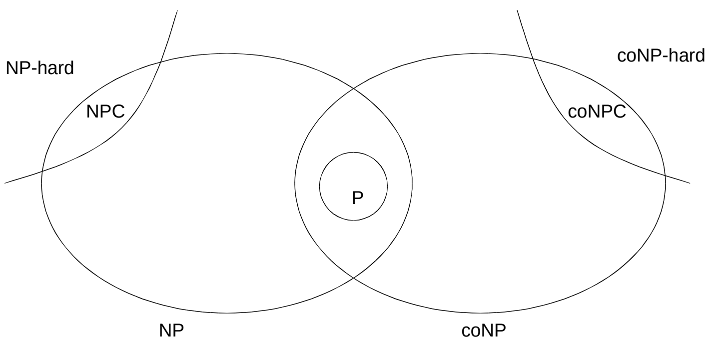
CSC 363 Summer 2005

Lecture Week 12

NP-completeness

- Cook, Levin, 1971: showed SAT is *NP-complete*
- Karp, 1972: showed 21 well-known problems are *NP-complete*
  - CLIQUE, VC, HAM-CYCLE, SUBSET-SUM
- Since then, hundreds of problems appearing in practice have been shown to be *NP-complete*
- Showing a problem L is *NP-hard* is very strong evidence there is no polynomial time algorithm solving L.
  - Otherwise, P=NP, and all these problems have polynomial time algorithms.

Most Believed View of the World



1972, P or NP-complete?

- Linear Programming ("LP")
  - given some constraint functions and an objective function, find a solution which satisfies the constraints and optimizes the objective
  - Simplex algorithm known, but not polynomial
  - Khachiyan, 1979: ellipsoid algorithm, LP is in P

1972, P or NP-complete?

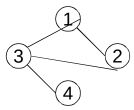
- Primality Testing ("PRIME")
  - given an integer *n* in binary (*log n* bits), decide whether it is prime or not
  - cannot try all divisors from 2 to n (or $n^{½}$), not polynomial in *log n*
  - easy: PRIME is in coNP
    - if *n* is not prime, guess divisor and check it
  - harder: PRIME is in NP
  - probably not NP-complete, unless NP=coNP

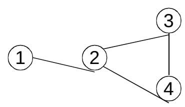
1972, P or NP-complete?

- Agrawal, Kayal, Saxena, 2002: PRIME is in P
  - there exists an algorithm which, given integer *n*, decides whether *n* is prime in time polynomial in *log n*
  - the output is only YES/NO
  - we still do not know how to (or if we can) actually compute a divisor of *n*
  - some cryptographic systems assume this is hard

1972, P or NP-complete?

- Graph Isomorphism
  - given two graphs, are they a permutation of each other?
  - to obtain isomorphic graphs:
    - draw the graph with all its edges
    - erase node labels
    - write down a new label for every node





1972, P or NP-complete?

- GRAPH-ISOMORPHISM is in NP
  - guess a permutation π (*n log n* bits)
  - check (u,v) in $E(G_1)$ iff $(\pi(u),\pi(v))$ in $E(G_2)$
- not known or believed to be in P
- not known or believed to be NP-complete
- still open today..

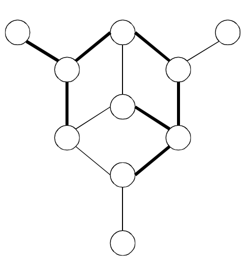
If P≠NP..

- There are problems inside NP which are neither in P nor NP-complete
- There are infinitely many classes, getting harder and harder, strictly between P and NP-complete
- GRAPH-ISOMORPHISM conjectured to be strictly in between P and NP-complete

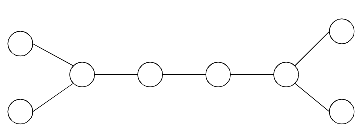
SUBGRAPH-ISOMORPHISM

- however, SUBGRAPH-ISOMORPHISM is NP-complete!
  - given G, H
  - can delete nodes of G, together with incident edges
  - do not delete any edges between remaining nodes
  - only afterwards match the remains of G with H
  - harder, because not clear what we should delete

