Randomized Rounding

Kent Quanrud

September 23, 2020

These notes – particularly the section on set cover – is still a work in progress. For more on set cover, we recommend either the handwritten notes prepared before class:

https://randomized-algorithms-fall-2020.s3.amazonaws.com/Randomized+rounding+(Fall+2020)+handwritten+notes.pdf

or the following YouTube video from Spring 2020:

https://youtu.be/rEBU8tixn7I

1 SAT

We will start from *the* NP-Hard problem, boolean satisfiability (SAT). For ease of notation, let `t` denote the boolean value "true" and let `f` denote the boolean value "false".[1] Recall that in SAT, we are given a boolean formula $f : \{\mathtt{t}, \mathtt{f}\}^n \to \{\mathtt{t}, \mathtt{f}\}$, such that

$$f(x_1, x_2, x_3) = (x_1 \vee x_2) \wedge \bar{x}_3$$

Here $\wedge$ stands for "and", $\vee$ stands for "or", and $\bar{x}$ stands for "not x". The goal is to choose an assignment $x_1, \ldots, x_n \in \{\mathtt{t}, \mathtt{f}\}$ such that $f(x_1, \ldots, x_n) = \mathtt{t}$. This problem is incredibly hard because boolean formulas are so expressive. In particular, **cook** observed that polynomial size boolean formulas can simulate polynomial time algorithms. This implies that *any* problem where a proof can be verified in polynomial time can be rewritten as a boolean satisfaction problem (namely, find the input that satisfies the (boolean formulation of the) verification algorithm.)

Since solving SAT formulas exactly is formidably hard, we instead consider *approximation algorithms* that try to satisfy "as much as possible". Of course, satisfying a SAT formula would appear to be an all or not thing proposition, so we assume a certain canonical form where the extent of satisfaction can be measured. Recall that "conjunction" is mathematical jargon for "and" and "disjunction" is mathematical jargon for "or". A SAT formula is a **conjunction** if it is the "and" of several variables:

$$f(x_1, \ldots, x_n) = x_1 \wedge \bar{x}_3 \wedge x_4$$

A SAT formula is in **conjunctive normal form** if it is a conjunction of clauses that are each disjunctions: e.g., of the form

$$f(x_1, x_2, \ldots) = (x_1 \vee x_2 \vee \not{x}_3) \wedge (x_2)$$

[1] Alternatively, `nil`?

We note that any boolean formula can be converted to be a CNF. Each disjuntion (i.e., each paranthesized "or") is called a **clause**. Given a boolean formula f in CNF, we can consider the quantifiable problem of trying to satisfy as many of the disjunctions as possible. It is of course also NP-Hard, since satisfying as many clauses as possible will in particular satisfy the entire formula if possible. But we can also talk about *approximation algorithms* that provably satisfy *almost* as many clauses as possible, where "almost" is quantified precisely as a fraction of the maximum possible.

Given a CNF formula f, let OPT refer to the maximum number of that is satisfied. An assignment $x_1, \ldots, x_n \in \{\mathtt{t}, \mathtt{f}\}$ is an α-approximation if $f(x_1, \ldots, x_n)$ satisfies at least $\alpha \,\mathrm{OPT}$ clauses. Our goal is to design and analyze an algorithm that guarantees an α-approximate solution, for α as large as possible. It is NP-Hard to design an algorithm with $\alpha = 1$. However, for any value $\alpha < 1$, it is not necessarily NP-Hard to get an α-approximation.

We first consider the subclass of **3-SAT** formulas, which are boolean formulas in CNF that have exactly 3 variables per clause. (More generally, a ***k*-SAT formula** is a formula in CNF where each clause has exactly k clauses in it.) It is well known that 3-SAT is NP-Hard, but 2-SAT is actually polynomial time solvable.

We challenge the reader to try to think of an approximation algorithm for 3-SAT.

(Hint: you're probably overthinking it.)

1.1 An approximation algorithm for 3-SAT

Consider the following oblivious, randomized algorithm.

> *For each variable x_i, assign $x_i = \mathtt{t}$ with probability 1/2, or else $x_i = \mathtt{f}$ (with remaining probability 1/2).*

Theorem 1. *Given a k-SAT formula, The oblivious rounding algorithm described above satisfies $(1 - 2^{-k})$-fraction of clauses in expectation.*

Remarkably, the above algorithm is best possible for 3-SAT. This is an implication of an extremely efficient *probabilistically checkable proof* in a seminal result by Håstad [1]. Probabilistically checkable proofs are proofs where the verifier randomly samples only a few bits from the proof.

Theorem 2 (Håstad [1])**.** *For any fixed $\epsilon > 0$, it is NP-Hard to get a $(7/8 + \epsilon)$-approximation to 3-SAT.*

We plan to return to probabilistically checkable proofs later. For the moment, Håstad [1]'s result is just a glimpse in to the profound implications of randomization in complexity theory.

