CMPSCI 611 Advanced Algorithms **Fall 2000**

Lecture 23: December 15th

Lecturer: Micah Adler *Scribe: Gary King and AnYuan Guo*

23.1 Review of Approximation Algorithm Classes

Most approximation algorithms fall into one of several well-defined categories. We review these and provide examples of each kind.

23.1.1 The worst case: n^ϵ

Examples in this class include *Clique*, *Independent Set*, and the *Graph-Coloring* problem. Problems in this class have, in essence, no good approximations. For example, if $\mathbf{P} \neq \mathbf{NP}$, then there is no approximation algorithm for *Clique* within $|V|^{1-\epsilon}$. Furthermore, if all problems in $\mathbf{NP}$ cannot be solved by Las Vegas type randomized algorithms in polynomial time, the best known approximation is $O(\frac{|V|}{\log |V|^2})$. Interestingly, although *Clique* and *Vertex-Cover* seem to be closely related $\mathbf{NP}$-Complete problems, they have wildly divergent approximation algorithms (*Vertex-Cover* has a 2-approximation).

It is not surprising that *Independent Set* also falls into this class as it is closely coupled to *Clique* (to convert from one to the other, just use $\overline{G}$, the complement of the graph G).

The *Graph-Coloring* problem is also in this class. It takes as input a graph $G = (V, E)$ and outputs a coloring of G using the smallest number of colors (this number is called the *chromatic number* of G). A coloring is a mapping, f:

$$f : V \longrightarrow \text{colors}$$

from the vertices of G to colors such that if v_1 is adjacent to v_2 then $f(v_1) \neq f(v_2)$. That is, adjacent vertices must have different colors. The graph below is an example of a graph whose chromatic number is 3.

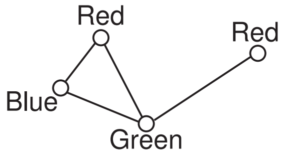

Figure 23.1: A graph whose chromatic number is 3

Coloring has practical applications in scheduling where we map:

$$\begin{array}{lcl} \text{vertices} & \longrightarrow & \text{items} \\ \text{edges} & \longrightarrow & \text{constraints} \\ \text{colors} & \longrightarrow & \text{time slots} \end{array}$$

In this case, we want to minimize the time-slots required to schedule all of the items while maintaining the constraints specifying that some items may not be scheduled at the same time.

Graph-Coloring is clearly related to *Clique* since the chromatic number of a graph is at least as large as the maximum clique. As in *Clique*, the news is not good. if $\mathbf{P} \neq \mathbf{NP}$, then there is no approximation algorithm for *Graph-Coloring* within $|V|^{(\frac{1}{7}-\epsilon)}$. Furthermore, if all of **NP** cannot be solved by Las Vegas type randomized algorithms in polynomial time, then there is no $|V|^{(1-\epsilon)}$ approximation for $\epsilon \geq 0$. The best known approximation algorithm is within $O(\frac{|V|(\log\log(|V|)^2}{(\log|V|)^2})$.

23.1.2 $O(\log n)$ Approximations

The *Set-Cover* problem is analogous to *SAT* for this class in that it is (relatively) easy to prove things about *Set-Cover* and this makes it possible to prove things about other problems in this class. *Set-Cover* takes as input a finite set S and a collection, C, of subsets of S. The output is a minimal subset $C' \subseteq C$ such that every element of S is contained in at least one element of C'. For example:

$$\begin{aligned} S &= \{e_1, e_2, e_3, e_4\} \\ C &= \{\{e_1, e_2\}, \{e_1, e_3\}, \{e_1, e_4\}, \{e_1, e_2, e_4\}\} \\ \text{both } C' &= \{\{e_1, e_2\}, \{e_1, e_3\}, \{e_1, e_4\}\} \\ \text{and } C'' &= \{\{e_1, e_2\}, \{e_1, e_2, e_4\}\} \end{aligned}$$

are solutions but C'' is minimal.

There exist $\log|S|$ approximation algorithms (in fact a greedy algorithm works for this) but if $\mathbf{P} \neq \mathbf{NP}$, then there is no $c|S|$ approximation algorithm for some c. Furthermore there are other conditions believed to be false such that if they are false then we cannot find a $(1-\epsilon)|S|$ approximation for any $\epsilon > 0$. That is, the $\log|S|$ approximation is the best that we do.

23.1.3 Reasonable categories: *Constant*, PTAS and FPTAS

We have seen all of these in previous lectures. Examples of them are:

Constant	*Vertex-Cover*
	Metric-TSP
	Max-Cut
PTAS	*Euclidean-TSP*
FPTAS	*Knapsack*

Another example of a class of *Constant* ratio approximation problems is *Max-k-SAT* which is like *k-SAT* in that its input is a boolean formula, ϕ, in *k-CNF* form but differs in that its output is not whether or not ϕ is satisfiable but rather an assignment that maximizes the number of clauses in ϕ that are satisfied. For *Max-k-SAT* with $k \geq 3$, we have $\frac{1}{1-2^{-k}}$ approximation (notice that as k gets larger, we have better and better approximations). If $\mathbf{P} \neq \mathbf{NP}$ then we cannot do better, i.e., there is no $\frac{1}{1-2^{-k}} - \epsilon$ approximation for any $\epsilon > 0$. Interestingly, *Max-2-SAT* is **NP**-Complete(!) and there is no 1.0476-approximation for *Max-2-SAT*.