SUBGRAPH-ISOMORPHISM



G



H

Decision, Search and Optimization Problems

- We developed our theory using languages, or, equivalently, *decision problems*
  - i.e. given input *x*, output YES or NO
- In practice, many problems are *search problems*
  - i.e. given input *x*, output some object *y*, if one exists
- Yet more general are *optimization problems*
  - i.e. given a set of input constraints and an objective function, output an object which satisfies all the constraints and optimizes (minimizes/maximizes) the objective function

Search Problems

- Given formula, output a satisfying assignment if one exists
- Given a graph *G* and an integer *k*, output a clique/vertex cover/independent set of *G* of size *k*, if one exists
- Given a set of numbers and a target, output a subset of those numbers which sum up to the target, if such a subset exists

Optimization Problems

- Given graph *G*, output a clique *C* of *G* of maximum size
  - constraint: *C* is a set of vertices in *G*
  - constraint: *C* is a clique
  - objective: maximize |*C*|
- Given a set of weights w_1, ..., w_m and a capacity *W*, output a subset *S* of those weights which have a maximum weight, while not exceeding *W*
  - constraint: *S* is a subset of 1, ..., m
  - constraint: sum of weights w_i with i in S is at most *W*
  - objective: maximize sum of weights w_i with i in S

Relation Between Decision, Search and Optimization Problems

- In general:
  - decision problem is “easiest”
  - search problem is “harder”
  - optimization problem is “the hardest”
- Meaning:
  - IF we can solve the search problem in polytime, THEN we can solve the decision problem in polytime
- In most cases, *but not all*, these problems are polytime equivalent:
  - IF we can solve the decision problem in polytime, THEN we can solve the optimization problem in polytime
  - e.g. MAX-CLIQUE, MIN-VC, MAX-SK
- Notable exception:
  - PRIME is in P, but PRIME-SEARCH maybe not in P

Dealing with NP-completeness

- NP-complete problems appear in practice
- We can’t hope to solve them in polynomial time, but we still have to solve them somehow..
- Approaches
  - problem restrictions
  - heuristics
  - randomization
  - approximation

Problem Restrictions

- Maybe problem statement is too general, make extra assumptions about input:
  - degree of vertices is bounded in a graph
  - graph is planar
  - weights are not too large
- Hopefully, problem becomes easier
- Example: a graph with maximum vertex degree *d* can be coloured with *d+1*colours by a simple Greedy algorithm

Heuristics

- Use an algorithm that works well in most practical cases, but
  - output not necessarily correct in all cases
  - runtime not necessarily good in all cases
  - worst-case runtime may be unknown or exponential, hopefully it doesn’t occur often
- Example: Simplex algorithm for solving Linear Programming
  - worst case runtime is exponential
  - in practice it works well
  - still used even after polytime algorithms discovered

Randomization

- Allow Turing Machine to “flip coins”
  - output is correct with some high probability
  - runtime is polynomial with some high probability
- BPP = class of decision problems which have algorithms with
  - worst case runtime is always polynomial
  - probability of error < 1/3
- Idea: if probability of error can be made very low (say, 2^{-100}), then it is more likely that the machine will crash then that it will give a wrong answer
- Example: Primality testing
- However, conjectured BPP≠NP

Approximation Algorithms

- For optimization problems, compromise on optimizing the objective function
- Output a solution which
  - satisfies all the constraints
  - not necessarily optimal
- Runtime is polynomial
- Need some measure of how useful the algorithm really is
- Approximation ratio = ratio between
  - objective value achieved by some (hypothetical) optimal solution
  - objective value achieved by algorithm
  - always >= 1
- How can we argue about an optimal solution??

Approximation Algorithms

- Minimization problem *P*:
  - input constraints *C*, objective function *f*
  - output a solution *S* satisfying *C*
  - minimize *f(S)*
- An *r*-approximation algorithm for *P*
  - output a solution *S* satisfying *C*
  - let *O* be an optimal solution
  - *f(O) <= f(S)* **<=** *r*f(O)*
  - runtime is polynomial
- For a maximization problem
  - *(1/r)*f(O)* **<=** *f(S) <= f(O)*

Approximation Algorithms

- Confusing: must compare the objective value achieved with the optimal objective value *without computing the optimal objective value*
  - Every given instance has some optimum solution
  - Approximation algorithm must get “close enough” to that optimum
- Even more confusing: maybe for “small” instances, algorithm gets closer to the optimum than for “large” instances
  - approximation ratio may depend on the size of the input

