Optimal Set Covering with Application

Samuel Abebe
I.D No GSR/0434/01

Addis Ababa University
Natural Science College
Faculty of Computer and Mathematical Sciences
Department of Mathematics

"In partial fulfillment of the requirements for the degree of Master of Science in Mathematics,"

Advisor Berhanu Guta (Dr.)

January, 2011

Addis Ababa, Ethiopia

Declaration

“I declare that this project has been composed by me and that no part of the project has formed the basis for the award of any Degree, Diploma, Associate ship, fellowship or any other similar title to me

_____

Author’s Signature”

Permission

“This is to certify that this project is compiled by Mr. Samuel Abebe in the department of mathematics, Addis Ababa University, under my supervision. I hereby also confirm that the project can be submitted for evaluation by examiners and eventual defense.

_____

Advisor’s Signature

Acknowledgment

This project paper was successful because of the great enthusiasm and wise support from my advisor Dr. Berhanu Guta and a never-ending support from my fiancé S\r Mistir Chirotaw. May God be with you, many thanks.

Summary of the project

The set covering problem (SCP) is a fundamental problem in the class of covering problems. Given a finite set E and a family $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ of subsets of E $(i.e., S_j \subseteq E,\ j = 1,2, \dots, n)$ the set covering problem is to find the minimum cardinality $J \subseteq \{1,2, \dots, n\}$ such that $\bigcup_{j \in J} S_j = E$. The elements of E are called *points*. Given $J \subseteq \{1,2, \dots, n\}$, a *point* is said to be *covered* if it belongs to $\bigcup_{j \in J} S_j$. In the case of minimum cost covering each set S_j have a cost $c_j > 0$ and our aim will be minimizing $\sum_{j \in J} c_j$ such that $\bigcup_{j \in J} S_j = E$.

In this project I focused, in general, on the subject matter of the problem and the main applications together with some basic algorithms for obtaining optimal (near-optimal) solutions including some MatLab programming.

1. Introduction

The *set covering problem* (SCP) is a classical question in computer science and complexity theory. It is a problem "whose study has led to the development of fundamental techniques for the entire field" of approximation algorithms. [6]

In this project work I tried to present explanations concerning the SCP and a number of basic algorithms to find near optimal solutions and their analysis with examples for the greedy and matrix reduction algorithms; together with applications in which the problem could arise and finally concluding remarks on the algorithms and a programming code in MatLab for the Greedy-Set-Cover algorithm, without association of costs for the subsets.

1.1. What is the set cover problem?

The set covering problem is an abstraction of many commonly arising combinatorial problems. As an input we are given several sets. They may have some elements in common. We must select a minimum number of these sets so that the sets we have picked contain all the elements that are contained in any of the sets in the input. In other words, given a finite set E and a family $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ of subsets of E $(i.e., S_j \subseteq E,\ j = 1,2, \dots, n)$ the set covering problem is to find the minimum cardinality $J \subseteq \{1,2, \dots, n\}$ such that $\bigcup_{j \in J} S_j = E$ [6]. In the minimum-cost set covering problem, each set $S_j, 1 \leq j \leq n,$ has a cost $c_j > 0$, and the problem is to find $J \subseteq \{1,2, \dots, n\}$ such that each point is *covered* and $\sum_{j \in J} c_j$ is minimum and $\bigcup_{j \in J} S_j = E$ [14].

1.2. Why it is useful?

The SCP is a main model for several important applications [2] and many applications arise having the covering structure. Delivery and routing problems, scheduling problems and location problems often take on a set covering structure whereby one wishes to assure that every customer is served by some location, vehicle or person [3]. Other applications include switching theory, the testing of Very-large-scale integration (VLSI) circuits, and line balancing. In general, Successful real-world applications of SCP are plenty, which can be classified in three major areas like (1) *facility location* (2) *scheduling or staffing of personnel*, and (3) *dispatching trucks or vehicles to routes or customers* [14].

1.3. How can we solve it?

The set covering problem (SCP) was one of Karp's 21 $\mathcal{NP}$-*complete* problems shown to be $\mathcal{NP}$-*complete* in 1972 [4]. But it is also $\mathcal{NP}$-*hard* in the strong sense. [2][8]

Once the problem has been formulated as a set-covering problem, the search for an optimal (or near-optimal) solution remains.

Most solution approaches start by considering the linear programming (LP) relaxation of this 0-1 linear programming problem [3].

The most recent and effective algorithms for SCP are both heuristic and exact approaches [2], which are based on the branch-and-bound approach.

The most natural heuristic approaches are the greedy algorithm [9] and Matrix reduction algorithm which shown to be working by an example, even though its performance was not guaranteed [15].

Good quality solutions were also obtained by [18] using a genetic-based heuristic and [17] presented a simple, robust, and quite fast heuristic by using a probabilistic greedy search.

By using a dynamic sub-gradient heuristic incorporated with branch-and-bound [20] showed significantly better quality of solutions for the SCP.

A polynomial time approximation algorithm relying on the transformation of an instance of SCP into an instance of a particular flow problem was presented in the paper by [16] and based on graph theoretic relaxation [12] designed a tree search algorithms for SCP. [7] found better computational results for larger instances of problems involving up to 400 rows x 4000 columns by using a Lagrangian heuristic, feasible solution exclusion constraints, Gomory *f*-cuts and an improved branching strategy and by applying the meta-heuristic Meta-RaPS (Meta-heuristic for Randomized Priority Search) [8] developed an effective heuristic to solve the SCP which had been tested on 80 SCP instances from the OR-Library. The sizes of the problems were up to 1000 rows x 10,000 columns for non-unicost SCP and 28,168 rows x 11,264 columns for the unicost SCP.

2. Preliminaries

2.1. Decision problems

Definition 2.1 A decision problem is any arbitrary yes-or-no question on an infinite set of inputs.

Definition 2.2 An algorithm is said to be *polynomial time* if its running time is upper bounded by a polynomial in the size of the input for the algorithm, i.e. $\mathcal{T}(n) = \mathcal{O}(n^k)$ for some constant k.

Definition 2.3 *Polynomial time-*$\boldsymbol{\mathcal{P}}$ is the class of all (decision) problems that have an algorithm that solves it in polynomial time (i.e. class of decision problems that can be efficiently solved)

Definition 2.4 *Nondeterministic Polynomial Time-*$\boldsymbol{\mathcal{NP}}$ is the class of all problems that have efficient certifiers (i.e. class of decision problems whose solutions can be efficiently verified)

Definition 2.5 $\boldsymbol{\mathcal{NP}}$*-complete* problems are the hardest problems in $\boldsymbol{\mathcal{NP}}$ in the sense that they have a *polynomial-time* algorithm if and only if $\boldsymbol{\mathcal{P}} = \boldsymbol{\mathcal{NP}}$[1].

2.2. Linear programming

Linear programming describes a broad class of optimization tasks in which both the constraints and the optimization criterion are linear functions. It turns out an enormous number of problems can be expressed in this way.

