CS270 Combinatorial Algorithms & Data Structures **Spring 2003**

Lecture 21: 4.8.03

Lecturer: Christos Papadimitriou *Scribe: Brian Milch, Rachel Rubin*

21.1 Approximation Algorithms...with a twist

- NP-complete problems are all disguises of the same thing
- But approximation algorithms are diverse!
- So far we have seen approximation algorithms for vertex cover, set cover and TSP with triangle inequality. We have celebrated small victories, like an improvement of a factor of 2
- But now there is a different class of approximation algorithms...

21.2 Approximation of the Knapsack Problem

We have a knapsack that can only hold a certain weight, and we want to maximize the total value of the items in the knapsack. We have a group of n items to choose from each with a positive weight w_i and a positive value v_i. There is only one copy of each item. Let $V = \max_i v_i$.

The problem is to find $S \subseteq \{1...n\}$ that maximizes $\sum_S v_i$ subject to $\sum_S w_i \leq W$.

There are several things to note about the KNAPSACK problem:

1. It is NP-complete
2. It can be solved in $O(nW)$ time with dynamic programming by filling in a table that gives, for each possible total weight, the most valuable combination of items with this weight
3. It can be solved in a dual way by filling a table that gives, for each possible value, the lightest combination of items with this value. This takes $O(n^2V)$ time.
4. It is approximable by using #3 with a twist

For example, let's look at a particular instance of KNAPSACK. Let $n = 100$. You are given the values:

value	5831	7181	4029	5113	8158	...
weight	683	305	...	...	....	...

These values make a huge table (8158 * 100 items) that you would have to fill in with the algorithm of #3.

So, how do you cut corners? Round the values!

21.2.1 Rounding the Values for Approximation

In this case, we will round down to the nearest 100.

value	5800	7100	4000	5100	8100	...
weight	683	305	...	...	....	...

Can we get any bound on the error of this approximation? Let S be the optimum solution of the original problem. Let S' be the optimum with the rounded values. We will use v_i' to denote the rounded version of v_i.

$$\begin{aligned}
\sum_S v_i &\geq \sum_{S'} v_i \quad \text{because } S \text{ is optimum for } v_i\text{'s} \\
&\geq \sum_{S'} v_i' \quad \text{because } v_i' \leq v_i \\
&\geq \sum_{S} v_i' \quad \text{because } S' \text{ is optimum for } v_i'\text{'s} \\
&\geq \sum_{S} v_i - n * 100 \quad \text{because } |S| \leq n \text{ and no number is rounded down by more than 99}
\end{aligned}$$

$$\Rightarrow \sum_S v_i - \sum_{S'} v_i \leq n * 100$$

Relative error:

$$\frac{\sum_S v_i - \sum_{S'} v_i}{\sum_S v_i} \leq \frac{n * 100}{V}$$

The inequality holds because V is an upper bound on $\sum_S v_i$. Note that if V is small, dynamic programming is OK. If V is large, the equation above shows that the relative error is small.

21.2.2 Approximation by Truncation

How much time does it take to solve the problem with rounded values? Once we round, we can truncate the last few digits of the values, yielding smaller numbers. Back to the example.

val	58	71	40	51	82	...
wts	683	305	...	...	....	...

What if instead of truncating off the last 2 digits, I truncate off the last b bits of the values? Then the 100 changes to 2^b:

$$\frac{\sum_S v_i - \sum_{S'} v_i}{\sum_S v_i} \leq \frac{n * 2^b}{V} = \epsilon$$

Suppose we have some ϵ that we want to achieve. Then we can solve for b.

$b = \log \frac{V\epsilon}{n}$

The maximum v_i' value is $V/2^b$. So we can solve the new problem in time $O(n^2 \frac{V}{2^b}) = O(\frac{n^3}{\epsilon})$

So the running time depends on the accuracy that you want! There is a family of polynomial-time algorithms, one for every ϵ. This is called a *polynomial-time approximation scheme* (PTAS).

