More Approximation Algorithms

Lecture 26

Dec 5, 2019

Formal definition of approximation algorithm

An algorithm $\mathcal{A}$ for an optimization problem X is an α-approximation algorithm if the following conditions hold:

- for each instance I of X the algorithm $\mathcal{A}$ correctly outputs a valid solution to I
- $\mathcal{A}$ is a polynomial-time algorithm
- Letting $OPT(I)$ and $\mathcal{A}(I)$ denote the values of an optimum solution and the solution output by $\mathcal{A}$ on instances I, $OPT(I)/\mathcal{A}(I) \leq \alpha$ *and* $\mathcal{A}(I)/OPT(I) \leq \alpha$. Alternatively:
  - If X is a minimization problem: $\mathcal{A}(I)/OPT(I) \leq \alpha$
  - If X is a maximization problem: $OPT(I)/\mathcal{A}(I) \leq \alpha$

Definition ensures that $\alpha \geq 1$

To be formal we need to say $\alpha(n)$ where $n = |I|$ since in some cases the *approximation ratio* depends on the size of the instance.

Formal definition of approximation algorithm

Unfortunately notation is not consistently used. Some times people use the following convention:

- If X is a minimization problem then $\mathcal{A}(I)/OPT(I) \leq \alpha$ and here $\alpha \geq 1$.
- If X is a maximization problem then $\mathcal{A}(I)/OPT(I) \geq \alpha$ and here $\alpha \leq 1$.

Usually clear from the context.

Part I

Approximation for Set Cover

Set Cover

Input: Universe $\mathcal{U}$ of n elements and m subsets $S_1, S_2, \ldots, S_m$ such that $\cup_i S_i = \mathcal{U}$.

Goal: Pick fewest number of subsets to cover all of $\mathcal{U}$ (equivalently, whose union is $\mathcal{U}$.

Set Cover

Input: Universe $\mathcal{U}$ of n elements and m subsets $S_1, S_2, \ldots, S_m$ such that $\cup_i S_i = \mathcal{U}$.

Goal: Pick fewest number of subsets to cover all of $\mathcal{U}$ (equivalently, whose union is $\mathcal{U}$.

Greedy$(\mathcal{U}, S_1, S_2, \ldots, S_m)$

```
    Uncovered = U
    While Uncovered ≠ ∅ do
        Pick set S_j that covers max number of uncovered elements
        Add S_j to solution
        Uncovered = Uncovered − S_j
    endWhile
    Output chosen sets
```

Greedy for Weighted Set Cover

Input: Universe $\mathcal{U}$ of n elements and m subsets $S_1, S_2, \ldots, S_m$ such that $\cup_i S_i = \mathcal{U}$. For each set S_i a non-negative weight w_i.

Goal: Pick smallest weight collection of subsets to cover all of $\mathcal{U}$.

Greedy$(\mathcal{U}, S_1, S_2, \ldots, S_m, w_1, w_2, \ldots, w_m)$

```
    Uncovered = U
    While Uncovered ≠ ∅ do
        Let j = argmax_h |S_h ∩ U| / w_j
        Add S_j to solution
        Uncovered = Uncovered − S_j
    endWhile
    Output chosen sets
```

Greedy and Hardness

Theorem

Greedy gives a $(1 + \ln d)$*-approximation for (weighted) Set Cover where* d *is the maximum set size. Approximation ratio is at most* $(1 + \ln n)$ *where* n *is number of elements.*

Greedy and Hardness

Theorem

Greedy gives a $(1 + \ln d)$*-approximation for (weighted) Set Cover where* d *is the maximum set size. Approximation ratio is at most* $(1 + \ln n)$ *where* n *is number of elements.*

Theorem

Unless $P = NP$ *there is no* $(1 - \epsilon) \ln n$*-approximation for Set Cover for any fixed* $\epsilon > 0$*.*

Bad examples for Greedy

$n = 2(1 + 2 + 2^2 + 2^p) = 2(2^{p+1} - 1)$, $m = 2 + 2(p + 1)$, $OPT = 2$, Greedy picks $p + 1$ and hence ratio is $\Omega(\ln n)$.