23.1.4 Summary

A useful list of the known good news and bad news about many important problems can be found at `http://www.nada.kth.se/~viggo/problemlist/compendium.html`. This reference includes information on general problems and special cases.

23.2 Linear Programming

A linear program is an optimization problem which has a linear objective function and linear inequalities as constraints. Many optimization problems of interest can be expressed as a linear program.

23.2.1 The Diet Problem

An example is the Diet Problem: designing a diet such that the daily nutritional requirement will be satisfied with minimum cost.

input:	an integer n, the number of different foods; an integer m, the number of required nutrients; a matrix a_{ij}, containing the amount of nutrient i in a single unit of food j; a vector b_i, containing the minimum daily requirement of nutrient i; and a vector c_i, containing the cost of a unit of food j.
goal:	Find vector $x = \{x_1, x_2, .., x_n\} \in \Re^n$ that minimizes $\sum_{j=1}^{n} c_j x_j$, where x_j is the quantity of food j to consume per day.
requirements:	For each $i \in 1..m, \sum_{j=1}^{n} a_{ij} x_j \geq b_i$, and for each $j \in 1..n, x_j \geq 0$.

23.2.2 Maximum flow

input:	A directed graph $G = (V, E)$, a matrix $C_{u,v}$ holding the edge capacities(0 if no edge from u to v), and two distinguished node s and t.

The familiar Max-Flow problem can also be stated as a linear programming problem.

Variables:	f_{uv}: flow from u to v for every pair of vertices (u, v), F: value of the flow.
Objective function:	Maximize variable F.
Subject to constraints:	capacity: $\forall u, v \in E, u \neq v, f_{uv} \leq C_{uv}$, skew-symmetry: $f_{uv} + f_{vu} = 0$, and conservation: $B \cdot f + F \cdot d = 0$.

The rows of the matrix B represent vertices of G and the columns represent ordered pairs of vertices (u, v) such that $u \neq v$.

Here is how to fill in values of the matrix B. For every column (u, v):

If $(u, v) \in E$ put +1 in row u, -1 in row v, and 0 elsewhere in the column.
If $(u, v) \notin E$ then fill the column with zeroes.

For example, given G:

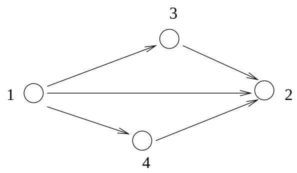

Figure 23.2:

The matrix B would be (all skipped columns are 0's):

B	$(1,2)$	$(1,3)$	$(1,4)$	$(2,1)$	$\ldots$	$(3,2)$	$\ldots$	$(4,2)$	$\ldots$
1	$+1$	$+1$	$+1$	0	$\ldots$	0	$\ldots$	0	$\ldots$
2	-1	0	0	0	$\ldots$	-1	$\ldots$	-1	$\ldots$
3	0	-1	0	0	$\ldots$	$+1$	$\ldots$	0	$\ldots$
4	0	0	-1	0	$\ldots$	0	$\ldots$	$+1$	$\ldots$

and d is a $|V|$-vector which has $+1$ at vertex t, -1 at vertex s and 0 everywhere else.

23.2.3 General Linear Programming problems

Linear programming problems often have the form:

Minimize:	$C \cdot x$, such that
Set of constraints:	$a_j x \geq b_i$ or
	$a_j x = b_i$ or
	$a_j x \leq b_i$ and
	$x_j \leq 0$ or x_j unconstrained
	$x_j \geq 0$

We want to convert general linear programming problems into a standard form so they can be more easily solved.

Minimize: $c \cdot x$,
Subject to: $Ax = b$, and $x \geq 0$
where x is a n-vector,
c is a n-vector,
b is a m-vector, and
A is a $m \times n$ matrix.

The conversion proceeds as follows:

- If $a_i x \geq b_i$ is one of the constraints then replace the constraint with the new constraint $a'_i x \leq b'_i$, and let $a'_i = -a_i$, and $b'_i = -b_i$.
- If $x_j \leq 0$ is one of the constraints then replace it with $x_j \geq 0$, multiply the jth column of A by -1 and the jth entry of c by -1.
- If x_j is unconstrained then replace all occurrences of it with $x'_j - x''_j$ and add constraints $x'_j \geq 0$ and $x''_j \geq 0$, where x'_j and x''_j are new variables.
- If $a_i x \leq b_i$ is one of the constraints then replace it with $a_i x + y = b_i$ and $y \geq 0$. y is called a surplus variable.