The general problem of linear programming is to find values for real variables $x_1, x_2, \dots, x_n$, which yield an extreme value (maximum or minimum) for a linear function

- Minimization linear program*:*

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j$$

[1] $\boldsymbol{\mathcal{NP}}$*-completeness* only means that (assuming $\boldsymbol{\mathcal{P}} \neq \boldsymbol{\mathcal{NP}}$) the problem does not have an algorithm that solves it *exactly* on every input

$$\text{Subject to} \quad \sum_{j=1}^{n} a_{ij}x_j \geq b_j, \qquad i = 1,2,\dots,m$$
$$x_j \geq 0, \qquad j = 1,2,\dots,n$$

- Maximization linear program:

$$\text{Maximize} \quad \sum_{i=1}^{m} b_i y_i$$

$$\text{Subject to} \quad \sum_{i=1}^{m} a_{ij}y_i \leq c_j, \qquad j = 1,2,\dots,n$$
$$y_i \geq 0, \qquad i = 1,2,\dots,m$$

  Where a_{ij}, b_i, and c_j are given rational numbers.

In the case where, if there is an assignment of integer numbers to $x_1, x_2, \dots, x_n$. The linear programming is said to be *Integer Programming.*

2.3. Duality Theory

The dual of an LP in standard form

$$(\text{P}) \quad \text{Maximize} \quad b^T y$$
$$\text{Subject to} \quad A^T y \leq c, \;\; y \geq 0$$

Is the LP

$$(D) \quad \text{Minimize} \quad c^T x$$
$$\text{Subject to} \quad Ax \geq b, \quad x \geq 0$$

Let the minimization program be the primal (P) program

Proposition 2.1 (**Weak-Duality**) [21]

If $\bar{x} = (x_1, x_2, \dots, x_n)$ and $\bar{y} = (y_1, y_2, \dots, y_m)$ are feasible solutions for the primal and dual program, respectively, then

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i \qquad \dots \quad (1)$$

By the LP-duality theorem, (1) holds with equality iff both $\bar{x}$ and $\bar{y}$ are optimal solutions.

Proposition 2.2 (Primal complementary slackness conditions)

Let $\beta \geq 1$

For each $1 \leq j \leq n$, either $x_j = 0$ or $c_j/\beta \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$

Proposition 2.3 (Dual complementary slackness conditions)

Let $\gamma \geq 1$

For each $1 \leq i \leq m$, either $y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \gamma . b_i$

By the LP-duality theorem solutions $\bar{x}$ and $\bar{y}$ are both optimal iff $\beta = 1$ and $\gamma = 1$.

Proposition 2.4 If $\bar{x}$ and $\bar{y}$ are primal and dual feasible solutions satisfying the slackness conditions, then

$$\sum_{j=1}^{n} c_j x_j \leq \beta\gamma \sum_{i=1}^{m} b_i y_i$$

3. The Problem and its Complexity

3.1. Set Covering Problem

Definition 3.1 An instance $(E, \mathcal{F})$ of the *set-covering problem* consists of a finite set E and a set $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ of subsets of E. We assume that each element of E appears in at least one of the subsets in $\mathcal{F}$,

$$E = \cup_{S_j \in \mathcal{F}} S_j \quad \dots \quad (2)$$

The set-covering problem requires us to determine a minimum sized subset of $\mathcal{F}$ that covers all the elements of E. In other words, given an instance $(E, \mathcal{F})$, we are required to find a minimum cardinality $J \subseteq \{1,2, \dots, n\}$ such that,

$$E = \cup_{j \in J} S_j \quad \dots \quad (3)$$

Which is called a *set-covering* of E, and S_j for $j \in J$ are called *covering sets*. If, with each $S_j \in \mathcal{F}$, there is associated a *(positive)* cost c_j, we wish to find that set-covering of E which has minimum cost, the cost being $\sum_{j \in J} c_j$.

In general, if all the subsets S_j have a unit cost $c_j > 0$; the problem is called the *unicost* SCP. Otherwise, it's called the *non-unicost (weighted)* SCP and if there is a common usage including both cases we just call it a SCP.

An instance of the set-covering problem is presented in Figure 1. As can be seen in the figure, the minimum sized set cover for this instance is $J = \{2,5,6\}$, that is S_2, S_5, S_6.

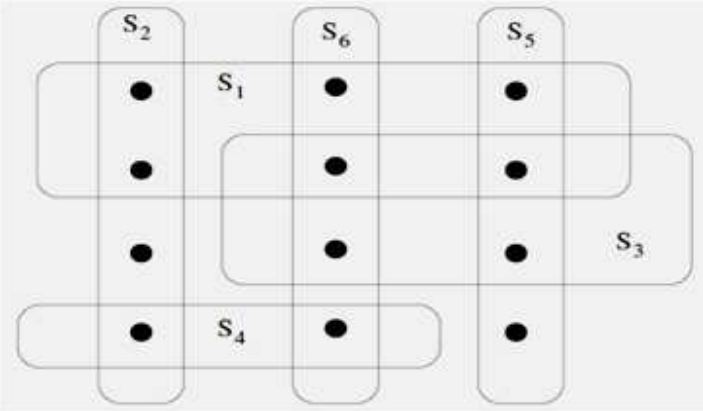

Figure 1: An instance of the set-covering problem. Each dark circle represents an element of E.
The set $\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$ with each boxes indicating members of $\mathcal{F}$. The minimal set cover is given by $J = \{2,5,6\}$. i.e., the sets S_2, S_5 and S_6 are minimal covers.

Proposition 3.1: The decision version of the set cover is $\mathcal{NP}$-complete [13].

The SCP is an abstraction of many commonly arising combinatorial problems [1].

Example 3.1 Suppose that E represents a set of skills that are needed to solve a problem and we have a given set of people available to work on the problem. We wish to form a committee containing as few people as possible, such that for every request skill in E, there is a member of the committee having that skill. In the decision version of the SCP, if we ask whether or not a covering exists with size at most k, where k is an additional parameter specified in the problem instance, the decision version of the problem is $\mathcal{NP}$-complete.

3.2. Linear Programming formulation of SCP

We define an incidence matrix A of a set covering problem as follows. There are $m = |E|$ rows in A, one for each point $e_i \in E$, 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 j^{th} column) is one if point e_i is in set S_j, otherwise a_{ij} is zero.