21.3 The Family of Approximation Algorithms

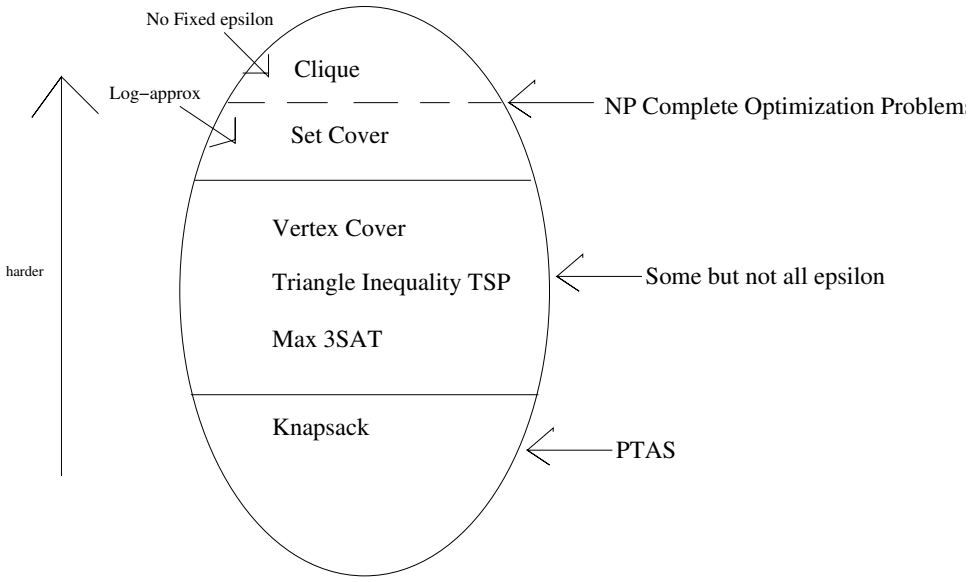

Figure 21.1: There are several categories of NP-complete problems in terms of their approximability.

As illustrated in Figure 21.1, different NP-complete optimization problems can be approximated to different degrees of accuracy in polynomial time. Why can't we just reduce each problem to, say, KNAPSACK, and approximate it using our PTAS for KNAPSACK? The reason is that reductions between NP-complete problems only preserve polynomial-time solvability, not polynomial-time approximability. If A reduces to B, that does not imply that an approximation algorithm for B provides a similarly good approximation algorithm for A.

How do we know that certain problems don't belong lower down in this diagram — for instance, that there is no PTAS for VERTEX-COVER? The claim is that this diagram is correct unless P = NP. We can actually do hardness proofs for approximations: for instance, later in this lecture we prove that unless P = NP, the general traveling salesman problem (without the triangle inequality) has no ϵ-approximation algorithm for any ϵ. Some results of this type have been known for a long time. However, a big breakthrough — the proof that problems such as MAX-3SAT and triangle-inequality TSP have no PTAS — occurred only ten years ago [1]. It's encouraging to know that two of the authors of this paper were Berkeley grad students at the time.

21.4 Approximating Max-3SAT

Before we move on to an approximation hardness proof, we examine a problem in Figure 21.1 that we haven't talked about before: MAX-3SAT. The problem is, given a logical formula in 3-CNF, to find a truth assignment that maximizes the number of satisfied clauses. The decision version of this problem is obviously NP-complete, since it has 3SAT as a special case.

Consider what happens if we just select a random truth assignment for a given formula. Each clause has 3 literals, and each literal is satisfied independently with probability 1/2, so the probability that all three literals are unsatisfied is $(1/2)^3 = 1/8$. So any given clause is satisfied with probability $1 - 1/8 = 7/8$.

Thus, the expected fraction of clauses satisfied in the whole formula is 7/8. To see this, let C be the set of clauses, and for each $i \in C$ define the random variable:

$$S_i = \begin{cases} 1 & \text{if clause } i \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

Then:

$$\begin{aligned} (\#\text{true clauses}) &= \sum_{i \in C} S_i \\ \mathcal{E}(\#\text{true clauses}) &= \sum_{i \in C} \mathcal{E} S_i \quad \text{by linearity of expectation} \\ &= \frac{7}{8} n \end{aligned}$$

where n is the number of clauses.

The maximum number of satisfied clauses is no more than n, so the *expected* relative error of simply picking a truth assignment at random (as an approximation algorithm for MAX-3SAT) is $\leq 7/8$. In fact it is possible to de-randomize this algorithm, choosing appropriate values for the variables (rather than flipping coins) so that the relative approximation error is guaranteed to be $\leq 7/8$.

