6.854 Advanced Algorithms	
Lecture 15: October 29, 2003	Lecturer: Erik Demaine and David Karger
Scribes: Shannon McDonald	

15.1 Introduction

In this lecture we review the two rounding schemes for the max-satisfiability problem discussed in the last class. We contrast these with a third approximation scheme for a new problem, called minimum congestion. We also introduce fixed-parameter algorithms, which, although exponential, use a clever method to limit the extent of intractable complexity. We demonstrate two methods for constructing fixed parameter algorithms: the bounded search tree method and the kernelization method.

15.2 Review

In the previous lecture we discussed two approximation schemes using rounding for the **max-satisfiability problem**:

> Given a collection of "or" clauses over a set of Boolean variables $\{x_i\}$, find an assignment of true and false values to the variables so that the number of true clauses is maximized.

The first approximation scheme employs random guessing. We randomly set each variable to true or false with probability $\frac{1}{2}$. The probability that a clause with k variables is false is the product of the probabilities that each variable is false. Hence we have:

$$p(\text{clause with } k \text{ variables is true}) = 1 - 2^{-k} \geq \frac{1}{2}. \tag{15.1}$$

The expected number of true clauses is the sum of the probabilities that each clause is true:

$$E[\text{number of true clauses}] = \sum_j \left(1 - 2^{-k_j}\right) \geq \frac{1}{2}(\text{number of clauses}) \geq \frac{1}{2}\text{OPT}, \tag{15.2}$$

where j indexes over the clauses, and the jth clause has k_j variables.

In a second, more sophisticated approximation scheme, we begin by writing the integer linear program corresponding to the max-satisfiability problem.

$$\begin{aligned}
\text{maximize} \quad & \sum z_j & (15.3)\\
\text{subject to the constraints} \quad & \sum_{\substack{i \text{ s.t. } x_i \text{ positive} \\ \text{in clause } j}} y_i + \sum_{\substack{i \text{ s.t. } x_i \text{ negated} \\ \text{in clause } j}} (1 - y_i) \geq z_j & (15.4)\\
& 0 \leq y_i \leq 1 \quad \text{for all } i & (15.5)\\
& 0 \leq z_j \leq 1 \quad \text{for all } j & (15.6)
\end{aligned}$$

We relax the problem to the fractional linear program. Relaxation does not decrease the value of OPT, so $\sum z_j \geq$ OPT. We solve the fractional linear program, and for every i, we round each variable x_i by setting it to true with probability y_i. To bound the expected number of true clauses, consider the sequence

$$\{\beta_k\}_{k=1}^{\infty} = \left\{1 - (1 - \frac{1}{k})^k\right\}_{k=1}^{\infty} = \left\{1, \frac{3}{4}, \frac{19}{27}, \dots\right\}. \tag{15.7}$$

This is a decreasing sequence with limit $1 - \frac{1}{e}$. We showed last time that if clause j has k_j variables, then it is true with probability greater than or equal to $\beta_{k_j} z_j$. Hence we have

$$E[\text{number of true clauses}] \geq \sum_j \beta_{k_j} z_j \geq \left(1 - \frac{1}{e}\right) \sum_j z_j \geq \left(1 - \frac{1}{e}\right) \text{OPT}. \tag{15.8}$$

We can further improve our result by combining these two rounding schemes. We run each algorithm once, and take the better solution. The expected number of true clauses (summed over both algorithms) is:

$$\begin{aligned} E[\text{number of satisfied clauses}] &= \sum_j \beta_{k_j} z_j + \sum_j (1 - 2^{-k_j}) && (15.9) \\ &\geq \sum_j (\beta_{k_j} + 1 - 2^{-k_j}) z_j && (15.10) \\ &\geq \frac{3}{2} \sum_j z_j \quad \text{since } \beta_{k_j} \geq 1 - \frac{1}{e} \text{ and } -2^{-k_j} \geq -\frac{1}{2} && (15.11) \\ &\geq \frac{3}{2}\text{OPT}. && (15.12) \end{aligned}$$

Hence at least one of the two schemes achieves at least $\frac{3}{4}$OPT satisfied clauses in expectation.

