CS 561, Approximation Algorithms

Jared Saia
University of New Mexico
Outline

- SET-COVER
- MAX-SAT
SET-COVER

- Given a universe of elements $U = \{1, \ldots, m\}$, and a family of subsets of $U$ called $\mathcal{S}$
- For every $S \in \mathcal{S}$, there is a weight $w_S$
- Goal: Find a cover $C \subseteq \mathcal{S}$ of minimum weight $\sum_{S \in C} w_S$.
- A set $C$ is a cover, if for all $i \in U$, there is a set $S \in C$ such that $i \in S$. 
SET-COVER

- SET-COVER is NP-HARD (to show, reduce from VERTEX-COVER)
- Want to solve this problem frequently in e.g. computational biology
- There is an interesting approximation algorithm for it though
- IDEA: Solve an LP; Use the setting in the solution to assign probabilities to indicator rv’s; Round these rv’s
The Integer Program (IP)

Minimize: \( \sum_{S \in S} w_S x_S \)

Subject to:

\( \sum_{S: i \in S} x_S \geq 1, \forall i \in U \)

\( x_S \in \{0, 1\}, \forall S \in S \)
Minimize: $\sum_{S \in \mathcal{S}} w_S x_S$

Subject to:

$\sum_{S: i \in S} x_S \geq 1, \forall i \in U$

$0 \leq x_S \leq 1, \forall S \in \mathcal{S}$
Analysis

- IDEA: Solve this LP in polynomial time
- PROBLEM: It gives us $x_S \in [0, 1]$ for all $S$. How do we decide whether to choose each set?
- IDEA: Choose set $S$ with **probability** $x_S$
Example

- $U = \{a, b, c\}$
- $S_1 = \{a, b\}; \ S_2 = \{a, c\}; \ S_3 = \{b, c\}$
- $w_S = 1$ for all sets $S$
Example

- $U = \{a, b, c\}$
- $S_1 = \{a, b\}; S_2 = \{a, c\}; S_3 = \{b, c\}$
- $w_S = 1$ for all sets $S$

- LP Solution: $x_1^* = x_2^* = x_3^* = 1/2$
- Let $R$ be the sets in the rounding
- Example Rounding: $R = \{S_1, S_2\}$
- Success! This gives a cover with optimal weight
Fact 1: Expected weight of $R$ is no more than expected weight of OPT.

- Proof: For each possible set $S$, let $X_S$ be an indicator r.v. that is 1 iff $S \in R$. Then we have

$$E \left( \sum_{S \in R} w_S \right) = E \left( \sum_S w_S X_S \right) = \sum_S w_S E(X_S) = \sum_S w_S x^*_S$$

- The last term is the weight of the LP solution which is at most the weight of the optimal solution.
Fact 2: Every element $i \in U$ is covered by $R$ with probability at least $1 - 1/e$

• Proof: Fix an element $i \in U$. Let $T$ be the sets in $S$ that contain $i$. Then

$$Pr(i \text{ is not covered by } R) = \prod_{S \in T} Pr(S \notin R)$$
$$= \prod_{S \in T} (1 - x^*_S)$$
$$\leq \prod_{S \in T} e^{-x^*_S}$$
$$= e^{-\sum_{S \in T} x^*_S}$$
$$\leq e^{-1}.$$
Problem: May not always get a cover

- Problem: Each item covered with probability $1 - \frac{1}{e}$, but likely that some item not covered.
- Idea: Round multiple times to get a cover with high probability.
- Increases the weight, but only by a logarithmic amount
Algorithm 1

1. Let $x^*$ be a solution to the relaxed LP
2. For $t = 1$ to $2 \ln m$ do
   (a) Add each set $S$ to $R_t$ with probability $x^*_S$ independently
3. Return $\bigcup_t R_t$
Analysis

**Theorem 1**: In one run with probability $1/4$, Algorithm 1 (1) returns a cover, (2) with total weight at most $4 \ln m \cdot OPT$.

Proof: (1) For a fixed $i$, By Fact 1 and independence, we have

$$Pr(i \text{ not covered}) \leq e^{-2 \ln m} = m^{-2}$$

Thus, by a union bound:

$$Pr(\text{any of the } m \text{ elements uncovered}) \leq m^{-1} \leq \frac{1}{4}.$$ 

(2) By Fact 2, expected weight of sets added in one iteration of the for loop is at most $OPT$. By linearity, expected weight over $2 \ln m$ iterations is at most $2 \ln m \cdot OPT$. Let $W$ be the weight of the sets returned by the algorithm. By Markov’s inequality, $Pr(W \geq 2E(W)) \leq 1/2$.

By a final union bound, with probability at least $1/4$ we have a cover with the weight at most $4 \ln m \cdot OPT$. 
In 4 expected runs, Algorithm 1 will return a $4 \ln m$ approximation to the optimal weight set-cover.

Thus it is a $O(\log m)$-approximation algorithm!

It critically relies on a solution to the LP to guide the randomized part of the algorithm.

Next, we’ll see another example of this approach for the MAX-SAT problem.
Imagine that we have some CNF boolean function, where each clause has exactly $k$ literals for some integer $k$.

