More Approximation Algorithms

Lecture 25
April 26, 2018

Most slides are courtesy Prof. Chekuri
Formal definition of approximation algorithm

An algorithm \( \mathcal{A} \) for an optimization problem \( X \) is an \( \alpha \)-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.
Part I

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.
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 $J_1, J_2, \ldots, J_n$ 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: 3 machines and 8 jobs with sizes 4, 3, 1, 2, 5, 6, 9, 7.
Different list: 9, 7, 6, 5, 4, 3, 2, 1
Two lower bounds on $OPT$

$OPT$ is the optimum load

Lower bounds on $OPT$: 

\[ OPT \geq \frac{\sum_{i=1}^{n} s_i}{m}. \text{ Why?} \]

\[ OPT \geq \max_{i=1}^{n} s_i. \text{ Why?} \]
**Two lower bounds on** \( OPT \)

**\( OPT \)** is the optimum load

**Lower bounds on \( OPT \):**
- average load: \( OPT \geq \sum_{i=1}^{n} s_i / m \). Why?
- maximum job size: \( OPT \geq \max_{i=1}^{n} s_i \). 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.
Lemma

\[ L - s_i \leq \left( \sum_{\ell=1}^{i-1} s_{\ell} \right) / m. \]

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

\[ L - s_i \leq \left( \sum_{\ell=1}^{i-1} s_{\ell} \right) / m. \]
Analysis continued

But then

\[ L \leq \frac{\left( \sum_{\ell=1}^{i-1} s_\ell \right)}{m} + s_i \leq \frac{\left( \sum_{\ell=1}^{n} s_\ell \right)}{m} + (1 - \frac{1}{m}) s_i \leq \]

\[ \leq \]

\[ \leq \]
But then

\[ L \leq \frac{\sum_{\ell=1}^{i-1} s_\ell}{m} + s_i \]

\[ \leq \frac{\sum_{\ell=1}^{n} s_\ell}{m} + (1 - \frac{1}{m})s_i \]

\[ \leq OPT + (1 - \frac{1}{m})OPT \]

\[ \leq (2 - \frac{2}{m})OPT \]

\[ = 2(1 - \frac{1}{m})OPT. \]
A Tight Example

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

**Question:** Is the analysis of the algorithm tight? That is, are there instances where $L$ is $2\left(1 - \frac{1}{m}\right)OPT$?

**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 Greedy will give makespan of
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 Greedy will give makespan of $m + m - 1 = 2m - 1$. 

Ordering jobs from largest to smallest

**Obvious heuristic:** Order jobs in decreasing size order and then use Greedy.
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 $\frac{4}{3}$-approximation and this is essentially tight.
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 \( \frac{4}{3} \)-approximation and this is essentially tight.**_
Analysis

Not so obvious.

Example: $m + 1$ jobs of size $1$

OPT = $2$
average load is $\frac{1 + 1}{m}$
and max job size is $1$

Need another lower bound
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 $OPT \geq s_m + s_{m+1} \geq 2s_{m+1}$. 

Proof.
Consider OPTimal schedule of the first $m + 1$ jobs $J_1, \ldots, J_{m+1}$.
By pigeon hole principle two of these jobs on same machine.
$OPT \geq \text{Load on that machine} \geq \text{the sum of the smallest two job sizes in the first } m + 1 \text{ jobs} = s_m + s_{m+1}$.
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 OPTimal schedule of the first $m + 1$ jobs $J_1, \ldots, J_{m+1}$. By pigeon hole principle two of these jobs on same machine.
Lemma

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

Proof.

Consider OPTimal schedule of the first \( m + 1 \) jobs \( J_1, \ldots, J_{m+1} \). By pigeon hole principle two of these jobs on same machine.
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 OPTimal schedule of the first $m + 1$ jobs $J_1, \ldots, J_{m+1}$. By pigeon hole principle two of these jobs on same machine.

$OPT \geq$ Load on that machine
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 \( \text{OPT} \geq s_m + s_{m+1} \geq 2s_{m+1} \).

Proof.

Consider OPTimal schedule of the first \( m + 1 \) jobs \( J_1, \ldots, J_{m+1} \). By pigeon hole principle two of these jobs on same machine.

\( \text{OPT} \geq \) Load on that machine \( \geq \) the sum of the smallest two job sizes in the first \( m + 1 \) jobs
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 $\text{OPT} \geq s_m + s_{m+1} \geq 2s_{m+1}$.

**Proof.**

Consider OPTimal schedule of the first $m + 1$ jobs $J_1, \ldots, J_{m+1}$. By pigeon hole principle two of these jobs on same machine.

$\text{OPT} \geq \text{Load on that machine} \geq \text{the sum of the smallest two job sizes in the first } m + 1 \text{ jobs} = s_m + s_{m+1}$. 
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 $4/3$. 
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

$L - s_i \leq \frac{1}{m} \sum_{\ell=1}^{i-1} s_\ell \leq OPT$. Now a more careful analysis.
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 $L - s_i \leq \frac{1}{m} \sum_{\ell=1}^{i-1} s_\ell \leq OPT$. Now a more careful analysis.

