SURVEY OF APPROXIMATION ALGORITHMS FOR SET COVER PROBLEM

Himanshu Shekhar Dutta

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2009

Dutta, Himanshu Shekhar. Survey of Approximation Algorithms for Set Cover Problem. Master of Science (Computer Science), December 2009, 76 pages, 10 figures, 1 table, references, 29 titles.

In this thesis, I survey 11 approximation algorithms for unweighted set cover problem. I have also implemented the three algorithms and created a software library that stores the code I have written.

The algorithms I survey are: 1. Johnson's standard greedy 2. f-frequency greedy 3. Goldsmidt, Hochbaum and Yu's modified greedy 4. Halldorsson's local optimization 5. Dur and Furer semi local optimization 6. Asaf Levin's improvement to Dur and Furer 7. Simple rounding 8. Randomized rounding 9. LP duality 10. Primal-dual schema 11. Network flow technique.

Most of the algorithms surveyed are refinements of standard greedy algorithm. Till now the best known algorithm has a performance ratio of $H(k) - \frac{196}{390}$.

ACKNOWLEDGEMENTS

My sincere thanks and regards to my advisor Dr. Farhad Shahrokhi for his constant support and guidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Set cover is an optimization problem which serves as a model to many real world problems. Its application includes wireless networks, semiconductor industry, flexible manufacturing, scheduling, routing etc..

One of the major application of set cover problem is in wireless network [1,2]. Take an example, where a set of customer locations have been provided. Assuming unit disk is geometric modeling of coverage area of base stations. The aim is to find the best locations for placing base stations, so that all the customers are serviced. Since placing base station also costs money, attempt is made to minimize the number of base stations, ensuring all the customers are serviced by the minimum collection of base station. Ideally, each base station should provide service to as many customers as possible. This is essentially a set cover problem in geometric settings.

Here n customer locations correspond to n points of the plane or elements of a universal set. $m \leq \binom{n}{2}$ possible base station or disk placements in the plane corresponds to a family of m sets [1,2]. The aim is to find minimum number of disks (or sets) in family of possible m disks (or sets), so that all the customer locations (or all the elements of universal set) are covered.

In the example shown in Fig-1.1, there are 22 points in the plane. The aim is to cover all these points with minimum number of disks. Consider applying a simple greedy heuristics which at each iteration selects a disk that covers maximum number of uncovered points. This algorithm selects the 3 disks shown in the boldface. At iteration-1, iteration-2 and iteration-3 it selected disks that cover 12, 6, 4 points respectively. This is not an optimal solution. The optimal solution is shown by two dashed disks. Each of the dashed disks covers 11 points each.

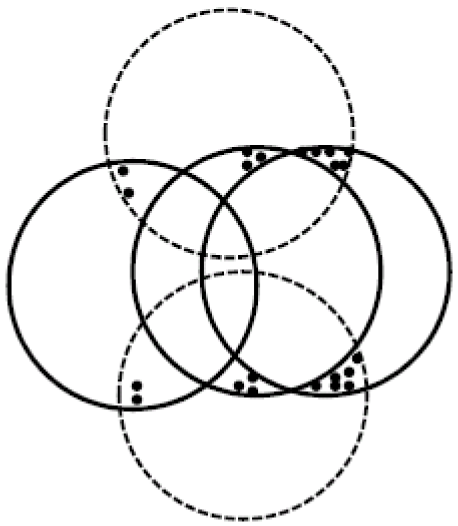
Fig-1.1 Set Cover in Geometry

Fig-1.2 shows another application of set cover problem. Here a set of unit rectangles are given and the aim is to hit all the rectangles with minimum number of points. This problem is relevant in setting up emergency facilities so that all the potential customers are within the coverage area of emergency facility [2]. Unit rectangles represent a set of customers (or cities) which need emergency facilities. This problem is known as hitting set problem which can be solved by transforming the hitting set problem to set cover problem and then applying any of the available algorithms for set cover problem. Clearly 4 such facilities should be enough for this example.

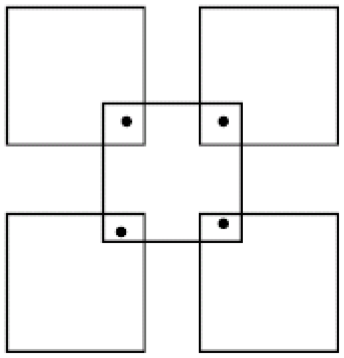

Fig-1.2 Hitting Set in Geometry

1.2 Preliminaries

Many combinatorial optimization problems are NP-hard, i.e., it is highly unlikely that a polynomial-time algorithm exists for solving the problem, unless P = NP [3,4]. Hence when one faces solving an NP-hard problem one should focus on finding near optimal solution in polynomial time [3,4,5,6].

Approximation algorithm is an algorithm that returns near optimal solution [3]. For maximization problems, the performance ratio of an approximate algorithm is the worst case ratio of the size of the optimal solution to the size of the approximate solution. For minimization problems, performance ratio is the inverse of this ratio, i.e., worst-case ratio of the size of the approximate solution to the size of the optimal solution.

In set cover problem, a universal set and collection of subsets are provided, such that each element of universal set is incident to at least one of the subsets [8,9,10,11,12]. The aim is to find minimum number of subsets, whose union is universal set. Basically, attempt is to cover all the elements of the universal set. For example, a company requires at least a programmer, to cover its programming need of different languages, example C, C++, JAVA, SQL etc.. There are people who know multiple languages. The aim is to find minimum number of people so that for every language there is at least one person who knows it, i.e., find minimum number of people so that all the languages are covered.

More precisely,

Let U, $|U| = m$ be a universal set and a family of subsets $S \subseteq 2^U$

A sub collection C ($C \in S$) covers the universal set U, if

$$U = \bigcup_{C \in S} C$$

Minimum set cover is sub collection C of minimum cardinality.

Example-1.1: Find the set cover for following collection of sets.

$U = \{1, 2, 3, 4\}$

$S_1 = \{1, 2\}$

$S_2 = \{2, 3\}$

$S_3 = \{3, 4\}$

$S_4 = \{4, 1\}$

S_1 and S_3 form the minimum set cover of size 2. Minimum set cover is not unique as S_2 and S_4, too, form the set cover of size 2.

Dual of set cover problem is known as hitting set problem [12]. In hitting set problem, a family of sets *S*, |*S*| = *m* are given. The aim is to create a subset *H* such that *H* has at least one element in each of the sets, i.e., every set has non-empty intersection with *H*. Minimum hitting set is creating a set with minimum cardinality.

$$H \bigcap_{s \subseteq U} S \neq \varnothing$$

For collection of sets provided in Example-1.1, resulting hitting set would be:

H = {1, 3}

This is non-unique, as {2, 4} is another hitting set of size 2.

Set packing is a sub collection P' ($P' \in S$), where all the members of this sub collection are mutually disjoint [12]. Maximum set packing is the sub collection of maximum cardinality. In the above example, both S_1 and S_3 or S_2 and S_4, form the maximum set packing of size 2.

An interesting concept related to set cover is that of an edge cover [7]. Edge cover of an undirected graph $G = (V,E)$ is a subset $E' \subseteq E$ of edges such that every vertex of *V* is incident to some edge of E'. Minimum edge cover is one with minimum cardinality. It is important to note that minimum edge cover problem is special instance of the set cover problem, where each subset has maximum cardinality of two. In fact some of the algorithms surveyed in this thesis use the concept of edge cover together with greedy approach, to get better performance guarantee.

In the Fig-1.3, edges ab and cd form the minimum edge cover, of cardinality 2, where G has a path length of 4.

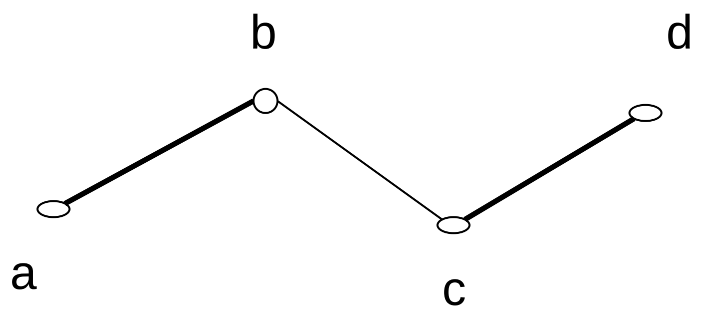

Fig-1.3 Example Illustrating Edge Cover and Maximum Matching

A matching in a graph G is a set of non loop edges with no shared points [7]. Maximal matching is a matching which can not be extended by adding any edge. Maximum matching is a matching of maximum cardinality. Perfect matching covers all the nodes of the graph [assumption - there are even number of vertices].

In the Fig-1.3, edge bc form the maximal matching. If this edge is selected in the matching set, then no other edge can be added. But clearly this is not the maximum matching. Edges ab and cd, form the maximum matching of cardinality 2. Incidentally this is also the edge cover and therefore it also represents prefect matching.

Maximum matching is special instance of the set packing problem, where each subset has maximum cardinality of two. For a graph *G* with *n* vertices having no isolated vertex, minimum edge cover and maximum matching, sum to the total number of vertices.

In a given undirected graph $G = (V, E)$ a vertex cover is a subset $V' \in V$ such that for every edge $uv \in E$, at least one of the end point or vertex belongs to V'. The minimum vertex cover is subset V' of minimum cardinality [3].

In the Fig-1.4, vertex a and d form the minimum vertex cover. All the edges are incident to these 2 vertices.

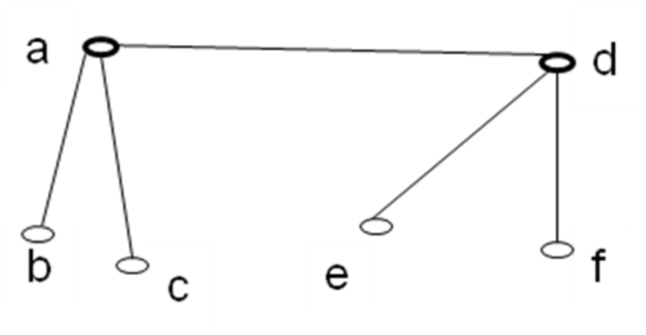

Fig-1.4 Example Illustrating Vertex Cover

Minimum vertex cover problem is a special instance of the dual set cover problem, where each subset has maximum cardinality of two.

CHAPTER 2

APPROXIMATION ALGORITHMS BASED ON STANDARD GREEDY

As described earlier, in set cover problem, a universal set and collection of subsets are provided, such that each element of universal set is incident to at least one of the subsets [8,9,10,11,12]. The aim is to find minimum number of subsets, whose union is universal set. Basically, the attempt is to cover all the elements of the universal set. Set cover problem has practical applications in many areas, such as logic design, semi conductor industry, fault testing etc..

