CS 5114: Theory of Algorithms

Clifford A. Shaffer

Department of Computer Science
Virginia Tech
Blacksburg, Virginia

Spring 2014

Title page

Students should be familiar with inductive proofs, recursion, data structures, and programming at the CS3114 level.

Reductions

A **reduction** is a transformation of one problem to another

Purpose: To compare the relative difficulty of two problems

Example:
Sorting reals **reduces to** (in linear time) the problem of finding a convex hull in two dimensions

- Use CH as a way to solve sorting

We argued that there is a lower bound of $\Omega(n \log n)$ on finding the convex hull since there is a lower bound of $\Omega(n \log n)$ on sorting

This example we have already seen.

NOT reduce CH to sorting – that just means that we can make CH as hard as sorting! Using sorting isn't necessarily the only way to solve the CH problem, perhaps there is a better way. So just knowing that sorting is ONE WAY to solve CH doesn't tell us anything about the cost of CH. On the other hand, by showing that we can use CH as a tool to solve sorting, we know that CH cannot be faster than sorting.

Reduction Notation

- We denote names of problems with all capital letters.
  - Ex: SORTING, CONVEX HULL
- What is a problem?
  - A relation consisting of ordered pairs (**I**, **SLN**).
  - **I** comes from the set of **instances** (allowed inputs).
  - **SLN** is the solution to the problem for instance **I**.
- Example: SORTING = (**I**, **SLN**).
  I is a finite subset of $\mathcal{R}$.
  - Prototypical instance: $\{x_1, x_2, ..., x_n\}$.
- **SLN** is the sequence of reals from **I** in sorted order.

no notes

Black Box Reduction (1)

The job of an algorithm is to take an instance **I** and return a solution **SLN**, or to report that there is no solution.

A **reduction** from problem **A**(**I**, **SLN**) to problem **B**(**I**', **SLN**') requires two transformations (functions) T, T'.
T: $\mathbf{I} \Rightarrow \mathbf{I}'$

- Maps instances of the first problem to instances of the second.

T': $\mathbf{SLN}' \Rightarrow \mathbf{SLN}$

- Maps solutions of the second problem to solutions of the first.

no notes

Black Box Reduction (2)

Black box idea:

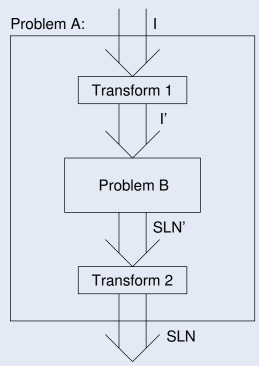
1. Start with an instance **I** of problem **A**.
2. Transform to an instance **I**’ $=$ T(**I**), an instance of problem **B**.
3. Use a “black box” algorithm for **B** as a subroutine to find a solution **SLN**’ for **B**.
4. Transform to a solution **SLN** $=$ T’(**SLN**’), a solution to the original instance **I** for problem **A**.

2014-04-09 CS 5114

Black Box Reduction (2)

no notes

Black Box Diagram



2014-04-09 CS 5114

Black Box Diagram

no notes

More Notation

If (**I**, **SLN**) reduces to (**I**$'$, **SLN**$'$), write:
(**I**, **SLN**) $\leq$ (**I**$'$, **SLN**$'$).

This notation suggests that (**I**, **SLN**) is **no harder than** (**I**$'$, **SLN**$'$).

Examples:

- SORTING $\leq$ CONVEX HULL

The time complexity of T and T’ is important to the time complexity of the black box algorithm for (**I**, **SLN**).

If combined time complexity is $O(g(n))$, write:
(**I**, **SLN**) $\leq_{O(g(n))}$ (**I**$'$, **SLN**$'$).

2014-04-09 CS 5114

More Notation

Sorting is no harder than Convex Hull. Conversely, Convex Hull is *at least as hard as* Sorting.

If T or T’ is expensive, then we have proved nothing about the relative bounds.

Reduction Example

SORTING $=$ (**I**, **SLN**)
CONVEX HULL $=$ (**I**’, **SLN**’).

