Harvard-MIT Division of Health Sciences and Technology
HST.951J: Medical Decision Support, Fall 2005
Instructors: Professor Lucila Ohno-Machado and Professor Staal Vinterbo

Algorithmic Complexity and Application to Problem Analysis

Staal A. Vinterbo

Harvard-MIT Division of Health Science and Technology

Decision Systems Group, BWH

Harvard Medical School

Nov 2005: HST 951/MIT 6.873 Class

Motivation

Problem

We have a new sequence of nucleotides. Which of the ones we already have does it match the best?

How do we address this problem?

- Has it been solved?
- Is there a problem that is close enough such that we can use it to obtain a solution?
- Is the problem feasible?
- How feasible?

Introduction

Algorithms and Computational Model

Definition (Program)

A finite sequence of computational instructions.

Definition (Computational Model)

The abstract representation of a device that can execute programs.

Definition (Algorithm)

An program for the solution of a particular problem.

Introduction

Computational Model

Convenient: present programs in a "Pascal" like language.

Example

An abstract "Pascal" machine, composed by a control and processing unit able to execute "Pascal" statements, and a set of memory locations identified by all variable and constant identifiers defined in the algorithm.

Introduction
Computational Model: Algorithm Example

Example

EXP(x, y)
(1) $r \leftarrow 1$
(2) **while** $y \neq 0$
(3) $r \leftarrow r * x$
(4) $y \leftarrow y - 1$
(5) **return** r

Introduction
Computational Model Cost

Uniform Cost

We also assume that all memory locations have the same size, and that all values involved in the computation are not larger than that they can be stored in a memory location.

Introduction
Computational Model: Example

Example

Our program EXP(x, y)
has cost $2 + 3y$.

Example

Alternative: logarithmic cost:
$a \leftarrow 5 + v$
has a cost proportional with the sum of logarithms of values involved:

$$\log 5 + \log |v|$$

Preliminaries
What quantities for Algorithms?

We need to decide

1. Execution cost
   - computational steps: the “dominant” operation
   - memory used
2. Input size, which characteristic parameter describing the input is it whose growth towards infinity gives asymptotic computation cost.

Preliminaries

Big O Notation

$$
\begin{aligned}
O(g(n)) &= \{f(n) | \text{there exists } c > 0 \text{ and } n_0 > 0 \text{ s.t.} \\
&\quad 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\} \\
o(g(n)) &= \{f(n) | \text{for any } c > 0 \text{ there exists } n_0 > 0 \text{ s.t.} \\
&\quad 0 \le f(n) < cg(n) \text{ for all } n \ge n_0\} \\
\Omega(g(n)) &= \{f(n) | \text{there exists } c > 0 \text{ and } n_0 > 0 \text{ s.t.} \\
&\quad 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}
\end{aligned}
$$

- $O(g(n))$ – the set of functions that are asymptotically bounded from above by g.
- $\Omega(g(n))$ – the set of functions that are asymptotically bounded from below by g.

Preliminaries

Big O Notation

Example

What is $x^2 - x$?

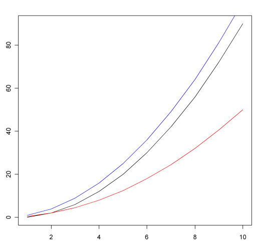
$$x^2 - x \le x^2 \text{ for } x_0 > 0 \Rightarrow x^2 - x \in O(x^2)$$

$$
\begin{aligned}
cx^2 &\le x^2 - x \Rightarrow \\
c &\le \frac{x^2 - x}{x^2} = 1 - \frac{1}{x} \overset{x \to \infty}{\to} 1 \Rightarrow \\
cx^2 &\le x^2 - x \text{ for } c = 1/2 \text{ and } x_0 = 2 \Rightarrow \\
x^2 - x &\in \Omega(x^2)
\end{aligned}
$$

Preliminaries

Big O Notation

Example



Black – $x^2 - x$, blue – x^2, red – $x^2/2$.

Preliminaries

Big O Notation

We had that $x^2 - x \in O(x^2) \cap \Omega(x^2)$. In general

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n)).$$

The set $\Theta(g(n))$ is then the set of functions for which g is a *tight* asymptotic bound.