21.5 Hardness of Approximating the General TSP

In this section, we discuss the general version of the traveling salesman problem, without assuming distances satisfy the triangle inequality.

Theorem 21.1 *Unless $P = NP$, there is no ϵ-approximation algorithm for the TSP for any $\epsilon > 0$.*

This means, for example, that no TSP algorithm is guaranteed to return a solution whose cost is less than 100 times the optimal cost.

Proof: Recall that we proved TSP to be NP-hard by the following reduction from HAMILTON, the problem of seeing whether a graph has a Hamiltonian cycle:

$$\begin{array}{lll} \text{HAMILTON} & \rightarrow & \text{TSP} \\ G = (V, E) & & d(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 2 & \text{otherwise} \end{cases} \end{array}$$

Note that the distances 1 and 2 always satisfy the triangle inequality $d_{ij} + d_{jk} \leq d_{ik}$. But now suppose we replace the 2 with $2 + \epsilon n$:

$$d(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 2 + \epsilon n & \text{otherwise} \end{cases} \tag{21.1}$$

We claim that with this reduction, an ϵ-approximation algorithm for TSP will allow us to solve HAMILTON.

In our old reduction, we said:

$$\begin{array}{lll} \exists \text{ Hamiltonian cycle} & \rightarrow & \text{optimum TSP cost } n \\ \nexists \text{ Hamiltonian cycle} & \rightarrow & \text{optimum TSP cost} \geq n + 1 \end{array}$$

Now we have:

$$\begin{aligned} \exists \text{ Hamiltonian cycle} &\rightarrow \text{optimum TSP cost } n \\ \nexists \text{ Hamiltonian cycle} &\rightarrow \text{optimum TSP cost} \geq n + 1 + \epsilon n \end{aligned}$$

Thus, there is a gap in the set of possible solution costs: no solution has cost greater than n but less than $n + 1 + \epsilon n$.

So here is an algorithm for solving HAMILTON, assuming we have an ϵ-approximation algorithm A for TSP:

- given a graph G, create $d(i, j)$ as in Eq. 21.1, and run A on it
- if the tour returned by A has cost n, output "yes"
- if the tour returned by A has cost $> n$, output "no"

Note that because of the gap, if the cost of the tour returned is $> n$, it must be $> n(1 + \epsilon)$. If there were a Hamiltonian cycle, the true optimum cost would be n, so the tour returned by our ϵ-approximation algorithm would have cost $\leq n(1 + \epsilon)$. So we are justified in returning "no" in the second case.

Thus, we have a polynomial-time reduction of HAMILTON to ϵ-approximation of TSP. This implies ϵ-approximation of TSP can't be done in polynomial time unless P = NP. ■

The challenge in proving a problem hard to approximate is to create a gap, as we did in the proof above. This can be extremely hard for problems other than TSP.

21.6 Approximating Max-Clique

Figure 21.1 shows MAX-CLIQUE in the top part of the diagram, as a problem with no constant-factor approximation. We are not yet ready to prove this result, but we can prove that MAX-CLIQUE belongs in either the top or the bottom section of the diagram, not in the middle.

Theorem 21.2 *If* $\exists \epsilon > 0$ *such that* MAX-CLIQUE *is* ϵ*-approximable, then* MAX-CLIQUE *is* ϵ*-approximable for all* $\epsilon > 0$ *(i.e., it has a PTAS).*

Proof: It suffices to show that for any ϵ, the existence of an ϵ-approximation implies the existence of an $(\frac{\epsilon}{2})$-approximation. We do this using a self-reducibility argument, reducing one MAX-CLIQUE probem to another MAX-CLIQUE problem on a different-sized graph.

For any graph G of size n, we can construct a graph G^2 as shown in Figure 21.2: we replace each node of G with a cluster of nodes that is a copy of the entire graph G. If there is an edge (u, v) in G, then we connect every node in the cluster corresponding to u to every node in the cluster corresponding to v. There are no other edges between clusters.