Example 3.2 If $\boldsymbol{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ and $\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5\}$ such that
$S_1 = \{e_1, e_2, e_3, e_5\}$, $S_2 = \{e_3, e_4, e_7\}$, $S_3 = \{e_1, e_4, e_6\}$, $S_4 = \{e_1, e_2, e_3, e_4, e_7\}$, and $S_5 = \{e_5\}$.
Then,

$$A = \begin{array}{c|ccccc} & S_1 & S_2 & S_3 & S_4 & S_5 \\ \hline e_1 & 1 & 0 & 1 & 1 & 0 \\ e_2 & 1 & 0 & 0 & 1 & 0 \\ e_3 & 1 & 1 & 0 & 1 & 0 \\ e_4 & 0 & 1 & 1 & 1 & 0 \\ e_5 & 1 & 0 & 0 & 0 & 1 \\ e_6 & 0 & 0 & 1 & 0 & 0 \\ e_7 & 0 & 1 & 0 & 1 & 0 \end{array}$$

Hence, an integer linear programming formulation of the SCP is as follows. For each set S_j, we have a zero-one variable x_j. Such that, $x_j = 1$ iff set S_j is chosen in the optimal solution otherwise $x_j = 0$. Let 1^m denote a vector of m ones and $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ denote a column vector. Therefore, the linear programming formulation will be:

$$\text{(IP)} \qquad \text{Minimize} \quad \boldsymbol{z} = \sum_{j=1}^{n} x_j \qquad \dots \quad (4)$$

$$\text{Subject to} \quad Ax \geq 1^m \qquad \dots \quad (5)$$

$$x_j \in \{0,1\}, \ (j = 1,2, \dots, n) \qquad \dots \quad (6)$$

For the minimum-cost set covering problem if the value $c_j > 0$ $(j = 1,2, \dots, n)$ represents the cost of column j, then the objective function becomes:

$$\boldsymbol{z} = \sum_{j=1}^{n} c_j x_j \qquad \dots \quad (7)$$

Equation (5) ensures that each row is covered by at least one column and equation (6) is the integrality constraint.

Proposition 3.2 The 0-1 integer programming problem is $\mathcal{NP}$*-complete*. [4][21]

In linear programs we are not allowed to require that the decision variables x_j are integers. But, there exist $\mathcal{P}$-time algorithms for linear programming problems and in cases such as SCP we are still able to derive useful information from linear programs. For instance, if we replace the constraints $x_j \in \{0,1\}$ with the constraints $0 \leq x_j \leq 1$, we obtain the following linear program, which can be solved in polynomial time:

$$\text{(LP)} \qquad \text{Minimize} \quad \boldsymbol{z} = \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to} \quad Ax \geq \mathbf{1}^m \qquad \ldots \quad (8)$$

$$0 \leq x_j \leq 1, \ \ (j = 1,2, \ldots, n)$$

The linear program (LP) in (8) is called a *relaxation* of the original program (IP) and since the (LP) is less constrained than the (IP); the following are immediate:

i. Every feasible solution for the original program (IP) is feasible for this program (LP).
ii. The objective value *(cost)* of any feasible solution for the integer program (IP) has the same objective value *(cost)* in linear program (LP) and
iii. The optimal objective value for the LP is less than or equal to the optimal objective value for the IP.

Hence, if we let z_{LP}^* denote the optimum value of the linear program and z_{IP}^* denote the optimum value of the integer program, we get the relationship $z_{LP}^* \leq z_{IP}^* = OPT$ holds true, since z_{LP}^* finds a feasible solution of lowest possible value. Where *OPT* is the optimal solution, z_{IP}^*, for the SCP.

4. Algorithms

4.1. Heuristics

Since no *polynomial-time* algorithm is available for finding the optimal solution of SCPs, 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 solution, the goal is to find whether the solution returned by the method is within a fixed factor of optimal, for every instances of the problem. Hence, by [10] the usual metric for measuring the nearness to optimality of a solution is by the ratio of its cost to that of an optimal solution. This is important because, [5], finding a good lower bound on the optimal solution *(OPT)* is a basic starting point in the design of heuristic algorithms for a minimization problem.

4.1.1. The matrix reduction algorithm (MRA)

The matrix reduction algorithm is a heuristic for the SCP. Given an instance of the problem in the form of an incidence matrix A, the algorithm "*reduces*" the matrix by repeatedly attempting to *eliminate* rows and columns and in the process the algorithm constructs $J \subseteq \{1,2,\dots,n\}$ such that $J' \subseteq \{1,2,\dots,n\}$ is an optimal solution of the reduced instance. If the reduction algorithm terminates with no matrix remaining, an optimal solution has been identified. Then J is an optimal solution of the original instance. But if the algorithm terminates with a matrix A' remaining; an integer programming (like branch and bound, which is discussed in section 4.2) will be applied for this matrix to get the remaining solution J'. Then $J \cup J'$ will be an optimal solution of the original instance of the problem.

Algorithm 1 MATRIX REDUCTION -SET-COVER

Input: incidence matrix A

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;

$J \leftarrow \emptyset$;

repeat

Step 1:

If row i has exactly one nonzero entry, say, in column j

then $J \leftarrow J \cup \{j\}$; (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 row 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 $i, a_{ij} \leq a_{i'j}$

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$A$;

Example 4.1 If we take the example 3.2 we have a solution using algorithm 1 (*MRA)* as follow.

Initially, $J = \emptyset$.

Step 1 adds 3 to J, and eliminates column S_3 and row e_1, e_4 and e_6,

Step 2 eliminates row e_3 since it dominates row e_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 columns S_1 and rows e_2 and e_5, and then it adds 4 to J, and eliminates column S_4 and row e_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.

4.1.2. A greedy approximation algorithm

The greedy method works by picking, at each stage, the *set* S_j that covers the greatest number of remaining elements that are uncovered, in the *unit-cost* case or the set whose *cost-effectiveness* is smallest in the case of *non-unicost* case.

Algorithm 2 GREEDY-SET-COVER $(E, \mathcal{F})$ FOR UNICOST

Input: family $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ of subsets of a finite set E

Output: $J \subseteq \{1,2, \dots, n\}$ such that $E = \bigcup_{j \in J} S_j$

1. $U \leftarrow E$; (U the set points that are currently uncovered)
2. $J \leftarrow \emptyset$;
3. **while** $U \neq \emptyset$ **do**
4. select $S_{j*} \in \mathcal{F}$ that maximize $|S_j \cap U|$
5. $U \leftarrow U \backslash S_{j*}$;
6. $J \leftarrow J \cup \{j^*\}$;
7. **end(while)**;
8. **output** J;

A greedy algorithm for the set-covering problem (unicost case) is presented above. The algorithm maintains a set of elements of E that are as yet uncovered in U. In each iteration of the while loop, the algorithm greedily chooses the set S_j that maximally covers the elements of U, until all of the elements of E are covered. The set J contains all the sets that have been chosen as part of the set cover at any point during the operation of the algorithm.

Example 4.2 Let's take example 3.2 again as above and we have a solution using algorithm 2 as follow:

Step 1 $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

Step 2 $J = \emptyset$

Step 3 $U \neq \emptyset$

Step 4 selects S_4

Step 5 $U = \{e_5, e_6\}$

Step 6 $J = \{4\}$

Step 3 $U \neq \emptyset$

Step 4 selects either S_1 and S_3 or S_3 and S_5.

Step 5 $U = \emptyset$

Step 6 $J = \{1,3,4\}$ or $J = \{3,4,5\}$

Now at step 3, since $U = \emptyset$, the algorithm stops and step 8 outputs optimal solutions $J = \{1,3,4\}$ or $J = \{3,4,5\}$ i.e. either S_1, S_3, S_4 or S_3, S_4, S_5 form a minimum set-covering.