More precisely, let U, $|U| = m$ be the ground set (or universal set) and S be a family of subsets $S \subseteq 2^U$. A sub collection C ($C \in S$) covers the universal set U, if

$$U = \bigcup_{C \in S} C$$

In this chapter, several approximate solutions to set cover problem will be reviewed which are based on Johnson's standard greedy algorithm.

2.1 Johnson's Standard Greedy Algorithm for Set Cover [8]

Greedy approximation of SC was originally presented by Johnson in his paper, "Approximation Algorithms for Combinatorial Problems". The standard greedy algorithm for set cover works by picking a subset that covers the most number of remaining uncovered elements, at each stage of algorithm.

2.1.1 Standard Greedy Algorithm [8]

```
1. Set $C = \varnothing$, UNCOV = $U$.
   SET(i) = $S_i$, $1 \le i \le N$.
2. If UNCOV = $\varnothing$, halt and return $C$.
3. Choose j $\le$ N such that |SET(j)| is maximized.
4. Set $C = C \cup \{S_j\}$, UNCOV = UNCOV - SET(j),
   SET(i) = SET(i) - SET(j), $1 \le i \le N$.
5. Go to 2.
```

Example-2.1: This example finds the set cover for following collection of sets, using standard greedy algorithm.

$S_1 = \{1, 2, 3, 4, 5, 6\}$

$S_2 = \{5, 6, 8, 9\}$

$S_3 = \{1, 4, 7, 10\}$

$S_4 = \{2, 5, 7, 8, 11\}$

$S_5 = \{3, 6, 9, 12\}$

$S_6 = \{10, 11\}$

$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$

ITERATION – I:

$C = \varnothing$

UNCOV = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

UNCOV $\neq \varnothing$, therefore continue

For j = 1, SET(1) = 6 has maximum cardinality

$C = \{S_1\}$

UNCOV = {7, 8, 9, 10, 11, 12}

$S_2 = \{8, 9\}$

$S_3 = \{7, 10\}$

$S_4 = \{7, 8, 11\}$

$S_5 = \{9, 12\}$

$S_6 = \{10, 11\}$

ITERATION – II:

UNCOV $\neq \varnothing$, therefore continue

For j = 4, SET(4) = 3 has maximum cardinality

$C = \{S_1, S_4\}$

UNCOV = {9, 10, 12}

$S_2 = \{9\}$

$S_3 = \{10\}$

$S_5 = \{9, 12\}$

$S_6 = \{10\}$

ITERATION – III:

UNCOV $\neq \varnothing$, therefore continue

For j = 5, SET(5) = 2 has maximum cardinality

$C = \{S_1, S_4, S_5\}$

UNCOV = {10}

S_3 = {10}

S_6 = {10}

ITERATION – IV:

UNCOV $\neq \varnothing$, therefore continue

For j = 3, SET(3) = 1 has maximum cardinality

$C = \{S_1, S_4, S_5, S_6\}$

UNCOV = $\varnothing$

ITERATION – V:

UNCOV $\neq \varnothing$, therefore stop.

$C = \{S_1, S_4, S_5, S_6\}$

Therefore, Johnson's standard greedy algorithm produces a set cover $\{S_1, S_4, S_5, S_6\}$ of size 4. This is not the optimum set cover. Careful analysis shows that optimum set cover is $\{S_3, S_4, S_5\}$.

2.1.2 Standard Greedy Algorithm's Ratio Bound [3]

The complexity of standard greedy algorithm described above is of $O(mn)$. But, by maintaining a priority queue, standard greedy algorithm can be implemented to run in time $O(\sum_{S_i \in S} |S_i|)$.

The standard greedy algorithm has a ratio bound of $H(k) = \sum_{i=1}^{k} 1/i$, where k is the size of the largest set. $H(k)$ is also known as harmonic series.

Lemma 1 [3]: Johnson's standard greedy set cover has a ratio bound $H(k)$, where k is the size of the largest set.

Proof [3]**:** Assign the cost to each subset selected. For unweighted set cover problem the cost of selecting a subset is 1. Let C^* denote the size of an optimal set cover C^*, C the size of the set cover C returned by the standard greedy algorithm and S_i the *i*th subset selected. When algorithm adds S_i to set cover, it incurs a cost of 1. This cost of selecting S_i is evenly spread among the elements covered for first time by S_i. Let c_u denote the cost/weight allocated to element u, for each element $u \in U$. Each element is assigned a cost only for the first time it is covered. If u is covered for the first time by S_i then

$$c_u = \frac{1}{|S_i - (S_1 \cup S_2 \cup ... \cup S_{i-1})|}$$

The algorithm finds a set cover solution C which has a total cost $|C|$ as each set incurs a cost of 1. Since the optimal cover C^* also covers U. Therefore,

$$|C| = \sum_{u \in U} c_u$$

$$\leq \sum_{S \in C^*} \sum_{u \in S} c_u \tag{2.1}$$

Also, for any subset S_i belonging to the family of subsets S

$$\sum_{u \in S_i} c_u \leq H(|S_i|) \tag{2.2}$$

For inequalities (2.1) and (2.2), it follows

$$|C| \leq \sum_{S_i \in C^*} H(|S_i|)$$

$$\leq |C^*| \cdot H(\max(|S_i|)$$

$$= |C^*| \cdot H(k) \tag{2.3}$$

Here k is the size of largest set, i.e., max(|S|).

Ratio bound for standard greedy set cover is $O(1) + \ln|k|$. This is true for non-weighted as well as weighted set cover. This can be proved using the fact that harmonic series has ratio bound $O(1) + \ln|k|$.

The above result implies standard greedy algorithm solution is not too larger than optimal solution. If the maximum set size is 2 then this ratio bound is = 1 + ½ = 3/2. For maximum set size of 3, this ratio bound becomes 1 + 1/2 + 1/3 = 11/6.

2.1.3 Analysis of the Standard Greedy Algorithm

Lund and Yanakakis [13] established that set cover problem cannot be approximated with ratio c $\log_2 m$ for any c < l/4 unless all NP problems are solvable in $DTIME(m^{poly \log m})$. The Lund and Yanakakis result, basically rules out any drastic improvement to standard greedy algorithm. Therefore, major efforts have been made to apply improvements to greedy algorithm rather developing a new algorithm from scratch.

Even for maximum set size of 3, set cover problem is NP-hard. This can be proved using the fact that when the edges of a graph are covered by triangles the problem becomes a set cover problem [17]. Here each triangle represents a set. Covering the edges of a graph by triangles is NP-hard problem. For the triangle covering problem, the standard greedy algorithm's performance ratio is 11/6.

Standard greedy set cover algorithm can be used to solve vertex cover problem for a graph of degree of 3 at most [3]. As mentioned above, the solution found by standard greedy set cover algorithm is bound by $H(3) = 1 + 1/2 + 1/3 = 11/6$ times the optimal solution. This approximation ratio is better than that of approximate vertex cover.

2.1.4 A Tight Analysis of the Standard Greedy Algorithm by Petr Slavik [15]

Johnson showed that the performance ratio for set cover standard greedy algorithm is $H(m)$, which lies between $\ln m$ and $\ln m + 1$. For 20 years this remained the bound for standard greedy set cover algorithm. Petr Slavik in 1996 [15] proved that the performance ratio of standard greedy algorithm is exactly $\ln m - \ln\ln m + \Theta(1)$. Here m is the size of ground set or universe. Also the lower and the upper bound differ only by less than 1.1. Slavik's analysis was first tight analysis of standard greedy algorithm. For a function going to infinity with m, his analysis provided the first upper bound for standard greedy algorithm that lies below H(m).

Slavik used the hardness results shown by Feige. Feigh proved a very strong result showing that for any $\varepsilon > 0$, set cover cannot be approximated within $(1 - \varepsilon) \ln m$ by any polynomial time algorithm, unless $NPTIME[m^{O(\log\log m)}]$. Therefore, for any polynomial time algorithm maximum achievable improvement on performance ratio $H(m)$, is at most a function $f(m) = o(\ln m)$.

All the approximations till this point were generally based on assumption of some prior knowledge of m and $c_{\min}$. Then various algorithms are used to obtain the bounds on c_{greedy}. But Petr Slavik started with $c_{\min}$ and c_{greedy}, and obtained bounds on m.

Slavik's analysis showed that for any set U with $|U| = m \geq 2$

$$c_{greedy} \leq c_{\min} (\ln m - \ln c_{\min} + 1)$$

Also

$$c_{greedy} \leq (c_{\min} - ½) (\ln m - \ln c_{\min} + 1) + c_{\min}$$

Above two results give upper and lower bound for standard greedy algorithm.

The upper bound comes as:

$$\frac{c_{greedy}}{c_{\min}} < \ln m - \ln \ln m + 0.78$$

and

The lower bound

$$\frac{c_{greedy}}{c_{\min}} > \ln m - \ln \ln m - 0.31$$

It can be clearly seen that upper and lower bound differ by only 1.11. The upper and lower bound provided above by Petr Slavik are tight up to a constant.

2.2 Hitting Set

Dual of set cover problem is hitting set problem. In hitting set problem, a family of sets *S*, |*S*| = *m* are given. The aim is to create a subset *H* such that *H* has at least one element in each of the sets, i.e., every set has non-empty intersection with *H*. Minimum hitting set is creating a set with minimum cardinality [12].

$$H \bigcap_{s \subseteq U} S \neq \varnothing$$

Hitting set problem can be converted into set cover problem. To achieve that, all the elements of U are replaced by names of subset that contain it. Now this is set cover problem with roles of U and S reversed.

2.2.1 Algorithm for Solving Hitting Set Problem

1. Replace all the elements of *U* by names of subset that contain it. This forms the family of subsets.
2. Family of subsets originally provided, now form the universal set U'.
3. Apply any set cover algorithm, of new set cover problem formed by reversing roles of U and S. The set cover found is the hitting set or dual of set cover.

Example-2.2: Builds hitting set, for following collection of subsets and universal set.

S_1 = {1, 2}

S_2 = {2, 3}

S_3 = {3, 4}

$S_4 = \{4, 1\}$

$U = \{1, 2, 3, 4\}$

Replace the elements of U by names of subset that contain it.

