# Chapter 2

# The Set Covering Problem

*In this note, we look at the classical set covering problem and analyze the performance of the greedy approximation algorithm in three different ways.*

## 2.1 The problem and its complexity

The set covering problem is a fundamental problem in the class of covering problems. Given a finite set $X$ and a family $\mathcal{F} = S_1, S_2, \ldots, S_n$ of subsets of $X$ (i.e., $S_j \subseteq X, \quad j = 1, \ldots, n$) the set covering problem is to find a minimum cardinality $J \subseteq \{1, \ldots, n\}$ such that $\bigcup_{j \in J} S_j = X$. The elements of $X$ are called *points*. Given a $J \subseteq \{1, \ldots, n\}$, a point is said to be *covered* if it belongs to $\bigcup_{j \in J} S_j$. In the minimum-cost set covering problem, each set $S_j$, $1 \leq j \leq n$, has a cost $c_j$, and the problem is to find a $J \subseteq \{1, \ldots, n\}$ such that each point is covered and $\sum_{j \in J} c_j$ is minimum.

Define the incidence matrix $A$ of a set covering problem as follows. There are $|X|$ rows in $A$, one for each point $x_i \in X$, and $n$ columns in $A$, one for each set $S_j$. The entry $a_{ij}$ of $A$ (the entry at the intersection of the $i$th row and the $j$th column) is one if point $x_i$ is in set $S_j$, otherwise $a_{ij}$ is zero. Figure 2.1 shows two examples of the set covering problem, along with their incidence matrices.

An integer linear programming formulation of the set covering problem is as follows. For each set $S_j$, we have a zero-one variable $s_j$. The intention is that $s_j = 1$ iff set $S_j$ is chosen in the optimal solution. Let $\mathbf{1}^m$ denote a vector with $m$ entries, each of which is 1; it will be clear from the context whether the vector is a row vector or a column vector (either is possible). Let $s$ denote the $n$-element column vector $\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}$.

$$A = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} S_1 & S_2 & S_3 \end{array} \\ \boxed{\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{array}} \end{array}$$

$$A = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{array} \begin{array}{c} \begin{array}{cccccc} S_1 & S_2 & S_3 & S_4 & S_5 & S_6 \end{array} \\ \boxed{\begin{array}{cccccc} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{array}} \end{array}$$

Figure 2.1: Two examples of the set covering problem.

$$\boxed{\begin{array}{lllll} (IP) & \text{minimize} & \displaystyle\sum_{j=1}^{n} s_j & & \\ & \text{subject to} & As & \geq & \mathbf{1}^m \\ & & s_j & \in & \{0,1\}, \qquad (j = 1, 2, \ldots, n). \end{array}}$$

For the minimum-cost set covering problem the objective function becomes

$$\sum_{j=1}^{n} c_j s_j.$$

**Proposition 2.1** *The set covering problem is NP-hard.*

A rough translation of the proposition is that if there exists a polynomial-time algorithm for solving the set covering problem, then there exist polynomial-time algorithms for solving each of the following "hard problems" as well as many other hard problems:

1. integer linear programming,

2. finding Steiner minimal trees,
3. finding a cut that has the maximum number of edges, and
4. the traveling salesman problem.

In other words, showing that a problem is NP-hard gives evidence that it has no polynomial-time algorithm, but no lower bound on the running time required for solving the problem is obtained. The formal proof of the above proposition starts with a problem already known to be NP-hard and shows that each instance of the problem can be transformed in polynomial running time to an instance of the set covering problem such that an optimal solution to the latter instance can be used to find an optimal solution to the former instance. In other words, to prove that the set covering problem is NP-hard, we show that a problem already known to be NP-hard is *polynomially transformable* to the set covering problem. Consequently, if a polynomial-time algorithm were available for solving set covering problems, then it could be used to solve the original NP-hard problem in polynomial time. For many NP-hard problems, the best algorithms known so far are rather inefficient, and amount to searching through all possible solutions of the input instance. Hence, it is commonly assumed that no efficient (i.e., polynomial-time) algorithm will ever be found for an NP-hard problem.

## 2.2 Applications

The set covering problem has many applications. Three are listed here.

(a) Information retrieval:
There are $n$ files $S_1, \ldots, S_n$, where $S_j$ has length $c_j$, and there are $m$ requests for information. Each unit of information is stored in at least one file. Find a subset of the files of minimum total length such that searching these will retrieve all the requested information.

