Set-Cover approximation through Dual Fitting

K. Subramani

Lane Department of Computer Science and Electrical Engineering
West Virginia University

March 25, 2014
Outline

1 Preliminaries
1 Preliminaries

2 Greedy Algorithms
Outline

1. Preliminaries
2. Greedy Algorithms
3. Dual-Fitting based Analysis of Greedy Algorithm
The Set Cover Problem

Given,
1. A ground set \( U = \{e_1, e_2, \ldots, e_n\} \),
2. A collection of sets \( S = \{S_1, S_2, \ldots, S_m\} \), where \( S_i \subseteq U \),\( i = 1, 2, \ldots, m \),
3. A weight function \( c: S_i \to \mathbb{Z}^+ \),

find a collection of subsets \( S_i \), whose union covers the elements of \( U \) at minimum cost.

Note: If all weights are unity (or the same), the problem is called the Cardinality Set Cover problem.
The Set Cover Problem

Given,
The Set Cover Problem

Given,

1. A ground set $U = \{e_1, e_2, \ldots, e_n\}$,
2. A collection of sets $S = \{S_1, S_2, \ldots, S_m\}$, $S_i \subseteq U$, $i = 1, 2, \ldots, m$,
3. A weight function $c: S_i \rightarrow \mathbb{Z}^+$,

find a collection of subsets $S_i$, whose union covers the elements of $U$ at minimum cost.

Note: If all weights are unity (or the same), the problem is called the Cardinality Set Cover problem.
The Set Cover Problem

Given,

1. A ground set \( U = \{e_1, e_2, \ldots, e_n\} \),
### The Set Cover Problem

Given,

1. A ground set \( U = \{e_1, e_2, \ldots, e_n\} \),
2. A collection of sets \( S_P = \{S_1, S_2, \ldots, S_m\} \), \( S_i \subseteq U \), \( i = 1, 2, \ldots, m \)
The Set Cover Problem

Given,

1. A ground set $U = \{e_1, e_2, \ldots, e_n\}$,
2. A collection of sets $S_P = \{S_1, S_2, \ldots S_m\}$, $S_i \subseteq U, i = 1, 2, \ldots, m$
3. A weight function $c : S_i \rightarrow \mathbb{Z}_+$,
The Set Cover Problem

Given,

1. A ground set $U = \{e_1, e_2, \ldots, e_n\}$,
2. A collection of sets $S_P = \{S_1, S_2, \ldots S_m\}$, $S_i \subseteq U$, $i = 1, 2, \ldots, m$
3. A weight function $c : S_i \rightarrow \mathbb{Z}_+$,

find a collection of subsets $S_i$, whose union covers the elements of $U$ at minimum cost.
The Set Cover Problem

Given,
1. A ground set $U = \{e_1, e_2, \ldots, e_n\}$,
2. A collection of sets $S_P = \{S_1, S_2, \ldots S_m\}$, $S_i \subseteq U$, $i = 1, 2, \ldots, m$
3. A weight function $c : S_i \rightarrow \mathbb{Z}^+$,

find a collection of subsets $S_i$, whose union covers the elements of $U$ at minimum cost.

Note

If all weights are unity (or the same),
The Set Cover Problem

Given,

1. A ground set $U = \{e_1, e_2, \ldots, e_n\}$,
2. A collection of sets $S_P = \{S_1, S_2, \ldots S_m\}$, $S_i \subseteq U, i = 1, 2, \ldots, m$
3. A weight function $c : S_i \rightarrow \mathbb{Z}^+$,

find a collection of subsets $S_i$, whose union covers the elements of $U$ at minimum cost.

Note

If all weights are unity (or the same), the problem is called the Cardinality Set Cover problem.
The Greedy Algorithm (Cardinality)

Let $C$ be the empty set.

while (there exists an uncovered element in $U$) do

1. Find the set $S_j$ with the largest number of uncovered elements.
2. Set $C = C \cup S_j$.
3. Throw out all the covered elements from $U$.

endwhile
The Greedy Algorithm (Cardinality)

Greedy Approach

1. \( C = \emptyset \).
2. While there exists an uncovered element in \( U \).
3. Find the set \( S_j \) with the largest number of uncovered elements.
4. Set \( C = C \cup S_j \).
5. Throw out all the covered elements from \( U \).
6. Endwhile
The Greedy Algorithm (Cardinality)

Greedy Approach

1. \( C = \emptyset \).
2. while (there exists an uncovered element in \( U \))
3.    Find the set \( S_j \) with the largest number of uncovered elements.
4.    Set \( C = C \cup S_j \).
5.    Throw out all the covered elements from \( U \).
6. endwhile
The Greedy Algorithm (Cardinality)

Greedy Approach

1. Cover $C = \emptyset$. 

while (there exists an uncovered element in $U$) 

2. Find the set $S_j$ with the largest number of uncovered elements.

3. Set $C = C \cup S_j$.

4. Throw out all the covered elements from $U$.

endwhile
The Greedy Algorithm (Cardinality)

Greedy Approach

1. Cover $C = \emptyset$.
2. while (there exists an uncovered element in $U$)
The Greedy Algorithm (Cardinality)

Greedy Approach

1. Cover \( C = \emptyset \).
2. **while** (there exists an uncovered element in \( U \))
3. Find the set \( S_j \) with the largest number of uncovered elements.
The Greedy Algorithm (Cardinality)

Greedy Approach

1. Cover \( C = \emptyset \).
2. **while** (there exists an uncovered element in \( U \))
3. Find the set \( S_j \) with the largest number of uncovered elements.
4. Set \( C = C \cup S_j \).
The Greedy Algorithm (Cardinality)

Greedy Approach

1. Cover $C = \emptyset$.
2. while (there exists an uncovered element in $U$)
3.     Find the set $S_j$ with the largest number of uncovered elements.
4.     Set $C = C \cup S_j$.
5.     Throw out all the covered elements from $U$. 
The Greedy Algorithm (Cardinality)

Greedy Approach

1. Cover $C = \emptyset$.
2. while (there exists an uncovered element in $U$)
3.     Find the set $S_j$ with the largest number of uncovered elements.
4.     Set $C = C \cup S_j$.
5.     Throw out all the covered elements from $U$.
6. endwhile
Analysis of the greedy approach

Let $OPT$ denote the size of the optimal set cover.

1. To begin with, there exists at least one set $S_i$ with $n_{OPT}$ or more uncovered elements.

2. The set picked by the greedy algorithm has at least $n_{OPT}$ uncovered elements.

3. The number of elements uncovered after the first iteration is at most $n - n_{OPT} = n \cdot (1 - 1_{OPT})$.

4. What happens if greedy picked one of $OPT$'s sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.

5. Hence there is at least one set with $n \cdot (1 - 1_{OPT}) \cdot (OPT - 1)$ uncovered elements.

6. However, we can safely assume that there is at least one set with $n \cdot (1 - 1_{OPT}) \cdot OPT$ uncovered elements!

7. The number of uncovered elements after the second iteration is at most $n \cdot (1 - 1_{OPT}) - n \cdot (1 - 1_{OPT}) \cdot OPT = n \cdot (1 - 1_{OPT})^2$. 