It is clear that G^2 has n^2 nodes. Also, if there is a clique of size k in G, then there is a clique of size k^2 in G^2: in each of the k clusters corresponding to nodes in the clique, there is a similar clique of size k, and all these k^2 nodes form a big clique.

Conversely, if there is a clique of size k^2 in G^2, then either:

- the nodes of this clique reside in $\geq k$ separate clusters, in which case those k clusters must correspond to nodes that form a clique of size k in G;

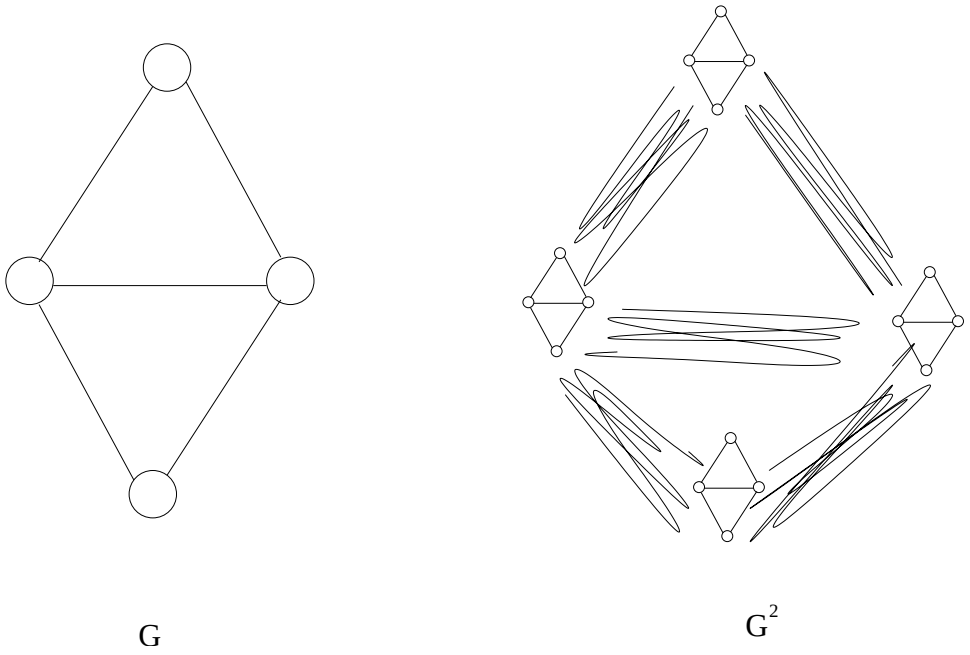

Figure 21.2: A pair of graphs G and G^2. The scribbled edges between clusters in G^2 represent edges between all nodes in one cluster and all nodes in the other cluster.

- some single cluster contains $\geq k$ of the nodes in this clique, in which case those k nodes form a clique in the cluster, and the corresponding k nodes form a clique in G.

So in either case, if we find a clique of size k^2 in G^2, we can construct a clique of size k in G.

Now suppose we have an ϵ-approximation algorithm A for MAX-CLIQUE. Given a graph G, we can obtain an $\frac{\epsilon}{2}$-approximation to MAX-CLIQUE(G) as follows:

- Run A on G^2. Let t be the size of the clique returned.
- Reconstruct the corresponding clique of size $\sqrt{t}$ in G as described above, and return this clique.

To see that this algorithm has relative error $\leq \frac{\epsilon}{2}$, suppose the maximum clique size in G is c. Then, by the arguments above, the maximum clique size in G^2 is c^2. So:

$$\begin{aligned} t &\geq \frac{c^2}{1+\epsilon} \quad \text{because } A \text{ gives an } \epsilon\text{-approximation} \\ \sqrt{t} &\geq \sqrt{\frac{c^2}{1+\epsilon}} = \frac{c}{\sqrt{1+\epsilon}} \\ \sqrt{t} &\geq \frac{c}{1+\frac{\epsilon}{2}} \quad \text{because } (1+\epsilon)^{(1/2)} \leq 1+\tfrac{\epsilon}{2} \end{aligned}$$

So given an ϵ-approximation algorithm, we can construct an $\frac{\epsilon}{2}$-approximation algorithm, which means MAX-CLIQUE either has a PTAS or has no constant-factor approximation. ■

References

[1] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 14–23, 1992.