CS 420: Advanced Algorithm Design and Analysis
Spring 2015 – Lecture 17

Department of Computer Science
University of British Columbia

March 10, 2015

Announcements

Guest Lecturer... Patrice Bellville

Assignments...

- Asst6/7...(due March 19)

Midterm III...

- Q/A session...March 24; 5:30-7:00; DMPT 110
- Exam...March 25; 5:30-7:00; DMPT 110
- ...on *all* course material up to and including March 19 lecture

Announcements (cont.)

Readings...

- matchings and network flows [Kleinberg&Tardos, Chapt. 7], [Cormen et al., Chapt. 26], [Dasgupta et al., Chapter 7]
- reductions and NP-hardness [Kleinberg&Tardos, Chapt. 8, 11], [Cormen et al., Chapt. 34,35]

Last day...

Matchings and Network Flows

- relationship with bipartite matchings (cont.)
- two applications

Reductions and relative hardness of problems

- reductions
  - definitions
  - examples of reductions encountered in course
  - role(s) in establishing relative hardness

Today...

Reductions and relative hardness of problems

- reductions...treated more formally
- overview of problems with efficient algorithms
  ... and related problems with no known efficient algorithm
- the complexity classes **P** and **NP**
- **NP**-hardness and **NP**-completeness

Reductions and relative hardness of problems

We write A $\textcircled{\leq}$ B to denote the fact that problem A is *reducible to* problem B. Informally, this means

1. a subroutine for solving problem B can be used as a *black box* in solving problem A
2. instances of problem A can be transformed to instances of problem B in such a way that a solution to the latter can be transformed back into a solution of the former

Reductions and relative hardness of problems

We have seen many examples throughout the course...

- element-distinctness $\textcircled{$\leq$}$ closest-pair $\textcircled{$\leq$}$ sorting
- transitive-closure $\textcircled{$\leq$}$ Boolean-matrix-product
- all-pairs-shortest-paths-with-arbitrary-weights
  $\textcircled{$\leq$}$ all-pairs-shortest-paths-with-non-negative-weights
- edit-distance (sequence-alignment)
  $\textcircled{$\leq$}$ single-source-shortest-path

Reductions and relative hardness of problems

We have seen many examples throughout the course...

- bipartite-matching ⧀ bipartite-vertex-cover ⧀ bipartite-matching
- bipartite-matching ⧀ unit-capacitated-network-flow ⧀ bipartite-matching
- integer-capacitated-network-flow ⧀ unit-capacitated-network flow

Reductions and relative hardness of problems

and in homework assignments...

- maximum-width s, t-path ≤⃝ maximum-weight-spanning-tree
- generalized network-flow ≤⃝ standard network-flow
- minimum-colour-transition-path ≤⃝ min-cost-shortest-path
- vertex-cover ≤⃝ minimum-colour-path

Reductions and relative hardness of problems

Often very general problems, such as sorting, min-cost-paths, network-flow or linear-programming, serve as the target of reductions.

Reductions and relative hardness of problems

Often very general problems, such as sorting, min-cost-paths, network-flow or linear-programming, serve as the target of reductions.

Other times we are interested in reducing a general case to a restricted case, or demonstrating reductions that go in both directions.

Reductions and relative hardness of problems

In the event that the reduction *overhead* is low, A $\textcircled{\leq}$ B implies:

Reductions and relative hardness of problems

In the event that the reduction *overhead* is low, A $\textcircled{$\leq$}$ B implies:

- solving problem A is *not much harder* than solving problem B (upper bounds on the cost of solving B translate to upper bounds on the cost of solving A)

Reductions and relative hardness of problems

In the event that the reduction *overhead* is low, $A \mathrel{\bigcirc\mkern-15mu\leq} B$ implies:

- solving problem A is *not much harder* than solving problem B (upper bounds on the cost of solving B translate to upper bounds on the cost of solving A)
- solving problem B is *not much easier* than solving problem A (lower bounds on the cost of solving A translate to lower bounds on the cost of solving B)

Reductions and relative hardness of problems

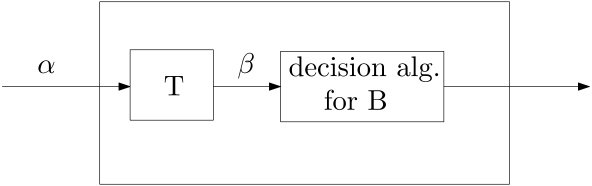
In order to be more precise about “relative hardness” it helps to take note of the actual cost of the reduction itself. We write A $\text{\textcircled{$\leq$}}_{t(n)}$B to denote the fact that the reduction from A to B can be carried out in $t(n)$ time, for instances of size n.

Reductions and relative hardness of problems

When using reductions to establish lower bounds, it is useful to focus on *decision versions* of problems rather than *optimization versions*: that is, determine if a solution of at least some value exists (yes/no) as opposed to determining the solution with the largest/best value.