Analysis of Greedy for Set Cover

- Let k^* be minimum number of sets to cover $\mathcal{U}$. Let k be number of sets chosen by Greedy.
- Let α_i be number of new elements covered in iteration i.
- Let β_i be number of elements uncovered at end of iteration i. $\beta_0 = n$.

Analysis of Greedy for Set Cover

- Let k^* be minimum number of sets to cover $\mathcal{U}$. Let k be number of sets chosen by Greedy.
- Let α_i be number of new elements covered in iteration i.
- Let β_i be number of elements uncovered at end of iteration i. $\beta_0 = n$.

Lemma

$\alpha_i \geq \beta_{i-1}/k^*$.

Proof.

Let $\mathcal{U}_i$ be uncovered elements at start of iteration i. All these elements can be covered by some k^* sets since all of $\mathcal{U}$ can be covered by k^* sets. There exists one of those sets that covers at least $\mathcal{U}_i/k^*$ elements. Greedy picks the best set and hence covers at least that many elements. Note $\mathcal{U}_i = \beta_{i-1}$. □

Analysis of Greedy contd

Lemma

$\alpha_i \geq \beta_{i-1}/k^*$.

$$\beta_i = \beta_{i-1} - \alpha_i \leq \beta_{i-1} - \beta_{i-1}/k^* = (1 - 1/k^*)\beta_{i-1}.$$

Hence by induction,

$$\beta_i \leq \beta_0(1 - 1/k^*)^i = n(1 - 1/k^*)^i.$$

Thus, after $k = k^* \ln n$ iterations number number of uncovered elements is at most

$$n(1 - 1/k^*)^{k^* \ln n} \leq ne^{-\ln n} \leq 1.$$

Thus algorithm terminates in at most $k^* \ln n + 1$ iterations. Total number of sets chosen is number of iterations.

Analysis contd

Theorem

Greedy gives a $(\ln n + 1)$*-approximation for Set Cover.*

- Can show a tighter bound of $(\ln d + 1)$ where d is maximum set size.
- Analysis a bit harder for weighted case.

LP and Randomized Rounding for Set Cover

Write (weighted) Set Cover problem as an integer linear program

$$\begin{array}{lll} \text{Minimize} & \sum_{i=1}^{m} w_i x_i & \\ \text{subject to} & \sum_{i:j \in S_i} x_i \geq 1 & \text{for } 1 \leq j \leq n \\ & x_i \in \{0,1\} & \text{for each } S_i \end{array}$$

LP and Randomized Rounding for Set Cover

Write (weighted) Set Cover problem as an integer linear program

$$\begin{array}{lll} \text{Minimize} & \sum_{i=1}^{m} w_i x_i & \\ \text{subject to} & \sum_{i:j \in S_i} x_i \geq 1 & \text{for } 1 \leq j \leq n \\ & x_i \in \{0,1\} & \text{for each } S_i \end{array}$$

Relax integer program to a linear program

$$\begin{array}{lll} \text{Minimize} & \sum_{i=1}^{m} w_i x_i & \\ \text{subject to} & \sum_{i:j \in S_i} x_i \geq 1 & \text{for } 1 \leq j \leq n \\ & x_i \geq 0 & \text{for each } S_i \end{array}$$

LP and Randomized Rounding for Set Cover

Write (weighted) Set Cover problem as an integer linear program

$$\begin{array}{lll} \text{Minimize} & \sum_{i=1}^{m} w_i x_i & \\ \text{subject to} & \sum_{i:j \in S_i} x_i \geq 1 & \text{for } 1 \leq j \leq n \\ & x_i \in \{0,1\} & \text{for each } S_i \end{array}$$

Relax integer program to a linear program

$$\begin{array}{lll} \text{Minimize} & \sum_{i=1}^{m} w_i x_i & \\ \text{subject to} & \sum_{i:j \in S_i} x_i \geq 1 & \text{for } 1 \leq j \leq n \\ & x_i \geq 0 & \text{for each } S_i \end{array}$$

Can solve linear program in polynomial time.