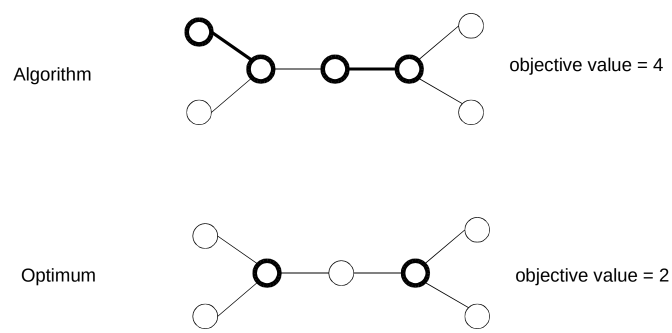
Vertex Cover

- Consider following algorithm:

```
on input G:
C = empty
E’ = all edges in G
while E’ is not empty
        let (u,v) be some edge in E’
        C = C + u + v
        remove from E’ every edge touching u or v
return C
```

- Runtime: $O(n^2)$. Loop executed O(n) times.

Vertex Cover



Vertex Cover

- Claim: the previous algorithm is a 2-approximation
- Output *C* is a vertex cover
  - edges are removed only when one of their endpoints is included in the cover
- Let *O* = an optimal vertex cover
  - so, |O| <= |C|
- Let *A* = set of edges picked in main loop
  - edges in A share no endpoints
  - to cover all edges in A, *any* vertex cover needs at least |A| vertices
  - in particular, |O| >= |A|
  - but |C| = 2*|A|
  - so, |C| <= 2*|O|

Approximation Algorithms

- The 2-approximation to MIN-VC is pretty simple
  - can we do better? maybe 3/2-approximation?
- Does every optimization problem have such a "nice" approximation algorithm?

Inapproximability Results

- Results of the form
  - " If there exists an *r*-approximation to this problem, then something very unlikely happens"
- They are "negative" results, seen as strong evidence that an *r*-approximation algorithm does not exist
- Highly technical
- Example:
  - If there exists a 1.36-approximation to Vertex Cover, then P=NP

Travelling-Salesman Problem

- Theorem: If there exists a constant ratio approximation to TSP, then P=NP
  - holds for *any constant*
  - maybe a (*log n*)-approximation exists
- Suppose there exist a constant *r >= 1* and a polynomial time algorithm *A* such that *A* is an *r*-approximation for TSP
- We develop a new algorithm *B*, using *A*, that solves HAM-CYCLE in polynomial time
- Since HAM-CYCLE is NP-complete, this implies P=NP

Travelling-Salesman Problem

- Algorithm B

  on input G:

  G' = add all missing edges to *G*

  define weight function *w*:

  w(original edge) = *1*

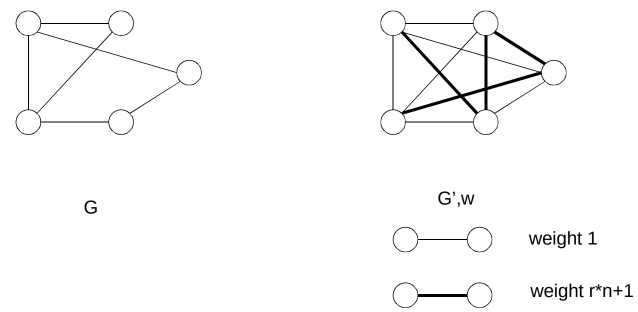
  w(new added edge) = *r*n + 1*

  run *A* on input *G',w* to get a cycle C

  if *w(C) <= r*n*, output YES

  otherwise, output NO
- Runtime: $O(n^2 + n^s)$, where $O(n^s)$ is runtime of A

Travelling-Salesman Problem



Travelling-Salesman Problem

- If *G* has a hamcycle *D*, then *D* is a hamcycle in *G'* with weight *n*
  - no other hamcycle in *G'* has smaller weight
  - *A* is an *r*-approximation, and outputs *C*
  - so, $w(C) <= r*n$
- If G has no hamcycle, then *any* hamcycle in *G'* must use at least one "heavy" edge
  - weight of any hamcycle of *G'* >= $r*n+1$
  - so, $w(C) > r*n$