15.3 Minimum Congestion

The approximation schemes above were reviewed to contrast them with the next approximation scheme, which approximates the **minimum congestion problem**:

> Suppose we have an electronic chip or other substrate, on which there are various points which we wish to connect with wires. The wires run through channels, and each channel can hold only a small number of wires. We wish to find a wiring that does not violate the channel capacity constraints.
>
> More formally, given a graph G, vertex demand pairs $\{s_i, t_i\}$, and edge capacities $w \in Z$ (each edge has the same capacity), find a set of paths such that there is a path from s_i to t_i for each i, and fewer than w paths traverse each edge. The case $w = 1$ corresponds to edge disjoint paths.

This is reminiscent of the single-source, single-sink maximum flow problem, for which an exact integral solution can be found. Unfortunately, if the maximum flow problem is posed with two demand pairs $\{(s_1, t_1), (s_2, t_2)\}$, the problem of finding edge-disjoint paths becomes NP-complete. We relax the minimum congestion problem to a soluble problem, the **multicommodity flow problem**:

> Find a flow on G that obeys capacity constraints, and that sends one unit of flow from s_i to t_i for each i. Commodities cannot be exchanged between different terminals. We may split up the flow from each s_i to t_i along several paths, so the solution need not be integral.

The multi-commodity flow problem has a linear programming solution. The linear program includes the maximum flow balance constraints for every s_i and t_i separately. We require that the sum of the flow through each edge is bounded by the capacity. The linear program is:

$$\text{maximize} \quad \sum_i f^i_{e=(t_i,s_i)} \tag{15.13}$$

$$\text{subject to} \quad \underbrace{\sum_x f^i_{e=(v,x)}}_{\text{outgoing flow}} - \underbrace{\sum_x f^i_{e=(x,v)}}_{\text{incoming flow}} = 0 \quad \text{for all } v \text{ and } i \tag{15.14}$$

$$\sum_i f^i_e \leq u_e = w \quad \text{for all edges not of the form } (t_i, s_i) \tag{15.15}$$

$$f^i_e \geq 0 \quad \text{for all } e \text{ and } i. \tag{15.16}$$

The variable f^i_e represents the flow of commodity i on edge e. Note that as in the linear program for the max flow problem (lecture 11), arcs of infinite capacity from t_i to s_i have been added.

Solving this linear program gives us the fractional flow. We want to use the solution to choose one path from s_i to t_i for each commodity pair. We saw with maximum flow that we can decompose the flow of commodity i into paths from s_i to t_i of total capacity one. Since the sum of the flows is one, the flows define a probability distribution on the paths. We draw a path from this distribution.

We calculate the expected number of paths through one edge as:

$$E[\text{number of paths through edge } e] = \sum_i p(\text{path } i \text{ chosen at edge } e) = \sum_i f^i_e \leq w \tag{15.17}$$

This is more challenging than the maximum satisfiability problem because we must ensure that our solution satisfies the capacity constraints to be valid. At any edge, by how much does the flow exceed w? To answer this question, we require the following two theorems.

Theorem 1 **[Chernoff's Bound]** *Let $X_1, ..., X_n$ be independently (though not necessarily identically) distributed random variables taking on values in the interval $[0, 1]$. Define $\mu = \sum_i E[X_i]$ to be the sum of their expectations. Then if $\epsilon \leq 1$,*

$$p\left(\sum_i X_i \leq (1-\epsilon)\mu\right) \leq e^{-\frac{\epsilon^2 \mu}{2}} \tag{15.18}$$

and

$$p\left(\sum_i X_i \geq (1+\epsilon)\mu\right) \leq e^{-\frac{\epsilon^2\mu}{4}}. \tag{15.19}$$

If $\epsilon \geq 1$, *then*

$$p\left(\sum_i X_i \geq (1+\epsilon)\mu\right) \leq 2^{-(1+\epsilon)\mu}. \tag{15.20}$$

■

To use this bound in the analysis of the minimum congestion problem (with $w = 1$), consider a particular edge. Set

$$X_i = \begin{cases} 1 & \text{if edge } e \text{ is used to connect } s_i \text{ and } t_i \\ 0 & \text{otherwise.} \end{cases} \tag{15.21}$$

Then $p(X_i = 1) = f_e^i$, so

$$\mu = E[\sum X_i] = \sum f_e^i \leq w = 1. \tag{15.22}$$

If we set $t = (1+\epsilon)\mu$, then by Chernoff's bound (equation 15.20) we obtain:

$$p(\sum X_i \geq t) \leq 2^{-t}. \tag{15.23}$$

The second theorem is a result from elementary probability theory.