(b) Airline crew scheduling:
An airline has $m$ flights $x_1, \ldots, x_m$. These flights can be combined into "flight legs" $S_1, \ldots, S_n$ such that the same crew can service all the flights in leg $S_j$. Find the minimum number of crews required to service all flights. Note that the number of flight legs may be much larger than the number of crews.

EXAMPLE

| | Leg 1 | Leg 2 | Leg 3 | Leg 4 |
|---|---|---|---|---|
| Toronto Paris | 1 | 0 | 0 | 0 |
| Toronto New York | 0 | 1 | 0 | 0 |
| Toronto Chicago | 0 | 0 | 1 | 1 |
| Chicago Vancouver | 0 | 0 | 1 | 0 |
| Chicago New York | 0 | 0 | 0 | 1 |
| New York Miami | 0 | 1 | 0 | 1 |

(c) Minimum path separator:
An undirected graph $G = (V, E)$ is given, together with $m$ special paths $P_1, P_2, \ldots, P_m$. Here, $m$ may be much greater than $|V|$ or $|E|$. For each edge $e_j$ of $G$, we denote the set of paths

$$\begin{array}{ll}
P_1 = v_2v_3v_4 & S_1 = \{P_2\} \\
P_2 = v_2v_1v_4 & S_2 = \{P_5\} \\
P_3 = v_2v_5v_4 & S_3 = \{P_2\} \\
P_4 = v_3v_4v_5 & S_4 = \{P_5\} \\
P_5 = v_3v_1v_5 & S_5 = \{P_1, P_6\} \\
P_6 = v_3v_2v_5 & S_6 = \{P_1, P_4\} \\
 & S_7 = \{P_3, P_4\} \\
 & S_8 = \{P_3, P_6\}
\end{array}$$

Figure 2.2: Modeling the minimum path separator problem as a set covering problem.

$P_i$, $1 \leq i \leq m$, such that $e_j$ is in $P_i$ by $S_j$. The problem is to find a minimum cardinality set of edges whose deletion from $G$ disconnects each of the paths $P_1, P_2, \ldots, P_m$.

## 2.3 The matrix reduction algorithm for set covering

Since no polynomial-time algorithm is available for finding the optimal solution of set covering problems, it is worthwhile to study heuristics for solving the problem with the goal of obtaining a performance guarantee (or approximation guarantee) on the heuristic. That is, given a simple method that finds feasible but not necessarily optimal solutions, the goal is to find whether the solution returned by the method is within a fixed factor of optimal, for every instance of the problem.

The matrix reduction algorithm is a heuristic for the set covering problem. Given an instance of the problem in the form of the incidence matrix, the algorithm "reduces" the matrix by repeatedly attempting to eliminate rows and columns and in the process the algorithm constructs a $J \subseteq \{1, \ldots, n\}$ such that if $J' \subseteq \{1, \ldots, n\}$ is an optimal solution of the reduced instance, then $J \cup J'$ is an optimal solution of the original instance. The algorithm may find the optimal solution on some instances, since the reduced instance may be trivial and so an optimal solution $J'$ for it is easily found. The algorithm may fail completely on other instances, since it may not succeed in eliminating even one row or column from the original incidence matrix. Thus there is *no* performance guarantee for the matrix reduction algorithm.

---

**Algorithm** Matrix Reduction Algorithm for Set Covering
**input**: Incidence matrix $A$ (with one row for each point $x_i$ in $X$ and one column for each set $S_j$ in $\mathcal{F}$).
**output**: Partial solution $J$ and "reduced" incidence matrix $A'$ such that for any optimal solution $J'$ of $A'$, $J' \cup J$ is an optimal solution of $A$.

STEP 0: (feasibility check)
  **if** $A$ has a row with all entries zero
  **then stop**, no feasible solution exists;
  **end**;

  $J := \emptyset$;

**repeat**
  STEP 1:
    **if** row $i$ has exactly one nonzero entry, say, in column $j$
    **then** $J := J \cup \{j\}$;  ($S_j$ must be chosen in the optimal solution)
      eliminate column $j$, and all rows having a "one" in that column;
    **end**;
  STEP 2:
    **if** rows $i$ "dominates" row $i'$, i.e., for all columns $j$, $a_{ij} \geq a_{i'j}$
    **then** eliminate row $i$;  (every solution that covers $i'$ will cover $i$ too)
    **end**;
  STEP 3:
    **if** column $j$ is "dominated by" column $j'$, i.e., for all rows $i$, $a_{ij} \leq a_{ij'}$
    **then** eliminate column $j$;  ($j'$ can replace $j$ in every solution)
    **end**;
**until** (matrix $A$ is empty) **or** (no rows nor columns are eliminated in the last iteration);

