College of Computer & Information Science — Fall 2007
Northeastern University — 11 September 2007
CS G399: Algorithmic Power Tools I — Scribe: Eric Robinson

Lecture Outline:

- Set Cover: Alternate Analysis
- Linear Programming: The Set Cover Problem
- Linear Programming: An Overview
- Linear Programming: The Forms
- Linear Programming: Vertices

In this lecture, an alternate analysis of the approximation to the set cover problem is provided. The set cover problem is then used to demonstrate an application in linear programming. An overview of linear programming is then provided and the three forms for linear programs are shown. Finally, a vertex is defined and it is positted that all linear programs must either have no vertices, have an infinite solution, or have a solution on a vertex.

1 Set Cover

Set cover was described in the previous class.

1.1 Alternate Analysis

The previous analysis of the set cover problem focussed on the relative cost for each set picked. Another approach is provided here which focuses instead on the relative cost for each element covered. First, recall the goal of the set cover problem. The input to the problem is,

- A set U of n elements, and
- A set S of subsets of U.
- A cost function for the subsets, $c : S \longrightarrow Q^+$

The goal is to pick the least costly set of subsets, S', such that $\bigcup(s \in S') = U$, or the set of subsets covers all elements in U.

The greedy algorithm for this is simple, it says, at each step, to pick the set with the smallest ratio of cost to newly covered elements, or, where U' is the set of elements covered currently by the solution, find the set:

$$\max_{s \in S} \frac{c(s)}{|s - U'|}$$

The previous proof worked by iterating over the cost of sets chosen in the greedy solution, here a different approach is shown iterating over the cost of new elements covered as they are chosen. The cost of an element is the normalized cost of the first set to cover that element, or, where s' is the first set to cover an element u, and elements U' have been seen in the current solution:

$$p(u) = \frac{c(s')}{|s' - U'|}$$

From this, the total cost of the greedy solution can be found to be:

$$\sum_{u \in U} p(u)$$

For simplicity's sake, the assumption that the elements are covered in order will be made. This is a safe assumption as renumbering the elements in U does not affect the solution. With this, we can evaluate the cost of e_i in the greedy solution:

$$\begin{aligned} p(e_i) &\leq \frac{\text{OPT}}{n-i+1} \\ &= \text{OPT} \times \frac{1}{n-i+1} \end{aligned}$$

This must be true because an optimal solution covers these with at most a cost of OPT, because of averaging over the remaining costs, there exists a set that covers thes these with at most $\frac{\text{OPT}}{n-i+1}$. From this, now the entire cost of the greedy solution can be bounded:

$$\begin{aligned} \text{ALG} &= \sum_{u \in U} p(u) \\ &\leq \sum_{u \in U} \text{OPT} \times \frac{1}{n-i+1} \\ &= \text{OPT} \times \sum_{u \in U} \frac{1}{n-i+1} \\ &= \text{OPT} \times H_n \approx \text{OPT} \times ln(n) \end{aligned}$$

As can be seen, the bound of H_n is found as it was with the previous analysis.

2 Linear Programming

Here linear programming is introduced. First, an example of the set cover problem written as a linear program is provided. After this, linear programming itself is discussed, the different linear programming forms are provided, and the definition of a vertex is given. Two excellent reference sources (among many others) are Michel Goemans's lecture notes on linear programming [Goe94], and Howard Karloff's text on linear programming [Kar91].

2.1 The Set Cover Problem

Here the set cover problem will be put into a linear programming framework. First, let's look at the result, our goal is to minimize the cost of each set. We will allow the variable x to represent the inclusion of each set, where:

$$x_s = \begin{cases} 1 & \text{if set is selected} \\ 0 & \text{otherwise} \end{cases}$$

From this, the linear programming set cover problem can be formulated. Note that each element u should be in some chosen subset, or $u \in \bigcup_{s \in S} x_s = 1 \forall e \in U$:

$$\begin{array}{lll} \min & \sum_{s \in S} x_s \times c(s) & \\ \text{s.t.} & \sum_{s \in S}(x_s \times s_u) \geq 1 & \forall u \in U \\ & x_s \in \{0,1\} & \forall s \in S \end{array}$$

An important note here is that if $x_s \in \{0,1\}$ could be replaced with $x_s \geq 0$ then the problem would become solvable in polynomial time. Linear programming is a problem in P, the same cannot be said for integer linear programming, which is known to be NP-complete.

In addition, one can envision taking an approximation by solving the linear programming problem, then approximating an integer solution by rounding the result in an intelligent way. Consider the following example:

$$\begin{aligned} U &= \{A, B, C\} \\ S &= \{\{A, B\}, \{B, C\}, \{A, C\}\} \\ c(s \in S) &= 1 \end{aligned}$$

The optimal integer solution here is to choose any two of the sets, for a total cost of 2. However, a non-integer solution could pick half of each set, leading to each value being covered half-ways by two sets. The total cost of this solution is only 1.5. The fractional solution is always a bound on the optimal solution for integer linear programming. If the gap between the fractional solution and an algorithm can be determined, this is an adequate proof of the upper bound of the gap between the integer solution and the algorihtm as well.