Analysis of the greedy approach

1. Let $\text{OPT}$ denote the size of the optimal set cover.
2. To begin with, there exists at least one set $S_i$ with $n_{\text{OPT}}$ or more uncovered elements. (Why?)
3. The set picked by the greedy algorithm has at least $n_{\text{OPT}}$ uncovered elements. (Why?)
4. The number of elements uncovered after the first iteration is at most $n - n_{\text{OPT}} = n \cdot (1 - 1_{\text{OPT}})$.
5. What happens if greedy picked one of OPT's sets? The remaining uncovered elements will have to be covered by at most $(\text{OPT} - 1)$ sets.
6. Hence there is at least one set with $n \cdot (1 - 1_{\text{OPT}}) (\text{OPT} - 1)$ uncovered elements.
7. But we don't know that we were that lucky. However, we can safely assume that there is at least one set with $n \cdot (1 - 1_{\text{OPT}}) \text{OPT}$ uncovered elements!
8. The number of uncovered elements after the second iteration is at most $n \cdot (1 - 1_{\text{OPT}}) - n \cdot (1 - 1_{\text{OPT}}) \text{OPT} = n \cdot (1 - 1_{\text{OPT}})^2$. 
Analysis of the greedy approach

1. Let $OPT$ denote the size of the optimal set cover.

2. To begin with, there exists at least one set $S_i$ with $n_{OPT}$ or more uncovered elements. (Why?)

3. The set picked by the greedy algorithm has at least $n_{OPT}$ uncovered elements. (Why?)

4. The number of elements uncovered after the first iteration is at most $n - n_{OPT} = n \cdot (1 - 1_{OPT})$.

5. What happens if greedy picked one of $OPT$'s sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.

6. Hence there is at least one set with $n \cdot (1 - 1_{OPT}) (OPT - 1)$ uncovered elements.

7. But we don't know that we were that lucky. However, we can safely assume that there is at least one set with $n \cdot (1 - 1_{OPT}) OPT$ uncovered elements!

8. The number of uncovered elements after the second iteration is at most $n \cdot (1 - 1_{OPT}) - n \cdot (1 - 1_{OPT}) OPT = n \cdot (1 - 1_{OPT})^2$. 


Let $OPT$ denote the size of the optimal set cover.
Analysis of the greedy approach

Analysis

1. Let $OPT$ denote the size of the optimal set cover.
2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements.
Let $OPT$ denote the size of the optimal set cover.

To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)
Analysis of the greedy approach

<table>
<thead>
<tr>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let $OPT$ denote the size of the optimal set cover.</td>
</tr>
<tr>
<td>2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)</td>
</tr>
<tr>
<td>3. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements.</td>
</tr>
</tbody>
</table>
Analysis of the greedy approach

Let $OPT$ denote the size of the optimal set cover.

To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)

The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)
Analysis of the greedy approach

### Analysis

1. Let $OPT$ denote the size of the optimal set cover.
2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)
3. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)
4. The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT}$.
Analysis of the greedy approach

Analysis

1. Let $OPT$ denote the size of the optimal set cover.
2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)
3. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)
4. The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT} = n \cdot \left(1 - \frac{1}{OPT}\right)$. 
Analysis of the greedy approach

Let $OPT$ denote the size of the optimal set cover.

To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)

The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)

The number of elements uncovered after the first iteration is at most

$$n - \frac{n}{OPT} = n \cdot (1 - \frac{1}{OPT}).$$

What happens if greedy picked one of OPT's sets?
## Analysis of the greedy approach

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Let $OPT$ denote the size of the optimal set cover.</td>
</tr>
<tr>
<td>2</td>
<td>To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. <em>(Why?)</em></td>
</tr>
<tr>
<td>3</td>
<td>The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. <em>(Why?)</em></td>
</tr>
<tr>
<td>4</td>
<td>The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT} = n \cdot (1 - \frac{1}{OPT})$.</td>
</tr>
<tr>
<td>5</td>
<td>What happens if greedy picked one of OPT’s sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.</td>
</tr>
</tbody>
</table>
Analysis of the greedy approach

Let $OPT$ denote the size of the optimal set cover.

To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)

The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)

The number of elements uncovered after the first iteration is at most
$$n - \frac{n}{OPT} = n \cdot \left(1 - \frac{1}{OPT}\right).$$

What happens if greedy picked one of OPT's sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.

Hence there is at least one set with $\frac{n\cdot(1 - \frac{1}{OPT})}{(OPT - 1)}$ uncovered elements.
Analysis of the greedy approach

1. Let $OPT$ denote the size of the optimal set cover.
2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)
3. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)
4. The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT} = n \cdot \left(1 - \frac{1}{OPT}\right)$.
5. What happens if greedy picked one of OPT's sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.
6. Hence there is at least one set with $\frac{n \cdot \left(1 - \frac{1}{OPT}\right)}{(OPT - 1)}$ uncovered elements.
7. But we don’t know that we were that lucky.
Analysis of the greedy approach

Let $OPT$ denote the size of the optimal set cover.

1. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements.
   (Why?)

2. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)

3. The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT} = n \cdot (1 - \frac{1}{OPT})$.

4. What happens if greedy picked one of OPT's sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.

5. Hence there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{(OPT - 1)}$ uncovered elements.

6. But we don't know that we were that lucky. However, we can safely assume that there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{OPT}$ uncovered elements!
Analysis of the greedy approach

1. Let $OPT$ denote the size of the optimal set cover.
2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)
3. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)
4. The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT} = n \cdot (1 - \frac{1}{OPT})$.
5. What happens if greedy picked one of OPT's sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.
6. Hence there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{(OPT - 1)}$ uncovered elements.
7. But we don’t know that we were that lucky. However, we can safely assume that there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{OPT}$ uncovered elements!
8. The number of uncovered elements after the second iteration is at most
Analysis of the greedy approach

Analysis

1. Let $OPT$ denote the size of the optimal set cover.
2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)
3. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)
4. The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT} = n \cdot (1 - \frac{1}{OPT})$.
5. What happens if greedy picked one of OPT's sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.
6. Hence there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{(OPT - 1)}$ uncovered elements.
7. But we don't know that we were that lucky. However, we can safely assume that there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{OPT}$ uncovered elements!
8. The number of uncovered elements after the second iteration is at most $n \cdot (1 - \frac{1}{OPT}) - \frac{n \cdot (1 - \frac{1}{OPT})}{OPT}$.
Analysis of the greedy approach

Analysis

1. Let $OPT$ denote the size of the optimal set cover.

2. To begin with, there exists at least one set $S_i$ with $\frac{n}{OPT}$ or more uncovered elements. (Why?)

3. The set picked by the greedy algorithm has at least $\frac{n}{OPT}$ uncovered elements. (Why?)

4. The number of elements uncovered after the first iteration is at most $n - \frac{n}{OPT} = n \cdot (1 - \frac{1}{OPT})$.

5. What happens if greedy picked one of OPT’s sets? The remaining uncovered elements will have to be covered by at most $(OPT - 1)$ sets.

6. Hence there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{(OPT - 1)}$ uncovered elements.

7. But we don’t know that we were that lucky. However, we can safely assume that there is at least one set with $\frac{n \cdot (1 - \frac{1}{OPT})}{OPT}$ uncovered elements!

8. The number of uncovered elements after the second iteration is at most $n \cdot (1 - \frac{1}{OPT}) - \frac{n \cdot (1 - \frac{1}{OPT})}{OPT} = n \cdot (1 - \frac{1}{OPT})^2$. 
Analysis (contd.)

1. After $t = \text{OPT} \cdot \ln n$ iterations, the number of elements left is $n \cdot \left(1 - \frac{1}{\text{OPT}}\right)$.

2. What we have shown is that the greedy strategy finds a solution in $\text{OPT} \cdot \ln n$ iterations. Since exactly one set is picked in each iteration, the approximation factor of the greedy approach is $\ln n$. 

$$n \cdot \left(1 - \frac{1}{\text{OPT}}\right) < n \cdot \left(1 - \frac{1}{e}\right) = n \cdot e - \ln n = n \cdot n - 1 = 1$$
e.g., we are done.
After \( t = \OPT \cdot \ln n \) iterations, the number of elements left is
\[
n \cdot \left(1 - \frac{1}{\OPT}\right) \OPT \cdot \ln n < n \cdot \left(1 - e^{-\ln n}\right) = n \cdot e^{\ln n - 1} = 1
\]
i.e., we are done.

What we have shown is that the greedy strategy finds a solution in \( \OPT \cdot \ln n \) iterations.
Since exactly one set is picked in each iteration, the approximation factor of the greedy approach is \( \ln n \).
Analysis (contd.)