Obviously, this algorithm works not just for 3-SAT. The general approximation factor for k-SAT is $1 - 1/2^k$.

k	**oblivious APX**
1	1/2
2	3/4
3	7/8
4	15/16
5	31/32
⋮	
k	$1 - 1/2^k$

Of the above, we point out that the first row – $k = 1$ – is rather embarassing. The reader should pause to think of an optimal algorithm for $k = 1$.

Seemingly one can adjust the algorithm to handle 1-SAT formulas differently. But 1-SAT points to a more general problem - general SAT formulas in CNF, with varying variables in the number of clauses. Given a general SAT formula in CNF, one can still apply the oblivious algorithm of flipping a coin for every variable. If every clause has *at least* k variables, then we obtain a $1 - 1/2^k$ approximation ratio. But the presence of even some one variable clauses leaves us with an approximation ratio of only $1/2$.

1.2 A continuous approach

1. For $i = 1, \ldots, m$, each z_i indicates whether ($z_i = 1$) we satisfy the ith clauses or not ($z_i = 0$).
2. For $j = 1, \ldots, n$, each y_j indicates whether ($y_j = 1$) we set y_j to be true or not ($y_j = 0$).

$$\begin{aligned}
\text{maximize } & \sum_{i=1}^{m} z_i \text{ over } y_1, \ldots, y_n, z_1, \ldots, y_m \in \mathbb{R} \\
\text{s.t. } & \sum_{j : x_j \in C_i} y_j + \sum_{j : \bar{x}_j \in C_i} (1 - y_j) \geq z_i \text{ for all clauses } C_i \\
& 0 \leq z_i \leq 1 \text{ for all } i = 1, \ldots, m \\
& 0 \leq y_j \leq 1 \text{ for all } j = 1, \ldots, n
\end{aligned}$$

Lemma 3. *The probability that a clause C_i is satisfied is $\geq (1 - 1/e) z_i$.*

$$\begin{aligned}
\mathrm{P}[C_i \text{ not satisfied}] &= \prod_{j : x_j \in C_i} \mathrm{P}[x_j = \mathtt{f}] \prod_{j : \bar{x}_j \in C_i} \mathrm{P}[x_j = \mathtt{t}] = \prod_{j : x_j \in C_i} (1 - y_j) \prod_{j : \bar{x}_j \in C_i} y_j \\
&\overset{\text{(a)}}{\leq} e^{-\left(\sum_{j : x_j \in C_i} y_j + \sum_{j : \bar{x}_j \in C_i} (1 - y_j)\right)} \\
&\leq e^{-z_i} \overset{\text{(b)}}{\leq} (1 - z_i) \cdot e^0 + (z_i) \cdot e^{-1} \\
&= 1 - (1 - 1/e) x_i.
\end{aligned}$$

(a) is by the inequality $1 + x \leq e^x$. (b) is because e^x is convex. By linearity expectation, we expect to satisfy a $(1 - 1/e)$-fraction of clauses.

Lemma 4. *Randomized rounding the above LP gives a $(1 - 1/e)$-approximation to SAT.*

Note that this bound is only interesting when there are clauses with one or two variables; otherwise the oblivious rounding is still better.

1.3 The best of both worlds

Let's start with a simple observation about the LP rounding scheme.

Observation 5. *If $|C_i| = 1$, then C_i is satisfied with probability z_i.*

k	oblivious	LP	average
1	1/2	z_i	$\geq 3/4 z_i$
2	3/4	$(1-1/e)z_i$	$\geq .691 z_i$
3	7/8	$(1-1/e)z_i$	$\geq 3/4$
4	15/16	$(1-1/e)z_i$	$\geq 3/4$
5	31/32	$(1-1/e)z_i$	$\geq 3/4$
$\vdots$			
k	$1-1/2^k$	$(1-1/e)z_i$	$\geq (3/4)z_i$.

Lemma 6. *The hybrid algorithm obtains an approximation ratio of* $\frac{7-4/e}{8} \geq .691$.

Refined analysis for $k = 2$.

Observation 7. *If* $|C_i| = 1$*, then* C_i *is satisfied with probability* z_i.

Lemma 8. *If* $|C_i = 2|$*, then* C_i *is satisfied with probability* $\geq 3z_i/4$.

Proof. Suppose for simplicity that $C_i = x_1 \vee x_2$. (It will be obvious how to generalize the analysis to other pairs of variables.) We have

$$\begin{aligned}
\mathrm{P}[C_i \text{ not satisfied}] = (1-y_1)(1-y_2) &\overset{\text{(a)}}{\leq} \left(\frac{(1-y_1)+(1-y_2)}{2}\right)^2 \\
&\leq \left(1-\frac{z_j}{2}\right)^2 \overset{\text{(b)}}{\leq} (1-z_j)\left(1-\frac{0}{2}\right)^2 + z_j\left(1-\frac{1}{2}\right)^2 \\
&= 1 - z_j + \frac{z_j}{4} = 1 - \frac{3}{4}z_j.
\end{aligned}$$

(a) is by AM-GM. (b) is by convexity of $f(x) = \left(1-\frac{x}{2}\right)^2$. ■

k	oblivious	LP	average
1	1/2	z_i	$\geq (3/4)z_i$
2	3/4	$(3/4)z_i$	$\geq (3/4)z_i$
3	7/8	$(1-1/e)z_i$	$\geq (3/4)z_i$
4	15/16	$(1-1/e)z_i$	$\geq (3/4)z_i$
5	31/32	$(1-1/e)z_i$	$\geq (3/4)z_i$
$\vdots$			
k	$1-1/2^k$	$(1-1/e)z_i$	$\geq (3/4)z_i$.

