Advanced Algorithms (I)

Chihao Zhang

Shanghai Jiao Tong University

Feb. 25, 2019

Advanced Algorithms

Advanced Algorithms

In the course, we will learn Approximation Algorithms

Advanced Algorithms

In the course, we will learn Approximation Algorithms

- linear programming, semi-definite programming
- spectral method
- random walks
- ...

Advanced Algorithms

In the course, we will learn Approximation Algorithms

- linear programming, semi-definite programming
- spectral method
- random walks
- ...

We will emphasize on

- tools for designing approximation algorithms
- rigorous analysis of algorithms

Course Info

Course Info

- Instructor: Chihao Zhang
- Course Homepage: `http://chihaozhang.com/teaching/AA2019spring/`
- Office Hour: every Monday, 7:00pm - 9:00pm

Course Info

- Instructor: Chihao Zhang
- Course Homepage:
  `http://chihaozhang.com/teaching/AA2019spring/`
- Office Hour: every Monday, 7:00pm - 9:00pm

Grading Policy

- Homework 30%
- Mid-term Exam 30%
- Course Project 40%

MaxSAT

MaxSAT

Given a CNF formula ϕ, is it satisfiable?

MaxSAT

Given a CNF formula ϕ, is it satisfiable?

$$\phi = (x_1 \vee x_3 \vee \bar{x}_{29}) \wedge (\bar{x}_3 \vee x_7) \wedge \cdots \wedge (\bar{x}_{33} \vee \bar{x}_{34} \vee x_{90} \vee x_{126})$$

MaxSAT

Given a CNF formula ϕ, is it satisfiable?

$$\phi = (x_1 \vee x_3 \vee \bar{x}_{29}) \wedge (\bar{x}_3 \vee x_7) \wedge \cdots \wedge (\bar{x}_{33} \vee \bar{x}_{34} \vee x_{90} \vee x_{126})$$

NP-hard, we look at its optimization version.

MaxSAT

Given a CNF formula ϕ, is it satisfiable?

$$\phi = (x_1 \vee x_3 \vee \bar{x}_{29}) \wedge (\bar{x}_3 \vee x_7) \wedge \cdots \wedge (\bar{x}_{33} \vee \bar{x}_{34} \vee x_{90} \vee x_{126})$$

NP-hard, we look at its optimization version.

> MaxSAT
>
> *Input:* A CNF formula $\phi = C_1 \wedge C_2 \cdots \wedge C_m$.
>
> *Problem:* Compute an assignment that satisfies maximum number of clauses.

MaxSAT

Given a CNF formula ϕ, is it satisfiable?

$$\phi = (x_1 \vee x_3 \vee \bar{x}_{29}) \wedge (\bar{x}_3 \vee x_7) \wedge \cdots \wedge (\bar{x}_{33} \vee \bar{x}_{34} \vee x_{90} \vee x_{126})$$

NP-hard, we look at its optimization version.

MaxSAT	
Input:	A CNF formula $\phi = C_1 \wedge C_2 \cdots \wedge C_m$.
Problem:	Compute an assignment that satisfies maximum number of clauses.

Harder than SAT, so we look for an approximate solution.

Tossing a Coin

Tossing a Coin

An instance ϕ

- The variable sets $V = \{x_1, x_2, \ldots, x_n\}$
- The set of clauses $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$
- Each clause C_i contains ℓ_i literals

Tossing a Coin

An instance ϕ

- The variable sets $V = \{x_1, x_2, \ldots, x_n\}$
- The set of clauses $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$
- Each clause C_i contains ℓ_i literals

We first cosnider the following simple algorithm:

Tossing a Coin

An instance ϕ

- The variable sets $V = \{x_1, x_2, \ldots, x_n\}$
- The set of clauses $C = \{C_1, C_2, \ldots, C_m\}$
- Each clause C_i contains ℓ_i literals

We first cosnider the following simple algorithm:

- For each variable x_i, toss an independent fair coin.

Tossing a Coin

An instance ϕ

- The variable sets $V = \{x_1, x_2, \ldots, x_n\}$
- The set of clauses $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$
- Each clause C_i contains ℓ_i literals

We first cosnider the following simple algorithm:

- For each variable x_i, toss an independent fair coin.
- If the coin goes `HEAD`, we set x_i **true**, otherwise we set x_i **false**.

Analysis

Analysis

The outcome of the algorithm is random, we are interested in its expectation.

Analysis

The outcome of the algorithm is random, we are interested in its expectation.

For this particular algorithm, it can be derandomized.

Analysis

The outcome of the algorithm is random, we are interested in its expectation.

For this particular algorithm, it can be derandomized.

$$\mathbf{E}[X] = \sum_{i=1}^{m} \mathbf{Pr}[C_i \text{ is satisfied}]$$

Analysis

The outcome of the algorithm is random, we are interested in its expectation.