Final steps

1. After $t = OPT \cdot \ln n$ iterations, the number of elements left is
After $t = OPT \cdot \ln n$ iterations, the number of elements left is

$$n \cdot (1 - \frac{1}{OPT})^{OPT \cdot \ln n}$$
After $t = OPT \cdot \ln n$ iterations, the number of elements left is

$$n \cdot (1 - \frac{1}{OPT})^{OPT \cdot \ln n} < n \cdot (\frac{1}{e})^{\ln n}$$
After $t = OPT \cdot \ln n$ iterations, the number of elements left is

$$n \cdot (1 - \frac{1}{OPT})^{OPT \cdot \ln n} \leq n \cdot (\frac{1}{e})^{\ln n}$$

$$= n \cdot e^{-\ln n}$$
After $t = OPT \cdot \ln n$ iterations, the number of elements left is

$$n \cdot \left(1 - \frac{1}{OPT}\right)^{OPT \cdot \ln n} \leq n \cdot \left(\frac{1}{e}\right)^{\ln n}$$

$$= n \cdot e^{-\ln n}$$

$$= n \cdot e^{\ln n - 1}$$
After $t = \text{OPT} \cdot \ln n$ iterations, the number of elements left is

$$n \cdot (1 - \frac{1}{\text{OPT}})^{\text{OPT} \cdot \ln n} \leq n \cdot \left(\frac{1}{e}\right)^{\ln n}$$

$$= n \cdot e^{-\ln n}$$

$$= n \cdot e^{\ln n^{-1}}$$

$$= n \cdot n^{-1}$$

i.e., we are done.
Final steps

After \( t = OPT \cdot \ln n \) iterations, the number of elements left is

\[
n \cdot \left( 1 - \frac{1}{OPT} \right)^{OPT \cdot \ln n} < n \cdot \left( \frac{1}{e} \right)^{\ln n}
\]

\[
= n \cdot e^{-\ln n}
\]

\[
= n \cdot e^{\ln n^{-1}}
\]

\[
= n \cdot n^{-1}
\]

\[
= 1
\]
After $t = OPT \cdot \ln n$ iterations, the number of elements left is

$$n \cdot \left(1 - \frac{1}{OPT}\right)^{OPT \cdot \ln n} < n \cdot \left(\frac{1}{e}\right)^{\ln n} = n \cdot e^{-\ln n} = n \cdot e^{\ln n^{-1}} = n \cdot n^{-1} = 1$$

i.e., we are done.
After \( t = \text{OPT} \cdot \ln n \) iterations, the number of elements left is

\[
n \cdot (1 - \frac{1}{\text{OPT}})^{\text{OPT} \cdot \ln n} < n \cdot \left(\frac{1}{e}\right)^{\ln n} = n \cdot e^{-\ln n} = n \cdot e^{\ln n^{-1}} = n \cdot n^{-1} = 1
\]

i.e., we are done.

What we have shown is that the greedy strategy finds a solution in \( \text{OPT} \cdot \ln n \) iterations.
Analysis (contd.)

Final steps

1. After \( t = OPT \cdot \ln n \) iterations, the number of elements left is

\[
n \cdot \left(1 - \frac{1}{OPT}\right)^{OPT \cdot \ln n} < n \cdot \left(\frac{1}{e}\right)^{\ln n}
\]

\[
= n \cdot e^{-\ln n}
\]

\[
= n \cdot e^{\ln n^{-1}}
\]

\[
= n \cdot n^{-1}
\]

\[
= 1
\]

i.e., we are done.

2. What we have shown is that the greedy strategy finds a solution in \( OPT \cdot \ln n \) iterations. Since exactly one set is picked in each iteration, the approximation factor of the greedy approach is \( \ln n \).
The Greedy Algorithm (Weighted)

Cost-effectiveness of a set is $c(S) - |S - C|$.

price($e$) is the average cost at which element $e$ is covered.

while ($C \neq U$) do
  Find the most cost-effective set in the current iteration, say $S$.
  Let $\alpha_S$ denote the cost-effectiveness of $S$.
  Observe that $\alpha_S = c(S) - |S - C|$.
  Pick $S$ and for each $e \in S - C$, set price($e$) = $\alpha_S$.
  $C \rightarrow C \cup S$.
end while

Output the picked sets.
The Greedy Algorithm (Weighted)

**Weighted Greedy Algorithm**

1. The cost-effectiveness of a set is $c(S) / |S - C|$.
2. The price of an element $e$ is the average cost at which element $e$ is covered.
3. While ($C \neq U$) do
   1. Find the most cost-effective set in the current iteration, say $S$.
   2. Let $\alpha_S$ denote the cost-effectiveness of $S$.
   3. Observe that $\alpha_S = c(S) / |S - C|$.
   4. Pick $S$ and for each $e \in S - C$, set price($e$) = $\alpha_S$.
   5. $C \rightarrow C \cup S$.
4. End while
5. Output the picked sets.
The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. \( C \rightarrow \emptyset. \)
The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. $C \rightarrow \emptyset$.
2. Cost-effectiveness of a set is $\frac{c(S)}{|S-C|}$. 
The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. $C \rightarrow \emptyset$.
2. Cost-effectiveness of a set is $\frac{c(S)}{|S-C|}$.
3. $price(e)$ is the average cost at which element $e$ is covered.
The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. \( C \rightarrow \emptyset \).
2. Cost-effectiveness of a set is \( \frac{c(S)}{|S - C|} \).
3. \( \text{price}(e) \) is the average cost at which element \( e \) is covered.
4. \textbf{while} \( (C \neq U) \) \textbf{do}


The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. $C \rightarrow \emptyset$.
2. Cost-effectiveness of a set is $\frac{c(S)}{|S-C|}$.
3. $\text{price}(e)$ is the average cost at which element $e$ is covered.
4. **while** $(C \neq U)$ **do**
   
   5. Find the most cost-effective set in the current iteration, say $S$. 

   6. Let $\alpha_S$ denote the cost-effectiveness of $S$.
   
   7. Observe that $\alpha_S = \frac{c(S)}{|S-C|}$.
   
   8. Pick $S$ and for each $e \in S-C$, set $\text{price}(e) = \alpha_S$.
   
   9. $C \rightarrow C \cup S$.
   
10. **end while**

11. Output the picked sets.
The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. $C \rightarrow \emptyset$.
2. Cost-effectiveness of a set is $\frac{c(S)}{|S-C|}$.
3. $\text{price}(e)$ is the average cost at which element $e$ is covered.
4. \textbf{while} ($C \neq U$) \textbf{do}
5. Find the most cost-effective set in the current iteration, say $S$.
6. Let $\alpha_S$ denote the cost-effectiveness of $S$. 

$C \rightarrow C \cup S$. 

\textbf{end while}

Output the picked sets.
The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

Let $C \rightarrow \emptyset$.

2 Cost-effectiveness of a set is $\frac{c(S)}{|S - C|}$.

3 $\text{price}(e)$ is the average cost at which element $e$ is covered.

while $(C \neq U)$ do

5 Find the most cost-effective set in the current iteration, say $S$.

6 Let $\alpha_S$ denote the cost-effectiveness of $S$.

7 Observe that $\alpha_S = \frac{c(S)}{|S - C|}$.
The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. $C \rightarrow \emptyset$.
2. Cost-effectiveness of a set is $\frac{c(S)}{|S-C|}$.
3. $price(e)$ is the average cost at which element $e$ is covered.
4. while ($C \neq U$) do
5.   Find the most cost-effective set in the current iteration, say $S$.
6.   Let $\alpha_S$ denote the cost-effectiveness of $S$.
7.   Observe that $\alpha_S = \frac{c(S)}{|S-C|}$.
8.   Pick $S$ and for each $e \in S - C$, set $price(e) = \alpha_S$. 
9. $C \rightarrow C \cup S$.
10. end while
11. Output the picked sets.
The Greedy Algorithm (Weighted)

**Weighted Greedy Algorithm**

1. $C \rightarrow \emptyset$.
2. Cost-effectiveness of a set is $\frac{c(S)}{|S-C|}$.
3. $price(e)$ is the average cost at which element $e$ is covered.
4. **while** $(C \neq U)$ **do**
   5. Find the most cost-effective set in the current iteration, say $S$.
   6. Let $\alpha_S$ denote the cost-effectiveness of $S$.
   7. Observe that $\alpha_S = \frac{c(S)}{|S-C|}$.
   8. Pick $S$ and for each $e \in S - C$, set $price(e) = \alpha_S$.
   9. $C \rightarrow C \cup S$. 