$1 = \{S_1, S_4\}$

$2 = \{S_1, S_2\}$

$3 = \{S_2, S_3\}$

$4 = \{S_3, S_4\}$

$U' = \{S_1, S_2, S_3, S_4\}$

Now the hitting set problem has been transformed into set cover problem. Here 1, 2, 3 and 4 form the family of subsets and the family of subsets originally provided, form the universal set. Clearly 1 and 3 is the solution set for new set cover problem as they cover the universal set U'. It can be easily verified that 1 and 3 is also the minimum hitting set solution, for original hitting set problem. This is non-unique as 2 and 4, too, form the hitting set of cardinality 2.

2.3 *f*-Approximation Greedy Algorithm for Set Cover [16]

f-approximation greedy algorithm provides a solution which is bounded by the maximum number of sets that an element belong. If each element occurs in at most f sets, then this algorithm finds a solution which has a performance ratio of *f*. This algorithm is very useful in low frequency systems.

Let’s assume [16],

Universal set $U = \{u_1, u_2, ..., u_m\}$.

Subsets $S_1, S_2, ..., S_n \subseteq U$.

The cost $c_j \geq 0$ for each set S_j. The cost function is 1 for unweighted set cover.

The aim is to find $I \subseteq \{1, ..., n\}$ that minimizes $\sum_{j \in I} c_j$ subject to $\bigcup_{j \in I} S_j = U$. In other words, the goal is to select the minimum cost collection of sets that cover the universal set. Frequency of an element in a set cover instance, is defined as the number of sets it is in, i.e., $f = \max_i |\{j : u_i \in S_j\}|$.

2.3.1 Greedy Algorithm for *f*-Approximation [16]

```
Initialize $I \leftarrow \emptyset$
While $U \neq \emptyset$
    Pick $u_i$ with maximum frequency. $u_i \in U$
    $I \leftarrow I \bigcup \{j : u_i \in S_j\}$
    $U \leftarrow U - \bigcup_{j \in I} S_j$
```

Example-2.3: This example finds the set cover for collection of sets in Fig-2.1, using *f*-approximation greedy algorithm.

S_1 = {1, 2, 3, 4, 5, 6}

S_2 = {5, 6, 8, 9}

S_3 = {1, 4, 7, 10}

S_4 = {2, 5, 7, 8, 11}

S_5 = {3, 6, 9, 12}

S_6 = {10, 11}

U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

ITERATION – I:

f_5 = 3 is the maximum frequency element.

Therefore, corresponding set in solution is S_1.

Hence,

C = {S_1}

UNCOV = {7, 8, 9, 10, 11, 12}

S_2 = {8, 9}

S_3 = {7, 10}

S_4 = {7, 8, 11}

S_5 = {9, 12}

S_6 = {10, 11}

ITERATION – II:

f_8 = 2 is the maximum frequency, in the remaining elements.

The corresponding set in solution is S_2.

Hence,

$C = \{S_1, S_2\}$

UNCOV = {7, 10, 11, 12}

$S_3 = \{7, 10\}$

$S_4 = \{7, 11\}$

$S_5 = \{12\}$

$S_6 = \{10, 11\}$

ITERATION – III:

$f_7 = 2$ is the maximum frequency, in the remaining elements.

The corresponding set in solution is S_3.

Hence,

$C = \{S_1, S_2, S_3\}$

UNCOV = {11, 12}

$S_4 = \{11\}$

$S_5 = \{12\}$

$S_6 = \{11\}$

ITERATION – IV:

$f_{11} = 1$ is the maximum frequency, in the remaining elements.

Therefore, corresponding set in solution is S_4.

Hence,

C = {S_1, S_3, S_2, S_4}

UNCOV = {12}

S_5 = {12}

ITERATION – V:

12 is the only uncovered element.

Therefore, set cover solution must include S_5.

Hence,

C = {S_1, S_3, S_2, S_4, S_5}

Therefore, the final set cover is,

C = {S_1, S_3, S_2, S_4, S_5}

2.4 Modified Greedy (MG) Algorithm by Goldschmidt et al. [17]

Olivier Goldschmidt, Dorit S. Hochbaum and Gang Yu proposed a solution with ratio bound of $(\ln |k| + 5/6)$, where k is the maximum set size. This algorithm works by applying the combination of more than one algorithm, approximate and exact, to obtain improvement over standard greedy algorithm. The final set cover solution is the union of the outputs for different algorithms applied. This algorithm uses the fact that optimal solution for set cover problem can be obtained in polynomial time, for maximum set size of 2. For maximum set size 2, the set cover problem reduces to edge cover problem.

For modified greedy algorithm, Goldschmidt et al. used following terminology: Given a family $\{S_1, S_2, ..., S_n\}$ of finite sets, define J = {1,2,….,n} and $I = \bigcup_{j \in J} S_j$. $J^* \subseteq J$ is called a cover if $I = \bigcup_{j \in J^*} S_j = I$. They assumed that no set is embedded in another set, i.e., for all $i, j \in J$ and $i \neq j$. If this assumption is not true then it requires a pre-processing step, to remove all the instances of sets being subsets of others. A k-set element is called set of size k. S_j^k denotes the elements still uncovered after k iteration of the algorithm. Index set for the first k sets added to the set cover is denoted J^k and the elements still uncovered after k rounds is denoted by I^k.

The modified greedy algorithm executes the Johnson's standard greedy algorithm, till the maximum set size becomes 2. Once the maximum set size becomes 2, it finds the optimal cover and gets the improvement over standard greedy algorithm.

2.4.1 MG (Modified Greedy) Algorithm [17]

Step 0: Set $\mathrm{J}^0 = \emptyset; \mathrm{S}_\mathrm{j}^0 = \mathrm{S}_\mathrm{j}, j \in J; I^0 = \mathrm{U}_{j \in j} S^0 = I; k = 0$.

Step 1: k = k + 1. Select j_k = argmax $_{j \in J} | S_j^{k-1} |$.

Step 2: If $\left| S_{j_k}^{k-1} \right| \geq 3,$ set $J^k = J^{k-1} \cup \ j_k$ and $S_j^k = S_j^{k-1} \setminus S_{j_k}^{k-1}$, $j \in J$,

$I^k = I^{k-1} \setminus S_{j_k}^{k-1}$ and go to step 1. Else, continue.

Step 3: If $I^k = \emptyset,$ stop the output cover $J^* = J^k$, otherwise, form the undirected graph G(V,E) with V=$U_{j \in J} \ S_j^k = I^k$ and $E = \ S_j^k : \left| S_j^k \right| = 2, j \in J$. After that, find the matching of maximum cardinality on graph *G*. Thereafter, include sets that correspond to the matching edges of G to J^k. Exclude the elements covered by the edges of the matching from S_j^k, $j \in J$. Include the remaining single element set in J^k. End processing and return $J^* = J^k$ as set cover.

Basically, algorithm applies the standard greedy algorithm till maximum set size becomes 2. At maximum set size of 2, the problem is equivalent to the edge cover problem. Edge cover problem can be solved in polynomial time. This can be done by finding maximum matching. Then the algorithm adds all the isolated vertices (single element set) not covered by maximum matching.

Note: For a graph G with n vertices having no isolated vertex, minimum edge cover and maximum matching, sum to the total number of vertices.

Lemma 2 [17]: For set size 2 or less, MG algorithm provides an optimal set cover.

Proof [17]**:** If $|S_j| \leq 2$, $j \in J$, step 1 and 2 can be ignored and algorithm can directly proceed to step 3. It can be easily observed that to obtain set cover each 1-element set needs to be included in the output. Therefore, algorithm requires only minimum number of 2-element sets for covering remaining elements. Basically, the aim is to solve the set cover problem with exact 2 elements per set. This problem is same as solving edge cover problem for graph. Edge cover of an undirected graph $G = (V,E)$ is a subset $E' \subseteq E$ of edges such that every vertex of V is incident to some edge of E'. Minimum edge cover is edge cover with minimum cardinality.

For modified greedy, a family of finite sets { $S_1, S_2, ..., S_n$ } = E and I = V are given. Each set is of size 2 since the edges are formed by joining the end points. There are known polynomial time algorithms to find solution of edge cover problem. These algorithms work by finding maximum matching and set of adjacent nodes uncovered by matching [18].

Let EC denote the set of edges for optimal edge cover and M' denote the set of edges in the maximum matching for the graph formed by EC, G(EC) = (V, EC). Clearly, $|M'| \leq |M|$, as M is maximum matching. The graph G(EC) also satisfies following relationship:

$|V| = 2\,|M'| + |EC - M'|$

Hence $|EC| = |V| - |M'|$. Edge cover $EC(M)$, can be obtained from maximum matching by adding the nodes (or vertices) that are not yet covered. This cover, too, satisfies the relationship

$|V| = 2\,|M| + |EC(M) - M|$

Hence $|EC(M)| = |V| - |M|$. This implies $|EC – EC(M)| = |M| - |M'| \geq 0$, which is possible only if $EC = EC(M)$, as EC is minimum edge cover.

The runtime of modified greedy algorithm is dominated by the time spent for solving the maximum matching problem. The maximum matching problem in a graph G = (V, E) can be solved in running time of $O(|E|\sqrt{|V|})$. In a graph of order n, maximum matching can be found in $O(n^{2.5})$ [19,20,21].

Lemma 3 [17]: Let J^* be a cover found by MG and let J be any other cover, i.e., $\bigcup_{j\in J} S_j = \bigcup_{j\in J^*} S_j = \bigcup_{j\in J'} S_j = I$. Let $n_j = |S_j|$, $j \in J'$, then

$$|J^*| \le \sum_{j\in J'} Q_j(n_j), \qquad (2.4)$$

where,

$$\mathrm{Q}_j(n_j) = \begin{cases} 0 & n_j = 0 \\ 1 & n_j = 1 \\ \sum_{i=1}^{n_j} 1/i - 1/6 & n_j \ge 2 \end{cases}$$

Proof [17]**:** The above can be proved by induction on largest size set.

When maximum set size is 1 or 2, then above holds true as step 3 of MG provides the optimal cover, and the right hand side of inequality (2.4) is greater than or equal to $|J'|$. As J^* is optimal, $|J^*|$ cannot exceed the number of sets required in any cover J'.

Assume that the result holds true for maximum set size = q - 1, then it needs to be proved that the result also holds true for $q \ge 3$. Let r denote the smallest index k such that $|S_j^k| \le q$, for all j, which means after r iterations of modified greedy algorithm is applied, there is no set that contains more than q – 2 elements. Let $U^r = I \setminus I^r$ denote the covered elements till iteration r. As U^r is covered by disjoint sets of size greater than or equal to q,

$$|J^r| \le |U^r| / q \qquad (2.5)$$

Let us now define $n_j^r = |S_j^r|$. After r - 1 iterations of MG algorithm, uncovered elements $I^r = I \setminus U^r$ and unused sets $S_j^r = S_j \setminus U^r,\ j \notin J^{r-1}$ - satisfies assumption for induction, as $n_j^r \le q-1,\ j \in J$.