**output** $J$ and the current matrix $A'$ obtained by reducing $A$;

---

## 2.4 The greedy algorithm for set covering

The greedy algorithm is a simple method that has a performance guarantee of roughly $\ln(|X|)$. This section describes the greedy algorithm, and the performance guarantee is proved in the next section.

$$A = \begin{array}{c|ccccc|} & S_1 & S_2 & S_3 & S_4 & S_5 \\ \hline x_1 & 1 & 0 & 1 & 1 & 0 \\ x_2 & 1 & 0 & 0 & 1 & 0 \\ x_3 & 1 & 1 & 0 & 1 & 0 \\ x_4 & 0 & 1 & 1 & 1 & 0 \\ x_5 & 1 & 0 & 0 & 0 & 1 \\ x_6 & 0 & 0 & 1 & 0 & 0 \\ x_7 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

Initially, $J = \emptyset$.
Step 1 adds 3 to $J$, and eliminates column $S_3$ and rows $x_1$, $x_4$ and $x_6$,
Step 2 eliminates row $x_3$ since it dominates row $x_2$, and
Step 3 eliminates columns $S_2$ and $S_5$ since they are dominated by columns $S_4$ and $S_1$, respectively.
Step 1 adds 1 to $J$, and eliminates column $S_1$ and rows $x_2$ and $x_5$, and then it adds 4 to $J$, and eliminates column $S_4$ and row $x_7$.
The reduced matrix is empty, so the algorithm stops and outputs an optimal solution $J = \{1, 3, 4\}$, i.e., $S_1, S_3, S_4$ form a minimum set covering.

Figure 2.3: The working of the matrix reduction algorithm is shown on an example (from Chapter 6, "Urban Operations Research", by Larson and Odoni).

---

**Algorithm** Greedy Algorithm for Set Covering
**input**: Family $\mathcal{F} = S_1, S_2, \ldots, S_n$ of subsets of a finite set $X$.
**output**: $J \subseteq \{1, 2, \ldots, n\}$ such that $\bigcup_{j \in J} S_j = X$.

$U := X$; ($U$ is the set of points that are currently uncovered)
$J := \emptyset$;

**while** $U \neq \emptyset$ **do**
  choose a subset $S_{j^*}$ in $\mathcal{F}$ such that $|S_{j^*} \cap U|$ is maximum among all $S_j$ in $\mathcal{F}$;
  $U := U \backslash S_{j^*}$ ;
  $J := J \cup \{j^*\}$;
**end** (while);

output $J$;

---

## 2.5 A performance guarantee for the greedy algorithm

For a nonnegative integer $d$, let the $d$th harmonic number $\sum_{i=1}^{d} \frac{1}{i}$ be denoted by $H(d)$ (take $H(0) = 0$).

**Fact 2.2** *For $d$ a positive integer, $\ln(d+1) \leq H(d) \leq 1 + \ln(d)$.*

For sufficiently large integers $d$, $H(d)$ can be approximated by $\ln(d) = \log_e(d)$, where $e$ denotes the base of the natural logarithm.

EXAMPLE

| $d$ | $H(d)$ | $\ln(d)$ | $\ln(d+1)$ |
|---|---|---|---|
| 5 | $1+\cdots+\frac{1}{5} = \frac{137}{60} \approx 2.28333$ | $1.60943\ldots$ | $1.79175\ldots$ |
| 10 | $1+\cdots+\frac{1}{10} = \frac{7381}{2520} \approx 2.92897$ | $2.30258\ldots$ | $2.39789\ldots$ |
| 25 | $3.81595\ldots$ | $3.21887\ldots$ | $3.25809\ldots$ |
| 100 | $5.18737\ldots$ | $4.60517\ldots$ | $4.61512\ldots$ |
| 1000 | $7.48547\ldots$ | $6.90775\ldots$ | $6.90875\ldots$ |
| 10000 | $9.78760\ldots$ | $9.21034\ldots$ | $9.21044\ldots$ |

