Advanced algorithms
Approximation algorithms Notes

Søren Dahlgaard

June 15, 2011

1 Disposition

1. Motivation. NP-Hard, probably intractable
2. Approximation factor $\max(C/C^*, C^*/C)$.
3. Approximation schemed $(1+\epsilon)$-approximation. Polynomial in ϵ.
4. Randomized. Expected value $C \Rightarrow \max(C?C^*, \ldots$
5. Simple example Vertex cover.
6. Say that linear programming relaxation can be okay.
7. Set-covering.
8. Proof of set-covering.

2 summary

Just because a problem is NP-Complete and (probably) intractable we don't give up. if C^* is the value (or cost) of the optimal solution and we have an algorithm that produces another solution C we say that the algorithm has an approximation ratio of $\rho(n)$ if

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

We call such an algorithm a $\rho(n)$-approximation algorithm. We assume that a solution has positive cost.

We also talk about approximation schemes. These take as input an $\epsilon > 0$ such that the scheme is a $(1+\epsilon)-$approximation algorithm. The running time of such a scheme can grow rapidly as ϵ gets smaller: $O(n^{2/\epsilon})$.

We call a scheme fully polynomial if its running time is polynomial in both $1/\epsilon$ and n. eg. $O((1/\epsilon)^2 n^3)$.

3 Vertex cover

An approximation scheme takes an edge $(u, v) \in E$ arbitrarily and adds u, v to S. It then removes all edges in E incident on either u or v and picks another edge if one is present.

Let A be the edges checked by the algorithm. Any vertex cover must include either u or v, so $|C^*| \geq |A|$. We have that $|C| = 2|A|$, so $|C| \leq 2|C^*|$. Thus this algorithm is a 2-approximation.

4 TSP

We focus on TSP with the triangle inequality. That is $c(u, w) \leq c(u, v) + c(v, w)$ for all $u, v, w \in V$. The approximation algorithm is to first create a MST of G. Then visit the vertices of a preorder walk of this MST.

It is clear that the size of a MST T is less than the size of an optimal traveling salesman tour. Eg. removing an edge from C^* yields a spanning tree which certainly cannot have lower weight than T. A preorder of the tree visits every vertex and uses every edge of T exactly twice. Let this full walk be W, we have $c(W) \leq 2c(T) \leq 2c(C^*)$. W is however likely not a tour, so we delete duplicate vertices and the triangle inequality gives us $c(C) \leq c(W) \leq 2c(C^*)$.

4.1 General TSP

We can show that an approximation algorithm for the general TSP if $P \neq NP$. We could use this approximation algorithm to solve the ham-cycle problem.

Because we know the approximation factor $\rho \geq 1$ we can create a special graph for which a TSP tour shorter than $\rho|V|$ corresponds to a hamiltonian cycle. Just use the following cost function for G':

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ \rho|V| + 1 & \text{otherwise} \end{cases}$$

Any tour using edges only in E will have length $|V|$. Any tour using at least one edge not in E will have length at least $(\rho|V| + 1) + (|V| - 1) > \rho|V|$, so the approximation algorithm cannot return a tour using edges not in E or it would not be a ρ-approximation.

5 Set-covering

Given a set of elements X and a family of subsets of X called F we wish to find the smallest size subset $C \subseteq F$, such that the union of C is X.

Our approximation algorithm works by picking the set that covers most uncovered elements. It is easy to see that this algorithm runs in polynomial time.

Proof. Let C be the subset chosen by the algorithm and let $S_1, \ldots, S_{|C|}$ be the sets in C. We assign a cost of 1 to each of these sets spread over the elements that the given set was the first to cover. So for each element in $x \in S_1$ we have

$c_x = 1/|S_1|$. For $x \in S_2$ we have either $c_x = 1/|S_1|$ if x was first covered by S_1. Otherwise we have $c_x = 1/|S_2 \setminus S_1|$, for S_3 this will be $c_x = 1/|S_3 \setminus (S_1 \cup S_2)|$.

In total we have $|C| = \sum_{x \in X} c_x$.

Each x is in at least one set $S \in C^*$. Therefore we must have

$$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x$$

Which gives:

$$|C| \leq \sum_{S \in C^*} \sum_{x \in S} c_x$$

We now use the fact that:

$$\sum_{x \in S} c_x \leq H(|S|) = 1 + 1/2 + 1/3 + \ldots + 1/|S|$$

This gives:

$$\begin{aligned} |C| &\leq \sum_{S \in C^*} H(|S|) && (1) \\ &\leq |C^*| \cdot H(\max\{|S| : S \in F\}) && (2) \end{aligned}$$

This proofs the theorem. □

Proof. We also need a proof for the "fact" given above. For any set $S \in F$ let $u_0 = |S|$. Let $u_1 = |S \setminus S_1|$, etc. Pick k such that $u_k = 0$ and $u_{k-1} > 0$. We then have:

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})}$$

We also have

$$|S_i \setminus (S_1 \cup \cdots \cup S_{i-1})| \geq |S \setminus (S_1 \cup \cdots \cup S_{i-1})| = u_{i-1}$$

Because otherwise S would've been picked before S_i. Also note that we can have some elements in S_i not covered by S.

