COMP 360 F2018 Algorithm Design

Bruce Reed

Basic Course Administration Information

All info is in the course outline, the first course module in MyCourses.

15% Assignments, 25% Midterm(October 19th in class), 60% Final

All references are either available as an eBook from the library or relevant sections will be posted on MyCourses.

Powerpoint presentations for each lecture will be available on Mycourses as will brief summaries with links to references.

Occasionally (warmup) exercises will be posted in advance of a lecture. Trying them even if you can not do them will help you get more out of the lecture.

The course overview from this lecture can be found in the overview section of MyCourses.

Algorithm Design and Analysis in Previous Courses (250 and 251/252)

Learned some techniques for algorithm design(dynamic programming, greedy algorithms, graph traversal techniques).

Learned how to bound the worst case running time of an algorithm.

Informally can tell if an algorithm is fast or slow. Usually wanted to design a fast (or even better, the fastest) algorithm for a problem.

(Big O notation and recurrence relations relevant in this regard).

We build on this but focus largely on problems for which we cannot find a fast algorithm.

Algorithm Design and Analysis in This Course

Complexity and NP-Completeness: How do we tell if a problem is hard or easy, i.e. whether or not there is some fast algorithm for it?

Linear Programming: One of the top 10 algorithms of the 20th century.

Common theme: Reductions between problems as an algorithmic design technique. Focus on certificates showing a solution is correct.

How to Handle Hard Problems: Approximation algorithms, pseudopolynomial time algorithms, randomized algorithms, average case analysis, special cases.

Two Important Algorithms: JPEG Compression and Google PageRank two more of the top 10 algorithms of the 20th Century. (+Quicksort).

NP-Completeness: determining if a problem is easy or hard.

Fast Versus Slow

We say an algorithm is *fast* if its worst case running time (number of steps taken on some model of a computer) on an instance of size n is O(n^d) for some fixed d. More formally, we say it runs in *polynomial time.*

To be precise we need to specify how we measure the size of an instance and what model of computation we use. However, the fact that an algorithm runs in polynomial time is typically invariant over all reasonable definitions of the size of the input and choice of the computer model. So we avoid setting out our choices, simply noting we have choices in mind.

We note that whether we say that the size of an input integer s is s or log s, does make a difference in whether algorithms are . An example making this clear is primality testing. We use the latter measure.

Hard versus Easy

A problem is easy if there is a fast, i.e. polynomial time, algorithm to solves it. Otherwise it is hard.

We discuss a million dollar conjecture characterizing which (decision) problems are hard.

Decision Problems

A decision problem is one where the output is yes or no.

Many problems have a corresponding decision problem such that either both are hard or both are easy

Decision-MWST:

Input(Graph G, weight w for each edge, integer k)

Question: Does the minimum weight spanning tree for this problem have weight at most k?

Certificates

A decision problem is *yes-certifiable* if for every instance for which the answer is yes, we can associate a certificate such that there is an algorithm which given the certificate and the instance in polynomial time in the size of the instance determines that the answer is yes.

Ex. n composite? certificate is integer pair $(f_1,f_2)$with $f_1f_2=n$, $f_1>1$,$f_2>1$

A decision problem is *no-certifiable* if for every instance for which the answer is no, we can associate a certificate such that there is an algorithm which given the certificate and the instance in polynomial time in the size of the instance determines that the answer is yes.

Ex. n prime? certificate is integer pair $(f_1,f_2)$with $f_1f_2=n$, $f_1>1$,$f_2>1$

Possibly Easy Problems

If a decision problem is easy then it must be both yes-certifiable and no-certifiable because we can just ignore the certificate and use the fast algorithm which solves it.

A Million Dollar Conjecture

Every decision problem which is yes-certifiable is easy

=

Every decision problem which is no-certifiable is easy.

A weaker conjecture:

Every decision problem which is both yes-certifiable and no-certifiable is easy

Using Reductions to Quickly Solve Problems

A reduction algorithm from decision problem π to decision problem π’ takes as input an instance of problem π and yields an instance of problem π’ with the same answer.

If there is a fast reduction algorithm from π to π’ and a fast algorithm for π’ then there is a fast algorithm for π. I.e. we can use a fast reduction from π to an easy problem to show π is easy.

This implies that if there is a fast reduction algorithm from π to π’ and no fast algorithm for π then there is no fast algorithm for π’. I.e. we can use a fast reduction from a hard problem to π’, to show π’ is hard.

NP-Complete Problems

A problem π is NP-Complete if there is a fast reduction to it from every yes-certifiable problem. This implies that if π is easy then every yes-certifiable problem is easy and the million dollar conjecture is true.

So, if π is NP-Complete it will likely be hard to develop a fast algorithm for it, and many believe this in fact implies π is hard, i.e. there is no fast algorithm for it.

Linear Programming

COURSE STARTS NOW!!!

Linear Programming

Will see how to reduce lots of problems to a Linear Programming Problem.

We will see how the theory of duality ties the existence of no-certificates for the decision version of Linear Programming to the algorithm we use to solve it.

The Diet Problem (Adapted from Chvatal Chapter 1)

- Principal Fortier (also known as Polly) spends as much of her money as she can on turning the Royal Victoria Hospital into a vibrant lively part of the McGill Campus which connects it to Mont Royal and the people of Montreal. So she needs to minimize how much she spends on food. She will rely on vitamin and mineral supplements as a cheap way to satisfy most of her nutritional needs. In choosing what food to eat, she needs only ensure that she meets daily requirements re energy (2000 calories), protein(55 g), and calcium(800 mg). Rather fussy with respect to what she eats, she restricts her attention to the six foods set out in the table on the right. Her (and our) job is to determine how to minimize her daily expenditure while ensuring her needs are met? How should we formalize the problem?

