Randomization in Approximation and Online Algorithms

Manos Thanos

Randomized Algorithms

NTUA

Randomized Rounding

- **Linear Programming** problems are problems for the optimization of a linear objective function, subject to linear equality and linear inequality constraints.
- If all the unknown variables are required to be integers, then the problem is called an **Integer Linear Programming** problem.
- Linear Programming has been proved to be **in P** by Leonid Khachiyan in 1979
- Integer Linear Programming is a **NP-hard** problem. There is a reduction from 3-SAT to ILP.

Randomized Rounding

- For problems in NP we **don't expect** to design **exact** algorithms with "good" running time
- We design **approximation algorithms** for these problems, e.g. algorithms that find a nearly optimal solution
- We say that **an algorithm is k-approximation** when the cost of the solution it finds, is at most k times the cost of the optimal solution for a minimization problem.

Randomized Rounding

The technique of randomized rounding

- Given an Integer Linear program, **we compute its relaxation**, e.g. the same linear program without the restriction for the variables to be integers.

- **We solve the relaxation** linear program to get an optimal solution OPT_f. In this solution the decision variables may not be integers.

- We use a randomize process to **round the fractional values** of the decision variables into integers. Within the process we have to achieve a "good" approximation factor.

Randomized Rounding (set cover)

- ***SET COVER:*** Given a universe X of m elements e_j, a collection of subsets of U, $S=\{s_1,\ldots,s_n\}$ and a cost function $c: S \to Q^+$ find a minimum cost subcollection of S that covers all elements of X

- *Linear Program:*

$$\text{minimize} \quad \sum_{i=1}^{m} c_i x_i$$

$$\text{subject to:} \quad \sum_{i:e_j \in S_i} x_i \geq 1 \qquad \forall j = 1,2,\ldots n$$

$$x_i \in \{0,1\} \qquad \forall i = 1,2,\ldots m$$

Randomized Rounding (set cover)

- A natural idea for rounding an optimal fractional solution is to **view fractions as probabilities**, flip coins with these biases and round accordingly. **Repeating** this process **O(logn) times** and picking a set if it is chosen in any of the iterations, we get a set cover with high probability having an expected cost at most O(logn) times the cost of the optimal solution.

Theorem: The above algorithm produces a set cover that covers all the elements and it has a cost at most O(logn)*OPT with probability greater or equal than 1/2

Randomized Rounding (set cover)

Proof:

Let x be an optimal solution to the linear program. For each set S pick S with probability x_s. Let C be the collection of sets picked. The expected cost of C is:

$$E[c(C)] = \sum_{s \in S} \Pr[s \text{ is picked}] \cdot c(s) = \sum_{s \in S} x_s c(s) = OPT_f$$

Next we compute the probability that an element e is covered by C. If e occurs in k sets of S that have probabilities to be picked $x_1,\ldots,x_k$, then since e is fractionally covered, $x_1+x_2+\ldots+x_k \geq 1$. The probability that e is covered, is minimized when $x_1=x_2=\ldots=x_k=1/k$. Thus,

$$\Pr[e \text{ is covered by } C] \geq 1-\left(1-\frac{1}{k}\right)^k \geq 1-\frac{1}{e}$$

Randomized Rounding (set cover)

In our algorithm we independently choose dlogn "set covers" and output their union C'. We choose d to be a constant such that:

$$\left(\frac{1}{e}\right)^{d\log n} \le \frac{1}{4n}$$

Now,

$$\Pr[e \text{ is not covered by } C'] \le \left(\frac{1}{e}\right)^{d\log n} \le \frac{1}{4n}$$

Summing over all elements, we get:

$$\Pr[\text{we haven't a valid set cover}] \le n\frac{1}{4n} \le \frac{1}{4} \qquad (1)$$

Randomized Rounding (set cover)

