfying assignment

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

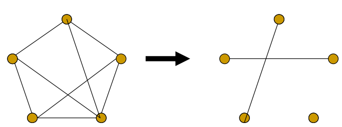


Example: Independent Set

- INDEPENDENT SET = { <G,k> | where G has an independent set of size k }
- An independent set is a set of vertices that have no edges
- How can we reduce this to clique?

Independent Set to CLIQUE

- This is the *dual problem*!



Doing Your Homework

- Think hard to understand the structure of both problems
- Come up with a “widget” that exploits the structure
- These are hard problems
- Work with each other!
- Advice from Grad Students: PRACTICE THEM

Take Home Message

- NP-Complete problems are the HARDEST problems in NP
- The reductions MUST take polynomial time
- Reductions are hard and take practice
- Always start with an instance of the known NP-Complete problem
- Next class: More examples and Minesweeper!

Papers

- Read *one* (or more) of these papers:
  - *March Madness is (NP-)Hard*
  - *Some Minesweeper Configurations*
  - *Pancakes, Puzzles, and Polynomials: Cracking the Cracker Barrel*

Each paper proves that a generalized version of a somewhat silly problem is NP-Complete by reducing 3SAT to that problem (March Madness pools, win-able Minesweeper configurations, win-able pegboard configurations)

- and Knuth’s *Complexity of Songs*

Optional, but it is hard to imagine any student who would not benefit from reading a paper by Donald Knuth including the sentence, “However, the advent of modern drugs has led to demands for still less memory…”