Theorem 9. *The hybrid algorithm obtains an approximation ratio of* $\geq 3/4$.

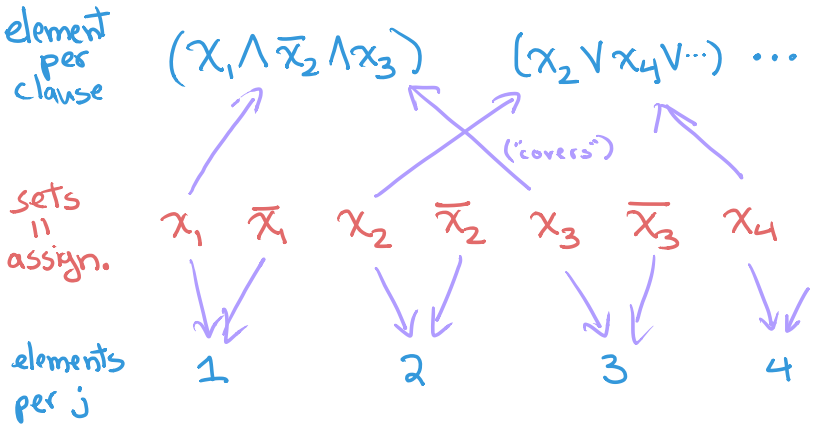
2 Set cover

In **set cover**, we are given m points $[m] = \{1, \ldots, m\}$, and n sets $S_1, \ldots, S_n \subseteq [m]$. The goal is to

find the minimum number of sets $S_{i_1}, \ldots, S_{i_k}$ *such that* $S_{i_1} \cup \cdots \cup S_{i_k} = [m]$.

$$f(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_2 \vee x_4 \vee x_5) \wedge \cdots .$$

(n variables, m clauses)



f satisfiable $\Leftrightarrow$ $\exists$ set cover of size n

Some natural extensions including adding costs for sets, pointwise-demands that require points to be covered be multiple sets, and generally coefficients $A_{ij} \in [0, 1]$ that say that set S_i gives A_{ij} fractional units of coverage to point j.

Set Cover was one of the original NP-Complete problems in Karp's seminal "Reducibility among combinatorial problems." A reduction from set cover to SAT is sketched out in Figure **??**.

$$\text{minimize } \sum_{j=1}^{n} x_j \text{ over } z \in \mathbb{R}^n_{\geq 0} \text{ s.t. } \sum_{S_j \ni i} x_j \geq 1 \text{ for all } i \in [m].$$

Until I finish, I recommend the handwritten notes at
https://randomized-algorithms-fall-2020.s3.amazonaws.com/Randomized+rounding+(Fall+2020)+handwritten+notes.pdf.

3 Exercises

Exercise 1. Extend the approximation algorithm for set cover to positive costs. For each set, there is a positive cost $c_j > 0$. The goal is to compute the minimum cost collection of sets that covers all the points.

Exercise 2. Consider an instance of (weighted) set cover defined by sets $S_1, \ldots, S_n \subseteq [m]$ and costs $c_i > 0$ for each set S_i. The goal is to compute the minimum cost collection of sets covering $[m]$. We say that solving the LP and then randomly rounding gives a $O(\log m)$ approximation. Here we consider a special case where all the sets are small and obtain a better approximation factor by a standard extension of randomized rounding called *alterations*.

Let $\Delta \in \mathbb{N}$ be such that $|S_j| \leq \Delta$ for all j. Conisder the algorithm `round-and-fix` for which some speudocode is given below. `round-and-fix` is similar to randomized rounding and has two stages. The first stage solves the LP and then *rounds* the solution scaled up by some factor $\alpha \geq 1$. It is possible that some of the elements $i \in [m]$ may not be covered. In the second stage, we *fix* each uncovered element by (deterministically) taking the cheapest set that covers it.

```
round-and-fix(sets S_1, ..., S_n ⊆ [m], costs c ∈ R^n_{>0}, α ≥ 1)

1. let x ∈ [0,1]^n solve the set cover LP

2. let F ⊆ {S_1, ..., S_n} sample each set S_i independently with probability min{1, αx_i}

3. for each i ∈ [m]

   A. if i is not covered by F

      1. add the cheapest set covering i to F

4. return F
```

Show that for an appropriate choice of α, this algorithm returns a $O(\log \Delta)$ approximation to the set cover instance (in expectation). (It is possible to get $\log \Delta + \log \log \Delta + O(1)$ with care.)

References

[1] Johan Håstad. "Some optimal inapproximability results". In: *J. ACM* 48.4 (2001), pp. 798–859. Preliminary version in STOC, 1997.