Clearly $E[c(C')] \leq OPT_f \cdot d \log n$

Applying Markov's Inequality we get:

$$\Pr[c(C') \geq OPT_f \cdot 4d \log n] \leq \frac{1}{4} \qquad (2)$$

As a result of (1) and (2) we have that algorithm produces a set cover that covers all the elements and it has a cost at most 4dlogn*OPT with probability greater or equal than 1/2

□

Randomized Rounding (MAX-SAT)

- ***MAX-SAT:*** Given a **conjunctive normal form formula f** on Boolean variables $x_1,\ldots x_n$, and **non-negative weights, w_c,** for each clause c of f, **find a truth assignment** to the Boolean variables that **maximizes the total weight of satisfied clauses**. Let *size(c)* denote the number of literals in c. Let random variable *W* denote the total weight of satisfied clauses and W_c denote the weight contributed by clause c to *W*.
- *Linear Program:*

$$\text{maximize} \quad \sum_{c \in C} w_c z_c$$

$$\text{subject to:} \quad \sum_{i:S_c^+} y_i + \sum_{i:S_c^-} (1 - y_i) \geq z_c \qquad \forall c \in C$$

$$y_i, z_c \in \{0,1\} \qquad \forall i, \forall c$$

Randomized Rounding (max-sat)

- The algorithm we will use is straightforward. We solve the relaxation of the linear program to get a fractional solution (**y*,z***). Independently we set x_i to True with probability y_i*. Finally we output the resulting truth assignment.

Theorem: The above algorithm produces a solution with expected profit more or equal than (1-1/e)*OPT

JOHNSON'S ALGORITHM FOR MAX-SAT

- The algorithm is the following:

  Set each Boolean variable to be True independently with probability 1/2 and output the resulting truth assignment.

Theorem: The above algorithm produces a solution with expected profit equal or greater than 1/2*OPT

Proof:

If the size of a clause is k, then it is not satisfied if all its literals are set to False. As a result $E[W_c] = (1-2^{-k})w_c$

For a clause c with $k \geq 1$, $E[W_c] \geq \frac{1}{2} w_c$. By linearity of expectation:

$$E[W] = \sum_{c \in C} E[W_c] \geq \frac{1}{2} \sum_{c \in C} w_c \geq \frac{1}{2} OPT$$

□

Derandomization Via the method of Conditional expectation

- **Johnson's algorithm can be derandomized** using the method of conditional probabilities

- **We will use the self-reducibility tree T for formula f**. Each internal node at level i corresponds to a setting for Boolean variables $x_1,\ldots,x_i$, and each leaf represents a complete truth assignment to the n variables.

- **We can compute the conditional expectation of any node in T in polynomial time**. Consider a node at level i. Let φ be the Boolean formula, on variables $x_{i+1}, \ldots, x_n$, obtained for this node via self-reducibility. Clearly the expected weight of satisfied clauses of φ can be computed in polynomial time. Adding to this the weight of clauses of f already satisfied gives the answer

Derandomization Via the method of Conditional expectation

- **We can compute in polynomial time, a path** from the root to a leaf of T such that **the conditional expectation of each node on this path is equal or greater than E[W].** Because of the fact that x_{i+1} is equally likely to be set to True or False, the conditional expectation of a node is the average of the conditional expectation of its children, i.e.,

$$E[W \mid x_1 = True, \ldots, x_i = True] = \frac{1}{2} E[W \mid x_1 = True, \ldots, x_i = True, x_{i+1} = True] + \frac{1}{2} E[W \mid x_1 = True, \ldots, x_i = True, x_{i+1} = False]$$

As a result of this fact, the child with the larger conditional expectation has at least the same value of its father's conditional expectation.

This path gives an assignment with total weight $\geq$ ½*OPT

A 3/4-FACTOR ALGORITHM FOR MAX-SAT

- We will combine the previous two algorithms as follows: We flip an unbiased coin and if the result is 1 then we run the first algorithm, else we run the second algorithm.

