Algorithmic Methods 22/11/2010

Lecture Notes 6: Approximations for MAX-SAT

Professor: Yossi Azar *Scribe:Alon Ardenboim*

1 Introduction

Although solving SAT is known to be NP-Complete, in this lecture we will cover some algorithms that give an approximated solution of the weighted version that comes close to the maximal satisfaction of the clauses within a constant factor. Towards the end of the lecture we will start discussing about the integer multi-commodity max-flow problem and give a linear programming algorithm that approximates it.

2 Notes about Weighted Vertex Cover Approximation

On the last lecture, we gave a linear programming algorithm that approximates the weighted version of the VC problem.

2.1 Reminder: Weighted-VC

In the weighted version of the VC problem you are given a graph $G = (V, E)$ as input, and a weight function on the vertices $w : V \mapsto \mathbb{R}^+$. The goal is to find a subset $S \subseteq V$ s.t. $\forall (u, v) \in E$ $u \in S$ or $v \in S$ and the weight of the cover $\sum_{s \in S} w(s)$ minimized.

2.2 Reminder: Approximation via Linear Programming

To find an approximation to the optimal solution, we defined an LP problem - we defined a variable X_v for every $v \in V$, and our goal function was:

$$minimize \sum_{v \in V} w(v) X_v$$

subject to:

$$X_v + X_u \geq 1 \quad \forall (u, v) \in E$$

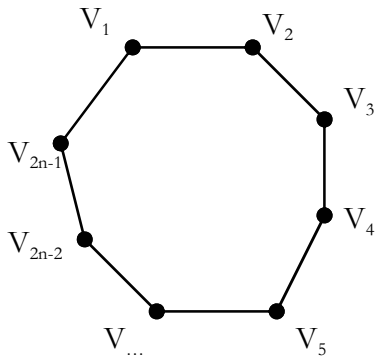
$$X_v \geq 0 \quad \forall v \in V$$

For each X_v we rounded the result; those who got a value $\geq \frac{1}{2}$ were rounded to 1, the others were rounded to 0. This way we got a 2-approximation of the problem. For more details, check the notes of Lecture 5.

2.3 Notes About the Approximation

1. **Possible Values**: After solving the linear program, we get that every X_v is assigned one of the values $\{0, \frac{1}{2}, 1\}$.

2. **Tight Bound**: The bound we get on approximating the weighted-VC using linear programming and rounding is tight. That is, it is possible to construct an example for which $ALG(G, w) \approx 2 \cdot OPT(G, w)$. Let's take a look at a graph $G = (V, E)$ which is a circle on $2n - 1$ vertices, and let's set $w(v) = 1$ for every $v \in V$. The optimal solution in for the LP problem would set $X_v = \frac{1}{2}$ for each $v \in V$, thus giving a total weight of $\frac{1}{2}(2n - 1) = n - \frac{1}{2}$, while the optimal solution for the problem would use n vertices for the cover, while giving us a total weight of n. After the rounding of the LP problem, we will set each $X_v = 1$, and will get a weight of $2n - 1$.



3 Approximating MAX-SAT

In the classic SAT problem we are given a conjunctive normal form (CNF) formula C that's comprised of a collection of clauses $\{C_1, ..., C_m\}$. Each clause C_j is a disjunction of literals $(X_{j_1} \vee X_{j_2} \vee \cdots \vee X_{j_k})$. Each one of the variables in the formula can be assigned a boolean value $(True/False)$, and our goal is to give an assignment to the variables that satisfies all the clauses. However, this problem is the most basic problem known to be NP-Complete. If we look at the 2-SAT problem where every clause contains at most 2 literals, we can check if it is satisfiable and find an assignment that satisfies it, if exists, in polynomial time. For any $k > 2$, k-SAT is known to be NPC. Let's look at the optimization problem.

3.1 MAX-SAT Definition

We are given a CNF formula C that's comprised of a collection of clauses $\{C_1, ..., C_m\}$ and a weight function $w : C \mapsto \mathbb{R}^+$ on the clauses, and our goal is to give an assignment to the variables $\sigma : Var \mapsto \{True, False\}$ so that the weight of satisfied clauses is maximal.
Let's denote the weight of the j^{th} clause as w_j and define an indicator function $\alpha : C \times \sigma \mapsto \{0, 1\}$ in which $\alpha(C_j, \sigma) = 1 \iff C_j$ is satisfied by assignment σ. The weight that

assignment σ gives to the formula is now defined:

$$w(C,\sigma) = \sum_{j=1}^{m} w_j \alpha(C_j, \sigma)$$

Although this problem seems easier than the classic SAT problem since we don't need to satisfy all the clauses, MAX-2-SAT is actually known to be NP-Hard. Let's try and give an approximation of the problem.