**Theorem 2.3** *Consider an instance of the set covering problem given by $\mathcal{F} = \{S_1, \ldots, S_n\}$, where each $S_j \subseteq X$, and $X = \bigcup_{j=1}^{n} S_j$. Let $J_{opt}$ denote an optimal solution, and let $J_{greedy}$ denote a solution found by the greedy algorithm. Then*

$$|J_{greedy}| \leq H(\max_{j=1,\ldots,n} |S_j|) \cdot |J_{opt}|.$$

**Proof**: Let $r$ denote $\max_{j=1,\ldots,n}(|S_j|)$.

Let $A$ be the $m \times n$ incidence matrix. Consider the linear programming (LP) relaxation (P) of the integer program (IP) for set covering, and the dual LP (D) of (P). Let $y$ denote the row vector $[y_1, \ldots, y_m]$ of the dual variables.

| | |
|---|---|
| $(P)$ minimize $\sum_{j=1}^{n} s_j$ subject to $As \geq \mathbf{1}^m$, $s_j \geq 0 \quad (j = 1, \ldots, n)$ | $(D)$ maximize $\sum_{i=1}^{m} y_i$ subject to $yA \leq \mathbf{1}^n$, $y_i \geq 0 \quad (i = 1, \ldots, m)$ |

The set covering problem has a feasible solution (since $X = \bigcup_{j=1}^{n} S_j$), hence, both (IP) and (P) have feasible solutions. Let $z_{IP}$ and $z_{LP}$ denote the optimal values of (IP) and (P), respectively; so $z_{IP} = |J_{opt}|$.

The key point of the proof is this:

> there is a way to construct a feasible solution $y = [y_1, \ldots, y_m]$ of the dual LP (D) such that the corresponding objective value $\sum_{i=1}^{m} y_i$ is $\geq |J_{greedy}|/H(r)$.

The theorem immediately follows because

$$|J_{opt}| = z_{IP} \geq z_{LP} \geq \sum_{i=1}^{m} y_i = |J_{greedy}|/H(r).$$

This sequence of inequalities is worth a detailed study. The first inequality ($z_{IP} \geq z_{LP}$) follows because (P) is a relaxation of (IP). In other words, every feasible solution of (IP), including the optimal (IP) solution, is a feasible solution of (P), hence, the optimal solution of (P) has objective value at most $z_{IP}$. The second inequality ($z_{LP} \geq \sum_{i=1}^{m} y_i$) is a direct consequence of linear programming weak duality applied to (P) and (D). The third inequality is guaranteed by the construction of $y$.

Here is the construction for $y$. For every iteration of the greedy algorithm, let $U$ denote the set of uncovered points at the start of that iteration. Consider an arbitrary iteration, and let the set chosen in that iteration be $\hat{S}$. We say that each point in $\hat{S} \cap U$ is *newly covered* by $\hat{S}$. In other words, a point $x$ is said to be newly covered by the set $S$ if $S$ is the first set in the sequence of sets chosen by the greedy algorithm such that $x \in S$. For each point $x \in \hat{S} \cap U$, define $wgt(x) = 1/|\hat{S} \cap U|$. Observe that for each point $x \in X$, the quantity $wgt(x)$ is well defined, because for each point $x \in X$, there exists a set $S$ in our family $\mathcal{F}$ such that $x$ is newly covered by $S$. To complete the construction, for $1 \leq i \leq n$, the dual variable $y_i$ of the point $x_i$ is assigned the value $wgt(x_i)/H(r)$. The next three claims show that $y = [y_1, \ldots, y_m]$ satisfies all the constraints of the dual LP (D) and that $\sum_{i=1}^{m} y_i \geq |J_{greedy}|/H(r)$. The first claim is the crucial step in the proof.

**Claim 1**: Consider an arbitrary set $S \in \mathcal{F}$. Let $q$ denote $|S|$. Order the points in $S$ in the reverse order in which they are covered by the greedy algorithm, and let this ordering be $x'_1, x'_2, \ldots, x'_q$. Then