Hence,

$$|J_1^*| \ \le \ \sum_{j \in J'} Q_j(n_j^r), \qquad (2.6)$$

where J_1^* is defined as the index set for the remaining cover, i.e., $J^* = J^r \bigcup J_1^*$.

For all combinations of values of n_j and $Q_j(n_j)$,

$$Q_j(n_j) \ge Q_j(n_j^r) \ + |S_j \setminus S_j^r|/q \qquad (2.7)$$

Now for any given j, consider the possible values of n_j and n_r, and in every scenario use the fact that $n_j^r \le n_j \le q$.

Case $n_j = 1$: $Q_j(n_j) = 1$. If $n_j^r = 0$, then $Q_j(n_j^r) = 0$ and $|S_j \setminus S_j^r| =$ $n_j - n_j^r = 1$. If $n_j^r = 1$, then $Q_j(n_j^r)$ = 1 and $|S_j \setminus S_j^r| = 0$. Therefore, in both cases $Q_j(n_j) \ge Q_j(n_j^r) \ + |S_j \setminus S_j^r|/q$ is true.

Case $n_j \ge 2$: $Q_j(n_j) =$ $H(n_j) - \frac{1}{6}$. There are 3 possible condition that correspond to the size of set S_j^r, n_j^r, after the first r iterations.

(i) $n_j^r = 0$: $Q_j(n_j^r) = 0$ and

$$Q_j(n_j) = \ H(n_j) - \frac{1}{6} \ = \ 1 + \frac{1}{3} + \frac{1}{3} + \frac{1}{4} + \ldots.. + \frac{1}{n_j}$$

$$\geq \frac{4}{3} > Q_j(n_j^r) + (n_j - n_j^r)/q$$

$$= Q_j(n_j^r) + |S_j \setminus S_j^r|/q$$

A strong inequality follows from $Q_j(n_j^r) = 0$ and $q \geq n_j$.

(ii) $n_j^r = 1$: $Q_j(n_j^r) = 1$ and

$$Q_j(n_j) = Q_j(n_j^r) + \frac{1}{3} + \frac{1}{3} + \frac{1}{4} + \ldots.. + \frac{1}{n_j}$$

$$\geq Q_j(n_j^r) + (n_j - 1)/n_j$$

$$\geq Q_j(n_j^r) + (n_j - n_j^r)/q \qquad \text{because } n_j \leq q$$

$$= Q_j(n_j^r) + |S_j \setminus S_j^r|/q$$

One thing to note here is that when $q = 3$, the inequality changes to equality

(iii) $n_j^r \geq 1$: $Q_j(n_j^r) = H(n_j^r) - \frac{1}{6} = 1 + \frac{1}{3} + \sum_{i=2}^{n_j^r} 1/i$

$$Q_n(n_j) = 1 + \frac{1}{3} + \sum_{i=2}^{n_j} \frac{1}{i}$$

$$= 1 + \frac{1}{3} + \sum_{i=2}^{n_j^r} \frac{1}{i} + \sum_{i-n_j^r+1}^{n_j} \frac{1}{i}$$

$$\geq Q_j(n_j^r) + (n_j - n_j^r)/q \qquad \text{since } \frac{1}{n_j} \geq \frac{1}{q}$$

Therefore, $Q_j(n_j) \geq Q_j(n_j^r) + |S_j \setminus S_j^r|/q$.

For $q \geq 3$, from (i), (ii) and (iii) it follows

$Q_j(n_j) \geq Q_j(n_j^r) + |S_j \setminus S_j^r|/q$ (2.8)

Applying inequality (2.8) and because $\sum_{j\in J'} |S_j \setminus S_j^r| \geq \bigcup_{j\in J'} |S_j \setminus S_j^r| = |U^r|$, following result is obtained:

$$\begin{aligned}\sum_{j\in J'} Q_j(n_j) &\geq \sum_{j\in J'} (Q_j(n_j^r) + |S_j \setminus S_j^r|/q) \\ &\geq |J_1^*| + |U^r|/q \\ &\geq |J_1^*| + |J^r|\end{aligned}$$

Lemma 3 gives a relationship between a set cover found by modified greedy algorithm and set cover provided by any other algorithm. If $J' = J_{opt}$, algorithm returns the worst case performance ratio.

Theorem [17]: The size of cover found by MG algorithm is no more than $H(k) - \frac{1}{6}$ times the size of optimal cover.

Proof [17]: The worst case performance ratio for MG algorithm is achieved, by setting $J' = J_{opt}$, $n_j = k$, $j \in J_{opt}$, i.e.,

$$|J^*| \leq \begin{cases} |J_{opt}| & k = 1 \\ (H(k) - \dfrac{1}{6})|J_{opt}| & k \geq 2 \end{cases}$$

Therefore, the size of cover found by modified greedy algorithm by Goldschmidt el al. is $H(k) - \frac{1}{6}$ times the size of an optimal cover, where k is the maximum set size. Therefore, modified greedy algorithm improves the performance ratio of standard greedy algorithm by 1/6. The upper bound is tight for $k \geq 3$.

For example, bound for standard greedy algorithm for k = 3 is 11/6, where as the MG algorithm has a ratio bound of 10/6. Therefore, standard greedy algorithm's bound for covering the edges of a graph by triangles, is 11/6. Modified greedy algorithm's bound for the same problem is 10/6.

It can be clearly observed that modified greedy by Goldschmidt el al., is mainly suited to the set cover problem for small k, where k is the maximum number of elements per set. The improvement comes from exact solution when set size is 2 or less. Modified greedy algorithm is important in the field of semiconductor industry or flexible manufacturing.

Example-2.4: Find the set cover for following collection of sets, using modified greedy algorithm.

$S_1 = \{1, 2\}$

$S_2 = \{1, 3\}$

$S_3 = \{1, 4\}$

$S_4 = \{2, 3\}$

$S_5 = \{2, 4\}$

$S_6 = \{3, 4\}$

The graph for problem, using MG algorithm is shown in Fig-2.1. The graph has edges: 1-2, 1-3, 1-4, 2-3, 2-4 and 3-4.

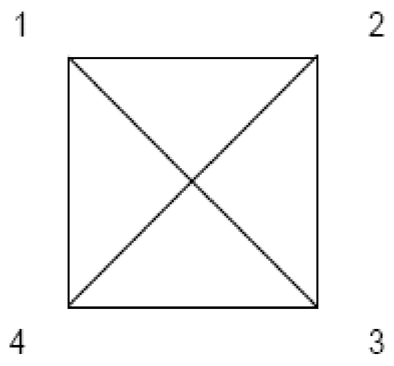

Fig-2.1 Graph Constructed for Example 2.4

1-2 and 3-4 form the maximum matching. This is also the perfect matching as it covers all the nodes of the graph. This means $\{S_1, S_6\}$ forms the edge cover. Clearly, $\{S_1, S_6\}$ is also a set cover. This is optimal solution, at the same time non-unique as well. $\{S_2, S_5\}$ and $\{S_3, S_4\}$ are other exact set cover solution for this problem.

Example-2.5: Find the set cover for following collection of sets. Here maximum set size is greater than 2.

S_1 = {1, 2, 3, 4, 5, 6}

S_2 = {5, 6, 8, 9}

S_3 = {1, 4, 7, 10}

S_4 = {2, 5, 7, 8, 11}

S_5 = {3, 6, 9, 12}

S_6 = {10, 11}

U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

STEP – I:

Apply standard greedy algorithm, till set size becomes 2.

This selects S_1 and S_4. Note: refer example 1.

Hence,

$C = \{S_1, S_4\}$

UNCOV = {9, 10, 12}

$S_2 = \{9\}$

$S_3 = \{10\}$

$S_5 = \{9, 12\}$

$S_6 = \{10\}$

STEP – II:

Find edge cover on remaining elements.

S_5 is the only edge.

S_3 covers the remaining isolated vertex 10.

Hence,

$C = \{S_1, S_4, S_3, S_5\}$

Therefore, the final set cover is

C = $\{S_1, S_4, S_3, S_5\}$

2.5 Set Cover via Local Improvement by Halldorsson [22]

Halldorsson proposed an old optimization technique and applied it to the approximation of collections of discrete items. He called this technique as local improvement. The approach consisted of extending a collection, by adding some items while removing others. Essentially, Halldorsson proposed a combination of different algorithms in several subinstances of the initial set cover instance.

Halldorsson described local search as tool to solve hard combinatorial optimization problems. Local search can be applied to greedy search or hill climbing, simulated annealing or randomized hill climbing, augmenting paths, and local changes. Local search technique has been successful in practical instances. But the technique fails many times as it doesn't have good worst case results. In this paper, Halldorsson presented several positive results on the quality of locally optimal solutions. The goal of local improvement is to improve the best performance ratios known for various optimization problems. At the same time, it also highlights the effectiveness of local improvement heuristics as approximation algorithms.

Local search approach starts with a maximal/minimal collection depending on maximization/minimization problem, and expands/shrinks the collection until no better solution can be found. Using this technique, the algorithm for 3-set cover, i.e., set cover with maximum set size of at most 3 gives a solution of performance ratio 11/7 ≈ 1.57. This generalizes to a $H(k) - \frac{11}{42}$ ratio for k-set cover, where k is the size of the largest set.

2.5.1 The Local Improvement Framework [22]

Halldorsson considered the problem of maximization. A solution is maximal if the addition of any item destroys the feasibility of current solution. A non-maximal solution can be extended by just adding an item.

Suppose A is a feasible solution and there are sets x, y, z where x $\in A$ and y, z $\notin A$ such that $A' = (A - \{x\}) \cup \{y, z\}$ is also a solution. Then, A' is an extension of A, and {x, y, z} a 2-improvement of A. Basically, a set of items I such that $A \oplus I$ (the union of A and I) is a solution and $|A \oplus I| > |A|$, is an improvement of A .

Halldorsson used the terminology t-improvements. A t-improvement adds n new items and removes n-1 items, for some $n \leq t$. Basically, improvement adds one more item than the number of items it removes. A solution is said to be t-locally optimal if no t-improvements is possible. Local search tries to improve initial solution by succession of improvements, until no improvement exists.

More precisely,

```
t-optimal solution:
$A \leftarrow$ maximal solution.
Repeat
    $I \leftarrow n$-improvement of $A$, where $n \leq t$.
    $A \leftarrow A \oplus I$ (the symmetric union of A and I).
    until no further improvement exists.
    Return $A$.
```

Minimization problem is very much like maximization problem, but *t*-improvement proceeds in opposite direction. For minimization problem, local search start with a minimal solution and it attempts to improve it by shrinking the solution set. Improvement removes one more item than the number of items it adds. A solution is said to be *t*-locally optimal if no *t*-improvement is possible.