Here, we actually have two alternative solutions in which the selection depends on our decision on step 4. And also if we take an inspection, a modification on the greedy step, we can see that a selection of $S_1, S_2,$ and S_3 is as well optimal. Hence, all of the three solutions we have found will be optimal. This means that, the minimal cover contains only three subsets. Even though not unique in this case, the selection of any of these three alternative solutions will form a minimal cover.

In the case of *non-unicost* case, the greedy strategy, iteratively, picks the most *cost-effective* set and remove the covered elements, until all elements are covered. If we let S' be the set of elements already covered at the beginning of the iteration. During this iteration, define the *cost-effectiveness*, α_j of a set S_j to be the average cost at which it covers new elements, i.e. $c(S_j)/|S_j - S'|$. Defining the price of an element to be the average cost at which it is covered or when a set S_j is picked, we can think of its cost being distributed equally among the new elements covered, to set their price.

Algorithm 3 GREEDY-SET-COVER $(E, \mathcal{F})$ FOR NON-UNICOST
Input: family $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ of subsets of a finite set E and their respective cost $c(S_j)$ for $j = 1,2, \dots, n$.
Output: $C \subseteq \{S_1, S_2, \dots, S_n\}$ such that $E = \cup_{S_j \subseteq C} S_j$

1. $U \leftarrow E$; (U the set points that are currently uncovered)
2. $S' \leftarrow \emptyset$; (S' the set of points that are currently covered)
3. $C \leftarrow \emptyset$; (C subsets currently selected)
4. **while** $S' \neq U$ **do**
5. Let $\alpha_j = \frac{c(S_j)}{|S_j - S'|}$, i.e. the cost-effectiveness of S_j, $\forall j$

6. find the set whose *cost-effectiveness* is smallest, say S_j^*.
7. Pick S_j^*, and for each $e_i \in S_j^* - S'$, set price$(e_i) = \alpha_j^*$
8. $S' \leftarrow S' \cup \{e_i : e_i \in S_j^* - S'\}$;
9. $C \leftarrow C \cup \{S_j^*\}$;
10. **end(while)**;
11. **output** C;

Example 4.3 Here we take the same example as before but we associate a cost of 1-unit to each $S_j \subseteq \mathcal{F}$. And we'll see how algorithm 3 works.

Step 1 $U = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$;

Step 2 $S' = \emptyset$;

Step 3 $C = \emptyset$;

Step 4 $S' \neq U$;

Step 5 $\alpha_1 = \frac{1}{4},\ \alpha_2 = \frac{1}{3},\ \alpha_3 = \frac{1}{3},\ \alpha_4 = \frac{1}{5},\ \alpha_5 = \frac{1}{1}\}$

Step 6 selects S_4 , since its *cost-effectiveness* is $\alpha_4 = \frac{1}{5}$, the smallest.

Step 7 sets price$(e_i) = \frac{1}{5}$, $\quad i = 1,2,3,4,7$

Step 8 $S' = \{e_1, e_2, e_3, e_4, e_7\}$;

Step 9 $C = \{S_4\}$

Step 4 $S' \neq U$

Step 5 $\alpha_1 = \frac{1}{1},\ \alpha_2 = \infty,\ \alpha_3 = \frac{1}{1},\ \alpha_5 = \frac{1}{1}$

Step 6 selects any one of the sets S_1 or S_3 or S_5 as they have the same *cost-effectiveness*, say S_1

Step 7 sets price $(e_5) = 1$

Step 8 $S' = \{e_1, e_2, e_3, e_4, e_5, e_7\}$;

Step 4 $S' \neq U$

Step 5 $\alpha_3 = 1$

Step 6 selects S_3, since it's the only remaining set containing the uncovered element

Step 7 price $(e_6) = 1$

Step 8 $C = \{S_1, S_3, S_4\}$

Step 9 $S' = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

Step 10 **end (while)**

Step 11 $C = \{S_1, S_3, S_4\}$

Hence, based on the assumptions that each subset has a cost associated with it, actually 1-unit cost in this case, we get a minimum cost set-cover of 3-units in either of the selections we take at step 5 in the second while loop. But, if we were associated different costs for each subset, the result would be different from what we have here.

The above procedures (Algorithm 1, 2 and 3) terminate with a valid set cover because in each of the iterations in the algorithms we have found a sub-cover and the set E is finite.

Linear programming plays a central role in the design and analysis of heuristic algorithms. Below are discussed some techniques which uses the theory of integer and linear programming for obtaining heuristic algorithms for the SCP.

4.1.3. Set cover via LP-rounding

After the linear programming relaxation of the SCP one way of converting it into an integral solution is to round all non-zero variables to 1. The *LP-rounding* works as follow, given the *LP-relaxation* solution x^*, we include subset S_j in our solution if and only if $x_j^* \geq \frac{1}{\text{f}}$, where f the maximum number of sets in which any element appears. That is, $\text{f} = \max_i |\{j: e_i \in S_j\}|$.

In effect, we round the fractional solution x^* to an integer solution $\hat{x}$ by setting $\hat{x}_j = 1$ if $x_j^* \geq \frac{1}{\text{f}}$, and $\hat{x} = 0$ otherwise.

Algorithm 4 SET-COVER via *LP-ROUNDING*

1. Find all optimal solutions to the *LP-relaxation*
2. Pick all sets S_j for which $x_j^* \geq \frac{1}{\text{f}}$ in this solution.

Let I denote the indices j of the subsets in this solution and let $\text{f}_i = |\{j: e_i \in S_j\}|$, the number of sets in which element e_i appears, $i = 1, \dots, m$; then $\text{f} = \max_i \text{f}_i$.

Theorem 4.1 The collection of subsets S_j, $j \in I$ in the above algorithm 4, is a set cover.

Proof: considering the solution provided by the algorithm, we call an element e_i covered if the solution contains a subset containing e_i, then we need to show that each element e_i is covered. Since the *LP-relaxation* solution x^* is feasible solution, we know that $\sum_{j:e_i \in S_j} x_j^* \geq 1$ for e_i and by the definition of f_i and f, there are $f_i \leq f$ terms in the summation, so at least one term must be at least $\frac{1}{f}$ .Thus for some j such that $e_i \in S_j$ we have $x_j^* \geq \frac{1}{f}$. Therefore $j \in I$, and element $e_i$is covered.

Example 4.4 Let $E = \{e, f, g\}$ and its subsets be $S_1 = \{e, f\}, S_2 = \{f, g\}, S_3 = \{e, g\}$, each of unit cost.

To see how *LP-rounding* works, we first solve the *LP-relaxation* of the problem and we will get a solution $x^* = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. The maximum number of any element in the subsets is equal to $f = 2$. Hence, as $x_1^* \geq \frac{1}{2}, x_2^* \geq \frac{1}{2}$ and $x_3^* \geq \frac{1}{2}$ according to algorithm 4 our integer solution $\hat{x}$ will be $\hat{x} = (1,1,1)$ with a cost of 3 units. But, an OPT must pick two of the sets for a cost of 2.