2.2 Overview

Integer linear programming is an NPC problem, whereas linear programming is in P. In addition, the non-integer solution is always guarenteed to be as good, if not better, than the integer solution. If an integer solution is desired, it is often possible to find a non-integer solution, and then through smart rounding approximate what an integer solution may be.

Below lists the three main linear programming algorithms listed historically:

Simplex - Invented by George Dantzig in 1947 to solve linear programming problems, this technique was fast for most practical applications. However, it is non-polynomial in worst case scenarios, and later examples were provided where simplex failed to perform efficiently.

Ellipsoid - Introduced by Naum Z. Shor, Arkady Nemirovsky, and David B. Yudin in 1972, and shown to be polynomial by Leonid Khachiyan, this technique, while polynomial, is in practice typically much slower than the simplex algorithm and was therefor rarely used. It was useful for showing that general linear programming was in P however.

Interior Point - Introduced and developed by Narendra Karmarkar in 1984, this mehtod has the advantage of being polynomial like the ellipsoid algorithm, but also fast in practice, like the simplex algorithm. For this reason it was used in practice for a long time. Now, however, hybrids and other variations are used in many cases.

2.3 The Forms

There are three main forms that linear programs may take. Those forms are listed below, a short description of each is provided, and a practical example of each is also given. The forms are in order according to restrictiveness. Fortunately, any linear program (a linear program in its general form), can be converted into the most restrictive form (the slack form) easily.

2.3.1 General Form

All forms of linear inequalities are allowed in the general form.

$$\begin{array}{lll} \text{min} & \sum_{i=1}^{n} c_i \times x_i & \\ \text{s.t.} & \sum_{i=1}^{n} a_{ij} \times x_i > b_j & \forall j \\ & \sum_{i=1}^{n} a_{ij} \times x_i = b_j & \forall j \\ & x_i \geq 0 & \forall i \\ & x_i \neq 0 & \forall i \end{array}$$

Note that upper bound constraints (with a $\leq$ sign) are not required, as they can be transformed into lower bound constraints simply by multiplying both sides of the constraint by -1.

2.3.2 Standard Form

The standard form does not allow for equality comparisons and requires that all variables be nonnegative.

$$\begin{array}{lll} \text{min} & \sum_{i=1}^{n} c_i \times x_i & \\ \text{s.t.} & \sum_{i=1}^{n} a_{ij} \times x_i \geq b_j & \forall j \\ & x_i \geq 0 & \forall i \end{array}$$

To get rid of equality comparisons, simply ensure that $\geq$ and $\leq$ both hold, which will require negating the $\leq$ comparison to make it a $\geq$, as described in the general form.

2.3.3 Slack Form

The slack form only allows for equality comparisons for the summations, and requires that all variables be nonnegative. An example is shown below.

$$\begin{array}{ll} \min & \sum_{i=1}^{n} c_i \times x_i \\ \text{s.t.} & \sum_{i=1}^{n} a_{ij} \times x_i = b_j \quad \forall j \\ & x_i \geq 0 \qquad\qquad\quad\ \forall i \end{array}$$

To obtain an equality comparison, an extra variable can be introduced (such variables are referred to as *slack variables*, hence the name slack form). For example

$$2x_1 + 3x_2 \geq 5$$

is the same as

$$\begin{aligned} 2x_1 + 3x_2 - t &= 5 \\ t &\geq 0 \end{aligned}$$

2.4 Vertices

A linear system always forms a polytope such that, for the linear system:

- The solution is unbounded (infinite)
- The solution is infeasible (no polytope exists)
- An optimal solution occurs on a vertex of the polytope

A linear system always produces a convex polytope. A convex polytope P, is a polytope where:

- A line between 2 points in the polytope remains in the polytope
- If $x, y \in P$, $\alpha x + (1-\alpha)y \in P$ for $\alpha \in [0, 1]$

A vertex, x, of a convex polytope P, is a "corner point" in the polytope and can be defined by any one of these three equivalent statements, which we will prove in the next lecture.

- $\neg\exists y \neq 0$ s.t. $x + y, x - y \in P$
- $\neg\exists y, z \in P$ s.t. $x = \alpha y + (1-\alpha)z, \alpha \in (0, 1)$
- $\exists n$ linear inequalities of P that are tight at x

References

[Goe94] M. Goemans. Introduction to Linear Programming. Lecture notes, available from http://www-math.mit.edu/˜goemans/, October 1994.

[Kar91] H. Karloff. *Linear Programming*. Birkhäuser, Boston, MA, 1991.