2.5.2 Approximate k-Set Cover Using Set Packing for (S, C, k) [22]

Halldorsson provided a generic algorithm using set packing. Halldorsson used *s* to denote the number of elements in the base set(or ground set or universal set or super set) *S*, *A* to denote the set cover output of the algorithm and *B* as the set cover output by any other algorithm. Here *C* is collection of sets and *k* is maximum size of sets in this collection.

For simplicity Halldorsson assumed that the input set collection *C* is monotone, i.e., whenever $X \in C$, so is the every subset $X^{'}$ of X. Set cover is a partition of the base set *S* into sets in *C*. This can be verified from the fact that replacing sets by an appropriate superset in the original input does not increase the size of the set cover solution, i.e., total number of sets. The Halldorsson’s approach can be viewed as a sequence of maximal solutions to set packing problems. It starts with k-set packing problem and ends with 1-set packing problem, for which the solution is trivial.

Halldorsson’s ApproxSetCover algorithm uses an approximate set packing algorithm to produce a set cover solution. If greedy algorithm is used to solve the set packing problem then the resulting solution of ApproxSetCover algorithm is also greedy.

For a collection *C* to S', algorithm uses the notation $C_{|S'}$ to denote restriction of *C* to S', be the collections of subsets of S', whose supersets are contained in *C*.

2.5.2.1 ApproxSetCover(S, C, k) Algorithm [22]

```
If k =1 return C.
$C_k$ ← The sets in C of size k.
$A_k$ ← ApproxSetPacking($C_k$, k).
$S_k$ ← $S - (\bigcup_{c \in A_k} C)$.
Return $A_k \cup$ ApproxSetCover($S_k$, $C_{|S_k}$, k-1).
```

Halldorsson compared the above algorithm to the improved bound of $H(k)-\frac{1}{6}$ for k-SC obtained by modified greedy algorithm by Goldschmidt, Hochbaum and Yu. MG algorithm works on finding maximum matching on maximum set size of 2, which is essentially finding set packing solution for maximum set size of 2. Therefore, 2-opt on 2-SP obtains the same ratio.

Lemma 4 [22]: ApproxSetCover, using 2-opt on the 2- SP sub-problem, finds an approximate 3-set cover with at most (s + 2b)/3 sets. Here *s* denotes the number of elements in the base set *S*, *a* be the number of sets in the set cover solution by current Halldorsson algorithm *A* and *b* be the number of sets in any other cover set cover Solution *B*.

Proof [22]**:** Let A_i (C_i), $i = 1, 2, 3,$ denote the sets in A (C) of size *i*, respectively. Let S_i denote the elements contained in sets in A_i.

Consider the restriction $B_{|S_2}$ of B to the elements contained in $A_1 \cup A_2$. Let $B_1^{'}$ and $B_2^{'}$ be the sets in $B_{|S_2}$ with 1 and 2 elements respectively.

This implies

$$s_2 = s - 3\ a_3,$$

so

$$b_1^{'} + 2b_2^{'} + 3a\ _3 = s$$

Therefore, the number of 2-sets in $B_{|s_2}$ is at least

$$\begin{aligned} b_2^{'} &= s - 3\ a_3 - (b_1^{'} + b_2^{'}) \\ &\geq s - 3\ a_3 - b \end{aligned}$$

Since the set packing algorithm discussed is 3/2 optimal, it will find at least two-thirds of the sets. Thus, the solution provided consists of a_3 3-set, at least $2(s - 3\ a_3 -\ b)/3$ 2-sets, remainder being 1-set.

Therefore, the cost of the solution will be at most

$$a_3 + \ \frac{2}{3}\ [s - 3\ a_3 - b] + [s - 3\ a_3 - \frac{4}{3}(s - 3\ a_3 - b)]$$

$$= \frac{s + 2b}{3}$$

This implies that 3-SC problem, can be solved using 2-opt on the 2-set packing sub-problem. The above algorithm finds approximate set cover which is 5/3 optimal. This result can be extended to k-set cover whose solution is $H(k)-\frac{1}{6}$ optimal. The assumption for k-set cover solution is, 2-SP problem is equivalent to maximum matching problem and the exact maximum matching can be found in polynomial time.

2.5.3 Halldorsson's Direct Application of Local Improvement [22]

Halldorsson provided the algorithm for 3-set cover problem, i.e., set cover problem with maximum set size of 3. He used the result of 3-SC, along with the modified greedy algorithm suggested by Goldschmidt et al. to get local improvements. Halldorsson suggested the local improvement method directly on the set cover solution. The method employed local shrinking of solution to achieve improvement.

For example, say

Sets $x_1, x_{2,}..., x_{k,} x_{k+1} \in \mathrm{A}, y_1, y_{2,}..., y_k \in \mathrm{C}$

such that A – $\{x_i\} \bigcup \{y_i\}$ is a set cover.

This would replace k + 1 sets in original solution A by only k sets, while the resultant solution still remaining a set cover.

The local improvement method seeks improvements using particular type of shrinking. The aim is to try to merge a set with some unit size sets, producing a number of sets of size two. The approach seeks: 1) a set in A_3 and three sets in A_1 for which there are three sets in C_2 who cover the same six elements, or 2) a set in A_2 and two sets in A_1

for which there are two sets in C_2 who cover the same four elements.

2.5.3.1 Algorithm for 3-SC Problem [22]

Find a maximal 3-set packing (3-SP).

Find an optimal 2-set cover (2-SC).

Repeat shrinking steps until obtained set cover is locally minimum.

The above algorithm finds a cover with at most (2s + 5b)/7 sets. 3-SC algorithm for set cover with maximum set size of at most 3, has a solution of performance ratio 11/7 = $H(k) - \frac{11}{42} \approx 1.57$.

Halldorsson proposed a method of finding a maximal collection of independent shirking improvement in linear time. It assumes collection of sets $C_2^{'}$ in C_2 which have one element in S_1. Then it marks each element in $S_1 \cup S_2$ with at most three sets in $C_2^{'}$ containing that element. Then test if for each set *X* in A_2, A_3, there can be chosen a group of |*X*| disjoint sets, one from the group of sets marked to each element. If answer is yes, then improvement is possible. Therefore, update S_i, $C_2^{'}$ accordingly to ensure that the improvements will be independent.

Halldorsson observed that under these shrinking improvements A_1 is monotone decreasing. Therefore, once a maximum collection of improvements have been obtained, further local improvements are not possible. Therefore, the entire shrinking process runs in the linear time. Parallel implementation of shrinking procedure is also possible, by finding a maximal independent set in the graph of all possible improvements.

Lemma 5 [22]: 3-SC algorithm finds a cover with at most (2*s* +5*b*)/7 sets.

Proof [22]: For two set collections $A,\ A^{'}$:

let $A\ \iota\ A^{'}$ denote { $X \in A:\ X \bigcap (\bigcup_{X^{'} \in A^{'}} X^{'})\ \neq \varnothing$ } or the collection of sets in A containing elements in some set in $A^{'}$.

Partition B into $B_1 = B\ \iota\ A_1,\ B_2 = (B - B_1)\ \iota\ A_2,$ and $B_3 = B - B_1 - B_2$. Let $b_1,\ b_2,\ b_3$ denote the sizes of these collections respectively.

Using the optimality of 2-set packing algorithm

$$a_1 \leq b_1 \qquad (2.9)$$

$$a_1 + a_2 \leq b_1 + b_2 \qquad (2.10)$$

Since the solution is locally minimal. Therefore, at least one element of each set in A_2 must belong to a set in either B_2 or B_3. Each set in B_2 contains an element in A_2. Therefore, slots available for elements from A_3 are at most 2, in such a set.

This implies

$$a_3 \leq 2b_2 + 3b_3 \qquad (2.11)$$

The total number of set in the solution is precisely s.

$$a_1 + 2a_2 + 3a_3 = s \qquad (2.12)$$

Now if add (2.9) twice , (2.10) three times, (2.11) once and (2.12) twice,

$$7(a_1 + a_2 + a_3)\ \leq\ 2s + 5(b_1 + b_2 + b_3) - 2b_3$$

$$(a_1 + a_2 + a_3)\ \leq\ (2s + 5(b_1 + b_2 + b_3))/7 - 2b_3/7$$

which implies 3-SC algorithm finds a set cover with (2s +5b)/7 sets at most.

Since $s \le 3b,$ this implies performance ratio of 11/7 = 2b, which implies that 3-SC algorithm has performance ratio of 11/7.

2.5.3.2 Algorithm for k-SC Problem [22]

Execute standard greedy algorithm, until set size becomes at most 3. Let A_1 be the obtained SC solution.

On the surviving set cover instance, apply 3-SC algorithm, to get solution A_2.

A = A_1 U A_2 is the final SC-solution.

The k-SC algorithm provides a solution which is $H(k) - \frac{11}{42}$ optimal.

Example 2.6: Find set cover for following collection of sets using k-SC algorithm.

S_1 = {1, 2, 3, 4, 5, 6}

S_2 = {5, 6, 8, 9}

S_3 = {1, 4, 7, 10}

S_4 = {2, 5, 7, 8, 11}

S_5 = {3, 6, 9, 12}

S_6 = {10, 11}

U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

STEP – I:

Apply standard greedy algorithm, till set size becomes 3.

S_1 = 6 has maximum cardinality.

Therefore, select S_1. Now remaining sets have maximum set size of 3.

After elements of S_1 is removed from remaining sets,

$C = \{S_1\}$

UNCOV = {7, 8, 9, 10, 11, 12}

$S_2 = \{8, 9\}$

$S_3 = \{7, 10\}$

$S_4 = \{7, 8, 11\}$

$S_5 = \{9, 12\}$

$S_6 = \{10, 11\}$

STEP – II:

Now find maximum set packing.

S_2 and S_3 form the maximum set packing. Note: this is non-unique.

After removing elements of S_2 and S_3 from remaining sets,

$C = \{S_1, S_2, S_3\}$

UNCOV = {11, 12}

$S_4 = \{11\}$

$S_5 = \{12\}$

$S_6 = \{11\}$

STEP – III:

Now find the minimum edge cover. But in this case, there are only isolated edges remaining.

Remaining element of S_4 and S_6 are essentially the same. Therefore, S_4 and S_5 complete the set cover.

$C = \{S_1, S_2, S_3, S_4, S_5\}$

STEP – IV:

Now apply local improvements to sets S_2, S_3, S_4, S_5.