For this particular algorithm, it can be derandomized.

$$\mathbf{E}[X] = \sum_{i=1}^{m} \mathbf{Pr}[C_i \text{ is satisfied}] = \sum_{i=1}^{m} \left(1 - 2^{-\ell_i}\right) \geq \frac{m}{2}.$$

Analysis

The outcome of the algorithm is random, we are interested in its expectation.

For this particular algorithm, it can be derandomized.

$$\mathbf{E}[X] = \sum_{i=1}^{m} \mathbf{Pr}[C_i \text{ is satisfied}] = \sum_{i=1}^{m} \left(1 - 2^{-\ell_i}\right) \geq \frac{m}{2}.$$

On the otherhand,

$$\mathbf{OPT} \leq m.$$

Analysis

The outcome of the algorithm is random, we are interested in its expectation.

For this particular algorithm, it can be derandomized.

$$\mathbf{E}[X] = \sum_{i=1}^{m} \mathbf{Pr}[C_i \text{ is satisfied}] = \sum_{i=1}^{m} \left(1 - 2^{-\ell_i}\right) \geq \frac{m}{2}.$$

On the otherhand,

$$\mathbf{OPT} \leq m.$$

Therefore,

$$\mathbf{E}[X] \geq \frac{1}{2} \cdot \mathbf{OPT}.$$

Can we improve the previous algorithm?

Can we improve the previous algorithm?

Observations

- the worst case happens when for some singleton clause, i.e., $\ell_i = 1$;

Can we improve the previous algorithm?

Observations

- the worst case happens when for some singleton clause, i.e., $\ell_i = 1$;
- for a singleton $C = x$, if there is no $C' = \bar{x}$, then we can increase the probability of x to be true;

Can we improve the previous algorithm?

Observations

- the worst case happens when for some singleton clause, i.e., $\ell_i = 1$;
- for a singleton $C = x$, if there is no $C' = \bar{x}$, then we can increase the probability of x to be true;
- otherwise, we can improve the upper bound for **OPT**! (x and $\bar{x}$ cannot be both satisfied)

Tossing a Biased Coin

Tossing a Biased Coin

Let the p-biased coin be with probability p to `HEAD`, $1 - p$ to `TAIL`.

Tossing a Biased Coin

Let the p-biased coin be with probability p to `HEAD`, $1 - p$ to `TAIL`.

Tossing a Biased Coin

Let the p-biased coin be with probability p to `HEAD`, $1 - p$ to `TAIL`.

- For each variable x_i, toss an independent p-biased coin.

Tossing a Biased Coin

Let the p-biased coin be with probability p to `HEAD`, $1-p$ to `TAIL`.

- For each variable x_i, toss an independent p-biased coin.
- If the coin goes `HEAD`, we set x_i **true**, otherwise we set x_i **false**.

Analysis

Analysis

Assumption for Clauses of ϕ

- More positive singletons than negative singletons.

Analysis

Assumption for Clauses of ϕ

- More positive singletons than negative singletons.
- There are t pairs of x and $\bar{x}$.

Analysis

Assumption for Clauses of ϕ

- More positive singletons than negative singletons.
- There are t pairs of x and $\bar{x}$.
- $p \geq \frac{1}{2}$.

$$\mathbf{E}\left[X\right] \geq t + (m - 2t) \min\left\{p, 1 - p^2\right\}.$$

Analysis

Assumption for Clauses of ϕ

- More positive singletons than negative singletons.
- There are t pairs of x and $\bar{x}$.
- $p \geq \frac{1}{2}$.

$$\mathbf{E}[X] \geq t + (m - 2t) \min\left\{p, 1 - p^2\right\}.$$

The new upper bound for **OPT**:

$$\mathbf{OPT} \leq m - t$$

Analysis

Assumption for Clauses of ϕ

- More positive singletons than negative singletons.
- There are t pairs of x and $\bar{x}$.
- $p \geq \frac{1}{2}$.

$$\mathbf{E}[X] \geq t + (m - 2t)\min\left\{p, 1 - p^2\right\}.$$

The new upper bound for **OPT**:

$$\mathbf{OPT} \leq m - t$$

Combine them and obtain

$$\mathbf{E}[X] \geq \alpha \cdot \mathbf{OPT}$$

where $\alpha \approx 0.618$.

Can we further improve the algorithm?

Can we further improve the algorithm?

We can use different coins for each variable, e.g., in

$$\phi = x_1 \wedge (x_1 \vee \bar{x}_2),$$

we prefer to set x_1 to **true**.

Can we further improve the algorithm?

We can use different coins for each variable, e.g., in

$$\phi = x_1 \wedge (x_1 \vee \bar{x}_2),$$

we prefer to set x_1 to **true**.

How can we make use of the information?

Can we further improve the algorithm?

We can use different coins for each variable, e.g., in

$$\phi = x_1 \wedge (x_1 \vee \bar{x}_2),$$

we prefer to set x_1 to **true**.

How can we make use of the information?