- Case 1: If $s_i$ is only job on $M_j$ then
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 $L - s_i \leq \frac{1}{m} \sum_{l=1}^{i-1} s_l \leq OPT$. Now a more careful analysis.

Case 1: If $s_i$ is only job on $M_j$ then $L = s_i \leq OPT$. 
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

$$L - s_i \leq \frac{1}{m} \sum_{\ell=1}^{i-1} s_\ell \leq OPT.$$ 

Now a more careful analysis.

- **Case 1**: If $s_i$ is only job on $M_j$ then $L = s_i \leq OPT$.

- **Case 2**: At least one more job on $M_j$ before $s_i$.
  
  We have seen that $L - s_i \leq OPT$. 

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

$$L - s_i \leq \frac{1}{m} \sum_{\ell=1}^{i-1} s_\ell \leq OPT.$$  Now a more careful analysis.

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

$$L - s_i \leq \frac{1}{m} \sum_{\ell=1}^{i-1} s_{\ell} \leq OPT.$$ 

Now a more careful analysis.

- **Case 1:** If $s_i$ is only job on $M_j$ then $L = s_i \leq OPT$.
- **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

Claim: \( s_i \leq \frac{OPT}{2} \)
Claim: \( s_i \leq OPT / 2 \)

Proof:
Since \( M_j \) had a job before \( s_i \) we have \( i > m \). Why?
Proof of Claim

Claim: $s_i \leq \frac{OPT}{2}$

Proof:
Since $M_j$ had a job before $s_i$ we have $i > m$. Why?

Hence $s_i \leq s_{m+1}$ because jobs were sorted by decreasing size.
Proof of Claim

Claim: \( s_i \leq \frac{OPT}{2} \)

Proof:
Since \( M_j \) had a job before \( s_i \) we have \( i > m \). Why?

Hence \( s_i \leq s_{m+1} \) because jobs were sorted by decreasing size. Since \( 2s_{m+1} \leq OPT \), we have \( s_i \leq s_{m+1} \leq \frac{OPT}{2} \).
Part II

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 $\bigcup_i S_i = \mathcal{U}$.

**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 $\bigcup_i S_i = \mathcal{U}$.

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

\[
\text{Greedy}(\mathcal{U}, S_1, S_2, \ldots, S_m) \\
\text{Uncovered} = \mathcal{U} \\
\text{While Uncovered} \neq \emptyset \text{ do} \\
\quad \text{Pick set } S_j \text{ that covers max number of uncovered elements} \\
\quad \text{Add } S_j \text{ to solution} \\
\quad \text{Uncovered} = \text{Uncovered} - S_j \\
\text{endWhile} \\
\text{Output chosen sets}
\]
Analysis of Greedy

Let $k^*$ be minimum number of sets to cover $\mathcal{U}$. Let $k$ be number of sets chosen by Greedy.

Let $\alpha_i$ be the number of new elements covered in iteration $i$.

Let $\beta_i$ be the number of elements uncovered at end of iteration $i$. $\beta_0 = n$. 

Let $k^*$ be minimum number of sets to cover $\mathcal{U}$. Let $k$ be number of sets chosen by Greedy.

Let $\alpha_i$ be $\#$ new elements covered in iteration $i$.
Let $\beta_i$ be $\#$ elements uncovered at end of iteration $i$. $\beta_0 = n$.

Lemma

\[ \alpha_i \geq \beta_{i-1}/k^*. \]
Analysis of Greedy

- Let $k^*$ be minimum number of sets to cover $\mathcal{U}$. Let $k$ be number of sets chosen by Greedy.
- Let $\alpha_i$ be \# new elements covered in iteration $i$.
- Let $\beta_i$ be \# 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.
Analysis of Greedy

- Let $k^*$ be minimum number of sets to cover $\mathcal{U}$. Let $k$ be number of sets chosen by Greedy.
- Let $\alpha_i$ be number of new elements covered in iteration $i$.
- Let $\beta_i$ 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.
Analysis of Greedy

- Let $k^*$ be minimum number of sets to cover $U$. Let $k$ be number of sets chosen by Greedy.
- Let $\alpha_i$ be $\#$ new elements covered in iteration $i$.
- Let $\beta_i$ be $\#$ elements uncovered at end of iteration $i$. $\beta_0 = n$.

Lemma

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

Proof.

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

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

\[ \beta_i = \beta_{i-1} - \alpha_i \]
### Lemma

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

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

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

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

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

- Algorithm generalizes to weighted case easily. Pick sets in each iteration based on ratio of elements covered divided by weight. Analysis a bit harder but also gives a \((\ln n + 1)\)-approximation.
Algorithm generalizes to weighted case easily. Pick sets in each iteration based on ratio of elements covered divided by weight. Analysis a bit harder but also gives a \((\ln n + 1)\)-approximation.