Let x^* be an optimum solution to the linear program.

Lemma

$OPT \geq \sum_{i=1}^{m} w_i x_i^*$.

Rounding the LP solution

How do we round x^* to obtain a feasible integer solution?

Recall deterministic rounding for Vertex Cover:

- Round x_i^* to $\mathbf{1}$ if $x_i^* \geq \mathbf{1/2}$, otherwise to $\mathbf{0}$
- Produced a feasible solution because each edge belonged to only two sets
- For general Set Cover instances, to obtain a feasible solution we need to round x_i^* to $\mathbf{1}$ if $x_i^* \geq \mathbf{1}/f$ where f is the maximum number of sets any element belongs to. Gives an f-approximation
- f can be as large as n

Randomized Rounding

Randomized rounding: an important idea in approximation

- Interpret x_i^* as the probability of choosing S_i
- Pick each S_i with probability x_i^*

Randomized Rounding

Randomized rounding: an important idea in approximation

- Interpret x_i^* as the probability of choosing S_i
- Pick each S_i with probability x_i^*

Questions:

- Will we obtain a feasible integer solution?
- What is the expected cost of the solution?

One round analysis

Let X_i be indicator random variable for S_i being chosen.

$$\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] = x_i^*$$

Total cost: is $C = \sum_i w_i X_i$ and hence

$$\mathbf{E}[C] = \sum_i w_i \, \mathbf{E}[X_i] = \sum_i w_i x_i^* \leq OPT.$$

One round analysis

Let X_i be indicator random variable for S_i being chosen.

$$\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] = x_i^*$$

Total cost: is $C = \sum_i w_i X_i$ and hence

$$\mathbf{E}[C] = \sum_i w_i \, \mathbf{E}[X_i] = \sum_i w_i x_i^* \leq OPT.$$

Let Y_j be indicator random variable for element j to be covered. What is $\mathbf{Pr}[Y_j = 1]$?

One round analysis

Let X_i be indicator random variable for S_i being chosen.

$$\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1] = x_i^*$$

Total cost: is $C = \sum_i w_i X_i$ and hence

$$\mathbf{E}[C] = \sum_i w_i \, \mathbf{E}[X_i] = \sum_i w_i x_i^* \leq OPT.$$

Let Y_j be indicator random variable for element j to be covered. What is $\mathbf{Pr}[Y_j = 1]$?

$$\mathbf{Pr}[Y_j = 0] = \prod_{i : j \in S_i} (1 - x_i^*) \leq \prod_{i : j \in S_i} e^{-x_i^*} \leq 1/e.$$

One round analysis contd

Let Y_j be indicator random variable for element j to be covered.

$$\mathbf{Pr}[Y_j = 0] = \leq 1/e \rightarrow \mathbf{Pr}[Y_j = 1] \geq (1 - 1/e).$$

$Y = \sum_{j=1}^{n} Y_j$ is number of elements covered. We have

$$\mathbf{E}[Y] = \sum_j \mathbf{E}[Y_j] \geq (1 - 1/e)n$$

One round analysis contd

Let Y_j be indicator random variable for element j to be covered.

$$\Pr[Y_j = 0] = \leq 1/e \rightarrow \Pr[Y_j = 1] \geq (1 - 1/e).$$

$Y = \sum_{j=1}^{n} Y_j$ is number of elements covered. We have

$$\mathbf{E}[Y] = \sum_j \mathbf{E}[Y_j] \geq (1 - 1/e)n$$

Thus after one round of randomized rounding

- Expected cost of sets chosen is at most OPT
- Expected number of elements covered is $(1 - 1/e)n$.