Each clause $C_j$ has a set of positive variables $P_j$ and a set of negative variables $N_j$.

Our goal is to set truth values to the variables in order to maximize the number of satisfied clauses.

IDEA: Solve an LP; Use the settings in this solution to assign probabilities to indicator r.v.’s; Round these r.v.’s.
Maximize: $\sum_j z_j$

Subject to:

$z_j \leq \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i), \forall C_j$

$0 \leq y_i \leq 1, \forall y_i$

$0 \leq z_j \leq 1, \forall z_j$
The Algorithm

- Write an LP for the boolean formula as in the previous slide
- Let $y_i^*$ be the settings found in the solution found for the LP
- For each variable $i$, set $i$ to TRUE with probability $y_i^*$ and FALSE otherwise
Analysis Background

- Convex/Concave Functions
- Arithmetic/Geometric Mean inequality
• A function, \( f \), is **convex** if for all inputs \( x \) and \( y \) and for all \( \lambda \in [0, 1] \):

\[
f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)
\]

• Key fact: If \( f \) has a second derivative, then \( f \) is convex iff the second derivative is always non-negative.
Concave Functions

- A concave function is the negative of a convex function
- A function, $f$, is **concave** if for all inputs $x$ and $y$ and for all $\lambda \in [0, 1]$:  
  \[ f(\lambda x + (1 - \lambda)y) \geq \lambda f(x) + (1 - \lambda)f(y) \]
- Key fact: If $f$ has a second derivative, then $f$ is concave iff the second derivative is always negative.
For any non-negative $x_1, x_2, \ldots, x_k$, the geometric mean is at most equal to the arithmetic mean

- $(x_1 x_2 \ldots x_k)^{1/k} \leq (1/k)(x_1 + x_2 + \ldots + x_k)$
- Easy to see this for 2 variables: $\sqrt{xy} \leq (1/2)(x + y)$
Probability $C_j$ is not satisfied

- Fix some clause $C_j$ and let $P_j$ be the set of positive and $N_j$ be the set of negative variables in $C_j$
- Then the probability that the clause is not satisfied is

$$\prod_{i \in P_j} (1 - y_i^*) \prod_{i \in N_j} y_i^* \leq \left( \frac{1}{k} \left( \sum_{i \in P_j} (1 - y_i^*) + \sum_{i \in N_j} y_i^* \right) \right)^k$$

$$= \left( 1 - \frac{1}{k} \left( \sum_{i \in P_j} y_i^* + \sum_{i \in N_j} (1 - y_i^*) \right) \right)^k$$

$$\leq \left( 1 - \frac{z_j^*}{k} \right)^k$$

First inequality holds since $GM \leq AM$. 

Using Concavity

- Probability that $C_j$ is satisfied is: $1 - \left(1 - \frac{z_j^*}{k}\right)^k$
- $f(z_j^*) = 1 - \left(1 - \frac{z_j^*}{k}\right)^k$ is concave over $z_j^* \in [0, 1]$
- Hence: For any $x$ and $y$ and all $\lambda \in [0, 1]$: 
  \[ f(\lambda x + (1 - \lambda)y) \geq tf(x) + (1 - \lambda)f(y) \]
- Specifically if $x = 0$ and $y = 1$, then 
  \[ f((1 - \lambda)) \geq (1 - \lambda)f(1) \]
- Setting $1 - \lambda$ to be $z_j^*$, we get that 
  \[ f(z_j^*) \geq z_j^* \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \]
Bounding with Concave Property

\[ f(z_j^*) = 1 - \left(1 - \frac{z_j^*}{k}\right)^k \]
Using Linearity of Expectation

- Probability that \( C_j \) is satisfied is \( \geq z_j^* \left( 1 - \left(1 - \frac{1}{k}\right)^k \right) \)
- Let \( W \) be the number of clauses satisfied by our algorithm, and let \( W_j \) be an indicator r.v. that is 1 iff \( C_j \) is satisfied.

\[
E(W) = \sum_j E(W_j) \\
\geq \sum_j z_j^* \left( 1 - \left(1 - \frac{1}{k}\right)^k \right) \\
\geq \min_k \left( 1 - \left(1 - \frac{1}{k}\right)^k \right) \sum_j z_j^* \\
\geq \min_k \left( 1 - \left(1 - \frac{1}{k}\right)^k \right) OPT \\
\geq (1 - 1/e)OPT \\
\geq .632 \cdot OPT
\]
• For step 5 of last slide, note that since $1 - x \leq e^{-x}$:

$$(1 - 1/k)^k \leq e^{-1}$$

• So for any value of $k$,

$$1 - (1 - 1/k)^k \geq 1 - 1/e$$

• Just FYI, it’s also true that

$$\lim_{k \to \infty} (1 - 1/k)^k = e^{-1}$$

Since

$$\lim_{k \to \infty} (1 + 1/k)^k = e$$
Take Away

• Many real-world problems can be shown to not have an efficient solution unless $P = NP$ (these are the NP-Hard problems)
• However, if a problem is shown to be NP-Hard, all hope is not lost!
• In many cases, we can come up with an provably good approximation algorithm for the NP-Hard problem.