Polly's Pantry

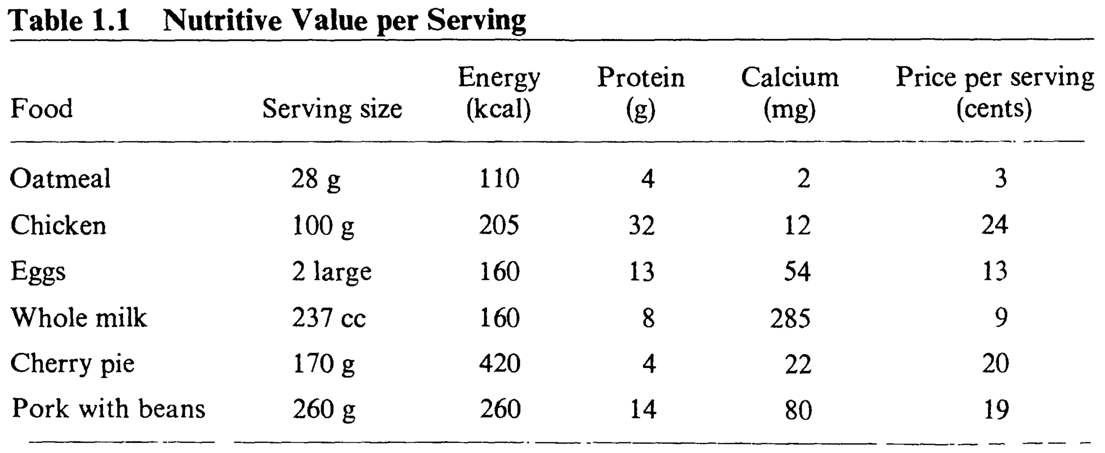

Table 1.1 Nutritive Value per Serving

Food	Serving size	Energy (kcal)	Protein (g)	Calcium (mg)	Price per serving (cents)
Oatmeal	28 g	110	4	2	3
Chicken	100 g	205	32	12	24
Eggs	2 large	160	13	54	13
Whole milk	237 cc	160	8	285	9
Cherry pie	170 g	420	4	22	20
Pork with beans	260 g	260	14	80	19

The LP Formulation

The Variables:

x_1 servings of oatmeal per day

x_2 servings of chicken per day

x_3 servings of eggs per day

x_4 servings of milk per day

x_5 servings of cherry pie per day

x_6 servings of pork with beans per day

The LP:

Minimize $3x_1+24x_2+13x_3+9x_4+20x_5+19x_6$

Subject to:

$$\begin{aligned}
110x_1+205x_2+160x_3+160x_4+420x_5+260x_6 &\geq 2000\\
4x_1+32x_2+13x_3+8x_4+4x_5+14x_6 &\geq 55\\
2x_1+12x_2+54x_3+285x_4+22x_5+80x_6 &\geq 800\\
x_1,x_2,x_3,x_4,x_5,x_6 &\geq 0
\end{aligned}$$

Some Definitions

Linear function: $\sum_{j=1}^{n} c_j\, x_j$ for real c_j

Linear equation: $f(x_1,\ldots,x_n)=b$ for some linear f and real b.

Linear inequality: $f(x_1,\ldots,x_n)\leq b$ or $f(x_1,\ldots,x_n)\geq b$ for some linear f and real b.

Linear constraint: either a linear equation or a linear inequality.

Linear Program: maximize or minimize a linear function subject to a set of linear constraints

The Best Solution?

Could do: 3 servings of milk + 14 servings of oatmeal.

27 cents + 42 cents =69 cents.

Can we do better??

Saw that the highest energy per cent was given by oatmeal.

So cost to satisfy energy is at least (2000/110)x 3 about 54 cents.

Highest calcium per cent is given by milk. So cost to satisfy is at least (800/285)x9= 25.26 about 26 cents.

Highest protein per cent also given by oatmeal so cost to satisfy protein is (55/4)x3 about 40 cents.

If we do 2.71 of milk and 14.25 of oatmeal: 42.75+24.3 67.2

Adding Further Constraints For Sustainability

Upper bound the number of servings of each type.

Always have milk with her oatmeal. Insist $x_4 - x_1 \leq 0$?

Final LP in book:

Minimize $3x_1+24x_2+13x_3+9x_4+20x_5+19x_6$

Subject to:

$$\begin{aligned}
110x_1+205x_2+160x_3+160x_4+420x_5+260x_6 &\geq 2000\\
4x_1+32x_2+13x_3+8x_4+4x_5+14x_6 &\geq 55\\
2x_1+12x_2+54x_3+285x_4+22x_5+80x_6 &\geq 800\\
x_1 \leq 4, x_2 \leq 3, x_3 \leq 2, x_4 \leq 8, x_5 \leq 2, x_6 &\leq 2,\\
x_1,x_2,x_3,x_4,x_5,x_6 &\geq 0
\end{aligned}$$

Some More Definitions

Feasible solution to an LP: assignment of values to the variables so all constraints are satisfied .

Infeasible LP: there are no feasible solutions.

Unbounded maximization (resp. minimization) problem: for all real r, there is a feasible solution with objective function value exceeding (resp. less than) r.

Integer Linear Program (ILP): Linear Program with added constraint that solutions are integral, i.e each x_i is an integer.