1. **I** $= \{x_1, x_2, ..., x_n\}$.
2. T(**I**) $=$ **I**’ $= \{(x_1, x_1^2), (x_2, x_2^2), ..., (x_n, x_n^2)\}$.
3. Solve CONVEX HULL for **I**’ to give solution **SLN**’ $= \{(x_{i[1]}, x_{i[1]}^2), (x_{i[2]}, x_{i[2]}^2), ..., (x_{i[n]}, x_{i[n]}^2)\}$.
4. T’ finds a solution to **I** from **SLN**’ as follows:
   1. Find $(x_{i[k]}, x_{i[k]}^2)$ such that $x_{i[k]}$ is minimum.
   2. Y $= x_{i[k]}, x_{i[k+1]}, ..., x_{i[n]}, x_{i[1]}, ..., x_{i[k-1]}$.

- For a reduction to be useful, T and T’ must be functions that can be computed by algorithms.
- An algorithm for the second problem gives an algorithm for the first problem by steps 2 – 4.

2014-04-09 CS 5114

Reduction Example

no notes

Notation Warning

Example: SORTING $\leq_{O(n)}$ CONVEX HULL.

WARNING: $\leq$ is NOT a partial order because it is NOT antisymmetric.

SORTING $\leq_{0(n)}$ CONVEX HULL.

CONVEX HULL $\leq_{O(n)}$ SORTING.

But, SORTING $\neq$ CONVEX HULL.

no notes

Bounds Theorems

Lower Bound Theorem: If $P_1 \leq_{O(g(n))} P_2$, there is a lower bound of $\Omega(h(n))$ on the time complexity of P_1, and $g(n) = o(h(n))$, then there is a lower bound of $\Omega(h(n))$ on P_2.

Example:

- SORTING $\leq_{O(n)}$ CONVEX HULL.
- $g(n) = n$. $h(n) = n \log n$. $g(n) = o(h(n))$.
- Theorem gives $\Omega(n \log n)$ lower bound on CONVEX HULL.

Upper Bound Theorem: If P_2 has time complexity $O(h(n))$ and $P_1 \leq_{O(g(n))} P_2$, then P_1 has time complexity $O(g(n) + h(n))$.

Notice o, not O.So, given good transformations, both problems take at least $\Omega(P_1)$ and at most $O(P_2)$.

System of Distinct Representatives (SDR)

Instance: Sets $S_1, S_2, \cdots, S_k$.
Solution: Set $R = \{r_1, r_2, \cdots, r_k\}$ such that $r_i \in S_i$.
Example:
Instance: $\{1\}, \{1, 2, 4\}, \{2, 3\}, \{1, 3, 4\}$.
Solution: $R = \{1, 2, 3, 4\}$.

Reduction:

- Let n be the size of an instance of SDR.
- SDR $\leq_{O(n)}$ BIPARTITE MATCHING.
- Given an instance of $S_1, S_2, \cdots, S_k$ of SDR, transform it to an instance $G = (U, V, E)$ of BIPARTITE MATCHING.
- Let $S = \cup_{i=1}^{k} S_i$. $U = \{S_1, S_2, \cdots, S_k\}$.
- $V = S$. $E = \{(S_i, x_j) | x_j \in S_i\}$.

Since it is a set, there are no duplicates.

Or, $R = \{1, 4, 2, 3\}$

U is the sets.
V is the elements from all of the sets (union the sets).
E matches elements to sets.

SDR Example

$\{1\}$	1
$\{1, 2, 4\}$	2
$\{2, 3\}$	3
$\{1, 3, 4\}$	4

A solution to SDR is easily obtained from a **maximum matching** in G of size k.

Need better figure here.

Simple Polygon Lower Bound (1)

- SIMPLE POLYGON: Given a set of n points in the plane, find a simple polygon with those points as vertices.
- SORTING $\leq_{O(n)}$ SIMPLE POLYGON.
- Instance of SORTING: $\{x_1, x_2, \cdots, x_n\}$.
  - In linear time, find $M = \max |x_i|$.
  - Let C be a circle centered at the origin, of radius M.
- Instance of SIMPLE POLYGON:

$$\{(x_1, \sqrt{M^2 - x_i^2}), \cdots, (x_n, \sqrt{M^2 - x_n^2})\}.$$

  All these points fall on C in their sorted order.
- The only simple polygon having the points on C as vertices is the convex one.

Need a figure here showing the curve.