$S_2 = \{8, 9\}$

$S_3 = \{7, 10\}$

$S_4 = \{7, 8, 11\}$

$S_5 = \{9, 12\}$

S_2 and S_3 form the maximum set packing. Note: this is non-unique.

After removing elements of S_2 and S_3 from remaining sets,

$C = \{S_1, S_2, S_3\}$

UNCOV = {11, 12}

$S_4 = \{11\}$

$S_5 = \{12\}$

In this case no further local optimization is possible. Therefore, the final set cover is

C = $\{S_1, S_2, S_3, S_4, S_5\}$

2.6 Duh and Furer Algorithm Using Semi Local Optimization [23]

Duh and Furer offered a new approach, semi-local optimization to solve set cover problem. This approach is based on local optimization. However this approach is more powerful. The advantage of this approach comes from global changes made to approximate solution. This approach applies standard greedy algorithm till the maximum set size becomes 3. For maximum set size of 5 and maximum set size of 4 it applies standard greedy algorithm with some restriction. In the end, for maximum set size 3, it uses semi local optimization. Semi local optimization for 3-set cover has greatly improved performance ratio of 4/3. Based on result of 3-set cover and a restrictive phase, performance ratio of k-set cover problem turns out to be $H(k) - \frac{1}{2}$.

2.6.1 3-Set Cover Using Semi Local Optimization [23]

In pure local optimization technique, current approximate solution gets better by replacing a fixed number of sets in the current set cover with a lesser number of sets to get a new set cover. Duh and Furer suggested following approach to solve 3-set cover problem and called it semi local optimization. In semi local optimization, once sets of size 3 has been selected for partial cover, the remaining sets with maximum set size of 2 are covered optimally in polynomial time, by finding maximum matching.

A semi-local(n,t) improvement step for 3-set cover consists of inserting up to n 3-sets and deleting up to t sets. Also, an arbitrary number of sets with maximum set size of 2 are optimally replaced. The algorithm looks for number of sets with set size equal 3, instead of all the smaller subsets. It allows global changes to the sets with maximum set size of 2. For quality of solution, algorithm first looks for number of sets in the set cover and second the number of sets with set size equal 1. Set cover with smallest size is preferred and among set covers of same size the one with fewer 1 element sets is preferred. Semi local-optimization algorithm for 3-set cover produces a set cover with performance ratio 4/3.

2.6.2 k-Set Cover Using Semi Local Optimization Algorithm [23]

```
1. The Greedy Phase:
    For j = k, down to 6 do
        Apply standard greedy algorithm to select maximum collection of j-sets.
        j-sets denote the sets covering exactly j new element.

2. Restricted Phase:
    For j = 5, down to 4 do
        Select maximal collection of j-sets with the condition that the choice of
        these j-sets would not increase the number of 1-sets.

3. Semi Local Improvement Phase for 3-Set Cover:
    Apply semi local optimization on the elements that are still not covered.
```

Semi local-optimization algorithm for k-set cover produces a set cover with a performance ratio of $H(k) - \frac{1}{2}$.

2.7 Modified Duh and Furer Algorithm by Asaf Levin [24]

In 2006, Asaf presented an improvement to Duh and Furer algorithm which has performance ratio $H(k) - \frac{196}{390}$. This is the best known improvement for standard greedy algorithm. The previous best known improvement of standard greedy algorithm was Duh and Furer algorithm which had a performance ratio of $H(k) - \frac{1}{2}$.

2.7.1 Asaf Levin's Algorithm [24]

1. The Greedy Phase:

    For j = k, down to 6 do

        Apply standard greedy algorithm to select maximum collection of j-sets.

        j-sets denote the sets covering exactly j new element.

2. Restricted Phase:

    Select a maximal collection of disjoint 5-sets with the condition that 5-sets would not increase the number of 1- sets.

3. Restricted Local Phase:

    (a) Select a maximal collection of disjoint 4-sets with the condition that these 4-sets would not increase the number of 1-sets.

    (b) While there are 4-sets $C \in \mathcal{C}$ (cover) and $C_1, C_2 \notin \mathcal{C}$ such that

    $\mathcal{C}' = (\mathcal{C} \setminus \{C\}) \cup \{C_1, C_2\}$ is a collection of disjoint 4-sets, so that it increases the number of 1-sets, replace C by $C^{'}$.

4. Semi Local Improvement Phase for 3-Set Cover:

Run semi local optimization on still uncovered elements.

Asaf Levin algorithm replaces the greedy construction for j = 4, in Duh and Furer algorithm by local search approach. The improvement of this algorithm, compared to Duh and Furer algorithm, comes from local search phase.

Above algorithm by Asaf Levin, for k-set cover produces a set cover with performance ratio $H(k)-\frac{196}{390}$. This is best known improvement for standard greedy algorithm. The time complexity of this algorithm is $O(m^5n^6)$.

CHAPTER 3

LINEAR PROGRAMMING TECHNIQUE FOR SET COVER PROBLEM

3.1 Linear Programming Introduction

Linear programming is the problem of optimizing a linear objective function, subject to linear inequality constraints [3]. Basically, the aim is to achieve best outcome, i.e., maximize or minimize linear function.

In linear programming problem for maximization [3], a m $\times$ n matrix A, a m-vector b and a n-vector c are given. The aim is to try and achieve a vector x of n elements that maximizes the objective function $\sum_{i=1}^{n} c_i x_i$, subject to the m constraints given Ax $\leq$ b. Feasible solution for linear programming problem is any vector x that satisfies Ax $\leq$ b. For minimization problem, constraints are of kind "$\geq$".

Many practical problems can be very easily expressed as linear programs [3]. Because of this a lots of research work has been done in the area of linear programming. General linear programs can be solved very quickly using the simplex algorithm.

If a problem can be defined as polynomial sized linear programming problem [3] then it means that there exists a polynomial time algorithm for the problem. Also in many special cases faster algorithms exist for the linear programming problem. Sometimes the objective function is ignored and an attempt is made to find whether any feasible solution exists or establish that no feasible solution exists.

For example, given a problem [26]:

minimize $7\ x_1\ +\ x_2\ +5\ x_3$

subject to $x_1\ -\ x_2\ +3\ x_3\ \geq 10$

$5\ x_1\ +2\ x_2 -\ x_3\ \geq\ 6$

$x_1, x_2,\ x_3 \geq 0$

This is the standard form of minimization problem [26]. Here all the constraints are of kind $\geq$ and all the variables are constraint to be non-negative. The importance of non-negative is that it doesn’t reverse the direction of constraint inequality. As stated earlier, any solution for the variables in the linear program which satisfies all the constraint is known as feasible solution. Let z^* represent the optimal value of this linear program and α be a given rational number. The question asked is if z^* is at most α . If answer to this question is yes, then it means that there is a feasible solution, whose objective function value is at most α . This α provides the upper bound for z^*. In the above example, consider if objective function value of at most 30, i.e., $z^* \leq 30$. x = (2,1,3) constitutes one solution as 7 * 2 + 1 + 5 * 3 = 30. Now the other objective is to find good lower bound on z^*. In the above example, one such bound can be obtained by the first constraint, using coefficient comparison term by term. Since the x_i ’s are restricted to be non-negative, clearly $7\ x_1\ +\ x_2\ +5\ x_3\ \geq\ x_1\ -\ x_2\ +3\ x_3\ \geq$ 10. Therefore, objective function is at least 10 for any feasible solution. A better lower bound is obtained by summing up the two constraints $7\ x_1\ +\ x_2\ +5\ x_3\ \geq (x_1\ -\ x_2\ +3\ x_3) + (5\ x_1\ +2$

x_2 - x_3) ≥ 16.

A lower bound is placed to the objective function so that suitable non-negative multipliers can be found for the constraints so that when their sum is taken, the coefficient of each x_i is dominated by the coefficient in the objective function [26]. Since x_i 's are restricted to be non-negative, right-hand side of this sum is the lower bound on z^*. Basically, the aim is to choose the multipliers in such a way that right hand side of the sum is as large as possible.

The problem of finding lower bound as linear program can be expressed as [26]:

$$\begin{array}{ll} \text{maximize} & 10\,y_1 + 6\,y_2 \\ \text{subject to} & y_1 + 5\,y_2 \leq 7 \\ & -y_1 + 2\,y_2 \leq 1 \\ & 3y_1 - y_2 \leq 5 \\ & y_1, y_2 \geq 0 \end{array}$$

Here y_1 and y_2 are chosen to be the non negative multipliers for the first and second constraint, respectively.

First linear program is called the primal linear program and the second linear program is called dual linear program [26]. Dual of dual linear program is primal linear program itself. Here it can be observed from dual of linear program is problem maximization, where primal linear program is minimization problem. By constructing, every feasible solution to the dual program gives a lower bound on the optimum value of the primal. The reverse also hold true, i.e., every feasible solution on the primal program provides the upper bound on the optimal value of the dual. Therefore, if the feasible solutions for the dual and the primal with matching value function are known, then both solutions must be optimal. In the example above, x = (7/4, 0, 11/4) and y = (2,1), both achieve the objective function value of 26. Therefore, both are optimal solutions.

L-P Duality Theorem [26]: The primal linear program has finite optimum if and only if its dual linear program has finite optimum. Moreover, if $x^* = (x_1^*, ...x_n^*)$ and $y^* = (y_1^*, ...y_m^*)$ are optimal solutions for primal linear program and the dual linear program respectively, then

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

In standard form, min-max dual theorem can be stated in the following way:

minimize $\sum_{j=1}^{n} c_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ $\quad$ i= 1,.....,m

$x_j \geq 0$ $\quad$ j= 1,.....,n

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

Then the dual linear program is:

maximize $\sum_{i=1}^{m} b_i y_i$ $\quad$ j= 1,.....,n

subject to $\sum_{i=1}^{m} a_{ij} y_i \leq c_j$ $\quad$ i= 1,.....,m

Weak L-P Duality Theorem [26]: If $x = (x_1, ...x_n)$ is a feasible solution for primal linear program and $y = (y_1, ...y_m)$ is a feasible solution for the dual linear program, then

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

Complementary Slackness Conditions [26]: Let x and y denote the solution for primal linear program and dual linear program respectively. Then x and y are both optimal if and only if each of the conditions stated below are satisfied:

Primal complementary slackness conditions for each $1 \leq j \leq n$; either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$; and

Dual complementary slackness conditions for each $1 \leq i \leq m$; either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$

3.2 Set Cover Using LP

For solving minimization problem for an NP-hard problem, using approximation linear programming algorithm, key step is to establish a good lower bound at the cost of the optimal solution [26]. For this, set cover problem is expressed as integer programming(IP). The cost of an optimal solution to LP-relaxation provides the desired lower bound.