3.2 First Attempt - "Guessing" the Assignment

Let's assign independently each variable a boolean value in a uniform way. Each variable would be given $'True'$ w.p. $\frac{1}{2}$ and $'False'$ w.p. $\frac{1}{2}$. Let's see what's the expectation of the the weight of the clauses that would get satisfied by this assignment. From the linearity of expectation we get that:

$$E_\sigma\left[w\left(C,\sigma\right)\right] = \sum_j w_j E_\sigma\left[\alpha\left(c_j,\sigma\right)\right] = \sum_j w_j Pr\left[\alpha\left(c_j,\sigma\right) = 1\right]$$

since each clause isn't satisfied w.p. $\frac{1}{2^{l_j}}$:

$$= \sum_j w_j \cdot \left(1 - \frac{1}{2^{l_j}}\right)$$

since each clause has at least one literal:

$$\geq \frac{1}{2}\sum_j w_j \geq \frac{1}{2} OPT(C,w)$$

If we know that each clause contains at least k different literals, then a random assignment would give us an approximation factor of $1 - \frac{1}{2^k}$. So with a random assignment we're expected to get as close to the optimal solution as a factor $\frac{1}{2}$. Can we achieve this result deterministically?

3.3 De-randomization

We'll use the conditioned expectation method to give a deterministic algorithm that would achieve an approximation factor $\frac{1}{2}$. We know that:

$$E_{\sigma_n}[w(C,\sigma_n)] = \frac{1}{2}\cdot E_{\sigma_{n-1}}[w(C,\sigma_{n-1})|X_1 = False] + \frac{1}{2}\cdot E_{\sigma_{n-1}}[w(C,\sigma_{n-1})|X_1 = True]$$

From that we know that at least one assignment of X_1 wouldn't decrease the expectation of the total weight of the formula. If we could calculate the expectation of the formula when some of the variables are assigned a value and some are set randomly, we could choose the assignment of X_1 which gives us the higher expectation and continue assigning values to all the variables until we have an assignment that gives us an approximation factor of $\frac{1}{2}$. Fortunately, we can. All we need to do is to calculate $E[\alpha(c,\sigma)]$ for every $c \in C$. Let's

look at a clause $c \in C$ and let's assume that variables $X_1, ..., X_l$ were assigned a value, and variables $X_{l+1}, ..., X_n$ are set randomly. If one of c's literals satisfied c than it's clear that $E[\alpha(c, \sigma_{n-l})] = 1$. Otherwise, if c has k different literals that weren't assigned a value, $E[\alpha(c, \sigma_{n-l})] = 1 - \frac{1}{2^k}$.

Example: We are given the the following set of clauses:

$$\overline{X_1} \quad X_1 \vee \overline{X_2} \quad X_1 \vee X_2 \vee \overline{X_3} \quad \overline{X_1} \vee \overline{X_2} \vee \overline{X_3} \quad X_1 \vee X_2 \vee X_3$$

where each clause has a weight of 1. We're expected to get a weight of:

$$\frac{1}{2} + \frac{3}{4} + \frac{7}{8} + \frac{7}{8} + \frac{7}{8} = 3\frac{7}{8}$$

If we set $X_1 = False$ we get an expectation of:

$$1 + \frac{1}{2} + \frac{3}{4} + 1 + \frac{3}{4} = 4$$

If we set $X_1 = True$ we get an expectation of:

$$0 + 1 + 1 + \frac{3}{4} + 1 = 3\frac{3}{4}$$

Therefore, we'll set $X_1 = False$. Let's continue to setting the second variable. If we set $X_2 = False$ we get an expectation of:

$$1 + 1 + \frac{1}{2} + 1 + \frac{1}{2} = 4$$

If we set $X_2 = True$ we get an expectation of:

$$1 + 0 + 1 + 1 + 1 = 4$$

Notice that if we set $X_2 = True$ we decide which of the clauses are satisfied deterministically, and we get that the total weight is higher than the initial expectation.