**Output the picked sets.**
### The Greedy Algorithm (Weighted)

**Weighted Greedy Algorithm**

1. \( C \rightarrow \emptyset \).
2. Cost-effectiveness of a set is \( \frac{c(S)}{|S-C|} \).
3. \( \text{price}(e) \) is the average cost at which element \( e \) is covered.
4. **while** \( (C \neq U) \) **do**
   5. Find the most cost-effective set in the current iteration, say \( S \).
   6. Let \( \alpha_S \) denote the cost-effectiveness of \( S \).
   7. Observe that \( \alpha_S = \frac{c(S)}{|S-C|} \).
   8. Pick \( S \) and for each \( e \in S - C \), set \( \text{price}(e) = \alpha_S \).
   9. \( C \rightarrow C \cup S \).
5. **end while**
The Greedy Algorithm (Weighted)

1. \( C \rightarrow \emptyset \).
2. Cost-effectiveness of a set is \( \frac{c(S)}{|S-C|} \).
3. \( \text{price}(e) \) is the average cost at which element \( e \) is covered.
4. \( \textbf{while } (C \neq U) \textbf{ do} \)
5. Find the most cost-effective set in the current iteration, say \( S \).
6. Let \( \alpha_S \) denote the cost-effectiveness of \( S \).
7. Observe that \( \alpha_S = \frac{c(S)}{|S-C|} \).
8. Pick \( S \) and for each \( e \in S - C \), set \( \text{price}(e) = \alpha_S \).
9. \( C \rightarrow C \cup S \).
10. \( \textbf{end while} \)
11. Output the picked sets.
Lemma

Let $e_1, e_2, ... , e_n$ denote the elements of $U$, in the order in which they were covered.

For each $k \in \{1, 2, ..., n\}$, $\text{price}(e_k) \leq \text{OPT}(n - k + 1)$.

Proof.

1. In each iteration, the remaining elements can be covered by the "leftover" sets of the optimal set cover at a cost of at most $\text{OPT}$.

2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most $\text{OPT} \bar{C}$, where $\bar{C} = U - C$.

3. When $e_k$ was covered, $|\bar{C}| \geq (n - k + 1)$.

4. Since our covering algorithm is greedy, we have, $\text{price}(e_k) \leq \text{OPT} \bar{C} = \text{OPT}(n - k + 1)$. 
Lemma

Let $e_1, e_2, \ldots, e_n$ denote the elements of $U$, in the order in which they were covered.
Lemma

Let $e_1, e_2, \ldots e_n$ denote the elements of U, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$,
Analysis

Lemma

Let \( e_1, e_2, \ldots, e_n \) denote the elements of \( U \), in the order in which they were covered. For each \( k \in \{1, 2, \ldots, n\} \), \( \text{price}(e_k) \leq \frac{\text{OPT}}{(n-k+1)} \).
Lemma

Let $e_1, e_2, \ldots e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, \( \text{price}(e_k) \leq \frac{\text{OPT}}{n-k+1} \).

Proof.

1. In each iteration, the remaining elements can be covered by the "leftover" sets of the optimal set cover at a cost of at most \( \text{OPT} \).
2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most \( \text{OPT} \).
3. When \( e_k \) was covered, \( |\bar{\mathcal{C}}| \geq (n-k+1) \).
4. Since our covering algorithm is greedy, we have, \( \text{price}(e_k) \leq \frac{\text{OPT}}{n-k+1} \).
Lemma

Let $e_1, e_2, \ldots, e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, \( \text{price}(e_k) \leq \frac{\text{OPT}}{n-k+1} \).

Proof.

1. In each iteration, the remaining elements can be covered by the “leftover” sets of the optimal set cover at a cost of at most $\text{OPT}$. 
Lemma
Let $e_1, e_2, \ldots, e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, $price(e_k) \leq \frac{OPT}{n-k+1}$.

Proof.
1. In each iteration, the remaining elements can be covered by the “leftover” sets of the optimal set cover at a cost of at most $OPT$.
2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most $\frac{OPT}{\bar{C}}$, where $\bar{C} = U - C$. 
Lemma

Let $e_1, e_2, \ldots, e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, $\text{price}(e_k) \leq \frac{\text{OPT}}{(n-k+1)}$.

Proof.

1. In each iteration, the remaining elements can be covered by the “leftover” sets of the optimal set cover at a cost of at most $\text{OPT}$.
2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most $\frac{\text{OPT}}{C}$, where $\bar{C} = U - C$.
3. When $e_k$ was covered $|\bar{C}| \geq (n - k + 1)$. 
Lemma

Let $e_1, e_2, \ldots, e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, $\text{price}(e_k) \leq \frac{\text{OPT}}{(n-k+1)}$.

Proof.

1. In each iteration, the remaining elements can be covered by the “leftover” sets of the optimal set cover at a cost of at most $\text{OPT}$.

2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most $\frac{\text{OPT}}{C}$, where $\tilde{C} = U - C$.

3. When $e_k$ was covered $|\tilde{C}| \geq (n - k + 1)$.

4. Since our covering algorithm is greedy, we have,
Analysis

Lemma

Let $e_1, e_2, \ldots e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, $\text{price}(e_k) \leq \frac{\text{OPT}}{(n-k+1)}$.

Proof.

1. In each iteration, the remaining elements can be covered by the “leftover” sets of the optimal set cover at a cost of at most $\text{OPT}$.

2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most $\frac{\text{OPT}}{\bar{C}}$, where $\bar{C} = U - C$.

3. When $e_k$ was covered $|\bar{C}| \geq (n - k + 1)$.

4. Since our covering algorithm is greedy, we have,

\[
\text{price}(e_k) \leq \frac{\text{OPT}}{(n-k+1)}
\]
Lemma

Let $e_1, e_2, \ldots e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, $\text{price}(e_k) \leq \frac{\text{OPT}}{(n-k+1)}$.

Proof.

1. In each iteration, the remaining elements can be covered by the “leftover” sets of the optimal set cover at a cost of at most $\text{OPT}$.

2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most $\frac{\text{OPT}}{\bar{C}}$, where $\bar{C} = U - C$.

3. When $e_k$ was covered $|\bar{C}| \geq (n - k + 1)$.

4. Since our covering algorithm is greedy, we have,

\[ \text{price}(e_k) \leq \frac{\text{OPT}}{\bar{C}} \]
Analysis

Lemma

Let $e_1, e_2, \ldots e_n$ denote the elements of $U$, in the order in which they were covered. For each $k \in \{1, 2, \ldots, n\}$, $price(e_k) \leq \frac{OPT}{(n-k+1)}$.

Proof.

1. In each iteration, the remaining elements can be covered by the “leftover” sets of the optimal set cover at a cost of at most $OPT$.
2. It follows that there is at least one set among the leftover sets with a cost-effectiveness of at most $\frac{OPT}{\bar{C}}$, where $\bar{C} = U - C$.
3. When $e_k$ was covered $|\bar{C}| \geq (n - k + 1)$.
4. Since our covering algorithm is greedy, we have,

$$
price(e_k) \leq \frac{OPT}{\bar{C}} = \frac{OPT}{(n-k+1)}.
$$
The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$.

