Set-Cover approximation through Dual Fitting

K. Subramani[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

March 25, 2014

Outline

Outline

Outline

Preliminaries

Preliminaries

The Set Cover Problem

Given,

Preliminaries

The Set Cover Problem

Given,

Preliminaries

The Set Cover Problem

Given,

1. A ground set $U = \{e_1, e_2, \ldots, e_n\}$,

Preliminaries

The Set Cover Problem

Given,

1. A ground set $U = \{e_1, e_2, \ldots, e_n\}$,
2. A collection of sets $S_P = \{S_1, S_2, \ldots S_m\}$, $S_i \subseteq U$, $i = 1, 2, \ldots, m$

Preliminaries

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 $\mathbf{c} : S_i \to Z_+$,

Preliminaries

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 $\mathbf{c} : S_i \to Z_+$,

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

Preliminaries

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 $\mathbf{c} : S_i \to Z_+$,

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),

Preliminaries

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 $\mathbf{c} : S_i \to Z_+$,

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)

The Greedy Algorithm (Cardinality)

Greedy Approach

The Greedy Algorithm (Cardinality)

Greedy Approach

The Greedy Algorithm (Cardinality)

Greedy Approach

1. Cover $C = \emptyset$.

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

Analysis of the greedy approach

Analysis

Analysis of the greedy approach

Analysis

Analysis of the greedy approach

Analysis

1. 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.

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?)

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.

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?)

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

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?

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.

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.

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.

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!

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

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.)

Analysis (contd.)

Final steps

Analysis (contd.)

Final steps

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

Analysis (contd.)

Final steps

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

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

Analysis (contd.)

Final steps

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

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

Analysis (contd.)

Final steps

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

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

Analysis (contd.)

Final steps

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

$$
\begin{aligned}
n \cdot (1 - \frac{1}{OPT})^{OPT \cdot \ln n} &< n \cdot (\frac{1}{e})^{\ln n} \\
&= n \cdot e^{-\ln n} \\
&= n \cdot e^{\ln n^{-1}}
\end{aligned}
$$

Analysis (contd.)

Final steps

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

$$
\begin{aligned}
n \cdot (1 - \frac{1}{OPT})^{OPT \cdot \ln n} &< n \cdot (\frac{1}{e})^{\ln n} \\
&= n \cdot e^{-\ln n} \\
&= n \cdot e^{\ln n^{-1}} \\
&= n \cdot n^{-1}
\end{aligned}
$$

Analysis (contd.)

Final steps

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

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

Analysis (contd.)

Final steps

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

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

i.e., we are done.

Analysis (contd.)

Final steps

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

$$\begin{aligned} n \cdot (1 - \frac{1}{OPT})^{OPT \cdot \ln n} &< n \cdot (\frac{1}{e})^{\ln n} \\ &= n \cdot e^{-\ln n} \\ &= n \cdot e^{\ln n^{-1}} \\ &= n \cdot n^{-1} \\ &= 1 \end{aligned}$$

   i.e., we are done.

2. What we have shown is that the greedy strategy finds a solution in $OPT \cdot \ln n$ iterations.

Analysis (contd.)

Final steps

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

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

   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)

The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

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. $C \to \emptyset$.
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**

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.

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 α_S denote the cost-effectiveness of S.

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 α_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 α_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 $price(e) = \alpha_S$.

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 α_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. $C \rightarrow C \cup S$.

The Greedy Algorithm (Weighted)

Weighted Greedy Algorithm

1. $C \to \emptyset$.
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 α_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. $C \to C \cup S$.
10. **end while**

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 α_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.

Analysis

Analysis

Lemma

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

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* $k \in \{1, 2, \ldots, n\}$,

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 $k \in \{1, 2, \ldots, n\}$, $price(e_k) \leq \frac{OPT}{(n-k+1)}$.

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.

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.

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 $\bar{C} = U - C$.

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 $|\bar{C}| \geq (n - k + 1)$.