Above suggests repeating the algorithm $\Theta(\log n)$ times to get an $O(\log n)$-approximation algorithm.

Round and Fix Algorithm

RRandFix$(\mathcal{U}, S_1, S_2, \ldots, S_m, w_1, w_2, \ldots, w_m)$

```
Solve LP relaxation and obtain optimum solution x*
Let α ≥ 1 be a parameter
Pick each S_i independently in solution with prob. min{1, α x*_i}
For each uncovered element j do
    Add cheapest set that covers it to the solution
Output chosen sets
```

Algorithm has two phases: randomized rounding and fixing/alteration.

Round and Fix Algorithm

RRandFix$(\mathcal{U}, S_1, S_2, \ldots, S_m, w_1, w_2, \ldots, w_m)$

```
Solve LP relaxation and obtain optimum solution x*
Let α ≥ 1 be a parameter
Pick each S_i independently in solution with prob. min{1, α x*_i}
For each uncovered element j do
    Add cheapest set that covers it to the solution
Output chosen sets
```

Algorithm has two phases: randomized rounding and fixing/alteration.

Observation: Algorithm outputs a feasible solution for any α

Main question: What should be α and how do we bound expected cost of the algorithm?

Analysis

For element j let Y_j be the indicator rv for j being covered after the randomized rounding phase.

Lemma

$\mathbf{Pr}[Y_j = 0] \leq e^{-\alpha}$.

Proof.

Similar to previous one round analysis. If for some i where $j \in S_i$ we have $\alpha x_i^* = 1$ then $Y_j = 0$. Otherwise

$$\mathbf{Pr}[Y_j = 0] = \prod_{i:j \in S_i} (1 - \alpha x_i^*) \leq e^{-\alpha}$$

since $\sum_{i:j \in S_i} x_i^* \geq 1$. □

Analysis contd

Lemma

For element j let β_j be cost of cheapest set covering j. Then $\beta_j \leq \sum_{i:j \in S_i} w_i x_i^*$.

Proof.

Because $\sum_{i:j \in S_i} x_i^* \geq 1$. □

Analysis contd

Lemma

For element j let β_j be cost of cheapest set covering j. Then $\beta_j \leq \sum_{i:j \in S_i} w_i x_i^*$.

Proof.

Because $\sum_{i:j \in S_i} x_i^* \geq 1$. □

Corollary

$\sum_{j=1}^{n} \beta_j \leq n \sum_{i=1}^{m} w_i x_i^* \leq nOPT$.

Analysis contd

Lemma

For $\alpha = \ln n$ the expected cost of the algorithm is at most $(1 + \ln n) \sum_i w_i x_i^* \leq (1 + \ln n) OPT$.

Analysis contd

Proof.

Let C be the cost of the algorithm. Let X_i be indicator rv for S_i being chosen in first random phase. Then $C \leq \sum_{i=1}^{m} w_i X_i + \sum_{j=1}^{n} (1 - Y_j)\beta_j$.

$$\begin{aligned}
\mathbf{E}[C] &\leq \sum_i w_i \alpha x_i^* + \sum_{j=1} \mathbf{Pr}[Y_j = 0]\, \beta_j \\
&\leq \alpha \sum_i w_i x_i^* + n e^{-\alpha} \sum_i w_i x_i^* \\
&\leq (\alpha + n e^{-\alpha}) \sum_i w_i x_i^* \leq (1 + \ln n) \sum_i w_i x_i^*.
\end{aligned}$$

Improved analysis

Lemma

Let d be maximum set size. For $\alpha = \ln d$ the expected cost of the algorithm is at most $(1 + \ln d) \sum_i w_i x_i^* \leq (1 + \ln d) OPT$.