Can show a tighter bound of \((\ln d + 1)\) where \(d\) is maximum set size.
Theorem

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

- Algorithm generalizes to weighted case easily. Pick sets in each iteration based on ratio of elements covered divided by weight.
- Analysis a bit harder but also gives a \((\ln n + 1)\)-approximation.
- Can show a tighter bound of \((\ln d + 1)\) where \(d\) is maximum set size.

Theorem

*Unless* \(P = NP\) *no* \((\ln n + \epsilon)\)-*approximation for Set Cover.*
A bad example for Greedy

\[ n = 2(1 + 2 + 2^2 + \cdots + 2^p) = 2(2^{p+1} - 1), \quad m = 2 + (p + 1), \]

\[ OPT = 2, \quad \text{Greedy picks } (p + 1) \quad \text{and hence ratio is } \Omega(\ln n). \]
Advantage of Greedy

Greedy is a simple algorithm. In several scenarios the set system is \textit{implicit} and exponentially large in $n$. 
Advantage of Greedy

Greedy is a simple algorithm. In several scenarios the set system is *implicit* and exponentially large in $n$. Nevertheless, the Greedy algorithm can be implemented efficiently if there is an oracle that each step picks the best set efficiently.

Example. Covering all the edges of a graph using minimum number of disjoint trees.
Advantage of Greedy

Greedy is a simple algorithm. In several scenarios the set system is *implicit* and exponentially large in $n$. Nevertheless, the Greedy algorithm can be implemented efficiently if there is an oracle that each step picks the best set efficiently.

**Example.** Covering all the edges of a graph using minimum number of disjoint trees.
Max $k$-Cover

**Input:** Universe $\mathcal{U}$ of $n$ elements and $m$ subsets $S_1, S_2, \ldots, S_m$ and integer $k$.

**Goal:** Pick $k$ subsets to maximize number of covered elements.
Max $k$-Cover

**Input:** Universe $\mathcal{U}$ of $n$ elements and $m$ subsets $S_1, S_2, \ldots, S_m$ and integer $k$.

**Goal:** Pick $k$ subsets to maximize number of covered elements.

\[
\text{Greedy}(\mathcal{U}, S_1, S_2, \ldots, S_m, k)
\]

\[
\text{Uncovered} = \mathcal{U}
\]

for $i = 1$ to $k$ do

Pick set $S_j$ that covers max number of uncovered elements

Add $S_j$ to solution

$\text{Uncovered} = \text{Uncovered} - S_j$

endWhile

Output chosen $k$ sets
Analysis

Similar to previous analysis.

- Let $OPT$ be max number of covered elements to cover $U$.
- Let $\alpha_i$ be number of new elements covered in iteration $i$.
- Let $\gamma_i$ be number of elements covered by greedy after $i$ iterations.

Let $\beta_i = OPT - \gamma_i$. Define $\beta_0 = OPT$. 

Lemma $\alpha_i \geq \beta_i - 1/k$. 

Proof: Exercise.
Analysis

Similar to previous analysis.

- Let $OPT$ be max number of covered elements to cover $U$.
- Let $\alpha_i$ be number of new elements covered in iteration $i$.
- Let $\gamma_i$ be number of elements covered by greedy after $i$ iterations.
- Let $\beta_i = OPT - \gamma_i$. Define $\beta_0 = OPT$.

Lemma $\alpha_i \geq \beta_i - \frac{1}{k}$. 
Proof: Exercise.
Analysis

Similar to previous analysis.

- Let $OPT$ be max number of covered elements to cover $\mathcal{U}$.
- Let $\alpha_i$ be number of new elements covered in iteration $i$.
- Let $\gamma_i$ be number of elements covered by greedy after $i$ iterations.
- Let $\beta_i = OPT - \gamma_i$. Define $\beta_0 = OPT$.

**Lemma**  
\[ \alpha_i \geq \beta_{i-1}/k. \]

**Proof:** Exercise.
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,

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

Thus, after $k$ iterations,

$$\beta_k \leq OPT(1 - 1/k)^k \leq OPT/e.$$
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,

\[ \beta_i \leq \beta_0 (1 - 1/k)^i = \text{OPT} (1 - 1/k)^i. \]

Thus, after \( k \) iterations,

\[ \beta_k \leq \text{OPT} (1 - 1/k)^k \leq \text{OPT} / e. \]

Thus \( \gamma_k = \text{OPT} - \beta_k \geq (1 - 1/e) \text{OPT}. \)
Theorem

Greedy gives a \((1 - 1/e)\)-approximation for Max \(k\)-Coverage.

Above theorem generalizes to submodular function maximization and has many applications.

Theorem (Feige 1998)

Unless \(P = NP\) there is no \((1 - 1/e - \epsilon)\)-approximation for Max \(k\)-Coverage for any fixed \(\epsilon > 0\).