Analysis

Lemma

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

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,

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,

$$price(e_k) \quad \leq$$

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,

$$price(e_k) \quad \leq \quad \frac{OPT}{\bar{C}}$$

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,

$$\begin{aligned} price(e_k) &\leq \frac{OPT}{\bar{C}} \\ &= \frac{OPT}{(n-k+1)}. \end{aligned}$$

□

Analysis (contd.)

Analysis (contd.)

Lemma

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

Analysis (contd.)

Lemma

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

Proof.

Analysis (contd.)

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.

Analysis (contd.)

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

Analysis (contd.)

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 $\sum_{k=1}^{n} price(e_k)$.

Analysis (contd.)

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

Analysis (contd.)

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

$$\sum_{k=1}^{n} price(e_k) \quad \leq$$

Analysis (contd.)

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

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

Analysis (contd.)

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

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

Analysis (contd.)

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

$$\begin{aligned}\sum_{k=1}^{n} price(e_k) &\leq \sum_{k=1}^{n} \frac{OPT}{(n-k+1)} \\ &= OPT \cdot (\frac{1}{1} + \frac{1}{2} + \ldots \frac{1}{n})\end{aligned}$$

Analysis (contd.)

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

$$\begin{aligned}\sum_{k=1}^{n} price(e_k) &\leq \sum_{k=1}^{n} \frac{OPT}{(n-k+1)} \\ &= OPT \cdot (\frac{1}{1} + \frac{1}{2} + \ldots \frac{1}{n}) \\ &= \end{aligned}$$

Analysis (contd.)

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

$$
\begin{aligned}
\sum_{k=1}^{n} price(e_k) &\leq \sum_{k=1}^{n} \frac{OPT}{(n-k+1)} \\
&= OPT \cdot (\frac{1}{1} + \frac{1}{2} + \ldots \frac{1}{n}) \\
&= H_n \cdot OPT.
\end{aligned}
$$

□

Formulating the Integer Program

IP formulation

Formulating the Integer Program

IP formulation

$$\min \sum_{S \in S_P} c(S) \cdot x_S$$

Formulating the Integer Program

IP formulation

$$\begin{array}{lll} & \min \sum_{S \in \mathcal{S}_P} c(S) \cdot x_S & \\ \text{subject to} & \sum_{S\,:\,e \in S} x_S \geq 1, & e \in U \end{array}$$

Formulating the Integer Program

IP formulation

$$
\begin{array}{llr}
 & \min \sum_{S \in S_P} c(S) \cdot x_S & \\
\text{subject to} & \sum_{S\,:\,e \in S} x_S \geq 1, & e \in U \\
 & x_S \in \{0, 1\}, & S \in S_P
\end{array}
$$

The Linear Program relaxation

The Linear Program relaxation

Relaxation

The Linear Program relaxation

Relaxation

$$\min \sum_{S \in S_P} c(S) \cdot x_S$$

The Linear Program relaxation

Relaxation

$$\begin{array}{lll} & \min \sum_{S \in S_P} c(S) \cdot x_S & \\ \text{subject to} & \sum_{S:\, e \in S} x_S \geq 1, & e \in U \end{array}$$

The Linear Program relaxation

Relaxation

$$\begin{array}{lll} & \min \sum_{S \in S_P} c(S) \cdot x_S & \\ \text{subject to} & \sum_{S:\, e \in S} x_S \geq 1, & e \in U \\ & x_S \geq 0, & S \in S_P \end{array}$$

The Linear Program relaxation

Relaxation

$$
\begin{array}{lll}
 & \min \sum_{S \in S_P} c(S) \cdot x_S & \\
\text{subject to} & \sum_{S:\, e \in S} x_S \geq 1, & e \in U \\
 & x_S \geq 0, & S \in S_P
\end{array}
$$

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

Relaxation

$$
\begin{array}{lll}
 & \min \sum_{S \in S_P} c(S) \cdot x_S & \\
\text{subject to} & \sum_{S:\, e \in S} x_S \geq 1, & e \in U \\
 & x_S \geq 0, & S \in S_P
\end{array}
$$