3.4 Second Attempt - Guessing with a "Hunch"

Let's take a formula C and try to "merge" all the clauses with a single literal of the same variable. For instance, if we have four clauses:

$$C_1 = X_1 \quad C_2 = \overline{X_1} \quad C_3 = X_1 \quad C_4 = \overline{X_1}$$

with weights:

$$w_1 = 17 \quad w_2 = 12 \quad w_3 = 42 \quad w_4 = 16$$

we can at first merge C_1 and C_3 to a single clause $C^{X_1} = X_1$ with weight $w^{X_1} = 17+42 = 59$ and C_2 and C_4 to a single clause $C^{\overline{X_1}} = \overline{X_1}$ with weight $w^{\overline{X_1}} = 12 + 16 = 28$. We can now again combine the two clauses into a single clause $\widetilde{C}^{X_1} = X_1$ with weight $\widetilde{w}^{X_1} = 59-28 = 31$. Note that every assignment of X_1 would give us a weight smaller by 28. Notice that although we reduce the weight of the formula, every approximation factor we get with the new formula

would also apply to the old formula. Let's denote the new formula after merging all the variables as $\widetilde{C}$, set the weight shift constant as $\beta > 0$ and the approximation factor of $\widetilde{C}$ as α, we can see that:

$$\frac{ALG_w(C)}{OPT_w(C)} = \frac{ALG_w(\widetilde{C}) + \beta}{OPT_w(\widetilde{C}) + \beta} \geq \frac{ALG_w(\widetilde{C})}{OPT_w(\widetilde{C})} \geq \alpha$$

Our approximation algorithm would first transform C into a reduced formula $\widetilde{C}$ where each variable can appear only once in a clause with a single literal. Afterwards, the algorithm will guess the assignment for each variable independently, but with a "hunch". Every literal in $\widetilde{C}$ that appears by himself in a clause would be set to $True$ w.p. $h > \frac{1}{2}$, and therefore, the complement to this literal would be set to $True$ w.p. $1 - h$. Every variable that doesn't have a literal of his in a clause of size 1 would be assigned a value randomly as before. Let's calculate the expectation of the indicator function α for each clause with the new algorithm. For clauses with a single literal:

$$E[\alpha(C_j, \sigma)] = h$$

and for clauses with $k \geq 2$ different literals:

$$E[\alpha(C_j, \sigma)] \geq 1 - h^k \geq 1 - h^2$$

Therefore:

$$E[\alpha(C_j, \sigma)] \geq min\{h, 1 - h^2\}$$

Because h is monotonically increasing and $1 - h^2$ is monotonically decreasing, we get that that the expectation reaches a maximum when $h = 1 - h^2 \Rightarrow h = \frac{\sqrt{5}-1}{2} \approx 0.618$.
A de-randomization using the conditioned expectation method could be applied as before to give us a deterministic algorithm with an approximation factor of ≈ 0.618.

3.5 Approximation using Linear Programming

Idealistically, we would like to define the following integer LP problem and find the optimum value - let's define an indicator $X_i \in \{0, 1\}$ for each variable $X_i (1 \leq i \leq m)$ and an indicator $Z_j \in \{0, 1\}$ for each clause $C_j (1 \leq j \leq n)$, and set the goal function to be:

$$max \sum_j w_j \cdot Z_j$$

subject to:

$$Z_j \leq \sum_{i \in C_j^+} X_i + \sum_{i \in C_j^-} (1 - X_i)$$

where C_j^+ are the variables that appear in C_j without negation, and C_j^- are the variables that appear in C_j with negation. If we could solve such a problem in polynomial time, we could do all sorts of cool stuff like solve 3-SAT and show that P=NP. Since ILP is an NP-Complete problem, we'll solve a relaxation of the problem. The goal function is the same and the constrains are the same, but the possible values of the variables are:

$$0 \leq X_i \leq 1 \quad 0 \leq Z_j \leq 1$$

After solving the LP problem we get $X_i = P_i$ for every variable. We'll do a probabilistic rounding of the result. For each X_i, independently, we'll set $X_i = True$ w.p. P_i and $X_i = False$ w.p. $1 - P_i$. The probability that $\alpha(C_j) = 0$ for a certain C_j is:

$$Pr[\alpha(C_j) = 0] = \prod_{i \in C_j^+} (1 - P_i) \prod_{i \in C_j^-} P_i$$

from the inequality of arithmetic and geometric means:

$$\leq \left(\frac{\sum_{i \in C_j^+} (1 - P_i) + \sum_{i \in C_j^-} P_i}{k} \right)^k$$

from the constraint on Z_j:

$$\leq \left(\frac{k - Z_j}{k} \right)^k = \left(1 - \frac{Z_j}{k} \right)^k$$

From that we get that:

$$Pr[\alpha(C_j) = 1] \geq 1 - \left(1 - \frac{Z_j}{k} \right)^k$$

If we can show that for some constant $\beta_k \geq \beta$:

$$1 - \left(1 - \frac{Z_j}{k} \right)^k \geq \beta_k \cdot Z_j \geq \beta \cdot Z_j$$

then we get a β-approximation of MAX-SAT since:

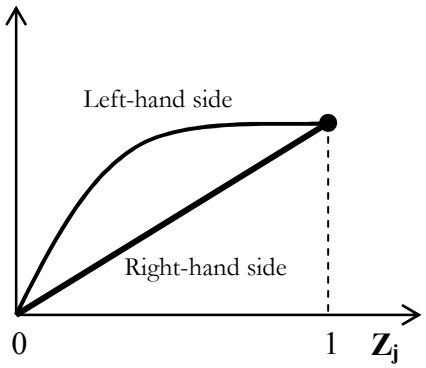
$$E[w(C, \sigma)] = \sum_j w_j \cdot Pr[\alpha(C_j) = 1] \geq \sum_j w_j \cdot Z_j \cdot \beta = \beta \cdot \sum_j w_j \cdot Z_j \geq \beta \cdot OPT_{relaxed}(C, \sigma) \geq \beta \cdot OPT(C, \sigma)$$

Claim: The inequality $1 - \left(1 - \frac{Z_j}{k} \right)^k \geq \left(1 - \left(1 - \frac{1}{k} \right)^k \right) \cdot Z_j$ holds for $0 \leq Z_j \leq 1$.

Proof. When $Z_j = 0$ we get that $0 \geq 0$. When $Z_j = 1$ we get:

$$1 - \left(1 - \frac{1}{k} \right)^k \geq \left(1 - \left(1 - \frac{1}{k} \right)^k \right) \cdot 1$$

It's easy to see that the second derivative of the left hand side of the inequality is negative when $0 \leq Z_j \leq 1$ and therefore it is concave there, and because the right hand side of the inequality is linear in Z_j, we get the following:



From that, it is easy to see that the inequality holds when $0 \leq Z_j \leq 1$. □

Since $\beta_k = 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e} \approx 0.63$, we get an approximation factor of $\beta \approx 0.63$ using linear programming and rounding for the MAX-SAT problem.

3.6 Combining Linear Programming and Random Assignment

Note that random assignment gives us a good approximation for clauses containing many literals, while linear programming gives us a good approximation for clauses containing few literals. We'll try combining them both - with probability $\frac{1}{2}$ we'll give all the variables the value they get using LP and rounding, and with probability $\frac{1}{2}$ we'll give all the variables a random assignment. Let's try and calculate the expected weight that the assignment gives on the formula (note that $k = k_j$):

$$\begin{aligned} E[w(C, \sigma_{combined})] &= \frac{1}{2}E[w(C, \sigma_{LP})] + \frac{1}{2}E[w(C, \sigma_R)] \\ = \frac{1}{2}\sum_j w_j \cdot Z_j \cdot \beta_{k_j} + \frac{1}{2}\sum_j w_j \cdot \left(1 - \frac{1}{2^{k_j}}\right) &\geq \sum_j w_j \cdot Z_j \left(\frac{1}{2}\beta_{k_j} + \frac{1}{2}\left(1 - \frac{1}{2^{k_j}}\right)\right) \end{aligned}$$