- $o(g(n))$ – the set of functions for which g is a lower bound that is not tight.

Preliminaries

Boundedness

Further we say that a function f is *polynomially bounded* if

$$f(n) \in O(n^k) = n^{O(1)}$$

for some constant k, and we say that f is *polylogarithmically* bounded if

$$f(n) \in O((\ln n)^k) = \ln^{O(1)} n$$

for some constant k. As we have that

$$(\ln n)^a \in o(n^k)$$

for any constant $k > 0$, we have that polylogarithmically bounded functions grow slower than polynomial functions.

Preliminaries

Other Useful Equalities

Using Stirling's approximation we have that

$$\begin{aligned} n! &= o(n^n) \\ \ln(n!) &= \Theta(n \ln n). \end{aligned}$$

We further have that

$$O(1) \subseteq O((\ln n)^k)) \subseteq O(n^k) \subseteq O(2^k) \subseteq O(n!) \subseteq O(n^n)$$

for some constant $k > 0$.

Analysis of Algorithms

Merge

Merge two sorted list l_1 and l_2 into a single sorted list.

```
MERGE(l1, l2)
    if ISEMPTY(l1)
        return l2
    if ISEMPTY(l2)
        return l1
    if ISLESSEQUAL(FIRST(l1),FIRST(l2))
        return (APPEND(LIST(FIRST(l1)),MERGE(REST(l1),l2)))
    return (APPEND(LIST(FIRST(l2)),MERGE(l1,REST(l2))))
```

We assume that all these functions can be done in a constant number of computational steps, i.e., $\Theta(1)$ steps.

Analysis of Algorithms

Merge

```
MERGE(l1, l2)
    if ISEMPTY(l1)
        return l2
    if ISEMPTY(l2)
        return l1
    if ISLESSEQUAL(FIRST(l1),FIRST(l2))
        return (APPEND(LIST(FIRST(l1)),MERGE(REST(l1),l2)))
    return (APPEND(LIST(FIRST(l2)),MERGE(l1,REST(l2))))
```

- $|l_1| + |l_2| = n$, $T(n)$ – number of steps needed to merge.
- $n = 1$: all we have to do is return non-empty list, $T(1) = \Theta(1)$.
- $n \neq 1$: $\Theta(1) + T(n-1)$
- $T(n) = \begin{cases} \Theta(1) & \text{for } n = 1, \\ T(n-1) + \Theta(1) & \text{for } n > 1. \end{cases}$

Analysis of Algorithms
Merge

Let us see what happens if we substitute a number for n.

$$\begin{aligned} T(4) &= T(3) + \Theta(1) \\ &= (T(2) + \Theta(1)) + \Theta(1) \\ &= ((T(1) + \Theta(1)) + \Theta(1)) + \Theta(1) \\ &= (((\Theta(1)) + \Theta(1)) + \Theta(1)) + \Theta(1) \\ &= 4\Theta(1) \end{aligned}$$

We see that $T(n) = n\Theta(1) = \Theta(n)$, meaning that $\textsc{Merge}(l_1, l_2)$ for a combined length of l_1 and l_2 of n requires $\Theta(n)$ steps.

Analysis of Algorithms
MergeSort

```
MERGESORT(l)
    if ISEMPTY(l)
        return l
    if ISSINGLETON(l)
        return l
    return (MERGE(
        MERGESORT(FIRSTHALF(l)),
        MERGESORT(SECONDHALF(l))))
```

$$T(n) = \begin{cases} \Theta(1) & \text{for } n = 1, \\ 2T(n/2) + \Theta(n) & \text{for } n > 1. \end{cases} = \Theta(n \ln n)$$

Think binary tree...

Analysis of Algorithms
Space Complexity

Similarly to time complexity, we can analyze algorithms in terms of space requirements. For input size n, $S(n)$ denotes the number of memory locations we need.