The general approach for solving set cover problem using LP is [16]:

1. Formulate the set cover problem as an IP.
2. Relax IP to a LP.

3. Use the LP solution to get a solution set cover for IP.

3.2.1 Set Cover via Rounding [16,26]

In the rounding strategy, first the linear program is solved and then the solution obtained for linear program is transformed to integral solution. At the same time, attempt is made to make sure that in the process the cost of the solution does not increase much. The integral and fractional solutions cost is compared to obtain the performance ratio. This strategy is known as rounding.

The set cover problem is defined as

Ground set or universe $U = \{ u_1, u_2, \ldots\ldots, u_m \}$

Subsets $S_1, S_2, \ldots\ldots, S_n \subseteq U$

weight $w_j \geq 0$ for each set S_j. The cost function is 1 for unweighted set cover.

The aim is to find $I \subseteq \{1, \ldots., n\}$ that minimizes $\sum_{j \in I} c_j$ subject to $\bigcup_{j \in I} S_j = U$.

Basically, the goal is to select the minimum weight collection of sets that cover the universe.

The frequency of an element in a set cover instance is defined to be the number of sets it is in, i.e.,

$$f = \max_i |\{ j : u_i \in S_j \}|$$

Define a variable x_j for each subset S_j, which can have the values of either 0 or 1. x_j will be set to 1 iff corresponding subset S_j is selected in the set cover. Clearly, the restriction being that, for each element $u_i \in U$, where the aim is that at least one of the set containing it should be picked.

For weighted SC problem the IP formulation is

$$\text{Min} \sum_{j=1}^{n} w_j x_j$$

Subject to:

$$\sum_{j:u_i \in S_j} x_j \geq 1 \qquad \forall u_i \in U$$

$$x_j \in \{0,1\}$$

The LP relaxation is obtained by changing the last constraint to $1 \geq x_j \geq 0$. Since the upper bound on x_j is redundant,

$$\text{Min} \sum_{j=1}^{n} w_j x_j$$

Subject to:

$$\sum_{j:u_i \in S_j}^{n} x_j \geq 1 \qquad \forall u_i \in U$$

$$\sum_{j=1}^{n} x_j \geq 0$$

Note: for unweighted set cover $w_j = 1 \qquad 1 \leq j \leq n$

One way of converting a solution to linear program into an integer solution is to round up all non-zero variables to 1. This algorithm achieves the performance guarantee of f. Following is the slightly modified algorithm which picks fewer sets in general.

Simple Rounding Algorithm for Set Cover [26]:

1. Solve LP-relaxation to get an optimal solution x^*.
2. Pick all sets S_j for which $x^*_j \geq \frac{1}{f}$ in this solution.

The performance ratio achieved by rounding algorithm is f.

Randomized Rounding Approach to Set Cover [26]:

Each element selected by rounding process has a constant probability. Repeating a process $O(\ln n)$ times set cover of high probability is obtained.

To get complete set cover, c $\ln n$ sub collections are picked independently and their union is computed. Here c is a constant. The probability that set cover returned by this approach is not a valid set cover is 1/4. With probability 1/2 this approach returns a set cover that has a performance ratio of $4\ln(4n)$. The above procedure is repeated until a valid set cover is found. The expected number of repetitions required is at most 2. Clearly, the expected cost of algorithm is of $O(\ln n)$ optimal.

3.2.2 Set Cover via LP Duality [26]

As already stated, any feasible solution to the dual gives a lower bound on the primal program. Therefore, it gives lower bound on the original integer program also. The algorithm presented in this section finds an integral solution to the primal and simultaneously a solution to the dual. The performance ratio is obtained by comparing the cost of these two solutions, i.e., solution to the primal and the dual. The main advantage of this algorithm over rounding algorithm is that instead of having to work with an arbitrary optimal solution to LP-relaxation, the two solutions can be carefully picked so that they have nice combinatorial properties. Also since the algorithm doesn't have to first solve LP-relaxation optimally, this algorithm can be made more efficient.

Integrality gap of a minimized integer program is defined as the maximum ratio of an optimal integer program and optimal fractional solution. The aim is to minimize integrality gap. Using the LP-duality approach, is essentially formulation of integer programming of the integrality gap.

Introduce a variable y_i corresponding to each element $u_i \in U$.

Now consider the primal linear program

$$\text{Min } \sum_{j=1}^{m} w_j x_j$$

Subject to:

$$\sum_{j:u_i \in S_j} x_j \geq 1 \qquad \forall u_i \in U$$

$$x_j \geq 0$$

The dual linear program to the above LP-relaxation for set cover is:

$$\text{Max } \sum_{u_i \in U} y_i$$

Subject to:

$$\sum_{i:u_i \in S_j} y_i \le w_j \qquad \forall S_j \subseteq U$$

$$y_i \ge 0 \qquad \forall u_i \in U$$

Simple Low Cost Dual-LP Algorithm for Set Cover [26]:

1. Find an optimal solution y^* to dual linear program.
2. Pick all sets S_j for which $\sum_{i:u_i \in S_j} y_i^* = w_j$ in this solution.

The performance ratio achieved by dual LP algorithm is *f*.

3.2.3 Primal-Dual Schema

LP-duality also provides a general schema for obtaining the approximation linear programming algorithm: the primal dual schema [16]. Primal dual schema behaves much like dual LP. But rather than finding the optimal dual solution, it constructs its own dual solution.

Now consider the primal program written in standard form [26]:

minimize $\sum_{j=1}^{n} c_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j \ge b_i$ $\qquad i= 1,.....,m$

$x_j \ge 0$ $\qquad j= 1,.....,n$

where a_{ij}, b_i and c_j are specified in the input.

Then the dual program is:

maximize $\sum_{i=1}^{m} b_i y_i$

subject to $\sum_{i=1}^{m} a_{ij} y_i \leq c_j$ $\quad j$= 1,.....,n

$y_i \geq 0$ $\quad i$= 1,.....,m

Algorithms using primal-dual schema run by ensuring the primal complementary slackness conditions and relaxing the dual conditions.

Primary Complementary Slackness Conditions [26]:

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

Relaxed Dual Complementary Slackness Conditions [26]:

For each $1 \leq i \leq m$: either $y_i = 0$ or $\sum_{i=1}^{m} a_{ij} x_j = \alpha \cdot b_i$

where α > 1 is an constant; if α becomes 1 then it is usual condition.

Proposition [26]: Let x and y be the primal and dual feasible solutions satisfying the conditions mentioned above the

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \sum_{i=1}^{m} b_i$$

3.2.3.1 Set Cover via Primal-Dual Schema [26]

Primal-dual schema starts by mentioning the primal complementary slackness condition and then the dual conditions are relaxed appropriately.

Primal condition can be written as

$$\forall S_j \subseteq U : x_j \neq 0 \Rightarrow \sum_{i:u_i \in S_j} y_i = w_j$$

Set S_j is called tight if $\sum_{i:u_i \in S_j} y_i = w_j$. Since the primal variables are integrally incremented, primal complementary slackness condition can be stated as:

Select only tight sets in the set cover.

To maintain the feasibility of dual, overpacking of any set is not allowed. A set is called overpacked if the total amount packed into its elements exceeds its cost. Primal and dual can be thought as covering and packing linear program respectively.

Dual conditions are relaxed with $\alpha = f$.

$$\forall u_i \in y_j : y_i \neq 0 \Rightarrow \sum_{j:u_i \in S_j} y_j \leq f$$

Since x has 0/1 solution, these conditions are equivalent to:

Cover each element with non-zero dual at most f times.

Since each element occurs in at most f sets, therefore, this condition is satisfied for all elements trivially.

Using the above two conditions:

Primal-Dual Algorithm for Set Cover [26]:

```
1. Initialize: $x \leftarrow 0; y \leftarrow 0$
2. Till all the elements are covered perform:
      Select an uncovered element, say I, and raise $y_i$ till some tight sets are found.
      Select each tight set in the cover and update $x$.
      Mark each of the elements occurring in these sets as “covered”.
```

3. Return the obtained set cover x.

The above algorithm achieves a performance ratio of f.

Greedy heuristics can be used to present set cover linear programming and in that case approximation factor will be $H(k)$.

CHAPTER 4

SET COVER SOLUTION USING NETWORK FLOW APPROACH

Mohamed Afif, Mhand Hifi, Vangelis Th. Paschos and Vassilis Zissimopoulos presented the set cover solution, which was based on network flow algorithm of Ford and Fulkerson. This polynomial time algorithm was based on transforming set cover problem into a particular network flow problem and proposing a flow algorithm which was based on Ford and Fulkerson algorithm.

4.1 Transformation of Set Cover into Flow Problem [27]

Mohamed et. al used the following definition for transforming set cover into a flow problem.

Definition 1 [27]: A bipartite graph *B* = (*S*, *C*, *E*) is constructed from set cover problem definition. In this bipartite graph *B* = (*S*, *C*, *E*), vertex set *S* corresponds to the family of sets $\delta,$ vertex set *C* corresponds to the ground set (or base set or universal set) *C*, and set of edges $E = \{s_i c_j : c_j \in S_i\}$. Once bipartite graph is created, a layered network *N* = (*X*, *A*, *c*, *b*) is constructed with vertex $X = S \cup C \cup s_0 \cup c_0$ where c_0 represents the source of the network and s_0 represents the sink of the network and $A = E \cup E_s \cup E_c$ is the arc set of the network.

$E_s = \{s_j s_0 : \forall s_j \in S\}$, $E_c = \{c_0 c_i : \forall c_i \in C\}$ and the arcs are oriented from c_0 to s_0. The vectors $c = (c_a)_{a\in A}$, $b = (b_a)_{b\in A}$ represent capacities and lower bounds respectively, on the arcs of N.

Therefore,

$$c_a = \begin{cases} 1 & a \in E_s \\ \dfrac{1}{|\Gamma^-(s_j)|} & a = c_i, \quad i \neq 0 \vee j \neq 0 \\ \sum\limits_{a' \in E_{c_i}} c_{a'} & a = c_0 c_i, \; i = 1,..,m \qquad E_{c_i} = \{c_i s_j : s_j \in \Gamma^+(c_i)\} \end{cases}$$

$$b_a = \begin{cases} 0 & a \in E_s \bigcup E \\ \min\{c_{a'} : a' \in \{c_i s_j : s_j \in \Gamma^+(c_i)\}\} & a = c_0 c_i, \quad i = 1,...,m \end{cases}$$

where $\Gamma(x)$ is the set of neighbors of *x*, $|\Gamma^+(x)|$ and $|\Gamma^-(x)|$ are the outer and inner degree of vertex *x*.