Linear Programming helps.

Tossing Clever Coins

Tossing Clever Coins

We introduce the following variables.

Tossing Clever Coins

We introduce the following variables.

- for every $i \in [n]$, y_i indicates whether x_i is true;
- for every $j \in [m]$, z_j indicates whether C_j is satisfied.

Tossing Clever Coins

We introduce the following variables.

- for every $i \in [n]$, y_i indicates whether x_i is true;
- for every $j \in [m]$, z_j indicates whether C_j is satisfied.

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{m} z_j \\
\text{subject to} \quad & \sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j, \quad \forall j \in [m] \text{ s.t. } C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{k \in N_j} \bar{x}_k \\
& z_j \in \{0, 1\}, \quad \forall j \in [m] \\
& y_i \in \{0, 1\}, \quad \forall i \in [n]
\end{aligned}
$$

Tossing Clever Coins

We introduce the following variables.

- for every $i \in [n]$, y_i indicates whether x_i is true;
- for every $j \in [m]$, z_j indicates whether C_j is satisfied.

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{m} z_j \\
\text{subject to} \quad & \sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k) \geq z_j, \quad \forall j \in [m] \text{ s.t. } C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{k \in N_j} \bar{x}_k \\
& z_j \in \{0, 1\}, \quad \forall j \in [m] \\
& y_i \in \{0, 1\}, \quad \forall i \in [n]
\end{aligned}
$$

This integer program is equivalent to MaxSAT.

Relaxation

Relaxation

There is no efficient algorithm for integer programming in general

Relaxation

There is no efficient algorithm for integer programming in general

Therefore, we relax its non-linear constriants

Relaxation

There is no efficient algorithm for integer programming in general

Therefore, we relax its non-linear constriants

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{m} z_j \\
\text{subject to} \quad & \sum_{i \in P_j} y_i + \sum_{k \in P_j} (1 - y_k) \ge z_j, \quad \forall j \in [m] \text{ s.t. } C_j = \bigvee_{i \in P_j} x_i \vee \bigvee_{k \in N_j} \bar{x}_k \\
& 0 \le z_j \le 1, \quad \forall j \in [m] \\
& 0 \le y_i \le 1, \quad \forall i \in [n]
\end{aligned}
$$

Relaxation

There is no efficient algorithm for integer programming in general

Therefore, we relax its non-linear constraints

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} z_j \\ \text{subject to} \quad & \sum_{i\in P_j} y_i + \sum_{k\in P_j}(1-y_k) \geq z_j, \quad \forall j \in [m] \text{ s.t. } C_j = \bigvee_{i\in P_j} x_i \vee \bigvee_{k\in N_j} \bar{x}_k \\ & 0 \leq z_j \leq 1, \quad \forall j \in [m] \\ & 0 \leq y_i \leq 1, \quad \forall i \in [n] \end{aligned}$$

We can solve this LP in poly-time

Algorithm

Algorithm

Let $\left(\left\{y_i^*\right\}_{i\in[n]}, \left\{z_j^*\right\}_{j\in[m]}\right)$ be an optimal solution of the LP.

Algorithm

Let $\left(\left\{y_i^*\right\}_{i\in[n]}, \left\{z_j^*\right\}_{j\in[m]}\right)$ be an optimal solution of the LP.

Algorithm

Let $\left(\left\{y_i^*\right\}_{i\in[n]}, \left\{z_j^*\right\}_{j\in[m]}\right)$ be an optimal solution of the LP.

- For each variable x_i, toss an independent y_i^*-biased coin.

Algorithm

Let $\left(\left\{ y_i^* \right\}_{i \in [n]}, \left\{ z_j^* \right\}_{j \in [m]} \right)$ be an optimal solution of the LP.

- For each variable x_i, toss an independent y_i^*-biased coin.
- If the coin goes `HEAD`, we set x_i **true**, otherwise we set x_i **false**.

Analysis

Analysis

A typical upper bound of **OPT** for LP based algorithms is

$$\mathbf{OPT} \leq \mathbf{OPT}(LP) = \sum_{j=1}^{m} z_j^*$$

Analysis

A typical upper bound of **OPT** for LP based algorithms is

$$\mathbf{OPT} \leq \mathbf{OPT}(LP) = \sum_{j=1}^{m} z_j^*$$

We can further establish

$$\mathbf{E}\left[X\right] \geq \left(1 - \frac{1}{e}\right) \sum_{j=1}^{m} z_j^*.$$

Analysis

A typical upper bound of **OPT** for LP based algorithms is

$$\mathbf{OPT} \leq \mathbf{OPT}(LP) = \sum_{j=1}^{m} z_j^*$$

We can further establish

$$\mathbf{E}[X] \geq \left(1 - \frac{1}{e}\right) \sum_{j=1}^{m} z_j^*.$$

Therefore, the LP rounding is a $\left(1 - \frac{1}{e}\right)$-approximation algorithm for MaxSAT