Analysis of Algorithms
Space Complexity: Merge

```
MERGE(l1, l2)
    if ISEMPTY(l1)
        return l2
    if ISEMPTY(l2)
        return l1
    if ISLESSEQUAL(FIRST(l1),FIRST(l2))
        return (APPEND(LIST(FIRST(l1)),MERGE(REST(l1),l2)))
    return (APPEND(LIST(FIRST(l2)),MERGE(l1,REST(l2))))
```

1. Arguments are given by reference.

$$S(n) = \begin{cases} \Theta(1) & \text{for } n = 1, \\ S(n-1) + \Theta(1) & \text{for } n > 1. \end{cases}$$

$S(n) = \Theta(n)$

Analysis of Algorithms

Space Complexity: Merge

```
MERGE(l1, l2)
    if ISEMPTY(l1)
        return l2
    if ISEMPTY(l2)
        return l1
    if ISLESSEQUAL(FIRST(l1),FIRST(l2))
        return (APPEND(LIST(FIRST(l1)),MERGE(REST(l1),l2)))
    return (APPEND(LIST(FIRST(l2)),MERGE(l1,REST(l2))))
```

2. Arguments are given by value (copied).

$$S(n) = \begin{cases} \Theta(1) & \text{for } n = 1, \\ S(n-1) + \Theta(n) + \Theta(1) & \text{for } n > 1. \end{cases}$$

$\sum_{i=1}^{n} i = n(n+1)/2 \Rightarrow S(n) = \Theta(n^2)$

What does that do to $T(n)$?

Analysis of Problems

The complexity of a problem can be described in terms of the time and space complexity of the algorithms that solve the problem.

An important property of an algorithm is the worst case time expenditure for a given problem size, i.e., the maximum time the algorithm takes over all problems of at most a given size.

Analysis of Problems

Relational View

Example

Binary relation R_{Sorted} on the set of finite lists of numbers.

- (l, l_s) is in R_{Sorted} if and only if l_s is the sorted version of l.

Example

I – $n \times m$ matrices M

S – $2^{\{1,2,\ldots,n\}}$

$(M, C) \in R_{\text{Cover}} \subseteq I \times S$ if and only if

$$\sum_{i=C} M[i,j] > 0$$

for all $j \in \{1, 2, \ldots, m\}$.

Analysis of Problems

NP-Relations

Definition (NP-Relation)

$R \subseteq I \times S$ is an NP-relation if the characteristic function χ_R of R is computable in polynomial time in $|x|$ for all $x \in I$.

Definition (P-Relation)

An NP relation $R \subseteq I \times S$ is an P-relation if we can compute $y \in R(x)$ or determine that $R(x) = \emptyset$ in polynomial time in $|x|$ for all $x \in I$.

Problems as NP-relations

I – problem instances

S – solutions

P-relations are problems that are *solvable* in polynomial time, NP-relations are problems that are *checkable* in polynomial time.

Analysis of Problems
Big Question

BIG Question

P = NP?

Not conclusively answered, although most believe it not true.

Analysis of Problems
Sat

Example (SAT)

Let V be a finite set of boolean variables, and let a *literal* be a boolean variable or its negation. Further let a be a set of literals. A clause is satisfied by a variable value assignment (setting) if at least one of the literals evaluates to true. If we let

- $I = 2^C - \emptyset$, where C is the set of all clauses over V,
- S be the set of all variable value assignments, and
- $R \subseteq I \times S$ such that $R(x)$ is the set of all variable value assignments such that all clauses in x are satisfied.

Then R is the SAT NP-relation.

Analysis of Problems
Reductions

Let R_1 and R_2 be two NP-relations. We define a reduction from R_1 to R_2 as a tuple of functions (f, g) such that

$$(x, g(x, y)) \in R_1 \iff (f(x), y) \in R_2.$$

We write $R_1 \leq R_2$.

$$\begin{tikzcd} x \arrow[r, "f"] & f(x) \arrow[d, "\mathcal{A}_{R_2}"] \\ g(x,y) & y \arrow[l, "g"] \end{tikzcd}$$

$$\mathcal{A}_{R_1}(x) = g(x, \mathcal{A}_{R_2}(f(x)))$$

Analysis of Problems
Reductions

Example

- R_{sort} is R_2
- R_{max} is R_1

Let $f(x) = x$, and $g(x, y) = \mathrm{last}(y)$, then $\max(x) = g(x, \mathrm{sort}(x)) = \mathrm{last}(\mathrm{sort}(x))$. We have that $R_{\text{max}} \leq R_{\text{sort}}$.