Therefore

$$\begin{aligned} \sum_{x \in S} c_x &\leq \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot 1/u_{i-1} && (3) \\ &= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} 1/u_{i-1} && (4) \\ &\leq \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} 1/j && (5) \\ &= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} 1/j - \sum_{j=1}^{u_i} 1/j \right) && (6) \end{aligned}$$

This gives us a telescoping series of harmonic numbers that results in $H(|S|)$. □

This algorithm is therefore a $(\ln |X| + 1)$-approximation. Note that it's the logarithm of $|X|$ not $|C^*|$!

6 Randomized approximation

Maybe skip this or introduce it along with normal approximation algorithms.

If an algorithm yields a result with expected cost C we call it a randomized $\rho(n)$-approximation algorithm.

If we want to know how many clauses of a 3-CNF we can satisfy at most, we can try setting each variable to 1 with probability 1/2. The probability that a clause is satisfied is then $1 - (1/2)^3 = 7/8$

Summing these up give $7m/8$ and since m is an upper bound it gives $m/(7m/8) = 8/7$.

7 Linear programming

Let minimum-weight vertex cover be just like vertex-cover except each vertex has an associated cost $w(v)$, and we want to minimize the cost of the vertex cover.

We can make the following integer program:

$$\text{min.} \sum_{v \in V} w(v)x(v)$$

$$\begin{aligned} \text{s.t. } x(u) + x(v) &\geq 1 && \text{for } (u,v) \in E && (7) \\ x(v) &\in \{0,1\} && \text{for } v \in V && (8) \end{aligned}$$

Linear relaxation is when we replace the constraint $x(v) \in \{0,1\}$ with $0 \leq x(v) \leq 1$.

Our algorithm solves this relaxed program and adds a vertex v to the solution iff $x(v) \geq 1/2$.

Proof. Let C^* be the optimal solution to the problem and let z be the optimal solution to the relaxed linear program. Because C^* is a feasible solution to this problem we must have $z \leq w(C^*)$.

If we by C denote the cover induced by the solution with obj. val. z we have that C is a vertex cover. Why? Because for any edge $(u,v) \in E$ we must have either $x(u) \geq 1/2$ or $x(v) \geq 1/2$. Thus every edge is covered.

We have:

$$z = \sum_{v \in V} w(v)x(v) \tag{9}$$

$$\geq \sum_{v \in V : x(v) \geq 1/2} w(v) \cdot 1/2 \tag{10}$$

$$= \sum_{v \in C} w(v) \cdot 1/2 \tag{11}$$

$$= 1/2 \cdot w(C) \tag{12}$$

So $w(C) \leq 2z \leq 2w(C^*)$.

□

8 Subset-sum

In the optimization version we want to find a subset $S' \subseteq S$ such that the sum of S' is as large as possible without exceding t.

We will give a fully polynomial-time approximation scheme. The idea is for each $x_i \in S$ we have a list of all possible values by combining $x_1, \ldots, x_i$. We remove values greater than t. This approach takes exponential time. That is, $L_i = L_{i-1} \cup L_{i-1} + x_i$. If we use a merge procedure like the one for merge sort this list will remain sorted.

The idea to make this polynomial is to use a parameter δ such that if we have two elements in $y \in L_{i-1}$ we might remove it if we have a $z \in L_i$ such that $y/1 + \delta \leq z \leq y$. For instance if $\delta = 0.1$ we have $11/1.1 \leq 10 \leq 11$, so 10 could represent 11 in our list.

We can trim a sorted list according to a δ in linear time. If we get an $0 < \epsilon < 1$ and want a $1 + \epsilon$ approximation factor we can use $\epsilon/2n$ as δ to trim all lists L_i. For instance if $\epsilon = 0.4$ and $n = 4$ we have $\delta = 0.05 = 0.4/8$.

Proof. It is obvious that the amount returned by the algorithm is indeed the sum of some subset because we don't add anything other than subset sums to the lists.

Let y^* be the optimal solution. We know that $z \leq y^*$, so we need to show that $y^*/z \leq 1 + \epsilon$ to show correctness of the algorithm.

For $y \in P_i, y \leq t$ we have some element $z \in L_i$ such that

$$\frac{y}{(1 + \epsilon/2n)^i} \leq z \leq y$$

This is because at each step we have a $\frac{y}{1+\delta} \leq z \leq y$, so through induction we must have $\frac{y}{(1+\delta)^i} \leq z \leq y$.

$$y^*/z \leq \left(1 + \frac{\epsilon}{2n}\right)^n$$

Because y^* is the largest element in P_n less than t and z is the same but for L_n. We have

$$\left(1 + \frac{\epsilon}{2n}\right)^n \leq 1 + \epsilon$$

This follows from a lot of shit scattered all over the book (and some excercises). This proves the approximation-factor.

For running time observe that the running time of the algorithm is polynomial in the lengths of $|L_i|$ for all the lists. After trimming a such list we have that two elements z_i, z_{i+1} must have $z_{i+1}/z_i > 1 + \epsilon/2n$. A list therefore has at most as many values as this list:

$$\{0, 1, 1 + \epsilon/2n, (1 + \epsilon/2n)^2, \ldots, (1 + \epsilon/2n)^k\}$$

where k is the largest integer such that the expression is $\leq t$. The list thus has:

$$\log_{1+\epsilon/2n} t + 2 = \frac{\ln t}{\ln(1 + \epsilon/2n)} + 2$$

elements. This is also $< \frac{3n \ln t}{\epsilon} + 2$ because $0 < \epsilon < 1$.

This is polynomial in the input size which must certainly be greater than n plus the amount of bits, $\lg t$ needed to represent t.

□