$$wgt(x'_\ell) \leq 1/\ell, \quad \forall \ell = 1, 2, \ldots, q.$$

To elaborate on the ordering $x'_1, x'_2, \ldots, x'_q$, focus on the last iteration of the greedy algorithm such that $S \cap U$ is nonempty; then we order the points in $S \cap U$ arbitrarily, and take them to be $x'_1, x'_2, \ldots, x'_{|S\cap U|}$. (Similarly, in an arbitrary iteration of the greedy algorithm such that the chosen set is, say, $S_j$ (where $1 \leq j \leq n$) we take the points in $S \cap U \cap S_j$ to be $x'_{|S\cap U|}, x'_{|S\cap U|-1}, \ldots, x'_{|S\cap U|-|S\cap U\cap S_j|+1}$.) Now, consider an arbitrary point $x'_\ell \in S$, $1 \leq \ell \leq q$. Let $x'_\ell$ be newly covered by the set $\hat{S}$. At the start of the iteration that chooses $\hat{S}$, all the points $x'_1, x'_2, \ldots, x'_\ell$ are uncovered, i.e., $\{x'_1, x'_2, \ldots, x'_\ell\} \subseteq S \cap U$. Hence, $|S \cap U| \geq \ell$. By the definition of the greedy strategy, the set $\hat{S}$ chosen in this iteration must have

$$|\hat{S} \cap U| \geq |S \cap U| \geq \ell.$$

Since the number of newly covered points in $\hat{S}$ is $\geq \ell$, $wgt(x'_\ell) = 1/|\hat{S} \cap U| \leq 1/\ell$.

**Claim 2**: For every set $S \in \mathcal{F}$, $\sum_{x \in S} wgt(x) \leq H(r)$.

Let $q = |S|$, and let $x'_1, x'_2, \ldots, x'_q$ be the ordering of the points in $S$ defined in Claim 1. Then

$$\sum_{x\in S} wgt(x) = wgt(x'_1)+wgt(x'_2)+\ldots+wgt(x'_i)+\ldots+wgt(x'_q) \leq 1+\frac{1}{2}+\ldots+\frac{1}{i}+\ldots+\frac{1}{q} = H(q) \leq H(r),$$

where the first inequality follows directly from Claim 1, and the last inequality follows since $q = |S| \leq \max_{j=1,\ldots,n}(|S_j|) = r$.

**Claim 3**: $\sum_{i=1}^{m} wgt(x_i) = |J_{greedy}|$.

To see this, focus on a set $\hat{S}$ chosen by the greedy algorithm and the corresponding set $U$ as defined above. Then $\sum_{x \in \hat{S} \cap U} wgt(x) = |\hat{S} \cap U| \cdot wgt(x \mid x \in \hat{S} \cap U) = |\hat{S} \cap U| / |\hat{S} \cap U| = 1$. In other words, for each set $\hat{S}$ chosen by the greedy algorithm, the newly covered points in $\hat{S}$ contribute a total weight of one. Summing over all sets chosen by the greedy algorithm gives Claim 3.

Consider $y = [y_1, \ldots, y_m]$, where $y_i = wgt(x_i)/H(r), \forall i = 1, \ldots, m$. Clearly, $y$ is $\geq 0$, and by Claim 2, for each set $S \in \mathcal{F}$, we have $\sum_{x_i \in S} y_i = \frac{1}{H(r)} \sum_{x_i \in S} wgt(x_i) \leq \frac{1}{H(r)} \cdot H(r) = 1$. This shows that $y$ is a feasible solution of (D). Moreover, Claim 3 shows that $\sum_{i=1}^{m} y_i = \frac{1}{H(r)} \sum_{i=1}^{m} wgt(x_i) = \frac{|J_{greedy}|}{H(r)}$. This completes the construction of $y$, and the proof of the theorem. ∎

## 2.6 A direct analysis

In this section, we look at an alternate analysis of the greedy heuristic for set cover that does not use linear programming duality. Recall that the greedy algorithm chooses sets in iterations, where in every iteration, the set chosen in one newly covering the maximum number of points.

The key point of the alternate proof is this:

> When set $S_{j^*}$ is chosen by the greedy algorithm, $|S_{j^*} \cap U| \geq \frac{|U|}{|J_{opt}|}$.

Note that $U$ represents the current set of uncovered points in the above statement. This observation is a direct consequence of the greedy choice, and the definition of the optimal solution: consider the optimal cover $J_{opt}$ which covers all the points in $X$, and hence also the points in $U$. By averaging among the sets in $J_{opt}$, the one which covers the maximum number of points in $U$ must cover at least $\frac{|U|}{|J_{opt}|}$. Since the greedy algorithm chooses among all the sets the one with the maximum new coverage, this coverage must be at least as much as claimed.