Theorem 2 [Union Bound] *If A_i is an event for $i = 1, 2, ...n$, then $p(\cup_i A_i) \leq \sum_i p(A_i)$.* ■

For our analysis, we consider the event that edge e has t or more paths passing through it. Applying the union bound gives the inequality:

$$p(\text{any edge has} \geq t \text{ paths}) \leq \sum_e p(\text{edge } e \text{ has} \geq t \text{ paths }) \leq 2^{-t}m. \tag{15.24}$$

In particular, if $t = 1 + \log m$, then $p(\text{any edge has } \geq t \text{ paths }) \leq \frac{1}{2}$. Thus we have bounded the probability that any edge has $1 + \log m$ paths passing through it, and so we have a $(1 + \log m)$-approximation to min congestion.

This is different than the max sat approximation algorithms, because here we have a feasible approximation with some probability. To get around this, we run the algorithm again and again until we obtain a solution. In fact, we don't have to solve the linear program multiple times; we need only choose paths according to their probability distribution multiple times.

Notice also that we have shown that there is some solution to minimum congestion that does not violate the capacity constraints, since there is a nonzero probability of obtaining a solution.

If w is large (for example, if $w = \log m$), then we can apply the middle Chernoff bound. Then we have

$$p\left(\sum_i X_i \geq (1+\epsilon)\log m\right) \leq e^{-\frac{\epsilon^2 \log m}{4}}. \tag{15.25}$$

We adjust ϵ to obtain a constant factor approximation.

15.4 Fixed Parameter Algorithms

For the past few lectures, we have discussed algorithms which give approximate solutions to hard problems, for which polynomial time algorithms do not exist. But what if an exact solution to a hard problem is needed? Fixed parameter algorithms provide exact solutions by dividing the problem into a tractable part with polynomial complexity, and a bad, intractably complex part. The goal is to ensure that most of the problem is in the former category. Part of the problem, however, contributes to the bad part. The parameter is in this part of the problem. [1]

We illustrate fixed parameter algorithms with the **k-vertex cover problem**:

> Given a graph G and a parameter k, does there exist a set C of at most k vertices such that for every edge e in G, there exists some vertex $v \in e \cap C$.

Hence in k-vertex cover, the size of the vertex cover is the parameter.

An **easy algorithm** iterates over each of the $\frac{n!}{k!(n-k)!}$ possible vertex covers with size k. This takes $O(n^k \cdot m)$ time. But we would prefer to decompose the problem size into a function of k and a polynomial function that does not depend on k. If we can achieve this, the running time will be $f(k) \cdot n^{O(1)}$.

15.4.1 Bounded Search Tree Method

A **smarter algorithm** uses the **bounded search tree method**. The idea of this method is to solve the problem in two steps [1]:

1. Compute the search space, which is an exponential tree.
2. Efficiently solve each branch of the tree using a search algorithm such as DFS.

In the case of k-vertex cover, the smarter algorithm works as follows:

1. Pick any edge $e = (v, w)$ in the graph. It must be covered by either v or w.
2. Guess which endvertex is in the vertex cover C. Each guess is essentially a duplication of the search space.
3. Add the vertex to C.
4. Remove that vertex and all of its incident edges from G. We now have a smaller graph.
5. Repeat.

This creates a binary tree, where one path from the root to a leaf specifies the vertices in one possible vertex cover. The height of the tree is at most k, since we want a vertex cover of size k. There are at most 2^k leaves, so the running time is $O(2^k \cdot n)$.

15.4.2 Definitions

Now that we have an intuitive understanding of fixed parameter algorithms, we formalize the main ideas:

Definition 1 *A* **parameter** *is a function from the space of problem instances to the natural numbers. A* **parameterized problem** *is a problem given with a parameter.*

Think of a parameter as a positive integer associated with each instance of a problem. For example, some parameters arising naturally in various problems are the size of a vertex cover, the number of processors in a parallel processing system, or the number of species whose DNA sequences are to be aligned [1]. We want to choose a parameter that is usually small in practice. This will ensure that the intractably complex part of the problem does not grow out of control.