Analysis of Problems

NP-Completeness

Definition (Polynomial time reduction)

If f and g are both computable in polynomial time, we call a reduction (f, g) a *polynomial time reduction*, and use $R_1 \leq_p R_2$ to indicate that we have a polynomial time reduction from R_1 to R_2.

Definition (NP-Complete NP-relation)

If $R \geq_p R'$ for all NP-relations R', then R is NP-Hard. If R is an NP-relation, R is NP-Complete.

NP-Complete NP-relations are the "hardest" NP-relations.

Analysis of Problems

NP-Completeness

Transitivity of reductions

Note that $\leq_p$ is transitive.

This means: reduction to one NP-complete relation is enough.

Cook's Theorem

Need a seed: Satisfiability is NP-complete (Cook 1971)

Analysis of Problems

NP Completeness of 3-Sat

Example (3-SAT)

3-SAT is the SAT problem where clauses are restricted to be of cardinality 3.

Theorem (3-SAT is NP-complete)

3-SAT is NP-complete.

Proof.

Each $c = \{z_1, \ldots, z_k\}$ is transformed as (using fresh y):

$$c \Rightarrow \begin{cases} \{\{z_1, y_1, y_2\}, \{z_1, \overline{y}_1, y_2\}, \{z_1, y_1, \overline{y}_2\}, \{z_1, \overline{y}_1, \overline{y}_2\}\} & \text{if } k = 1 \\ \{\{z_1, z_2, y\}, \{z_1, z_2, \overline{y}\}\} & \text{if } k = 2 \\ c & \text{if } k = 3 \\ \{z_1, z_2, y\} \cup \{\{y_i, z_{i+2}, \overline{y}_{i+1}\} | 1 \leq i \leq k - 4\} \cup \{\overline{y}_{k-3}, z_{k-1}, z_k\} & \text{if } k > 3 \end{cases}$$

Analysis of Problems

Optimization problems

Definition (Optimization problem)

An optimization problem is a three tuple $(R, m, \star)$, where

- $R \subseteq I \times S$, I are instances, S are solutions,
- m is a function $m : R \to \mathbb{N}$,
- $\star$ is an element of $\{\leq, \geq\}$.

Definition

For an optimization problem $(R, m, \star)$, the set $R(x)$ is the *set of feasible solutions for the instance* x, $m(x, y)$ is the *measure of solution* y *of instance* x, $m^*(x) = z$ such that $z = m(x, y)$ for some $y \in R(x)$ and $z \star m(x, y')$ for all $y' \in R(x)$. Also, $y(x) = \{y \in R(x) | m(x, y) = m^*(x)\}$.

Analysis of Problems
NPO problems

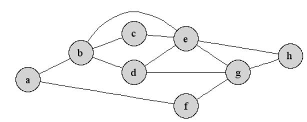
This means that $m^*(x)$ is the optimal measure for problem instance x, and $y(x)$ is the set of optimal solutions for problem instance x. Also, $\star$ is called the *goal*, and the problem is a *minimization* problem if $\star = \leq$, and a *maximization* problem if $\star = \geq$.

Definition (NP-Optimization (NPO) Problem)

An optimization problem $(R, m, \star)$ is an NPO problem if $R^n = \{(x, y) \in R | m(x, y) \star n\}$ is an NP-relation, and m is computable in polynomial time.

Analysis of Problems
Vertex Cover

Example (Vertex Cover)



If we let V be a universe of vertices

- I be the set of all graphs $G = (V', E)$, where $V' \subseteq V$, and $E \subseteq V' \times V'$,
- $S = 2^V$, and
- $R \subseteq I \times S$ such that $S \in R(x)$ is such that for all $(u, v) \in E$, we have that $S \cap \{u, v\} \neq \emptyset$,

Then $(R, m(S) = |S|, \leq)$ is the *Vertex Cover* minimization problem.

Analysis of Problems
NP-hard NPO problems

Definition (NP-hard NPO problem)

An NPO problem $(R, m, \star)$ is *NP-hard* if R^n is NP-complete.