The lemma follows, since $\sum_{k=1}^{n} \text{price}(e_k) \leq \sum_{k=1}^{n} \text{OPT}(n-k+1) = \text{OPT} \cdot \left(1 + \frac{1}{2} + \ldots + \frac{1}{n}\right) = H_n \cdot \text{OPT}$. 

Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

...
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$. The lemma follows, since $\sum_{k=1}^{n} \text{price}(e_k) \leq \sum_{k=1}^{n} \text{OPT}(n-k+1) = H_n \cdot \text{OPT}$. 
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$.
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$. The lemma follows, since
Lemma

The greedy algorithm is an \( H_n \) factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to \( \sum_{k=1}^{n} \text{price}(e_k) \).

The lemma follows, since

\[
\sum_{k=1}^{n} \text{price}(e_k) \leq \sum_{k=1}^{n-1} \text{price}(e_k) \leq \text{OPT} \cdot H_n.
\]
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$.

The lemma follows, since

$$\sum_{k=1}^{n} \text{price}(e_k) \leq \sum_{k=1}^{n} \frac{OPT}{(n-k+1)}$$
The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} price(e_k)$.

The lemma follows, since

$$\sum_{k=1}^{n} price(e_k) \leq \sum_{k=1}^{n} \frac{OPT}{(n - k + 1)}$$

$$= H_n \cdot OPT$$
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} price(e_k)$.

The lemma follows, since

$$
\sum_{k=1}^{n} price(e_k) \leq \sum_{k=1}^{n} \frac{OPT}{(n-k+1)} \\
= OPT \cdot \left( \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n} \right)
$$
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$.

The lemma follows, since

$$\sum_{k=1}^{n} \text{price}(e_k) \leq \sum_{k=1}^{n} \frac{OPT}{(n-k+1)}$$

$$= OPT \cdot \left( \frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n} \right)$$

$$=$$
Lemma

The greedy algorithm is an $H_n$ factor approximation algorithm for set cover.

Proof.

1. The cost of each set is distributed among the new elements covered.
2. It follows that the total cost of the set cover picked is equal to $\sum_{k=1}^{n} \text{price}(e_k)$.

The lemma follows, since

$$\sum_{k=1}^{n} \text{price}(e_k) \leq \sum_{k=1}^{n} \frac{\text{OPT}}{(n-k+1)}$$

$$= \text{OPT} \cdot \left(\frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n}\right)$$

$$= H_n \cdot \text{OPT}.$$
Formulating the Integer Program

IP formulation

\[
\min \sum_{S \in S} P_c(S) \cdot x_S \quad \text{subject to} \quad \sum_{S : e \in S} x_S \geq 1, \quad e \in U \quad x_S \in \{0, 1\}, \quad S \in SP
\]
Formulating the Integer Program

**IP formulation**

\[
\min \sum_{S \in \mathcal{P}} c(S) \cdot x_S
\]
Formulating the Integer Program

\[
\begin{align*}
\text{IP formulation} & \\
\min & \sum_{S \in \mathcal{P}} c(S) \cdot x_S \\
\text{subject to} & \quad \sum_{S : e \in S} x_S \geq 1, \quad e \in U
\end{align*}
\]
Formulating the Integer Program

\[
\begin{align*}
\text{min} & \quad \sum_{S \in S_P} c(S) \cdot x_S \\
\text{subject to} & \quad \sum_{S: e \in S} x_S \geq 1, \quad e \in U \\
& \quad x_S \in \{0, 1\}, \quad S \in S_P
\end{align*}
\]
The Linear Program relaxation

\[
\begin{align*}
\min_{S \in S} & \quad \sum_{e \in S} P_c(S) \cdot x_S \\
\text{subject to} & \quad \sum_{S : e \in S} x_S \geq 1, \\
& \quad x_S \geq 0, \\
& \quad S \in S_P
\end{align*}
\]

Example

Let \( U = \{ e, f, g \} \) and the specified sets be \( S_1 = \{ e, f \} \), \( S_2 = \{ f, g \} \) and \( S_3 = \{ e, g \} \), each of unit cost. Optimal integral cover is 2, whereas optimal fractional cover is \( \frac{3}{2} \).
The Linear Program relaxation

Example

Let $U = \{e, f, g\}$ and the specified sets be $S_1 = \{e, f\}$, $S_2 = \{f, g\}$ and $S_3 = \{e, g\}$, each of unit cost. Optimal integral cover is 2, whereas optimal fractional cover is $\frac{3}{2}$. 
The Linear Program relaxation

\[
\min \sum_{S \in \mathcal{P}} c(S) \cdot x_S
\]
The Linear Program relaxation

\[
\begin{align*}
\text{min} & \quad \sum_{S \in \mathcal{P}} c(S) \cdot x_S \\
\text{subject to} & \quad \sum_{S : e \in S} x_S \geq 1, \quad e \in U
\end{align*}
\]
The Linear Program relaxation

\[
\min \sum_{S \in S_P} c(S) \cdot x_S \\
\text{subject to } \sum_{S : e \in S} x_S \geq 1, \quad e \in U \\
\quad x_S \geq 0, \quad S \in S_P
\]

Example

Let \( U = \{ e, f, g \} \) and the specified sets be \( S_1 = \{ e, f \} \), \( S_2 = \{ f, g \} \) and \( S_3 = \{ e, g \} \), each of unit cost. Optimal integral cover is 2, whereas optimal fractional cover is \( \frac{3}{2} \).
The Linear Program relaxation

\[
\begin{align*}
\min & \sum_{S \in \mathcal{S}_P} c(S) \cdot x_S \\
\text{subject to} & \sum_{S : e \in S} x_S \geq 1, \quad e \in U \\
& x_S \geq 0, \quad S \in \mathcal{S}_P
\end{align*}
\]

**Example**

Let \( U = \{e, f, g\} \) and the specified sets be \( S_1 = \{e, f\}, S_2 = \{f, g\} \) and \( S_3 = \{e, g\} \), each of unit cost.
The Linear Program relaxation

\[ \min \sum_{S \in S_P} c(S) \cdot x_S \]
subject to
\[ \sum_{S: e \in S} x_S \geq 1, \quad e \in U \]
\[ x_S \geq 0, \quad S \in S_P \]

**Example**
Let \( U = \{e, f, g\} \) and the specified sets be \( S_1 = \{e, f\} \), \( S_2 = \{f, g\} \) and \( S_3 = \{e, g\} \), each of unit cost. Optimal integral cover is 2, whereas optimal fractional cover is \( \frac{3}{2} \).
The dual of the relaxation
The dual of the relaxation

\[
\max \sum_{e \in U} y_e \\
\text{subject to} \sum_{e \in S} y_e \leq c(S), \quad S \in \mathcal{S} \\
y_e \geq 0, \quad e \in U
\]
The dual of the relaxation

\[
\max \sum_{e \in U} y_e
\]
The dual of the relaxation

\[
\text{max } \sum_{e \in U} y_e \\
\text{subject to } \sum_{e : e \in S} y_e \leq c(S), \quad S \in S_P
\]
The dual of the relaxation

\[ \begin{align*} 
\text{max} \quad & \sum_{e \in U} y_e \\
\text{subject to} \quad & \sum_{e \in S} y_e \leq c(S), \quad S \in S_P \\
& y_e \geq 0, \quad e \in U 
\end{align*} \]
Understanding the dual

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $\text{OPT}_D = \text{OPT}_f \leq \text{OPT}$.
4. The cost of any dual feasible solution is a lower bound on $\text{OPT}_f$ and hence on $\text{OPT}$.
5. A good guess for dual values is $y_i = \text{price}(e_i)$. Unfortunately, this solution is not dual feasible. (Homework!) A better guess is $y_i = \text{price}(e_i) H_n$. 
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $\text{OPT}^{D} = \text{OPT}^{f} \leq \text{OPT}$.
4. The cost of any dual feasible solution is a lower bound on $\text{OPT}^{f}$ and hence on $\text{OPT}$.
5. A good guess for dual values is $y_{i} = \text{price}(e_{i})$. Unfortunately, this solution is not dual feasible. (Homework!) A better guess is $y_{i} = \text{price}(e_{i}) H_{n}$. 