Simple Polygon Lower Bound (2)

- As with CONVEX HULL, the sorted order is easily obtained from the solution to SIMPLE POLYGON.
- By the Lower Bound Theorem, SIMPLE POLYGON is $\Omega(n \log n)$.

no notes

Matrix Multiplication

Matrix multiplication can be reduced to a number of other problems.

In fact, certain special cases of MATRIX MULTIPLY are equivalent to MATRIX MULTIPLY in asymptotic complexity.

SYMMETRIC MATRIX MULTIPLY (SYM):

- Instance: a symmetric $n \times n$ matrix.

MATRIX MULTIPLY $\leq_{O(n^2)}$ SYM.

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix} = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}$$

Clearly SYM is not harder than MM. Is it easier? No...

So, having a good SYM would give a good MM. The other way of looking at it is that SYM is just as hard as MM.

Matrix Squaring

Problem: Compute A^2 where A is an $n \times n$ matrix.

MATRIX MULTIPLY $\leq_{O(n^2)}$ SQUARING.

$$\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}^2 = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix}$$

no notes

Linear Programming (LP)

Maximize or minimize a linear function subject to linear constraints.
Variables: vector $\mathbf{X} = (x_1, x_2, \cdots, x_n)$.

Objective Function: $\mathbf{c} \cdot \mathbf{X} = \sum c_i x_i$.
Inequality Constraints: $\mathbf{A_i} \cdot \mathbf{X} \leq b_i \quad 1 \leq i \leq k$.
Equality Constraints: $\mathbf{E_i} \cdot \mathbf{X} = d_i \quad 1 \leq i \leq m$.

Non-negative Constraints: $x_i \geq 0$ for some is.

Example of a "super problem" that many problems can reduce to.

Objective function defeinse what we want to minimize.

$\mathbf{A_i}$ is a vector – k vectors give the k b's.

Not all of the constraint types are used for every problem.

Use of LP

Reasons for considering LP:

- Practical algorithms exist to solve LP.
- Many real-world optimization problems are naturally stated as LP.
- Many optimization problems are reducible to LP.

no notes

Network Flow Reduction (1)

- Reduce NETWORK FLOW to LP.
- Let $x_1, x_2, \cdots, x_n$ be the flows through edges.
- Objective function: For $S =$ edges out of the source, maximize
$$\sum_{i \in S} x_i.$$
- Capacity constraints: $x_i \leq c_i \quad 1 \leq i \leq n$.
- Flow conservation:
  For a vertex $v \in V - \{s, t\}$,
  let $Y(v) =$ set of x_i for edges leaving v.
  $Z(v) =$ set of x_i for edges entering v.
$$\sum_{Z(V)} x_i - \sum_{Y(V)} x_i = 0.$$

Obviously, maximize the objective function by maximizing the X_i's!! But we can't do that arbirarily because of the constraints.

Network Flow Reduction (2)

Non-negative constraints: $x_i \geq 0 \quad 1 \leq i \leq n$.
Maximize: $x_1 + x_4$ subject to:

$$\begin{aligned} x_1 &\leq 4 \\ x_2 &\leq 3 \\ x_3 &\leq 2 \\ x_4 &\leq 5 \\ x_5 &\leq 7 \\ x_1 + x_3 - x_2 &= 0 \\ x_4 - x_3 - x_5 &= 0 \\ x_1, \cdots, x_5 &\geq 0 \end{aligned}$$

Need graph:
Vertices: s, a, b, t.

Edges:

- s $\rightarrow$ a with capacity $c_1 = 4$.
- a $\rightarrow$ t with capacity $c_2 = 3$.
- a $\rightarrow$ b with capacity $c_3 = 2$.
- s $\rightarrow$ b with capacity $c_4 = 5$.
- b $\rightarrow$ t with capacity $c_5 = 7$.

Matching

- Start with graph $G = (V, E)$.
- Let $x_1, x_2, \cdots, x_n$ represent the edges in E.
  - $x_i = 1$ means edge i is **matched**.
- Objective function: Maximize
$$\sum_{i=1}^{n} x_i.$$
- subject to: (Let $N(v)$ denote edges incident on v)
$$\sum_{N(V)} x_i \leq 1$$
$$x_i \geq 0 \quad 1 \leq i \leq n$$
- Integer constraints: Each x_i must be an integer.
- Integer constraints makes this INTEGER LINEAR PROGRAMMING (ILP).

no notes

Summary

NETWORK FLOW $\leq_{O(n)}$ LP.