4.1.4. Set Cover via Dual-rounding

It will be useful to consider the dual of a linear programming relaxation of a problem. We can obtain by introducing a variable y_i corresponding to each element $e_i \in E$ its dual formulation for the SCP using the usual known technique as follows:

$$\begin{aligned} &\text{Maximize} && \textstyle\sum_{i=1}^{m} y_i \\ &\text{Subject to} && \textstyle\sum_{i:e_i \in S_j} y_i \leq c_j \quad j = 1, \dots, n \qquad \dots \quad (9) \\ & && y_i \geq 0, \qquad i = 1, \dots, m \end{aligned}$$

One useful step when using dual programs for designing *(heuristics)* algorithms is to give *"meaning"* or *"interpretation"* for the dual solution.

We can interpret the dual, equation (9) as follows: we want to put things into the sets such that none of the sets are *"over packed"*, that is, the sum of the costs of elements in a set is at most the cost of the set.

Hence the information from the dual LP-relaxation can be used for derivation of good heuristic algorithm for the SCP as outlined below.

Let y^* be an optimal solution to the dual-LP, equation (9). Then form a solution by choosing all subsets for which the corresponding dual inequality is *tight*; that is, the inequality is met with equality for subset S_j, and $\sum_{i:e_i \in S_j} y_i^* = c_j$.

Let I' denote the indices of the subsets in this solution.

Theorem 4.2 The collection of subsets $S_j,\ j \in I'$ is a set cover.

Proof: suppose that there exists some uncovered element e_k. Then for each subset S_j containing e_k, it must be the case that

$$\sum_{i:e_i \in S_j} y_i^* < c_j$$

Let ϵ be the smallest difference between the right-hand side and the left-hand side of all constraints involving e_k; that is, $\epsilon = \min_{j:e_i \in S_j} \left(c_j - \sum_{i:e_i \in S_j} y_i^* \right)$. By the above inequality, we know that $\epsilon > 0$. Now, considering a new dual solution y' in which $y'_k = y_k^* + \epsilon$ and every other components of y' is the same as in y^*. Then y' is dual feasible solution since for each j such that $e_k \in S_j$,

$$\sum_{i:e_i \in S_j} y'_i = \sum_{i:e_i \in S_j} y_i^* + \epsilon \leq c_j,$$

by the definition of ϵ.

For each j such that $e_k \notin S_j$,

$$\sum_{i:e_i \in S_j} y_i' = \sum_{i:e_i \in S_j} y_i^* \leq c_j,$$

as before. Furthermore, $\sum_{i=1}^{m} y_i' > \sum_{i=1}^{m} y_i^*$ this contradicts the optimality of y^*. Thus, it must be the case that all elements are covered and I' is a set cover.

4.1.5. Set Cover via the Primal-Dual Schema

Primal-Dual schema is another method for designing heuristic algorithms for the SCP using linear programming. The overview of the algorithm is outlined as follow:

- Pick a primal infeasible solution $\bar{x}$, and a dual feasible solution $\bar{y}$, such that the slackness conditions (primal and dual) are satisfied for chosen β and γ.
- Iteratively improve the feasibility of $\bar{x}$ (integrally) and the optimality of $\bar{y}$, such that the conditions remain satisfied, until $\bar{x}$ become feasible.

To write the algorithm as in the previous sections, we need the following definition and formulations:

Put $\beta = 1$ and $\gamma = \mathfrak{f}$, where $\mathfrak{f}$ is the frequency of the most frequent element.

Definition 4.1 Set S_j is called *tight* if $\sum_{i:e_i \in S_j} y_i^* = c_j$, $\forall j$

And we have the following facts from the primal and dual slackness conditions;

1. **Primal conditions** "pick only tight sets in the cover" if:

$$\forall S_j \in \mathcal{F}: x_j > 0 \Rightarrow \sum_{i:e_i \in S_j} y_i^* = c_j, \forall j$$

2. **Dual conditions** "each $e_i, y_i > 0$, can be covered at most $\mathfrak{f}$ times" if:

$$\forall e_i: y_i > 0 \Rightarrow \sum_{j=1}^{n} x_j \leq \mathfrak{f}$$

Now, we can write the primal-dual algorithm for the SCP in formal way

Algorithm 5 SET COVER via Primal-Dual

1. Initialization: $\bar{x} \leftarrow 0, \bar{y} \leftarrow 0$; (i.e. set $\forall S_j,\ x_j = 0,\ \forall e_i, y_i = 0$)
2. **While** (all elements are not covered) **do**

   Pick an uncovered element e_i, and rise y_i until some set goes tight.

   Pick all tight sets in the cover and update $\bar{x}$ (i.e. set $x_j = 1$ for those tight sets S_j)

   Declare all elements occurring in these sets as "covered".
3. **Output** the set cover $\bar{x}$

Example 4.5 Let $E = \{1,2,3,4,5,6,7,8\}$ and its subsets be $S_1 = \{1,2,3,6\},\ S_2 = \{2,4,5,8\}, S_3 = \{1,3,5\},\ S_4 = \{2,5,7\},\ S_5 = \{1,6,7,8\},\ S_6 = \{1,4,6\},\ S_7 = \{6,7,8\}$, and $S_8 = \{2,4,6,7,8\}$ with their respective cost $c_1 = 4,\ c_2 = 6,\ c_3 = 1,\ c_4 = 3,\ c_5 = 9,\ c_6 = 2,\ c_7 = 10$ and $c_8 = 7$. Now, applying the above algorithm we have the following:

i. Initially we have $x = (0,0,0,0,0,0,0,0)$ and $y = (0,0,0,0,0,0,0,0)$

ii. For the while loop we have the table:

iteration	e_i (uncovered elements)	y_i	tight set(s)	covered elements
1	e_1	$y_1 = 1$	S_3	e_1, e_3, e_5
2	e_2	$y_2 = 3$	S_1, S_4	e_2, e_6, e_7
3	e_4	$y_4 = 1$	S_6	e_4
4	e_8	$y_8 = 2$	S_2	e_8

iii. The solution is $x = (1,1,1,1,0,1,0,0)$.

Hence, the cover found is S_1, S_2, S_3, S_4, S_6 with a total cost of 16.

4.2. Exact algorithms

4.2.1. Branch and Bound (B&B)

Branch and Bound (B&B) is by far the most widely used tool for solving large scale $\mathcal{NP}$-*hard* combinatorial optimization problems. In the first step, the linear programming relaxation of the SCP (obtained by removing the integrality constrained in the x_j variables) is solved using any available LP-solvers (algorithms e.g. simplex). The resulting optimal solution value is then a lower bounding on the value of the optimal integer solution. If we do not have any feasible integer solution we will take a sub-problem and branch on one of its variables x_j, $j = (1, \dots, n)$ based on the following branching and bounding strategies:

i. **Branching:** choose an active sub-problem; that is, the one we have not chosen before, and
ii. **Bounding:** choose the sub-problem with the lowest objective value.