Only change in analysis. $\sum_j \beta_j \leq d \sum_i w_i x_i^*$.
How?

$$\begin{aligned}
\sum_j \beta_j &\leq \sum_{j=1}^{n} \sum_{i : j \in S_i} w_i x_i^* \\
&\leq \sum_{i=1}^{m} w_i x_i^* |S_i| \\
&\leq d \sum_i w_i x_i^*.
\end{aligned}$$

Greedy vs LP Approach

- Greedy simple and efficient to implement. LP more involved to solve efficiently.
- Greedy works in implicit settings while LP approach is infeasible in those settings.
- LP provides a lower bound. Greedy does not.
- LP performs provably much better in many special cases of interest while Greedy can be behave badly in the same special cases.

Part II

Approximation for Load Balancing

Load Balancing

Given n jobs $J_1, J_2, \ldots, J_n$ with sizes $s_1, s_2, \ldots, s_n$ and m identical machines $M_1, \ldots, M_m$ assign jobs to machines to minimize maximum load (also called makespan).

Problem sometimes referred to as multiprocessor scheduling.
Example: 3 machines and 8 jobs with sizes $4, 3, 1, 2, 5, 6, 9, 7$.

Load Balancing

Given n jobs $J_1, J_2, \ldots, J_n$ with sizes $s_1, s_2, \ldots, s_n$ and m identical machines $M_1, \ldots, M_m$ assign jobs to machines to minimize maximum load (also called makespan).

Formally, an assignment is a mapping $f : \{1, 2, \ldots, n\} \rightarrow \{1, \ldots, m\}$.

- The load $\ell_f(j)$ of machine M_j under f is $\sum_{i:f(i)=j} s_i$
- Goal is to find f to minimize $\max_j \ell_f(j)$.

Greedy List Scheduling

List-Scheduling

Let $J_1, J_2, \ldots, J_n$ be an ordering of jobs
for $i = 1$ to n do
    Schedule job J_i on the currently least loaded machine

Greedy List Scheduling

List-Scheduling

```
Let J_1, J_2, ..., J_n be an ordering of jobs
for i = 1 to n do
    Schedule job J_i on the currently least loaded machine
```

Example: 3 machines and 8 jobs with sizes 4, 3, 1, 2, 5, 6, 9, 7.

Example

Example: $\mathbf{3}$ machines and $\mathbf{8}$ jobs with sizes $\mathbf{4, 3, 1, 2, 5, 6, 9, 7}$.
Different list: $\mathbf{9, 7, 6, 5, 4, 3, 2, 1}$

Two lower bounds on OPT

OPT is the optimum load

- average load: $OPT \geq \sum_{i=1}^{n} s_i/m$. Why?
- maximum job size: $OPT \geq \mathbf{max}_{i=1}^{n} s_i$. Why?

Analysis of Greedy List Scheduling

Theorem

Let L be makespan of Greedy List Scheduling on a given instance. Then $L \leq 2(1 - 1/m)OPT$ where OPT is the optimum makespan for that instance.

Analysis of Greedy List Scheduling

Theorem

Let L be makespan of Greedy List Scheduling on a given instance. Then $L \leq 2(1 - 1/m)OPT$ where OPT is the optimum makespan for that instance.

- Let M_h be the machine which achieves the load L for Greedy List Scheduling.
- Let J_i be the job that was last scheduled on M_h.
- Why was J_i scheduled on M_h? It means that M_h was the least loaded machine when J_i was considered. Implies all machines had load at least $L - s_i$ at that time.

Analysis continued

Lemma

$L - s_i \leq (\sum_{\ell=1}^{i-1} s_\ell)/m$.

Proof.