Definition 2 *A problem is* **fixed parameter tractable (FPT)** *if there exists some algorithm that solves it with running time $f(k) \cdot n^{O(1)}$. Such a problem is referred to as having* **fixed parameter tractability**.

For example, k-vertex cover is fixed parameter tractable.

Definition 3 *A problem is* **W[1]-hard** *if there is no fixed parameter algorithm to solve it.*

For example, k-clique is W[1]-hard. The class W[1] is analogous to the class NP when using a nondeterministic Turing machine that is allowed only k steps on any branch of computation.

15.4.3 Kernelization

A second method used in fixed parameter algorithms is kernelization.

Definition 4 **Kernelization** *is the method of reducing a problem to a kernel of size at most $f(k)$. A* **kernel** *is a problem which is "equivalent" to the original problem.*

There is a clever kernelization for the k-vertex cover problem. Assume that the graph is simple, i.e. it has no loops or multiple edges. The algorithm works as follows:

1. Set p equal to the number of vertices in G with degree greater than k. Any such vertex v must be in the cover, because otherwise each of the more than k vertices adjacent to v must be in the cover. Hence we may add v to the cover, and then remove it and its incident edges from consideration. After processing each vertex v with $d(v) > k$, we obtain a graph of maximum degree at most k. If $p > k$, then there cannot be a vertex cover. [1]

2. If there are at least $k^2 + 1$ edges, then there is no vertex cover, because each vertex has degree at most k, and hence covers at most k edges.

3. If the number of edges is at most k^2, then use a different algorithm to find a $(k-p)$-vertex cover. The problem of finding the $(k-p)$-vertex cover is the kernel. It is equivalent to the original problem because a k-vertex cover exists in G if and only if a $(k-p)$-vertex cover exists in the graph considered at this step.

There are at most $2k^2$ vertices, so if we apply the easy k-vertex cover algorithm, then step 3 takes $O\left(n_{\text{step 3}}^k \cdot m_{\text{step 3}}\right) = O\left((2k^2)^k k^2\right)$ time. In addition, the first step takes $O(m+n)$ time to compute the degree of each vertex, so the total running time is $O\left(2^k \cdot k^{2k+2}\right) + O(m+n)$.

If we apply the smarter algorithm, the third step takes $O\left(2^k \cdot n_{\text{step 3}}\right) = O\left(2^k \cdot 2k^2\right)$ time, so the total running time is $O(2^k \cdot 2k^2) + O(m+n)$.

Theorem 3 *Given a problem to be solved, the following are equivalent:*

1. *The problem is fixed parameter tractable, i.e. there exists some $f(k) \cdot n^{O(1)}$ algorithm to solve it.*
2. *There exists some $f(k) + n^{O(1)}$ algorithm to solve the problem.*
3. *There exists a kernelization for the problem.*

■

Note that this theorem is existential rather than constructive; it does not provide a method for arriving at the kernelization.

15.4.4 Connection to Approximation Algorithms

Any optimization problem may be converted to a decision problem, which asks whether or not OPT is greater than or equal to k. The value k can be thought of as a parameter in the decision problem.

The next theorem requires the following definition:

Definition 5 *An* **efficient polynomial time approximation scheme (EPTAS)** *is a polynomial time approximation scheme with running time $f(\frac{1}{\epsilon}) \cdot n^{O(1)}$. Hence the running time may be exponential in $\frac{1}{\epsilon}$, but must be polynomial in n.*

Theorem 4 *If the optimization problem has an EPTAS, then the associated decision problem is fixed parameter tractable.*

Proof: We assume that the problem is integral. Consider a maximization problem with an EPTAS for its solution. The associated decision problem asks whether or not OPT$\geq k$. Set $\epsilon = \frac{1}{2k}$, and run the EPTAS using that value of ϵ.

Let C be the solution obtained by the EPTAS. If $C \geq k$, then OPT$\geq C \geq k$, so the answer to the associated decision problem is YES.

If $C < k$, then

$$\text{OPT} \leq \left(1 + \frac{1}{2k}\right) C < k + \frac{1}{2}. \tag{15.26}$$

Since the problem is integral, this implies that OPT is at most k.

The argument for minimization problems is symmetric. ■

References

[1] R.G. Downey and M.R. Fellows. *Parameterized Complexity.* Springer-Verlag, New York, 1997.