Theorem: The above algorithm produces a solution with expected profit equal or greater than 3/4*OPT

Proof:

Let size(c)=k. We have proved that

$$E[W_c \mid coin = 0] \geq (1-2^{-k})w_c \geq (1-2^{-k})w_c z_c^* \qquad (1)$$

$$E[W_c \mid coin = 1] \geq \left(1-\left(1-\frac{1}{k}\right)^k\right) w_c z_c^* \qquad (2)$$

A 3/4-FACTOR ALGORITHM FOR MAX-SAT

Combining (1) and (2) we get:

$$E[W_c] = \frac{1}{2}E[W_c \mid coin = 0] + \frac{1}{2}E[W_c \mid coin = 1] \geq$$

$$\geq \frac{1}{2} w_c z_c^* \left[\left(1 - \left(1 - \frac{1}{k} \right)^k \right) + (1 - 2^{-k}) \right] \qquad (3)$$

For k=1 or k=2 we get from (3) that $E[W_c] \geq \frac{3}{4} w_c z_c^*$

For $k \geq 3$

$$E[W_c] \geq \left[\frac{7}{16} + \frac{(1-1/e)}{2} \right] w_c z_c^* \geq \frac{3}{4} w_c z_c^*$$

A 3/4-FACTOR ALGORITHM FOR MAX-SAT

By linearity of expectation:

$$E[W] = \sum_{c \in C} E[W_c] \geq \sum_{c \in C} \frac{3}{4} w_c z_c^* = \frac{3}{4} OPT$$

which completes the proof. □

- We can change the above ¾-factor algorithm to make it deterministic:
  - Use the derandomized 1/2 –factor algorithm to solve the problem
  - Use the derandomized (1-1/e) –factor algorithm to solve the problem
  - Output the better of the two assignments

An improved algorithm for max-2sat

- A **quadratic program** is a problem of **optimizing a quadratic function** of integer valued variables, **subject to quadratic constraints** on these variables**. If each monomial** in the objective function , as well as in each of the constraints, **is of degree 0 or 2**, then we will say that this is a **strict quadratic program**.

- **A strict quadratic program** over n variables **defines a vector program** over n vector variables in R^n. Every degree 2 term corresponds to an inner product

- **Vector programs are equivalent to semi-definite programs**

- For all $\varepsilon>0$ **semi-definite programs can be solved** within an additive error of ε, **in time polynomial in n** and $\log(1/\varepsilon)$.

An improved algorithm for MAX-2SAT

- We can construct a strict quadratic program for MAX-2SAT

  Corresponding to each Boolean variable x_i, we introduce variable y_i which is constrained to be +1 or -1. In addition, we introduce another variable y_0 which is also constrained to be +1 or -1. Variable x_i will be true if $y_i = y_0$ and false otherwise. Now:

$$v(x_i) = \frac{1 + y_0 y_i}{2} \qquad v(\overline{x_i}) = \frac{1 - y_0 y_i}{2}$$

$$v(x_i \vee x_j) = 1 - v(\overline{x_i})\, v(\overline{x_j}) = 1 - \frac{1 - y_0 y_i}{2} \cdot \frac{1 - y_0 y_j}{2}$$

$$= \frac{1 + y_0 y_i}{4} + \frac{1 + y_0 y_j}{4} + \frac{1 - y_i y_j}{4}$$

An improved algorithm for max-2sat

- *Strict Quadratic Program:*

$$\text{maximize} \quad \sum_{0 \le i < j \le n} \left(a_{ij}(1 + y_i y_j) + b_{ij}(1 - y_i y_j) \right)$$

$$\text{subject to:} \quad y_i^2 = 1 \qquad \forall 0 \le i \le n$$

$$y_i \in Z \qquad \forall 0 \le i \le n$$

- *Vector Relaxation Program:*

$$\text{maximize} \quad \sum_{0 \le i < j \le n} \left(a_{ij}(1 + u_i u_j) + b_{ij}(1 - u_i u_j) \right)$$

$$\text{subject to:} \quad u_i u_i = 1 \qquad \forall 0 \le i \le n$$

$$u_i \in R^{n+1} \qquad \forall 0 \le i \le n$$

An improved algorithm for max-2sat

- The algorithm is the following:

  We solve the vector program. Let $a_0,\ldots, a_n$ be an optimal solution. We pick a vector r uniformly distributed on the unit sphere in n+1 dimensions and we set $y_i=1$ iff $r \cdot a_i \geq 0$

  This gives a truth assignment for the Boolean variables.