Let the indices of the sets picked by the greedy algorithm in the order they were picked be denoted by $j(1), \ldots, j(g)$. Furthermore, for $t = 1, \ldots, g$, let $U_t$ denote the set $U$ just before the set $J_{j(t)}$ was picked. Thus, for example, $U_1 = X$. Define $U_{g+1} = \emptyset$. We can write a simple recurrence for $|U_t|$.

$$|U_{t+1}| = |U_t| - |S_{j(t)} \cap U_t|$$

By the observation above, we have $|S_{j(t)} \cap U_t| \geq \frac{|U_t|}{|J_{opt}|}$. Substituting and simplifying, we get

$$|U_{t+1}| \leq |U_t|(1 - \frac{1}{|J_{opt}|})$$

Expanding out the recurrence, we get

$$|U_t| \leq |X|(1 - \frac{1}{|J_{opt}|})^t$$

Taking natural logarithm and simplifying, we finally get

$$t \leq |J_{opt}| \ln \frac{|X|}{U_t} \tag{2.1}$$

Since $U_g \neq \emptyset$, this immediately implies that $|J_{greedy}| = g \leq |J_{opt}| \ln |X|$. Note that the performance ratio is slightly weaker, namely, $\ln |X|$ versus the earlier derived $\ln \max_{j=1,\ldots,n}(|S_j|)$. Despite this shortcoming, there are a couple of corollaries of this analysis that are simpler to derive than from the earlier proof.

**Corollary 2.4** *Suppose we run the greedy algorithm only until a constant fraction, say $0 < \alpha < 1$ of the points are covered. Then the number of sets used is at most $1 + |J_{opt}| \ln \frac{1}{\alpha}$.*

To prove the corollary, we use the first term of one to account for the last iteration before stopping. At the iteration just preceding this, we have $|U| > \alpha \cdot |X|$ by the stopping rule, and the claim follows from equation 2.1.

**Corollary 2.5** *Suppose at every step of the greedy algorithm, we are only able to choose a near-optimal greedy set, i.e., we choose a set $S_{j^*}$ whose new coverage is at least $\beta$ times the maximum among that of all $S_j$ in $\mathcal{F}$ for some $0 < \beta \leq 1$. Then the size of the cover output is at most $\beta |J_{opt}| \ln |X|$.*

This corollary is a direct consequence of substituting the weaker guarantee into the analysis and simplification of the recurrence relation for $|U|$.

## 2.7 A randomized algorithm

The greedy algorithm may be viewed as a derandomization of a simple randomized algorithm for choosing a set cover. The starting point of this algorithm is a solution to the linear program (P) for set cover. This solution assigns values $s_j$ between zero and one to the sets, that sum to $z_{LP}$. We can interpret these values as probabilities and choose independently, every set $S_j$ with probability $s_j$. By linearity of expectation, the expected number of sets chosen is $z_{LP}$.

Next we bound the probability that any point $x \in X$ is not covered by the sets chosen in one round of this randomized selection of sets. The point $x$ is not covered when none of the sets containing it is chosen. The probability of this event is $\prod_{j:S_j \ni x}(1 - s_j) \leq e^{-\sum_{j:S_j \ni x} s_j} \leq \frac{1}{e}$.

Suppose we repeat the randomoized selection for $\gamma \ln |X|$ rounds. Then the probability of not covering any given point $x \in X$ drops to at most $(\frac{1}{e})^{\gamma \ln |X|} \leq |X|^{-\gamma}$. By the union bound, the probability that some point in $X$ is not covered is at most $\sum_{x \in X} |X|^{-\gamma} \leq |X|^{1-\gamma}$. By choosing a sufficiently large constant $\gamma$, we can drive down this probability to any polynomially small factor.

An alternate oblivious rounding algorithm and its derandomization leading to the greedy algorithm is discussed by Young in [8]. The analysis of the greedy heuristic for set covering is due to

Johnson [3], Lóvasz [5], and Chvátal [1]. Our analysis using duality follows the treatment of Chvátal closely. The direct analysis to the best of our knowledge appeared first in a paper of Leighton and Rao [4] on approximating graph separators. A tight analysis of the greedy heuristic, with more on the second order terms is due to Slavik [7].