Dual Fitting
Dual-Fitting based Analysis of Greedy Algorithm
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
Understanding the dual

**Note**

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $\text{OPT}_D = \text{OPT}_f \leq \text{OPT}$.
Understanding the dual

**Note**

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $\text{OPT}_D = \text{OPT}_f \leq \text{OPT}$.
4. The cost of any dual feasible solution is a lower bound on $\text{OPT}_f$ and hence on $\text{OPT}$. 
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $OPT_D = OPT_f \leq OPT$.
4. The cost of any dual feasible solution is a lower bound on $OPT_f$ and hence on $OPT$.
5. A good guess for dual values is
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $OPT_D = OPT_f \leq OPT$.
4. The cost of any dual feasible solution is a lower bound on $OPT_f$ and hence on $OPT$.
5. A good guess for dual values is $y_i = price(e_i)$. 
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $\text{OPT}_D = \text{OPT}_f \leq \text{OPT}$.
4. The cost of any dual feasible solution is a lower bound on $\text{OPT}_f$ and hence on $\text{OPT}$.
5. A good guess for dual values is $y_i = \text{price}(e_i)$. Unfortunately, this solution is not dual feasible. (Homework!)
Understanding the dual

Note

1. The primal LP is a covering LP; the dual is a packing LP.
2. In the dual, the goal is to assign weights to elements of sets, such that no set is overpacked.
3. Observe that $\text{OPT}_D = \text{OPT}_f \leq \text{OPT}$.
4. The cost of any dual feasible solution is a lower bound on $\text{OPT}_f$ and hence on $\text{OPT}$.
5. A good guess for dual values is $y_i = \text{price}(e_i)$. Unfortunately, this solution is not dual feasible. (Homework!) A better guess is $y_i = \frac{\text{price}(e_i)}{H_n}$. 
The vector $y$ defined by $y_i = \text{price}(e_i)$ is dual feasible.

Proof. We will show that no set is overpacked by $y$.

1. Pick an arbitrary set $S \in \mathcal{S}$ with $k$ elements.

2. Number the elements of $S$ as $e_1, e_2, \ldots, e_k$ in the order that they were covered by the greedy algorithm.

3. Consider the iteration in which $e_i$ was covered. At this juncture, $S$ contains at least $(k-i+1)$ elements.

4. Thus, in the current iteration, $S$ itself can cover $e_i$ at an average cost of $c(S)(k-i+1)$.

5. Since our algorithm was greedy, $\text{price}(e_i) \leq c(S)(k-i+1)$.

6. Thus, $y_i \leq \frac{1}{H_n} \cdot c(S)(k-i+1)$. 

Analysis
The vector $y$ defined by $y_i = \text{price}(e_i)H_n$ is dual feasible.

Proof.

1. Pick an arbitrary set $S \in \mathcal{S}$ with $k$ elements.
2. Number the elements of $S$ as $e_1, e_2, ..., e_k$ in the order that they were covered by the greedy algorithm.
3. Consider the iteration in which $e_i$ was covered. At this juncture, $S$ contains at least $(k - i + 1)$ elements.
4. Thus, in the current iteration, $S$ itself can cover $e_i$ at an average cost of $c(S)(k - i + 1)$.
5. Since our algorithm was greedy, $\text{price}(e_i) \leq c(S)(k - i + 1)$.
6. Thus, $y_i \leq 1H_n \cdot c(S)(k - i + 1)$. 

The vector \( y \) defined by \( y_i = \frac{\text{price}(e_i)}{H_n} \) is dual feasible.
Analysis

Lemma

The vector \( y \) defined by \( y_i = \frac{\text{price}(e_i)}{H_n} \) is dual feasible.

Proof.

We will show that no set is overpacked by \( y \).
Analysis

Lemma

The vector $y$ defined by $y_i = \frac{\text{price}(e_i)}{H_n}$ is dual feasible.

Proof.

We will show that no set is overpacked by $y$. 


The vector $y$ defined by $y_i = \frac{\text{price}(e_i)}{H_n}$ is dual feasible.

Proof.
We will show that no set is overpacked by $y$.

1. Pick an arbitrary set $S \in S_P$ with $k$ elements.
Analysis

Lemma

The vector $y$ defined by $y_i = \frac{\text{price}(e_i)}{H_n}$ is dual feasible.

Proof.

We will show that no set is overpacked by $y$.

1. Pick an arbitrary set $S \in S_P$ with $k$ elements.
2. Number the elements of $S$ as $e_1, e_2, \ldots e_k$ in the order that they were covered by the greedy algorithm.
Lemma

The vector \( \mathbf{y} \) defined by

\[
y_i = \frac{\text{price}(e_i)}{H_n}
\]

is dual feasible.

Proof.

We will show that no set is overpacked by \( \mathbf{y} \).

1. Pick an arbitrary set \( S \in S_P \) with \( k \) elements.
2. Number the elements of \( S \) as \( e_1, e_2, \ldots, e_k \) in the order that they were covered by the greedy algorithm.
3. Consider the iteration in which \( e_i \) was covered.
Lemma

The vector \( y \) defined by \( y_i = \frac{\text{price}(e_i)}{H_n} \) is dual feasible.

Proof.

We will show that no set is overpacked by \( y \).

1. Pick an arbitrary set \( S \in S_P \) with \( k \) elements.
2. Number the elements of \( S \) as \( e_1, e_2, \ldots e_k \) in the order that they were covered by the greedy algorithm.
3. Consider the iteration in which \( e_i \) was covered. At this juncture, \( S \) contains at least \( (k - i + 1) \) elements.
**Lemma**

The vector $y$ defined by $y_i = \frac{\text{price}(e_i)}{H_n}$ is dual feasible.

**Proof.**

We will show that no set is overpacked by $y$.

1. Pick an arbitrary set $S \in S_P$ with $k$ elements.
2. Number the elements of $S$ as $e_1, e_2, \ldots e_k$ in the order that they were covered by the greedy algorithm.
3. Consider the iteration in which $e_i$ was covered. At this juncture, $S$ contains at least $(k - i + 1)$ elements.
4. Thus, in the current iteration, $S$ itself can cover $e_i$ at an average cost of
Lemma

The vector $y$ defined by $y_i = \frac{\text{price}(e_i)}{H_n}$ is dual feasible.

Proof.

We will show that no set is overpacked by $y$.

1. Pick an arbitrary set $S \in S_P$ with $k$ elements.
2. Number the elements of $S$ as $e_1, e_2, \ldots e_k$ in the order that they were covered by the greedy algorithm.
3. Consider the iteration in which $e_i$ was covered. At this juncture, $S$ contains at least $(k - i + 1)$ elements.
4. Thus, in the current iteration, $S$ itself can cover $e_i$ at an average cost of $\frac{c(S)}{(k-i+1)}$. 
Lemma

The vector \( y \) defined by \( y_i = \frac{\text{price}(e_i)}{H_n} \) is dual feasible.

Proof.

We will show that no set is overpacked by \( y \).

1. Pick an arbitrary set \( S \in S_P \) with \( k \) elements.
2. Number the elements of \( S \) as \( e_1, e_2, \ldots e_k \) in the order that they were covered by the greedy algorithm.
3. Consider the iteration in which \( e_i \) was covered. At this juncture, \( S \) contains at least \( (k - i + 1) \) elements.
4. Thus, in the current iteration, \( S \) itself can cover \( e_i \) at an average cost of \( \frac{c(S)}{(k-i+1)} \).
5. Since our algorithm was greedy, \( \text{price}(e_i) \leq \frac{c(S)}{(k-i+1)} \).
Analysis

Lemma

The vector \( \mathbf{y} \) defined by \( y_i = \frac{\text{price}(e_i)}{H_n} \) is dual feasible.