Since all machines had load at least $L - s_i$ it means that $m(L - s_i) \leq \sum_{\ell=1}^{i-1} s_\ell$ and hence

$$L - s_i \leq (\sum_{\ell=1}^{i-1} s_\ell)/m.$$

Analysis continued

But then

$$\begin{aligned} L &\leq (\sum_{\ell=1}^{i-1} s_\ell)/m + s_i \\ &\leq (\sum_{\ell=1}^{n} s_\ell)/m + (1 - \frac{1}{m})s_i \\ &\leq OPT + (1 - \frac{1}{m})OPT \\ &\leq 2(1 - \frac{1}{m})OPT. \end{aligned}$$

A Tight Example

Question: Is the analysis of the algorithm tight? That is, are there instances where L is $2(1 - 1/m)OPT$?

A Tight Example

Question: Is the analysis of the algorithm tight? That is, are there instances where L is $2(1 - 1/m)OPT$?

Example: $m(m - 1)$ jobs of size 1 and one big job of size m where m is number of machines.

A Tight Example

Question: Is the analysis of the algorithm tight? That is, are there instances where L is $2(1 - 1/m)OPT$?

Example: $m(m - 1)$ jobs of size 1 and one big job of size m where m is number of machines.

- $OPT = m$. Why?
- If the list has large job at end the schedule created by Greedy is $m + m - 1 = 2m - 1$.

Ordering jobs from largest to smallest

Obvious heuristic: Order jobs in decreasing size order and then use Greedy.

Does it lead to an improved performance in the worst case? How much?

Ordering jobs from largest to smallest

Obvious heuristic: Order jobs in decreasing size order and then use Greedy.

Does it lead to an improved performance in the worst case? How much?

Theorem

Greedy List Scheduling with jobs sorted from largest to smallest gives a $\mathbf{4/3}$*-approximation and this is essentially tight.*

Analysis

Not so obvious.

If we only use average load and maximum job size as lower bounds on OPT then we cannot improve the bound of 2

Example: $m + 1$ jobs of size 1

- $OPT = 2$
- average load is $1 + 1/m$ and max job size is 1

Analysis

Not so obvious.

If we only use average load and maximum job size as lower bounds on OPT then we cannot improve the bound of 2

Example: $m + 1$ jobs of size 1

- $OPT = 2$
- average load is $1 + 1/m$ and max job size is 1

Need another lower bound

Another useful lower bound

Lemma

Suppose jobs are sorted, that is $s_1 \geq s_2 \geq \ldots \geq s_n$ *and* $n > m$ *then* $OPT \geq s_m + s_{m+1} \geq 2s_{m+1}$.

Another useful lower bound

Lemma

Suppose jobs are sorted, that is $s_1 \geq s_2 \geq \ldots \geq s_n$ *and* $n > m$ *then* $OPT \geq s_m + s_{m+1} \geq 2s_{m+1}$.

Proof.

Consider the first $m+1$ jobs $J_1, \ldots, J_{m+1}$. By pigeon hole principle two of these jobs on same machine. Load on that machine is at least the sum of the smallest two job sizes in the first $m+1$ jobs. □

Proving a 3/2 bound

Using the new lower bound we will prove a weaker upper bound of **3/2** rather than the right bound of **4/3**.

As before let M_j be the machine achieving the makespan L and let J_i be the last job assigned to M_j. we have $L - s_i \leq \frac{1}{m} \sum_{\ell=1}^{i-1} s_\ell$. Now a more careful analysis.

- Case 1: If s_i is only job on M_j then $L \leq s_i \leq OPT$.
- Case 2: At least one more job on M_j before s_i.
  - We have seen that $L - s_i \leq OPT$.
  - **Claim:** $s_i \leq OPT/2$
  - Together, we have $L \leq OPT + s_i \leq 3OPT/2$.

Proof of Claim

Since M_j had a job before s_i we have $i > m$.

Hence $s_i \leq s_{m+1}$ becase jobs were sorted. Since $OPT \geq 2s_{m+1}$, we have $s_i \leq s_{m+1} \leq OPT/2$.

PTAS for Load Balancing

Known: For load balancing one can get a $(1+\epsilon)$-approximation for any fixed $\epsilon > 0$.