If we could find a polynomial time algorithm $\mathcal{A}$ for $(R, m, \star)$, we can find $y \in R^n(x)$ or determine that $R^n(x) = \emptyset$ in polynomial time as follows: if $m^*(x) > n$ then $R^n(x) = \emptyset$, otherwise return $y \in y(x)$. This means that if $(R, m, \star)$ is NP-hard, a polynomial time algorithm for this problem would mean NP $=$ P. Hence, it is believed that there exist no polynomial time algorithm for NP-hard NPO problems.

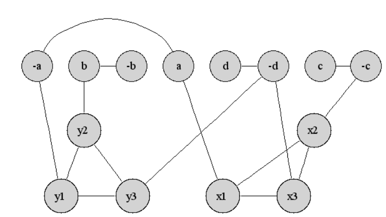
Analysis of Problems
Vertex Cover is NP-hard

Theorem

Vertex Cover is NP-hard

Proof.

Reduction from 3-SAT:
$U = \{a, b, c, d\}$,
$C = \{\{a, \overline{c}, \overline{d}\}, \{\overline{a}, b, \overline{d}\}\}$ C is satisfiable if and only if G in figure has a vertex cover of size $K = |U| + 2 * |C| = 8$ or less



$\square$

Analysis of Problems

Approximation properties

Definition (Performance ratio)

Let $(R, m, \star)$ be an NPO problem. Given an instance x and $y \in R(x)$, we define the *performance ratio* of y with respect to x as

$$\mathcal{R}_R(x, y) = \max\left(\frac{m_R(x, y)}{m_R^*(x)}, \frac{m_R^*(x)}{m_R(x, y)}\right).$$

Definition (r-approximation algorithm)

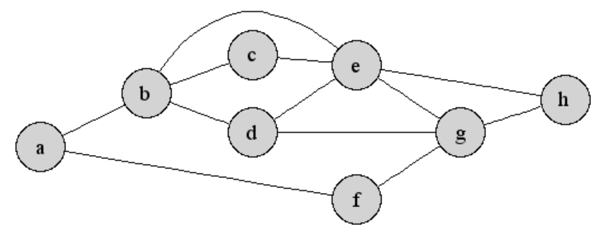
We say that a polynomial time algorithm $\mathcal{A}$ for problem $(R, m_R, *)$ is an $r(n)$-approximation algorithm if $\mathcal{R}_R(x, \mathcal{A}(x)) \leq r(|x|)$ for all instances x.

Analysis of Problems

Vertex Cover Example

Example (Vertex Cover)

Graph $G = (V, E)$. Find minimum cardinality $V' \subseteq V$ such that for all $(u, v) \in E$, we have that $V \cap \{u, v\} \neq \emptyset$.



Analysis of Problems

Vertex Cover Example

Example (Vertex Cover)

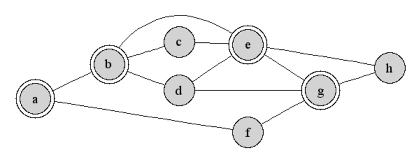
VC(V, E)
repeat
  choose any edge $(u, v) \in E$
  $V' \leftarrow V' \cup \{u, v\}$
  remove from E any e incident to either v or u
until $E = \emptyset$
return V'

Claim: VC is a 2-approximation algorithm.

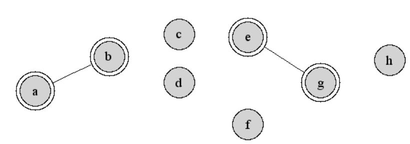
Analysis of Problems

Vertex Cover Example

Proof



Remove all edges except those in VC(V, E)



You need at least half of the vertices in VC(V, E)

Analysis of Problems

Good and Bad News

Good News

Existence of r-approximation algorithm

Bad News

Proof of non-existence of r'-approximation algorithm, unless P (typically $P =$P=NP).

Analysis of Problems

Good and bad problems

Good problems

FPTAS r-approximation possible in $p(|x|)p'(1/(1-r))$ time. Example: Maximum Knapsack.

Bad Problems

Problems where deciding whether $R(x)$ is empty or not is NP-hard. Examples: Max(min)imum Weighted Satisfiability, Minumum $\{0,1\}$-integer programming.

Analysis of Problems

Prediction of hardness

Results given so far are worst case analysis results.

- non-randomness of instance hardness?

Analysis of randomized instances by statistical mechanics and phase transitions between regions of hard and not-so-hard instances.