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

Dual

The dual of the relaxation

Dual

$$\max \sum_{e \in U} y_e$$

The dual of the relaxation

Dual

$$\max \sum_{e \in U} y_e$$

$$\text{subject to} \quad \sum_{e:\, e \in S} y_e \leq c(S), \quad S \in S_P$$

The dual of the relaxation

Dual

$$\begin{aligned} & \max \textstyle\sum_{e \in U} y_e & \\ \text{subject to} \quad & \textstyle\sum_{e:\, e \in S} y_e \leq c(S), & S \in S_P \\ & y_e \geq 0, & e \in U \end{aligned}$$

Understanding the dual

Understanding the dual

Note

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* $OPT_D = OPT_f \leq OPT$.

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.

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* $OPT_D = OPT_f \leq OPT$.
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* $y_i = price(e_i)$. *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* $OPT_D = OPT_f \leq OPT$*.*
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* $y_i = price(e_i)$*. Unfortunately, this solution is not dual feasible. (Homework!) A better guess is* $y_i = \frac{price(e_i)}{H_n}$*.*

Analysis

Analysis

Lemma

Analysis

Lemma

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

Analysis

Lemma

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

Proof.

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

Analysis

Lemma

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

Proof.

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

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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.

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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.

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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.

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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.

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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 $\frac{c(S)}{(k-i+1)}$.

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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, $price(e_i) \leq \frac{c(S)}{(k-i+1)}$.

Analysis

Lemma

The vector $\mathbf{y}$ *defined by* $y_i = \frac{price(e_i)}{H_n}$ *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, $price(e_i) \leq \frac{c(S)}{(k-i+1)}$.
6. Thus, $y_i \leq \frac{1}{H_n} \cdot \frac{c(S)}{(k-i+1)}$.

□

Analysis (contd.)

Proof (contd.)

Analysis (contd.)

Proof (contd.)

It follows that:

Analysis (contd.)

Proof (contd.)

It follows that:

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

Analysis (contd.)

Proof (contd.)

It follows that:

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

Analysis (contd.)

Proof (contd.)

It follows that:

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

$$=$$

Analysis (contd.)

Proof (contd.)

It follows that:

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

Analysis (contd.)

Proof (contd.)

It follows that:

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

Analysis (contd.)

Proof (contd.)

It follows that:

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

□

Approximation Guarantee

Approximation Guarantee

Lemma

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

Approximation Guarantee

Lemma

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

Proof.

The cost of the set cover picked is:

Approximation Guarantee

Lemma

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

Proof.

The cost of the set cover picked is:

$$\sum_{e \in U} price(e) \quad =$$

Approximation Guarantee

Lemma

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

Proof.

The cost of the set cover picked is:

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

Approximation Guarantee

Lemma

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

Proof.

The cost of the set cover picked is:

$$\begin{aligned}\sum_{e \in U} price(e) &= H_n \cdot \left(\sum_{e \in U} y_e\right) \\ &\leq H_n \cdot OPT_f\end{aligned}$$

Approximation Guarantee

Lemma

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

Proof.

The cost of the set cover picked is:

$$\begin{aligned} \sum_{e \in U} price(e) &= H_n \cdot (\sum_{e \in U} y_e) \\ &\leq H_n \cdot OPT_f \quad \text{(Why?)} \end{aligned}$$

Approximation Guarantee

Lemma

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

Proof.

The cost of the set cover picked is:

$$\begin{aligned}\sum_{e \in U} price(e) &= H_n \cdot \left(\sum_{e \in U} y_e\right) \\ &\leq H_n \cdot OPT_f \quad \text{(Why?)} \\ &\leq H_n \cdot OPT\end{aligned}$$

□

Tightness

Example

Tightness

Example

The above approximation guarantee cannot be improved with this integer programming formulation.

Tightness

Example

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.