Now we can apply the branch and bound process by means of the following algorithm:

ALGORITHM 6 THE SET COVER via Branch and Bound

1. Solve the LP-relaxation of the original problem (P_0); using the simplex algorithm.
2. **If** the solution is integer, **STOP**.
   else create two new sub-problems (P_k and P_l) by branching on a fractional variable x_j , from the last optimal table, with the highest fraction (if the fraction parts are equal take any selection)
3. Solve each of the new sub-problems by adding additional constraint $x_j = 0$ and $x_j = 1$ to P_k and P_l, respectively.
4. Select a sub-problem based on the branch and bound strategy. **Repeat** until there are no active sub-problems.

Example 4.6 Let $E = \{e, f, g\}$ and the subsets be $S_1 = \{e, f\}, S_2 = \{f, g\}, S_3 = \{e, g\}$ each of unit cost. The linear relaxation solution is $x_1 = 1/2, x_2 = 1/2, x_3 = 1/2$ with value $3/2$. And we know that (by the argument in section 3.2) for this minimization problem no integer solution

will have a value less than $3/2$. Unfortunately, since our solution $x = (x_1, x_2, x_3)$ is not integer, we do not have an integer solution yet.

Since the fractional part of each points of our solution is equal, we can select any of one of them, say $x_1 = 1/2$. We want x_1 to be integer. So we branch x_1, creating two sub-problems. In one, we will add the constraint $x_1 = 0$. In the other, we add the constraint $x_1 = 1$.

If we solve the linear relaxations of the sub-problems, we get from the last optimal tables of the simplex algorithm the following solutions:

- $x_1 = 0$: Objective value 2, $x_1 = 0,\ x_2 = 1,\ x_3 = 1$;
- $x_1 = 1$: Objective value 2, $x_1 = 1,\ x_2 = 1,\ x_3 = 0$

At this point, we know that the optimal integer solution is no less than $3/2$ (actually it is greater than or equal to 2). But, we have here two integer solutions with the same objective value 2. Hence we stop further branching and our integer solutions are $x = (0,1,1)$ or $x = (1,1,0)$. Besides to this, if our branching strategy were selected a different variable other than x_1 we would have another solution $x = (1,0,1)$. Therefore, we have three possibilities for minimal cost set cover.

4.2.2. A Cutting Plane Method

Definition 4.1 A cutting plane for an Integer LP problem is a linear constraint which does not exclude any integer feasible point.

After the LP-relaxation (simplex algorithm) fractional solution to P, the primal problem of SCP, we can generate a set of constraints that will cut-off this fractional solution, which is called the cutting plane approach. Then, adding these constraints to our SCP formulation, we can resolve the linear program, and if it's integer, we have found the optimal integer solution. Otherwise, if it's still fractional, then we continue generating constraints and resolving the linear program until an integer solution is possibly found. Below is discussed the cutting-plane method of Gomory.

Algorithm 6 GOMORY-CUTTING PLANE

Step 0: If the LP-relaxation (simplex algorithm) has no feasible solution. **STOP**, no solution

Step 1: solve problem P', (i.e. the LP-relaxation of P)

Step 2: If the optimal solution to P' is integer, **STOP**

else decompose the variables x_j of the solution to P' in the following way:

$$x_j = [x_j] + h_j \ , \quad 0 \leq h_j < 1$$

(this is only for those variables which are fractional)

Determine the index t such that $h_t = \max\{h_j : j = 1, \dots, n\}$. If t is not unique, then choose the smallest index with this property.

Step 3: decompose all a_{tj} according to:

$$a_{tj} = [a_{tj}] + h_{tj} \ , \quad 0 \leq h_{tj} < 1$$

Step 4: form the Gomory-constraint

$$x_G - \sum_{i=n+1}^{m} h_{ti} x_i = -h_t$$

Step 5: Add the Gomory-constraint and solve the linear program P' (using dual-simplex algorithm). If the problem P' has no solution, then there is no integer solution, **STOP**. Otherwise Go to **Step 2**.

The key success to this method is to be able to efficiently generate constraints that separate fractional solutions from all integer solutions, **step 2**.

The above two algorithms gives us exact solutions. But there will be a difficulty if our problem instance is very large; that is our incidence matrix may have large dimensions. However, these algorithms are still very important if we are able to reduce the dimension of our incidence matrix using heuristics like the MRA with some partial solutions. Then the remaining sub-problem can be solved using either of the algorithms.

5. Analysis of algorithms

Definition 5.1 a ϕ-approximation algorithm for an optimization problem is a polynomial time algorithm that for all instances of the problem produces a solution whose value is within a factor of ϕ of the value of an optimal solution.

For a ϕ-approximation algorithm, we will call ϕ the performance guarantee of the algorithm.

5.1. Matrix reduction analysis

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 or more column from the original incidence matrix. Thus there is no performance guarantee for the matrix reduction algorithm

5.2. Greedy algorithm analysis

Let's begin the analysis of the greedy algorithm with the presentation of Chavatal's [9] result which is a generalization of the SCP to the *non-unit-cost* case, in which each sets of the given collection $\mathcal{F}$ has an associated cost and we wish to find a cover with the minimum total cost.

Before we state the important theorem of Chavatal's, it's better to observe the following:

Lemma 5.1 For each $i \in \{1,2,\dots,m\}$, $price(e_i) \leq OPT/(m-i+1)$. where e_i is the i^{th} element covered.

Proof: Numbering the elements of U (noting that U represents the current set of uncovered points in the greedy algorithm) in the order in which they were covered by the algorithm. Let $e_1, e_2, \dots, e_m$ be this numbering and $O_1, O_2, \dots, O_p$ be the optimal sets. Then we have,

$$OPT = c(O_1) + c(O_2) + \cdots + c(O_p) \qquad \dots \quad (10)$$

Now assuming the greedy algorithm has covered the elements in S' so far. Then the uncovered elements, or $|U - S'|$, are at most the intersection of all of the optimal sets intersected with the uncovered elements:

$$|U - S'| \leq |O_1 \cap (U - S')| + |O_2 \cap (U - S')| + \cdots + |O_p \cap (U - S')| \quad \ldots \quad (11)$$

In the greedy algorithm, we select the set with the smallest cost effectiveness α, where:

$$\alpha \leq \frac{c(O_k)}{|O_k \cap (U - S')|}, k = 1,2, \ldots, p \quad \ldots. \quad (12)$$

The above result in equation (12) is true because the greedy algorithm will always choose the set with the smallest cost effectiveness, which will either be smaller than or equal to a set that the optimal algorithm chooses.

Hence, from the three results, $(10) - (12)$ we get:

$$c(O_k) \geq \alpha . |O_k \cap (U - S')|$$

which implies,

$$OPT = \sum_k c(O_k) \geq \alpha . \sum_k |O_k \cap (U - S')| \geq \alpha . |U - S'|$$

$$\therefore \quad \alpha \leq \frac{OPT}{|U - S'|},$$

Hence, since $|U - S'| = m - (i - 1)$ and $\alpha = price(e_i)$, $price$ of the i^{th} element is:

$$price(e_i) \leq \frac{OPT}{m - (i-1)} = \frac{OPT}{m - i + 1}$$ ■

Theorem 5.1 The greedy set cover algorithm (algorithm 2) is a $H(m)$ factor approximation algorithm for the minimum cost set cover problem, where $H(m) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{m}$.