Theorem: The above algorithm produces a solution with expected profit equal or greater than a*OPT where a>0,87856

An improved algorithm for max-2sat

Proof:

Let W be the random variable denoting the weight of the truth assignment produced by the algorithm. We will prove that E[W]≥a*OPT.

We can see that:

$$E[W] = 2 \sum_{0 \le i < j \le n} a_{ij} \Pr[y_i = y_j] + b_{ij} \Pr[y_i \neq y_j]$$

Let θ_{ij} denote the angle between a_i and a_j .Then:

$$\Pr[y_i \neq y_j] = \frac{\theta_{ij}}{\pi} \qquad \Pr[y_i = y_j] = 1 - \frac{\theta_{ij}}{\pi}$$

$$E[W] = 2 \sum_{0 \le i < j \le n} a_{ij} \left(1 - \frac{\theta_{ij}}{\pi}\right) + b_{ij} \frac{\theta_{ij}}{\pi} \qquad (1)$$

An improved algorithm for max-2sat

Now, if we choose

$$\mathsf{a} = \frac{2}{\pi} \min_{0 \le \theta \le \pi} \frac{\theta}{1 - \cos\theta}$$

We have that:

$$\frac{\theta_{ij}}{\pi} \ge \mathsf{a}\left(\frac{1-\cos\theta_{ij}}{2}\right) \qquad (2) \qquad\qquad 1 - \frac{\theta_{ij}}{\pi} \ge \mathsf{a}\left(\frac{1+\cos\theta_{ij}}{2}\right) \qquad (3)$$

(1) becomes from (2) and (3):

$$E[W] = 2 \sum_{0 \le i < j \le n} a_{ij}\left(1 - \frac{\theta_{ij}}{\pi}\right) + b_{ij}\frac{\theta_{ij}}{\pi} \ge \mathsf{a} \sum_{0 \le i < j \le n} a_{ij}(1+\cos\theta_{ij}) + b_{ij}(1-\cos\theta_{ij})$$

An improved algorithm for max-2sat

Because of the fact that vectors are on the unit sphere we have that $u_i u_j = \cos\theta_{ij}$. The minimization function was

$$\text{minimize} \quad \sum_{0 \le i < j \le n} \left(a_{ij}(1 + u_i u_j) + b_{ij}(1 - u_i u_j) \right)$$

And the angles θ_{ij} are the angles between the vectors that gives the optimal solution. Thus:

$$E[W] \ge \mathsf{a} \sum_{0 \le i < j \le n} a_{ij}(1 + \cos\theta_{ij}) + b_{ij}(1 - \cos\theta_{ij}) \ge \mathsf{a} \cdot OPT$$

Using elementary calculus we can prove that

$$\mathsf{a} = \frac{2}{\pi} \min_{0 \le \theta \le \pi} \frac{\theta}{1 - \cos\theta} \Rightarrow \mathsf{a} > 0.87856$$

The above algorithm is 0.87856-approximation

□

Online Algorithms

- **Online algorithms receive** and **process the input in partial amounts**. In a typical setting, an online algorithm receives a sequence of requests for service. **It must service each request before** it services **the next one**. In servicing each request the algorithm has a choice of several alternatives, each with an associated cost. **The choice influences the future requests**.