What will not discuss but is nevertheless another exciting aspect of research on the set covering problem is its non-approximability. The first result in this area due to Lund and Yannakakis [6] showed that there is no polynomial time approximation algorithm with performance ration better than $\frac{\ln |X|}{8}$ unless $NP \subseteq DTIME[n^{\text{polylog } n}]$, an unlikely complexity theoretic assumption akin to $NP \subseteq P$. After a series of strengthenings, the best known result [2] essentially matches the performance of the greedy algorithm and shows that there is no polynomial time approximation algorithm with performance ration better than $(1 - o(1)) \ln |X|$ unless $NP \subseteq DTIME[n^{\log \log n}]$.

## 2.8 Exercises

1. Given an undirected graph $G = (V, E)$ with a subset $T \subseteq V$ of nodes specified as *terminals*, and nonnegative costs $c : V \to \Re_+$ on the nodes, the Node Steiner Tree problem is to find a tree of minimum total cost containing all the terminals. Transform set covering to this problem to show that it is NP-hard.

   (Extra credit) Derive an $O(\log |V|)$ approximation algorithm for the Node Steiner Tree problem. Hint: Formulate this problem as a set cover problem in such a way that it would be possible to implement a single step of the greedy algorithm in polynomial time.

   Is there an approximation algorithm for this problem with performance ratio $o(\log |V|)$?

2. Construct an example of the set covering problem (one for each $|X|$) such that

   (P1) $|J_{greedy}| \geq \alpha \cdot \log_2 |X| \cdot |J_{opt}|$, where $\alpha$ is a constant,

   (P2) no two rows of the incidence matrix $A$ are identical and

   (P3) in the set family $\mathcal{F}$, no set $S_j$ is a subset of another set $S_k$.

3. Repeat the above replacing (P1) with $|J_{opt}| \geq \alpha \cdot \log_2 |X| \cdot z_{LP}$ where $\alpha$ is a constant.

4. The fixed-charge median problem is defined as follows: we are given a complete undirected graph $G = (V, E)$, with nonnegative fixed location costs $f : V \to \Re_+$ on the nodes and nonnegative distances $d : E \to \Re_+$ on the edges. The objective is to locate median nodes at a subset $M \subseteq V$ of the nodes. Every non-median node is defined to be serviced by the median node closest to it under the distances $d$. The total cost of this median placement is the sum of the fixed location costs of all the median nodes, and the sum over non-median nodes, of the distance of the node to its closest median. Formally, the cost of a placement of medians in $M$ is $c(M) = \sum_{v \in M} f_v + \sum_{v \in V-M} \min_{u \in M} d_{uv}$. The goal is to find a placement $M$ with minimum total cost.

   Derive an $O(\log |V|)$ approximation algorithm for this problem.

5. (Extra credit) In showing a 2-approximation algorithm for node-multiway cuts, we showed half-integrality of optimal solutions for a certain linear relaxation of the problem. The dual to this relaxation is a linear program whose objective is to maximize the total concurrent flow between terminals specified in the problem. Prove or disprove: for any specification of integral capacities, there is a maximum flow solution that is also half-integral.

6. Transform set covering to an instance of the Directed Steiner Tree problem: given a digraph $G = (V, A)$, with a root node $r$ and a subset $T \subseteq V$ of *terminals*, and nonnegative costs $c : A \to \Re_+$ on the arcs, find an outward-directed spanning tree of minimum total cost with $r$ as the root and containing all the nodes in $T$.

# Bibliography


[1] V. Chvátal, *A greedy heuristic for the set-covering problem*, Math. of OR **4:3** (1979), pp. 233-235.

[2] U. Feige, *A threshold of ln n for approximating set cover*, Proc. 28th Annual Symp. on Theory of Computing (1996), pp. 314-318.

[3] D. S. Johnson, *Approximation algorithms for Combinatorial Problems*, J. Computer System. Sci. **9** (1974), pp. 256-278.

[4] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms*," Proc. of the 29th Annual IEEE Conference on Foundations of Computer Science (1988), pp. 422-431.

[5] L. Lovàsz, *On the ratio of optimal integral and fractional covers*, Disc. Math. **13** (1975), pp. 383-390.

[6] C . Lund and M. Yannakakis, *On the Hardness of Approximating Minimization Problems*, Proc., 25th Annual ACM Symp. on Theory of Computing (1993), pp. 286-293.

[7] P. Slavik, *A tight analysis of the greedy algorithm for set cover*, Proc. 28th Annual Symp. on Theory of Computing (1996), pp. 435-441.

[8] N. Young, *Randomized rounding without solving the linear program*, Proc. of the 6th Annual ACM-SIAM Symp. on Disc. Algo. (1995), pp. 170-178.