- there is a long history of using *language recognition* problems as complexity benchmarks
- by focusing on decision problems, we avoid complexity that arises simply from describing the optimal solution
- optimization problems can often be expressed as a sequence of decision problems

Reduction A $\textcircled{\leq}$ B between decision problems



α

T

β

decision alg.
for B

decision alg. for A

A review of some graph problems

A review of some graph problems

left column

-spanning trees
    min cost
    maximum width

A review of some graph problems

left column

-spanning trees
    min cost
    maximum width

-path problems
    min-cost
    min colour-transitions
    Eulerian path

A review of some graph problems

left column

-spanning trees
    min cost
    maximum width

-path problems
    min-cost
    min colour-transitions
    Eulerian path

-graph colouring
    2-colouring (bipartite)
    4-colouring (planar graph)

A review of some graph problems

left column	**right column**
-spanning trees	
min-cost	bounded-degree MST
maximum width	bounded-diameter MST
-path problems	
min-cost	longest (simple) path
min colour-transitions	min total colours
Eulerian path	Hamiltonian path
-graph colouring	
2-colouring (bipartite)	3-colourability
4-colouring (planar graph)	3-colouring (planar graph)

A review of some graph problems

left column

-matchings etc.
  max size (bipartite)
  vertex cover (bipartite)
  general (non-bipartite)
  b-matchings

A review of some graph problems

left column

-matchings etc.
  max size (bipartite)
  vertex cover (bipartite)
  general (non-bipartite)
  b-matchings

-network flows etc.
  max value
  integral/general capacities
  vertex capacities
  minimum cut
  edge/vertex-disjoint paths

A review of some graph problems

left column	**right column**
-matchings etc.	
max size (bipartite)	3-d matching (triangle cover)
vertex cover (bipartite)	maximum independent set
general (non-bipartite)	vertex cover (tripartite)
b-matchings	
-network flows etc.	
max value	flows with edge costs
integral/general capacities	undirected flows with lower bounds
vertex capacities	
minimum cut	vertex-disjoint connecting paths
edge/vertex-disjoint paths	

A review of some graph problems

All of the **left column** problems have *efficient* solutions: their decision versions belong to the complexity class **P**, defined to be the family of decision problems (languages) that can be decided (recognized) in time bounded by some polynomial in the input size.

A review of some graph problems

All of the **left column** problems have *efficient* solutions: their decision versions belong to the complexity class **P**, defined to be the family of decision problems (languages) that can be decided (recognized) in time bounded by some polynomial in the input size.

Why are we interested in *polynomial time*?

- generous definition of tractable
- often equates to tractable in practice
- closure properties (composition)
- invariance under natural computation models

A review of some graph problems

None of the **right column** problems are known to have *efficient* solutions.

A review of some graph problems

None of the **right column** problems are known to have *efficient* solutions.

Nevertheless, their decision versions all admit efficient *certification*; i.e. a short proof/certificate that the answer is YES. They all belong to the complexity class **NP** is defined to be the family of decision problems (languages) whose membership can be certified/verified in time bounded by some polynomial in the input size.

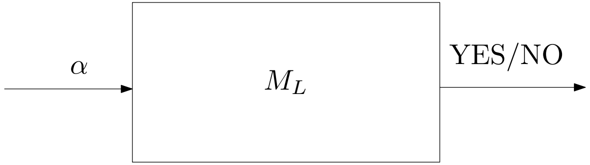
A review of some graph problems

None of the **right column** problems are known to have *efficient* solutions.

Nevertheless, their decision versions all admit efficient *certification*; i.e. a short proof/certificate that the answer is YES. They all belong to the complexity class **NP** is defined to be the family of decision problems (languages) whose membership can be certified/verified in time bounded by some polynomial in the input size.

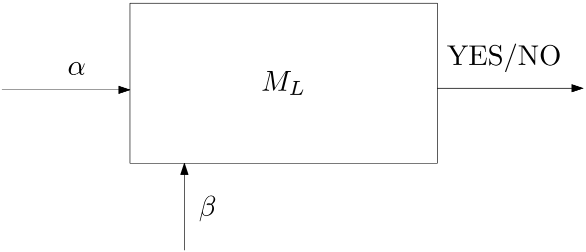
NP stands for *non-deterministic* polynomial-time: certification corresponds to acceptance by a non-deterministic machine.

Deterministic language acceptance



Machine M_L *accepts* L if:
$\alpha \in L$ if and only if M_L outputs YES on input α

Non-deterministic language acceptance



Machine M_L *non-deterministically accepts* L if:
$\alpha \in L$ if and only if there exists a string β such that M_L outputs YES on input (α, β).

The complexity classes **P** and **NP**

P denotes the set of languages that can be (deterministically) accepted in time bounded by some polynomial in the input length.
NP denotes the set of languages that can be (non-deterministically) accepted in time bounded by some polynomial in the input length.

Note:

- deterministic acceptance is equivalent to deterministic decision (**P** is closed under complement)
- **NP** is not known to be closed under complement

The complexity classes **P** and **NP**

It turns out that all of the **right column** problems are as hard as any problem in **NP**, up to polynomial factors, which is abbreviated **NP**-hard. Since they are also in **NP** they belong to the class **NP**-complete.

NP-hard problems have the property that they have polynomial-time solutions (i.e. they belong to **P** if and only if **P**=**NP**, i.e. all problems in **NP** have polynomial-time solutions.

The complexity classes **P** and **NP**

How could we possibly show that some problem X is **NP**-hard? We don't even know all of the problems in **NP**!

The complexity classes **P** and **NP**

How could we possibly show that some problem X is **NP**-hard? We don't even know all of the problems in **NP**!

- it is straightforward once we know some **NP**-hard problem A: simply demonstrate $\mathrm{A} \mathbin{\bigcirc\mkern-15mu\leq}_{t(n)} \mathrm{X}$, where $t(n)$ is some polynomial in n.
- the real breakthrough was the demonstration of a *first* **NP**-hard problem

Coming up...

Reductions and relative hardness of problems

- some examples of reductions establishing **NP**-hardness and **NP**-completeness
- approximation algorithms for hard problems