- In order to evaluate an online algorithm **we compare its total cost** on a sequence of requests **to the total cost of an offline algorithm** that services the same sequence of requests. **The worst case ratio over all possible request sequences is the competitive ratio** of an online algorithm and is the natural measure of the quality of the algorithm

Online Paging Problem

- ***PAGING PROBLEM:*** Consider the following **two level virtual memory system**. Each level can store a number of fixed-size memory units, called pages. The first level, called the **slow memory** can **store** a fixed set $P=\{p_1,\ldots,p_N\}$ of **N pages**. The second level, called the **fast memory**, can **store** any **k**-subset of P. Given a **request** for a page $\mathbf{p_i}$ the **system must bring $\mathbf{p_i}$ to the fast memory**. If p_i is already to the fast memory, then no cost incurs; otherwise a unit cost incurs. In order to move p_i to the fast memory another page has to be evicted. **The problem is to minimize the total cost** by making a wise choice when evicting a page. It is obvious that a page which will be requested in the near future shouldn't be evicted but the future is unknown as the problem is online.

Deterministic Algorithms for Paging

- *Least Recently Used* (**LRU**): evict the item in the cache whose most recent request occurred furthest in the past
- *Fist-in, First-out* (**FIFO**): evict the item that has been in the cash for the longest period
- Last-in, First-out (**LIFO**): evict the item that has been moved to the cash most recently
- *Least Frequently Used* (**LFU**): evict the item in the cash that has been requested least often
- LRU and FIFO are k-competitive. LIFO and LFU don't achieve a bounded competitiveness coefficient.

Deterministic Algorithms for Paging

Theorem: There isn't a deterministic online algorithm for the online paging problem that achieves a competitiveness coefficient better than k where k is the number of pages that could be stored in the cash

- Can we overcome the negative result of the above theorem using randomization?

Adversary models

- We can view the offline algorithm servicing the request sequence as an **adversary who is not only managing the cache, but it also generating the request sequence**. The adversary's **goal is to increase the cost of the given online algorithm, while keeping it down for the offline algorithm**.

- *Oblivious adversary*: must construct the request sequence in advance and pays optimally

- *Adaptive online adversary*: serves the current request online and then chooses the next request based on the online algorithm's actions so far

- *Adaptive offline adversary*: chooses the next request based on the online algorithm's actions thus far, but pays the optimal offline cost to service the resulting request sequence

A Randomized Algorithm for Paging

- *Algorithm MARKER*: The algorithm proceeds in a **series of rounds**. Each of the k cache **locations has a marker bit** associated with it. **At the beginning of every round**, all k **marker bits are reset to zero**. As memory requests come in, the algorithm processes them as follows. **If the item requested is** already **in** one of the k **cache** locations, the **marker bit** of that location **is set to one**. **If the request is a miss, the item is brought into the cache** and the item that is **evict**ed to make room for it is chosen as follows: **choose an unmarked** cache **location uniformly at random**, evict the item in it and set its marker bit to one. After all the locations have been thus marked, the round is deemed over on the next request to an item not in the cache.

A Randomized Algorithm for Paging

Theorem: Marker algorithm is $2H_k$-competitive against an oblivious adversary, where H_k is the k-th harmonic number

Proof:

We assume that algorithm MARK and algorithm OPT start with the same set of items in their caches; otherwise it is possible to attribute the difference to an additive constant.

For each phase i we call the pages in the cache immediately prior to the start of the phase **old**. We call the rest of the pages **new.**

Consider the i-th phase of the algorithm. Let m_i be the number of new pages requested in this phase. It is obvious that the worst scenario for MARK is that all new pages requests precede the old pages requests.

A Randomized Algorithm for Paging

Ordered in this way, the first m_i requests incur m_i page faults. We now investigate the page faults resulted for the first k-m_i requests to old pages.