Proof: Since the cost of each set picked is distributed among the new elements covered, the total cost of the set cover picked is equal to:

$$\sum_{e_i \in U} price(e_i) = cost\ of\ the\ greedy\ algorithm \quad \ldots \quad (13)$$

Putting the result of the above lemma and (13) together, we get the total cost of the set cover:

$$\sum_{i=1}^{m} price(e_i) \leq \sum_{i=1}^{m} \frac{OPT}{m-i+1} = OPT.\left(1+\frac{1}{2}+\cdots+\frac{1}{m}\right) = OPT.H(m) \qquad \blacksquare$$

The algorithmic factor bounds the ratio of the cost of the solution obtained by the greedy heuristic to the cost of the optimal solution.

Example 5.1 let the universe $X = \{1,2,3,\ldots,12\}$ and its subsets $Y = \{1,2,3,4,5\}$, $W = \{1,2,3,6,7\}$, and $Z = \{4,5,8,9,10,11,12\}$ with their respective cost $c(Y) = 6$ units, $c(W) = 15$ units and $c(Z) = 7$ units.

If we use algorithm 3, we will get a total cost of 28 units, but the greedy algorithm is not optimal in this case because an optimal solution would have chosen W and Z for a cost of 22 units.

5.3. LP-rounding analysis

The approximation guarantee is established by comparing the cost of the integral and fractional solutions.

Theorem 5.2 The *LP-rounding* is an ƒ-approximation algorithm for the SCP. [5][23]

Proof: It is clear that the algorithm runs in polynomial time. By the construction the algorithm in section 4.1.3, we have $1 \leq ƒ.x_j^*$ for each picked set S_j and from this, and the fact that each term $ƒ.c_j.x_j^*$ is nonnegative; we see that,

$$\sum_{j=1}^{n} c_j \leq \sum_{j=1}^{n} c_j.(ƒ.x_j^*)$$

$$= ƒ.\sum_{j=1}^{n} c_j x_j^*$$

$$= ƒ.z_{LP}^*$$

$$\leq ƒ.OPT$$

Where the final inequality follows from the argument in section 3.2 that, $z_{LP}^* \leq z_{IP}^* = OPT$. ■

5.4. Dual-rounding analysis

In order to establish the approximation guarantee, the cost of the solution produced by the algorithm needs to be compared with the cost of an optimal solution.

Theorem 5.3 The dual-rounding algorithm is an f-approximation algorithm for the set cover problem.

Proof: The central idea is the following "*charging*" argument: when we choose a set S_j to be in the cover we "*pay*" for it by charging y_i^* to each of its elements; each element is charged at most once for each set that contains it (and hence at most f times over all the subsets), and so the total cost is at most $f.\sum_{i=1}^{m} y_i^*$, or f times the dual objective function.

More formally, since $j \in I'$ only if $c_j = \sum_{i:e_i \in S_j} y_i^*$,

Then we have the cost of the set cover I' is:

$$\begin{aligned}\sum_{j \in I'} c_j &= \sum_{j \in I'} \sum_{i:e_i \in S_j} y_i^* \\ &= \sum_{i=1}^{m} \left|\{j \in I' : e_i \in S_j\}\right| . y_i^* \\ &\leq \sum_{i=1}^{m} f_i y_i^* \\ &\leq f \sum_{i=1}^{m} y_i^* \quad \text{, since } f \leq f_i \\ &\leq f . OPT\end{aligned}$$

The second equality follows from the fact that when we interchange the order of summation, the coefficient of y_i^* is equal to the number of times this term occurs overall. The final inequality follows from the weak duality property. ■

5.5. Primal-Dual analysis

An approximation guarantee of $\beta\gamma$ is achieved using the primal-dual schema, since

$$\sum_{j=1}^{n} c_j x_j \leq \beta\gamma \sum_{i=1}^{m} b_i y_i \leq \beta\gamma . z_{LP} \leq \beta\gamma . OPT$$

Where, z_{LP} is LP-relaxation optimal solution of the SCP and the inequalities holds true by proposition 2.4 and the LP-duality theorem.

Theorem 5.3 The primal-dual algorithm (algorithm5) is an f–approximation algorithm for the set cover problem.

Proof: There will be no uncovered elements and no over-packed sets at the end of the algorithm. Thus, the primal and dual solutions will both be feasible since they satisfy the primal and dual complementary slackness conditions with $\beta = 1$ and $\gamma =$ f. Hence by proposition 2.4 and the above argument the approximation is f. ■

6. Applications

The set-covering problem has many practical and useful applications. Successful real-world applications of SCP are plenty, which can be classified in three major areas (1) *facility location* (2) *scheduling or staffing of personnel*, and (3) *dispatching trucks or vehicles to routes or customers* [14]. For each area a sample applications are listed below:

6.1. Facility location

- Determine the optimal location for a new fire station that can cover a given set of dispersed subdivisions, taking into account the average response time from a fire station to a fire in each subdivision.
- Determine the least number of new supermarkets to be built to cover a number of geographically dispersed communities, taking into consideration the distance restriction and concentration of populations
- Determine which subset of a given number of potential transmission towers to constructed that can cover a given number contiguous geographical communities, taking into account their budgeted construction costs and maximization of potential population to be served
- (*information retrieval*) There are n files $S_1, S_2, \dots, S_n$ 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 minimum total length such that searching these will retrieve all the requested information.

6.2. Scheduling or staffing of personnel

- Given a set of scheduled flights and a set of "preferred" flight crews, the staffing problem is to identify a subset of crews to cover all flights at a minimal cost.
- Determine the minimal number of patrol officers required to cover a given set of areas in Addis Ababa, taking into account response times

6.3. Dispatching trucks/ vehicles to routes/customers

- Determine emergency medical service vehicle deployment in Addis Ababa
- Minimize the number of vehicles to meet a fixed periodic schedule

7. Concluding Remarks

- The MRA is heuristic for the SCP and gives us a valid set cover. But this is not guaranteed unless the reduced matrix is empty. However, it is a very important algorithm that helps us to produce a partial solution, in case of non empty reduced matrix, then the reduce instance of the problem can be solved efficiently with another algorithm. Hence we can solve large instance of the SCP in combination with MRA.
- The algorithm GREEDY-SET-COVER described in Section (4.2) is an approximation algorithm to the set covering problem, with a logarithmic approximation ratio (noting the fact that $H(m) \approx \ln(m)$).
- Since they are typically very easy to implement, the greedy algorithm and its variants stated in different literatures can be used to solve large-scale set covering problems almost in shorter time bound. However, their myopic nature may easily yield solutions far from optimality.
- The exact approaches like branch and bound, cutting planes and dual heuristic approaches solve the SCP optimally but they take a lot of time and therefore, they cannot be applied to large-scale instances of the SCP if the solution is needed in real time. However, we can use the combination of the algorithms to get improved feasible solutions.
- The heuristic approaches though they provide only approximate solutions, they are computationally efficient. This make them better candidates for solving large-scale instances of the SCP, as they obtain near optimal solutions in real time [19].
- The *primal-dual* gave an approximation algorithm for the SCP. Although it did not give a better performance guarantee than the *LP-rounding* algorithm, but in practice the *primal-dual* method gives much faster algorithms than those that require solving a linear program.[23]