If we'll show that:

$$\frac{1}{2}\beta_k + \frac{1}{2}\left(1 - \frac{1}{2^k}\right) \geq \frac{3}{4}$$

we'll get a $\frac{3}{4}$ approximation since:

$$E[w(C, \sigma_{combined})] \geq \sum_j w_j \cdot Z_j \left(\frac{1}{2}\beta_{k_j} + \frac{1}{2}\left(1 - \frac{1}{2^{k_j}}\right)\right) \geq \frac{3}{4}\sum_j w_j \cdot Z_j \geq \frac{3}{4}OPT(C, \sigma)$$

Claim: The following inequality holds:

$$\frac{1}{2}\left(1 - \left(1 - \frac{1}{k}\right)^k\right) + \frac{1}{2}\left(1 - \frac{1}{2^k}\right) \geq \frac{3}{4}$$

Proof. Let's first rearrange the equation:

$$\left(1-\frac{1}{k}\right)^k+\frac{1}{2^k}\leq\frac{1}{2}$$

For $k=1$ we get:

$$0+\frac{1}{2}\leq\frac{1}{2}$$

For $k=2$ we get:

$$\frac{1}{4}+\frac{1}{4}\leq\frac{1}{2}$$

For $k\geq 3$ it's clear that $\left(1-\frac{1}{k}\right)^k\leq\frac{1}{e}$ and $\frac{1}{2^k}\leq\frac{1}{8}$, and that gives us:

$$\left(1-\frac{1}{k}\right)^k+\frac{1}{2^k}\leq\frac{1}{e}+\frac{1}{8}\leq\frac{1}{2}$$

□

Great! We got a $\frac{3}{4}$ approximation when selecting the assignment of the LP approximation w.p. $\frac{1}{2}$ and the random assignment w.p. $\frac{1}{2}$. Can we improve the approximation factor if we choose which assignment to give to each variable w.p. $\neq\frac{1}{2}$? Apparently, $\frac{1}{2}\backslash\frac{1}{2}$ gives us the best approximation factor.
If we cannot improve the approximation factor, can we at least reach the approximation deterministically? This time the answer is positive. De-randomization, although a little more complicated than the previous ones, works here as well.

4 Integer Routing via Multi-Commodity Flow

Consider the following problem - you are given a graph $G=(V,E)$ with a capacity function on the edges $c:E\mapsto\mathbb{R}^+$ and a request set (s_i,t_i). For each request we get a value v_i if we find a path with capacity 1 from s_i to t_i. A feasible solution to this problem would be to find a path P_i for each request or set $P_i=\emptyset$ if we choose not to serve the request, and the value for this solution would be $\sum_{i|P_i\neq\emptyset}v_i$. Our goal is obviously to maximize this value. Let's denote $l_e=|\{i|e\in P_i\}|$. We want to ensure that the feasible solution would maintain $l_e\leq c_e$ for every $e\in E$.

4.1 Linear Programming Approximation - Phase 1

We'll make some relaxing assumptions:

1. We can replace the path from s_i to t_i with a flow from s_i to t_i.
2. We can serve a fraction of the request, and we'll get a fraction of the appropriate v_i.

Let's define some new variables. X_e^i will denote the flow for request i on an edge e and X^i would denote the total amount of flow for request i.
The goal function for the LP problem is:

$$maximize\sum_i v_iX_i$$

subject to:

$$\sum_i X_e^i \leq c_e \quad \forall e \in E \quad \text{(Make sure each edge isn't exceeding its capacity)}$$

$$\sum_{e|e=(u,v)} X_e^i - \sum_{e|e=(v,u)} X_e^i = 0 \quad \forall i \quad \forall u \in V \setminus \{s_i, t_i\} \quad \text{(Incoming flow equals to outgoing flow)}$$

$$\sum_{e|e=(s_i,v)} X_e^i - \sum_{e|e=(v,s_i)} X_e^i = X_i \quad \forall i \quad \text{(What leaves the source minus what enters it)}$$

As well as $0 \leq X^i \leq 1$ and $0 \leq X_e^i \leq 1$.
After we get a flow for each request, we'll disassemble the flow from s_i to t_i into at most $|E|$ different paths. The way of doing it is to take the flow graph for each request, which is a directed a-cyclic graph (DAG), and start removing paths from it, one by one. For every path we remove, we deduce it's weight, which is the weight of the lightest edge in the path, from each edge in the path, and continue on to the next path. Since every path removal removes at least one edge, we can re-iterate at most $|E|$ times. We'll continue on to phase 2 of the approximation algorithm at the beginning of lesson 7.