The j-th old page requested in this phase is in the cache with probability:

$$\frac{k - m_i - (j-1)}{k - (j-1)}$$

A page fault will occur only if the page is not in the cash, thus with probability:

$$\frac{m_i}{k - (j-1)}$$

A Randomized Algorithm for Paging

Hence, the expected number of faults during the i-th phase is:

$$m_i + \sum_{j=1}^{k-m_i} \frac{m_i}{k-(j-1)} = m_i + m_i(H_k - H_{m_i}) = m_i(H_k - H_{m_i} + 1) \le m_i H_k$$

During the i-th and (i-1)-th phase, at least $k+m_i$ pages are requested. As a result, OPT has made at least mi page faults. At the first phase OPT made at least m1 page faults. Thus the total number of page faults of OPT is:

$$\frac{1}{2}\sum_i m_i$$

Thus the competitive ratio is at least:

$$\frac{\sum_i m_i H_k}{\frac{1}{2}\sum_i m_i} = 2H_k$$

□

Lower Bound For Randomized paging Algorithms

Theorem: Any randomized algorithm for online paging can't achieve a competitiveness coefficient better than H_k against an oblivious adversary

Proof:

We will use *Yao's minimal principle*: The expected running time of the optimal deterministic algorithm for an arbitrarily chosen distribution ρ is a lower bound on the expected running time of the optimal randomized algorithm for problem P

It suffices to show that for an arbitrarily chosen distribution ρ any deterministic algorithm can't achieve a better competitiveness coefficient than H_k. Then H_k will be a lower bound for every randomized algorithm.

Lower Bound For Randomized paging Algorithms

We will use only k+1 different pages $I=\{p_1,\ldots p_{k+1}\}$. The distribution on requests will be the following: The request ri is chosen uniformly at random from the pages $I-\{r_{i-1}\}$.

We will divide the requests into rounds. A round ends when all different pages in I are requested. The expected length of a round is kH_k as it is equivalent to a random walk on a complete graph with k+1 vertices.

The offline algorithm makes a page fault in every round.

The expected page faults of every deterministic algorithm is H_k.

As a result H_k is a lower bound on every deterministic algorithm and this yields the result

□

ONLINE K-SERVER PROBLEM

- ***K-SERVER PROBLEM:*** Let k>1 be an integer and let M=(M,d) be a metric space where M is a set of points with |M|>k and d is a metric over M. An algorithm controls k mobile servers, which are located on points of M. The algorithm is presented with a sequence $\sigma=r_1,r_2,\ldots,r_n$ of requests where a request r_i is a point in the space. We say that a request r_i is served if one of the k servers lies on point r_i. By moving servers, the algorithm, must serve all the requests sequentially. Our goal is to minimize the total cost occurred which is the total distance moved by all servers until all requests are served.

Online k-server problem

- k-server is a generalization of the online paging problem.
- The greedy algorithm doesn't achieve a bounded competitiveness coefficient
- *k-server conjecture*: Any metric space allows for a deterministic k-competitive, k-server algorithm. (k has been proved to be a lower bound on the competitiveness coefficient of any deterministic algorithm)
- We have found a deterministic (2k-1)-competitive algorithm for all metric spaces.
- We have found deterministic k-competitive algorithms when the metric space is a line or a tree.

Online k-server problem

- We know of relatively few cases where randomization against an oblivious adversary beats the lower bound of k for deterministic algorithms.
- It is an open question if there is a randomized algorithm that can achieve a competitiveness coefficient better than H_k against an oblivious adversary.
- We have shown that the lower bound on every online algorithm against an adaptive online adversary is k.

Online k-server problem

- *Harmonic Algorithm*: Let d_i be the distance between the i-th server and the requested point. The algorithm chooses independently of the past to the j-th server to service the request with probability:

$$\frac{1/d_i}{\sum_1^k 1/d_i}$$

- We have found metrics that the Harmonic algorithm can't achieve a competitiveness coefficient better than k(k+1)/2

- We have shown that $\frac{5k2^k}{4}$ is an upper bound on Harmonic's competitiveness coefficient

Thank You