Proof.

We will show that no set is overpacked by \( \mathbf{y} \).

1. Pick an arbitrary set \( S \in S_P \) with \( k \) elements.
2. Number the elements of \( S \) as \( e_1, e_2, \ldots e_k \) in the order that they were covered by the greedy algorithm.
3. Consider the iteration in which \( e_i \) was covered. At this juncture, \( S \) contains at least \( (k - i + 1) \) elements.
4. Thus, in the current iteration, \( S \) itself can cover \( e_i \) at an average cost of \( \frac{c(S)}{(k - i + 1)} \).
5. Since our algorithm was greedy, \( \text{price}(e_i) \leq \frac{c(S)}{(k - i + 1)} \).
6. Thus, \( y_i \leq \frac{1}{H_n} \cdot \frac{c(S)}{(k - i + 1)} \).
It follows that:

\[ k \sum_{i=1}^{y} e_i \leq c(S) H_n \cdot \left( \frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{1} \right) = c(S) H_k H_n \leq c(S). \]
It follows that:
Analysis (contd.)

Proof (contd.)

It follows that:

\[ \sum_{i=1}^{k} y_{e_i} \leq \]

\[ \sum_{i=1}^{k} y_{e_i} \leq \]

\[ c(S) \cdot (1 + 1 - \ldots + 1) \leq c(S). \]
Analysis (contd.)

Proof (contd.)

It follows that:

$$\sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H_n} \cdot \left( \frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{1} \right)$$
It follows that:

\[ \sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H_n} \cdot \left( \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1} \right) \]

= 
It follows that:

\[ \sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H_n} \cdot \left( \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1} \right) \]

\[ = \frac{H_k}{H_n} \cdot c(S) \]
It follows that:

\[ \sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H_n} \cdot \left( \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{1} \right) \]

\[ = \frac{H_k}{H_n} \cdot c(S) \]
It follows that:

$$\sum_{i=1}^{k} y_{e_i} \leq \frac{c(S)}{H_n} \cdot \left( \frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{1} \right)$$

$$= \frac{H_k}{H_n} \cdot c(S)$$

$$\leq c(S).$$
Approximation Guarantee

Lemma

The approximation guarantee of the greedy set cover algorithm is $H_n$.

Proof.

The cost of the set cover picked is:

$$\sum_{e \in U} \text{price}(e) = H_n \cdot \left( \sum_{e \in U} y_e \right)$$

$$\leq H_n \cdot \text{OPT}$$

(Why?)

$$\leq H_n \cdot \text{OPT}$$
Approximation Guarantee

Lemma

The approximation guarantee of the greedy set cover algorithm is $H_n$. 

Proof. 
The cost of the set cover picked is: 
$$ \sum_{e \in U} \text{price}(e) = H_n \cdot \left( \sum_{e \in U} y_e \right) \leq H_n \cdot \text{OPT}(Why?) \leq H_n \cdot \text{OPT} $$
Approximation Guarantee

Lemma

The approximation guarantee of the greedy set cover algorithm is $H_n$.

Proof.

The cost of the set cover picked is:

$$\sum_{e \in U} \text{price}(e) = H_n \cdot \left( \sum_{e \in U} y_e \right) \leq H_n \cdot \text{OPT}$$
Lemma

The approximation guarantee of the greedy set cover algorithm is $H_n$.

Proof.

The cost of the set cover picked is:

$$
\sum_{e \in U} \text{price}(e) =
$$
**Lemma**

The approximation guarantee of the greedy set cover algorithm is $H_n$.

**Proof.**

The cost of the set cover picked is:

$$\sum_{e \in U} \text{price}(e) = H_n \cdot \left( \sum_{e \in U} y_e \right)$$
Dual-Fitting based Analysis of Greedy Algorithm

Approximation Guarantee

Lemma

The approximation guarantee of the greedy set cover algorithm is $H_n$.

Proof.

The cost of the set cover picked is:

$$\sum_{e \in U} \text{price}(e) = H_n \cdot \left( \sum_{e \in U} y_e \right) \leq H_n \cdot OPT_f$$
The approximation guarantee of the greedy set cover algorithm is $H_n$.

Proof.
The cost of the set cover picked is:

$$\sum_{e \in U} \text{price}(e) = H_n \cdot \left( \sum_{e \in U} y_e \right) \leq H_n \cdot OPT_f \quad (\text{Why?})$$
Approximation Guarantee

Lemma

The approximation guarantee of the greedy set cover algorithm is $H_n$.

Proof.

The cost of the set cover picked is:

$$\sum_{e \in U} \text{price}(e) = H_n \cdot \left( \sum_{e \in U} y_e \right) \leq H_n \cdot \text{OPT}_f \quad \text{(Why?)}$$

$$\leq H_n \cdot \text{OPT}$$
The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let \( n = 2^k - 1 \), where \( k \) is a positive integer.

2. Let \( U = \{e_1, e_2, \ldots, e_n\} \).

3. For \( 1 \leq i \leq n \), consider \( i \) as a \( k \)-bit number. This number is a \( k \)-dimensional vector over \( \mathbb{GF}_2 \).

4. Let \( i \) denote this vector.

5. Let \( S = \{S_1, S_2, \ldots, S_n\} \) and let \( c(S) = 1 \), for all \( S \in S \).

Observations

1. Each set contains \( n + 1 = 2^k - 1 \) elements.

2. Each element is contained in \( n + 1 = 2^k - 1 \) sets.

3. Thus, \( x_i = 2n + 1 \), \( 1 \leq i \leq n \) is a fractional set cover (optimal) of cost \( 2n + 1 \).
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation.
The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$. 
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. 
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let \( n = 2^k - 1 \), where \( k \) is a positive integer.
2. Let \( U = \{e_1, e_2, \ldots, e_n\} \).
3. For \( 1 \leq i \leq n \), consider \( i \) as a \( k \)-bit number. This number is a \( k \)-dimensional vector over \( GF[2] \). Let \( i \) denote this vector.
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. Let $i$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j | i \cdot j = 1\}$. 
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. Let $i$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j | i \cdot j = 1\}$.
5. Let $S_P = \{S_1, S_2, \ldots, S_n\}$ and let $c(S) = 1$, for all $S \in S_P$. 

Observations

1. Each set contains $n + 1 = 2^k - 1$ elements.
2. Each element is contained in $n + 1$ sets.
3. Thus, $x_i = 2n + 1$, $1 \leq i \leq n$ is a fractional set cover (optimal) of cost $2n$. 
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. Let $i$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j | i \cdot j = 1\}$.
5. Let $S_P = \{S_1, S_2, \ldots, S_n\}$ and let $c(S) = 1$, for all $S \in S_P$.

Observations

1. Each set contains $n + 1 = 2^k - 1$ elements.
2. Each element is contained in $n + 1$ sets.
3. Thus, $x_i = 2n - 1$, $1 \leq i \leq n$ is a fractional set cover (optimal) of cost $2n - 1$. 
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. Let $i$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j | i \cdot j = 1\}$.
5. Let $S_P = \{S_1, S_2, \ldots S_n\}$ and let $c(S) = 1$, for all $S \in S_P$.

Observations

1. Each set contains
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. Let $i$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \mid i \cdot j = 1\}$.
5. Let $S_P = \{S_1, S_2, \ldots S_n\}$ and let $c(S) = 1$, for all $S \in S_P$.

Observations

1. Each set contains $\frac{n+1}{2} = 2^{k-1}$ elements.
Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. Let $i$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j | i \cdot j = 1\}$.
5. Let $S_P = \{S_1, S_2, \ldots S_n\}$ and let $c(S) = 1$, for all $S \in S_P$.