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 $U = \{e_1, e_2, \ldots, e_n\}$.

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 $\mathbf{i}$ 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 $\mathbf{i}$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \,|\, \mathbf{i} \cdot \mathbf{j} = \mathbf{1}\}$.

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 $\mathbf{i}$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \mid \mathbf{i} \cdot \mathbf{j} = \mathbf{1}\}$.
5. Let $S_P = \{S_1, S_2, \ldots S_n\}$ and let $c(S) = 1$, for all $S \in S_P$.

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 $\mathbf{i}$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \mid \mathbf{i} \cdot \mathbf{j} = \mathbf{1}\}$.
5. Let $S_P = \{S_1, S_2, \ldots S_n\}$ and let $c(S) = 1$, for all $S \in S_P$.

Observations

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 $\mathbf{i}$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \,|\, \mathbf{i} \cdot \mathbf{j} = \mathbf{1}\}$.
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 $\mathbf{i}$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \,|\, \mathbf{i} \cdot \mathbf{j} = \mathbf{1}\}$.
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 $\mathbf{i}$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \,|\, \mathbf{i} \cdot \mathbf{j} = \mathbf{1}\}$.
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.*

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 $\mathbf{i}$ denote this vector.
4. For $1 \leq i \leq n$, let $S_i = \{e_j \,|\, \mathbf{i} \cdot \mathbf{j} = \mathbf{1}\}$.
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* $\frac{2 \cdot n}{n+1}$.

Tightness Analysis (contd.)

Tightness Analysis (contd.)

Lemma

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

Tightness Analysis (contd.)

Lemma

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

Proof.

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.

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

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 $\mathbf{A}$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $\mathbf{i_1}, \mathbf{i_2}, \ldots \mathbf{i_p}$ respectively.
3. The dimension of the null-space of $\mathbf{A}$ is at least 1.

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 $\mathbf{A}$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $\mathbf{i_1}, \mathbf{i_2}, \ldots \mathbf{i_p}$ respectively.
3. The dimension of the null-space of $\mathbf{A}$ 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 $\mathbf{A}$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $\mathbf{i_1}, \mathbf{i_2}, \ldots \mathbf{i_p}$ respectively.
3. The dimension of the null-space of $\mathbf{A}$ is at least 1. (Why?) Rank of $\mathbf{A}$ 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 $\mathbf{A}$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $\mathbf{i_1}, \mathbf{i_2}, \ldots \mathbf{i_p}$ respectively.
3. The dimension of the null-space of $\mathbf{A}$ is at least 1. (Why?) Rank of $\mathbf{A}$ is less than k!
4. The null-space of $\mathbf{A}$ contains a vector $\mathbf{j}$.

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

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 $\mathbf{A}$ be a $p \times k$ matrix over $GF[2]$, whose rows consist of $\mathbf{i_1}, \mathbf{i_2}, \ldots \mathbf{i_p}$ respectively.
3. The dimension of the null-space of $\mathbf{A}$ is at least 1. (Why?) Rank of $\mathbf{A}$ is less than k!
4. The null-space of $\mathbf{A}$ contains a vector $\mathbf{j}$.
5. Since $\mathbf{A} \cdot \mathbf{j} = \mathbf{0}$, 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

Conclusion of Analysis

Lemma

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

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.

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 $k = \log_2(n+1)$.

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

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

$$\frac{OPT_I}{OPT_f} =$$

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

$$\begin{aligned} \frac{OPT_I}{OPT_f} &= \frac{k}{\frac{2 \cdot n}{(n+1)}} \\ &= \frac{n+1}{2 \cdot n} \cdot \log_2(n+1) \end{aligned}$$

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

$$\begin{aligned}\frac{OPT_I}{OPT_f} &= \frac{k}{\frac{2\cdot n}{(n+1)}}\\ &= \frac{n+1}{2\cdot n}\cdot\log_2(n+1)\\ &> \frac{\log_2 n}{2}\end{aligned}$$

□