MATCHING $\leq_{O(n)}$ ILP.

no notes

Summary of Reduction

Importance:

1. Compare difficulty of problems.
2. Prove new lower bounds.
3. Black box algorithms for "new" problems in terms of (already solved) "old" problems.
4. Provide insights.

Warning:

- A reduction **does not** provide an algorithm to solve a problem – only a transformation.
- Therefore, when you look for a reduction, you are **not** trying to solve either problem.

no notes

Another Warning

The notation $P_1 \leq P_2$ is meant to be suggestive.

Think of P_1 as the easier, P_2 as the harder problem.

Always transform from instance of P_1 to instance of P_2.

Common mistake: Doing the reduction backwards (from P_2 to P_1).

DON'T DO THAT!

no notes

Common Problems used in Reductions

NETWORK FLOW

MATCHING

SORTING

LP

ILP

MATRIX MULTIPLICATION

SHORTEST PATHS

no notes

Tractable Problems

We would like some convention for distinguishing tractable from intractable problems.
A problem is said to be **tractable** if an algorithm exists to solve it with polynomial time complexity: $O(p(n))$.

- It is said to be **intractable** if the best known algorithm requires exponential time.

Examples:

- Sorting: $O(n^2)$
- Convex Hull: $O(n^2)$
- Single source shortest path: $O(n^2)$
- All pairs shortest path: $O(n^3)$
- Matrix multiplication: $O(n^3)$

Log-polynomial is $O(n \log n)$

Like any simple rule of thumb for catagorizing, in some cases the distinction between polynomial and exponential could break down. For example, one can argue that, for practical problems, 1.01^n is preferable to n^{25}. But the reality is that very few polynomial-time algorithms have high degree, and exponential-time algorithms nearly always have a constant of 2 or greater. Nearly all algorithms are either low-degree polynomials or “real” exponentials, with very little in between.

Tractable Problems (cont)

The technique we will use to classify one group of algorithms is based on two concepts:

1. A special kind of reduction.
2. Nondeterminism.

no notes

Decision Problems

(I, S) such that S(X) is always either “yes” or “no.”

- Usually formulated as a question.

Example:

- Instance: A weighted graph $G = (V, E)$, two vertices s and t, and an integer K.
- Question: Is there a path from s to t of length $\leq K$? In this example, the answer is “yes.”

Need a graph here.

Decision Problems (cont)

Can also be formulated as a language recognition problem:

- Let L be the subset of I consisting of instances whose answer is "yes." Can we recognize L?

The class of tractable problems $\mathcal{P}$ is the class of languages or decision problems recognizable in polynomial time.

Following our graph example: It is possible to translate from a graph to a string representation, and to define a subset of such strings as corresponding to graphs with a path from s to t. This subset defines a language to "recognize."

Polynomial Reducibility

Reduction of one language to another language.

Let $L_1 \subset I_1$ and $L_2 \subset I_2$ be languages. L_1 is **polynomially reducible** to L_2 if there exists a transformation $f : I_1 \to I_2$, computable in polynomial time, such that $f(x) \in L_2$ if and only if $x \in L_1$.
We write: $L_1 \leq_p L_2$ or $L_1 \leq L_2$.

Or one decision problem to another.

Specialized case of reduction from Chapter 10.

Examples

- CLIQUE $\leq_p$ INDEPENDENT SET.
- An instance I of CLIQUE is a graph $G = (V, E)$ and an integer K.
- The instance $I' = f(I)$ of INDEPENDENT SET is the graph $G' = (V, E')$ and the integer K, were an edge $(u, v) \in E'$ iff $(u, v) \notin E$.
- f is computable in polynomial time.

no notes

Transformation Example

- G has a clique of size $\geq K$ iff G' has an independent set of size $\geq K$.
- Therefore, CLIQUE $\leq_p$ INDEPENDENT SET.
- **IMPORTANT WARNING:** The reduction does not **solve** either INDEPENDENT SET or CLIQUE, it merely transforms one into the other.

Need a graph here.

If nodes in G' are independent, then no connections. Thus, in G they all connect.