Observations

1. Each set contains $\frac{n+1}{2} = 2^{k-1}$ elements.
2. Each element is contained in $\frac{n+1}{2}$ sets.
Example

The above approximation guarantee cannot be improved with this integer programming formulation. Consider the following instance:

1. Let $n = 2^k - 1$, where $k$ is a positive integer.
2. Let $U = \{e_1, e_2, \ldots, e_n\}$.
3. For $1 \leq i \leq n$, consider $i$ as a $k$-bit number. This number is a $k$-dimensional vector over $GF[2]$. Let $i$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j | i \cdot j = 1\}$.
5. Let $S_P = \{S_1, S_2, \ldots S_n\}$ and let $c(S) = 1$, for all $S \in S_P$.

Observations

1. Each set contains $\frac{n+1}{2} = 2^{k-1}$ elements.
2. Each element is contained in $\frac{n+1}{2}$ sets.
3. Thus, $x_i = \frac{2}{n+1}$, $1 \leq i \leq n$ is a fractional set cover (optimal) of cost $\frac{2\cdot n}{n+1}$.
Lemma

Any integral cover must pick at least $k$ of the above $n$ sets.

Proof.

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2, ..., i_p$ denote the indices of these sets.

2. Let $A$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $i_1, i_2, ..., i_p$ respectively.

3. The dimension of the null-space of $A$ is at least 1. (Why?) Rank of $A$ is less than $k$!

4. The null-space of $A$ contains a vector $j$.

5. Since $A \cdot j = 0$, the element $e_j$ is not in any of the $p$ sets.

6. Hence, the $p$ sets do not form a cover.
Lemma

Any integral cover must pick at least $k$ of the above $n$ sets.
Lemma

Any integral cover must pick at least \( k \) of the above \( n \) sets.

Proof.
Lemma

Any integral cover must pick at least $k$ of the above $n$ sets.

Proof.

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2 \ldots i_p$ denote the indices of these sets.
Lemma

*Any integral cover must pick at least k of the above n sets.*

Proof.

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2 \ldots i_p$ denote the indices of these sets.

2. Let $A$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $i_1, i_2, \ldots i_p$ respectively.
Lemma

Any integral cover must pick at least \( k \) of the above \( n \) sets.

Proof.

1. Consider the union of some \( p \) sets, where \( p < k \). Let \( i_1, i_2, \ldots, i_p \) denote the indices of these sets.
2. Let \( A \) be a \( p \times k \) matrix over \( GF[2] \), whose rows consist of \( i_1, i_2, \ldots, i_p \) respectively.
3. The dimension of the null-space of \( A \) is at least 1.
Lemma

*Any integral cover must pick at least k of the above n sets.*

Proof.

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2, \ldots i_p$ denote the indices of these sets.
2. Let $A$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $i_1, i_2, \ldots i_p$ respectively.
3. The dimension of the null-space of $A$ is at least 1. (Why?)
Tightness Analysis (contd.)

**Lemma**

*Any integral cover must pick at least $k$ of the above $n$ sets.*

**Proof.**

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2, \ldots, i_p$ denote the indices of these sets.

2. Let $A$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $i_1, i_2, \ldots, i_p$ respectively.

3. The dimension of the null-space of $A$ is at least 1. (Why?) Rank of $A$ is less than $k!$. 
Tightness Analysis (contd.)

Lemma

*Any integral cover must pick at least k of the above n sets.*

Proof.

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2, \ldots, i_p$ denote the indices of these sets.
2. Let $A$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $i_1, i_2, \ldots, i_p$ respectively.
3. The dimension of the null-space of $A$ is at least 1. (Why?) Rank of $A$ is less than $k$!
4. The null-space of $A$ contains a vector $j$. 
Tightness Analysis (contd.)

Lemma

*Any integral cover must pick at least $k$ of the above $n$ sets.*

Proof.

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2 \ldots i_p$ denote the indices of these sets.
2. Let $A$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $i_1, i_2, \ldots i_p$ respectively.
3. The dimension of the null-space of $A$ is at least 1. (Why?) Rank of $A$ is less than $k$!
4. The null-space of $A$ contains a vector $j$.
5. Since $A \cdot j = 0$, the element $e_j$ is not in any of the $p$ sets.
Lemma

Any integral cover must pick at least $k$ of the above $n$ sets.

Proof.

1. Consider the union of some $p$ sets, where $p < k$. Let $i_1, i_2 \ldots i_p$ denote the indices of these sets.
2. Let $A$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $i_1, i_2, \ldots i_p$ respectively.
3. The dimension of the null-space of $A$ is at least 1. (Why?) Rank of $A$ is less than $k$!
4. The null-space of $A$ contains a vector $j$.
5. Since $A \cdot j = 0$, the element $e_j$ is not in any of the $p$ sets.
6. Hence, the $p$ sets do not form a cover.
Conclusion of Analysis

Lemma

The integrality gap of the IP formulation of set cover discussed above is more than $\log_2 n^2$.

Proof. The previous lemma established that any integral set cover has cost at least $k = \log_2 (n+1)$. It follows that the lower bound on the integrality gap established by this example is $\frac{k}{2} \cdot n \cdot (n+1) = \frac{n+1}{2} \cdot n \cdot \log_2 (n+1) > \log_2 n^2$. 

Lemma

The integrality gap of the IP formulation of set cover discussed above is more than \( \frac{\log_2 n}{2} \).
The integrality gap of the IP formulation of set cover discussed above is more than $\frac{\log_2 n}{2}$.

Proof.

The previous lemma established that any integral set cover has cost at least $k = \log_2 (n + 1)$. It follows that the lower bound on the integrality gap established by this example is $\frac{k}{2} \cdot n (n + 1) = n + 1 \cdot 2 \cdot n \cdot \log_2 (n + 1) > \log_2 n^2$. 

Conclusion of Analysis
The integrality gap of the IP formulation of set cover discussed above is more than $\frac{\log_2 n}{2}$.

Proof.

The previous lemma established that any integral set cover has cost at least $k = \log_2(n + 1)$. 
Conclusion of Analysis

Lemma

The integrality gap of the IP formulation of set cover discussed above is more than $\frac{\log_2 n}{2}$.

Proof.

The previous lemma established that any integral set cover has cost at least $k = \log_2(n + 1)$. It follows that the lower bound on the integrality gap established by this example is
Lemma

The integrality gap of the IP formulation of set cover discussed above is more than \( \frac{\log_2 n}{2} \).

Proof.

The previous lemma established that any integral set cover has cost at least \( k = \log_2(n + 1) \). It follows that the lower bound on the integrality gap established by this example is

\[
\frac{OPT_I}{OPT_f} = \log_2(n + 1) \]
Conclusion of Analysis

Lemma

The integrality gap of the IP formulation of set cover discussed above is more than $\frac{\log_2 n}{2}$.

Proof.

The previous lemma established that any integral set cover has cost at least $k = \log_2 (n + 1)$. It follows that the lower bound on the integrality gap established by this example is

$$\frac{OPT_I}{OPT_f} = \frac{k}{\frac{2 \cdot n}{n+1}}$$
Conclusion of Analysis

**Lemma**

The integrality gap of the IP formulation of set cover discussed above is more than \( \frac{\log_2 n}{2} \).

**Proof.**

The previous lemma established that any integral set cover has cost at least \( k = \log_2 (n + 1) \). It follows that the lower bound on the integrality gap established by this example is

\[
\frac{OPT_I}{OPT_f} = \frac{k}{\frac{2 \cdot n}{n+1}}
\]

\[
= \frac{n + 1}{2 \cdot n} \cdot \log_2 (n + 1)
\]
The integrality gap of the IP formulation of set cover discussed above is more than $\frac{\log_2 n}{2}$.

Proof.

The previous lemma established that any integral set cover has cost at least $k = \log_2(n + 1)$. It follows that the lower bound on the integrality gap established by this example is

\[
\frac{OPT_I}{OPT_f} = \frac{k}{\frac{2 \cdot n}{(n+1)}} = \frac{n + 1}{2 \cdot n} \cdot \log_2(n + 1) > \frac{\log_2 n}{2}
\]