8. Programming

A MatLab programming for generating a minimal set cover for the greedy heuristic algorithm (non-unicost) was implemented based on the outlines of the algorithm in section 4.1 as shown below.

```
function J = greedyscp(E,F)

% JANUARY, 2011
% GREEDY-SET-COVER FOR UNICOST
% Samuel Abebe, Department of Mathematics, Optimization stream
% this code will take the universe set E and its family of subsets F such that their union initially
% covers E
% function [index of the selected sets] = GREEDYSCP (ELEMENTS, SUBSETS)
% the universe set E should be provided in vector form of integer entries from 1 to n, where n is
% the number of elements to be covered and the subset F should be provided in array form with
% entries in vector form.
% for example, E = [1 2 3 4 5] and F = {[1 3 5], [2 4 5], [2 4 5], [1 5]}
% then, the program will provide the selected subsets listed according to their index in F.
% if two subsets have the same cardinality at some step the algorithm will
% chooses the entry with the lowest index in F.


  U = E;        % the set points that are currently uncovered

  J = [ ];      % the indices of subsets currently selected (empty)

  NumberSubsets = numel (F);                % Number of subsets of E

  SetsCardinalities = cellfun (@numel,F);   % the cardinality of each subset

  % Labels each subsets according to their list in F
  LebelSubsets = [1: NumberSubsets];

  % to check wether each sub-set contains an element in E
     Y = [ ];
  for i=1:NumberSubsets
     X = intersect(U,F{i});
     Y = union(Y,X);
     sortY = sort(Y);
  end

  if length(U)==length(sortY)   % check if the sub-sets initially form a cover
    disp('The Minimal Set-Cover will choose:')
```

```
  % The Main routine of the algorithm

 while ~isempty(U) % continue untill the set of uncovered elements is empty
   %select the subset with maximum cardinality
    [SelectedSet,LebelSubsets] = max(SetsCardinalities);
    U = setdiff(U,F{LebelSubsets});% remove currently covered elements
    J = union(J,LebelSubsets);% selects the index of the selected subset
    F{LebelSubsets}=[];% removes the selected subset
    % create an empty cell array for the remainig subsets containing the
    % currently uncovered elements
    V = cell(1,NumberSubsets);
    for i=1:NumberSubsets
        V{i}=intersect(U,F{i}); % put in each cell the remaining
        % subsets containg the uncovered elements
    end
    F = V; % the new subsets containing the uncovered elements
    SetsCardinalities = cellfun(@numel,F); % cardinality of the new subsets

 end
else
   disp('ERROR:The input sub-set can not be a cover !! ')
   return;
end
```

References

1. S. DASGUPTA, C.H. PAPADIMITRIOU and U.V. VAZIRANI *(2006) Algorithm,* McGraw-Hill Higher Education. pp. 247-297.
2. A. CAPRARA, P. TOTH and M. FISCHETTI (2000) *Algorithms for the set covering problem,* Annals of Operation Research, pp. 353-354.
3. K. HOFFMAN and M. PADBERG (2006) *Set Covering, Packing and Partitioning Problems* unpublished article.
4. S. ARORA and B. BARAK (2007) *Computational Complexity: A Modern Approach,* Web draft, pp. 37-38, 50-52,58.
5. V. VAZIRANI (2010) *Approximation Algorithms.* pp. 16-17,100-121
6. H. GORMEN, E. LEISERSON and C. STEIN (2001) *Introduction to Algorithms,* McGraw-Hill Book Company.
7. J.E. BEASLEY and K. JØNSTEN (1992) *Enhancing an algorithm for set covering problems,* European Journal of Operation Research, pp. 293-300
8. G. LAN, W. DEPUY and E. WHITEHOUSE (2007) *an effective and simple heuristic for the set covering problem,* European Journal of Operation Research, 1387-1403.
9. V. CHAVATAL (1979) A *Greedy Heuristic for the Set-Covering Problem,* Mathematics of Operation Research, Vol. 4, No. 3., pp. 233-235
10. C. LUNDA and M. YANNAKAKIS (1994), *On the Hardness of Approximating Minimization Problems,* Journal of the Association for Computing Machinery, Vol. 41, No. 5, pp. 960-981.
11. THOMAS A. FEO. and MAURICIO G.C. RESENDE (1989) *A Probabilistic Heuristic for a Computationally difficult Set Covering Problem,* Operation Research letters 8 pp. 67-71.
12. ELIA EL-DARAZI and GAUTAM MITRA (1992) *Solution of Set-Covering and Partitioning Problems using Assignment Relaxation,* J. Operation Research Society, Vol. 43, No. 5, pp. 483-493
13. O. GOLDREICH (2010) *The basics of computational complexity,* Cambridge University Press pp. 114-115
14. DER-SAN CHEN, ROBERT G. BATSON and YU DANG **(**2010) *Applied Integer Programming***:** *Modeling and Solution,* John Wiley and Sons,
15. LARSON and ODONI. Urban Operations Research. Internet edition.

16. M. AFIF, M. HIFI, V, TH. PASCHOS and V. ZISSIMOPOULOS (1995) *A New Efficient Heuristic For the Minimum Set Covering Problem,* Journal of the Operational Research Society vol. 46, pp.1260-1268
17. M. HAOUARI and JS. CHAOUARI (2002) *A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem,* Journal of Operational research Society, 53, pp. 792-799.
18. J.E BEASLEY and P.C. CHU (1996) *A genetic algorithm for the set covering problem.* European Journal of Operational Research, 94, pp. 392-404.
19. K.S AL-SULTAN, M.F. HUSSAIN and J.S. NIZAMI (1996) *A genetic algorithm for the set covering problem.* Journal of Operational Research Society. 47, pp. 702-709.
20. E. BALAS and M.C. CARRERA (1996) *A Dynamic Sub-gradient-Based Branch-And-Bound Procedure for Set Covering*, INFORMS, Vol. 44, No. 6, pp. 875-890.
21. G.L NEMHAUSER and L.A WOLSEY (1999) *Integer and Combinatorial Optimization.* John Wiley & Sons.
22. P. TOTH and D. VIGO (2002) *The Vehicle Routing Problem (set-covering based algorithms for the capacitated VRP).* SIAM monographs on discrete mathematics & applications, pp. 85-108
23. D.P WILLIAMSON and D.B SHMOYS (2010) *The Design of Approximation Algorithms,* Cambridge University Press, pp.16-33, 161-164