Set of incoming arcs of vertex v_i are denoted by $I^-(v_i)$. Set of outgoing arcs of vertex v_i are denoted by $I^+(v_i)$.

Using construction implied by above definition set cover problem reduces to a minimum flow problem Φ on N.

$$\Phi = \begin{cases} \min \sum\limits_{a\in E_s} \varphi_a \\ \sum\limits_{a\in I^-(v_i)} \varphi_a = \sum\limits_{a\in I^+(v_i)} \varphi_a & v_i \in X \setminus \{s_0, c_0\} \\ b_a \leq \varphi_a \leq c_a & a \in A \\ \varphi_a \in \{0,1\} & a \in E_s \\ \varphi_a \in \mathbb{Q}^+ & a \in E \bigcup E_c \end{cases}$$

The set cover problem can be reduced to minimum flow problem Φ in polynomial time $O(mn)$. Also the (optimal) objective function values for the solution of set cover and Φ are equal.

Given an instance of I of set cover it takes $O(mn)$ steps for the construction of bipartite graph *B*. The construction of layered network N, too, takes $O(|E|) = O(mn)$ steps. Since computation of upper and lower capacities of arc can be performed in constant time, therefore, the entire transformation from set cover to network flow problem can be performed in $O(mn)$.

Example 4.1: Create the characteristic bipartite graph B and the layered network N, for following set cover problem.

$\delta = \{S_1, S_2, S_3, S_4\}$

$C = \{a, b, c, d, e\}$

$S_1 = \{a, b\}$

$S_2 = \{b, c, d\}$

$S_3 = \{d, e\}$

$S_4 = \{a, e\}$.

Using Definition 1, above set cover problem can be converted into network flow problem. Fig-4.1 shows the characteristic bipartite graph B for this and Fig-4.2 represents the layered network N.

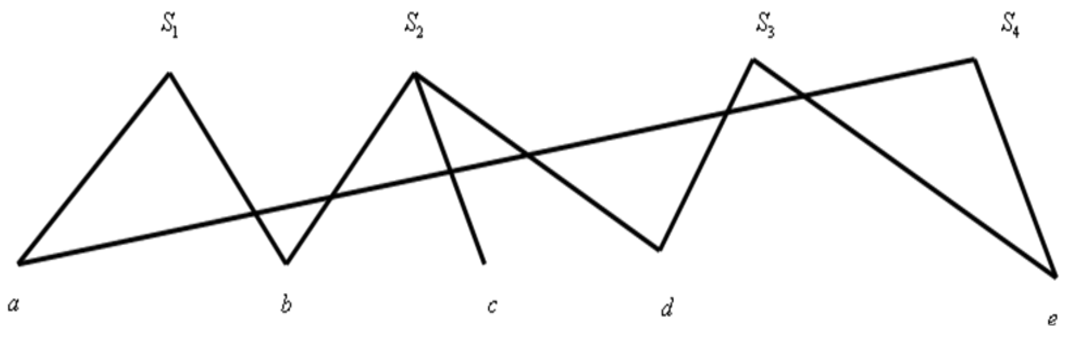

Fig-4.1 Bipartite Graph B for Set Cover Problem in Example 4.1

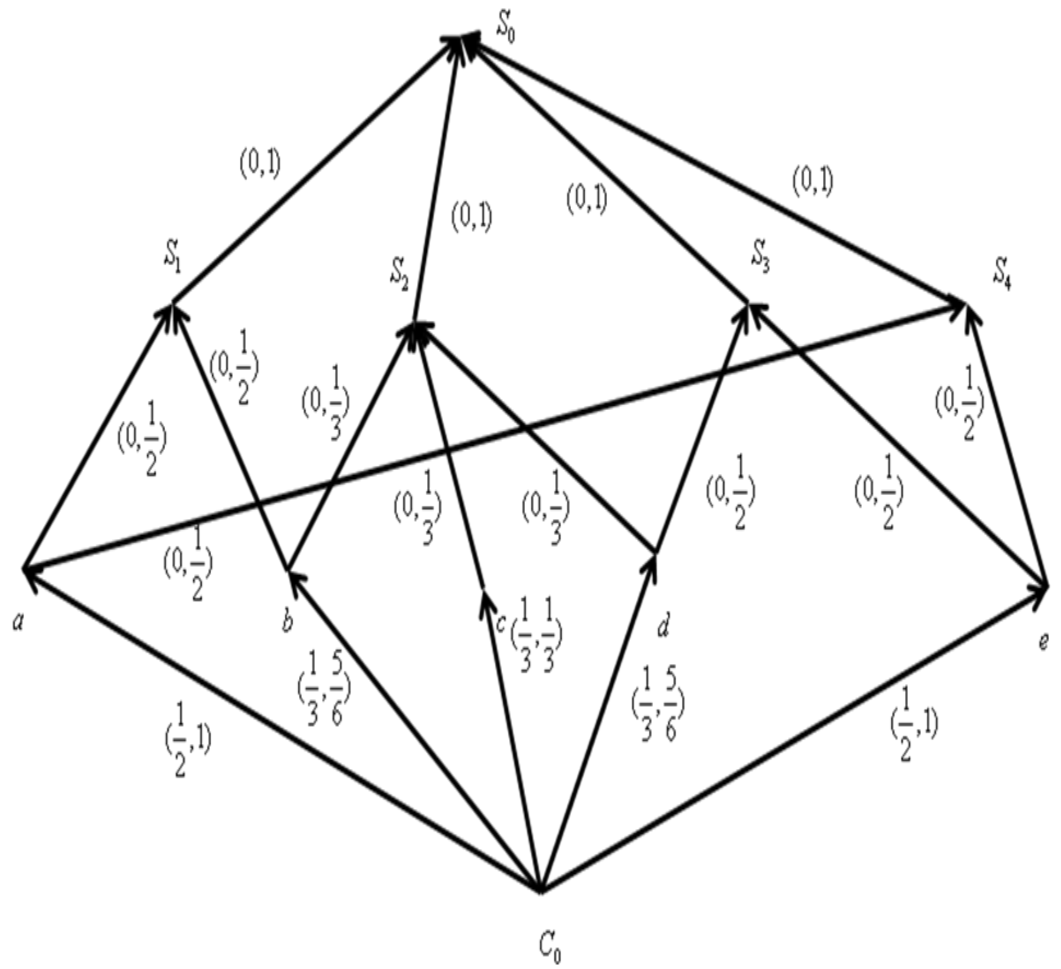

Fig-4.2 Layered Network N for Bipartite Graph in Fig-4.1

4.2 Minimum Flow Algorithm for Set Cover Problem [27]

Mohamed et. al gave following algorithm for solving set cover problem. Algorithm uses *t*(*a*) to denote the extremity of arc a and *i*(*a*) to denote initial extremity.

```
begin
    $S^* \leftarrow \varnothing$
    repeat
            CONSTRUCT(N);
            $\varphi \leftarrow \{c_a : \forall a \in A\}$
            repeat
                stop ← false; $\delta(c_0) \leftarrow \infty; X' \leftarrow \{c_0\}; A' \leftarrow \varnothing$
                while ($s_0 \notin X'$) $\wedge \neg$ stop do
                    if $\exists a = xy \in A, x \in X' \wedge y \in X \setminus X' \wedge \varphi_a > b_a$ then
                        $X' \leftarrow X' \cup \{y\};\ A'(y) \leftarrow a;$
                        $\delta(y) \leftarrow \min\{\delta(y), \varphi_a - \varphi_b\};\ A' \leftarrow A' \cup \{a\}$
                        else stop ← true
                    end-if
                end-while
                if $s_0 \in X'$ then
                    $\delta \leftarrow \delta(s_0)$
                  else $\delta \leftarrow 0$
                end-if
                if $\delta \neq 0$ then
                    $x \leftarrow s_0$;
                    while $x \neq c_0$ do
                        $a \leftarrow A^{'}(x);\ \varphi_a \leftarrow \varphi_a - \delta;\ x \leftarrow i(a)$
                    end-while
                end-if
            until $\delta = 0$
            $S^* \leftarrow S^* \bigcup \{s_i : s_i s_0 = \arg\max\{\varphi_a : a \in E_s\}\}$;
            $\overline{A} \leftarrow \{s_i s_0\} \bigcup \{a : t(a) = s_i\} \bigcup \{c_0 c_j : c_j \in \Gamma^-(s_i)\}$;
            $C \leftarrow C \setminus \Gamma^-(s_i);\ S \leftarrow S \setminus \{s_i\}; A \leftarrow A \setminus A$
    until $C = \varnothing$
end
```

Procedure CONSTRUCT uses the definition 1 to transform either the original set cover problem or its surviving instance after deleting of an *S*-vertex and its *C*-neighborhoods, on outer repeat loop.

The inner repeat loop gives the minimum flow algorithm. It computes a flow reduction δ and a reducing path along which the flow will be reduced by δ. Algorithm starts with a feasible solution, it saturates all arcs of N, and tries to reduce flow from c_0 to s_0. The reduction step δ along a path $[c_0 c_j s_i s_0]$ is defined recursively as follows:

$\delta(c_0) = \infty,$ for each arc $a = xy \in A,$ $\delta(y) = \min\{\delta(x), \varphi_a - b_a\}$ and finally $\delta = \delta(s_0).$

Once a minimum flow has been obtained, algorithm checks if the flows on the arcs of E_s are integral or not. If the flows are not integral and given that flows are rational numbers, algorithm chooses the *S*-vertex through which the maximum flow closest to 1 is sent. Algorithm then deletes this vertex and its neighbors in *C*. Then upper and lower capacities in the surviving network are re-calculated. Thus at the end of each iteration, a number of *C*-vertices of N are deleted.

Therefore, after a number of steps all *C*-vertices will be removed, which implies they are covered by *S*-neighbors. The algorithm treats every path of algorithm only once because the way φ_a and φ_b are computed for $a \in E$.

Example 4.2: Find set cover solution, for the problem defined in Example 4.1, using minimum flow network algorithm.

First the flow values need to be calculated. Fig-4.3 shows the flow values, calculated by the network flow algorithm, for the network shown in Fig-4.2. Algorithm selects the vertex S_2, because of the maximum flow value on the arc S_2S_0, on the arcs of E_s.

After selecting S_2, this vertex and all its neighbors are deleted. Fig-4.4 shows the surviving network after deleting vertex S_2 and its neighborhood. There after flow values are recalculated on the surviving network. Fig-4.5 shows recalculated flow value for surviving network.

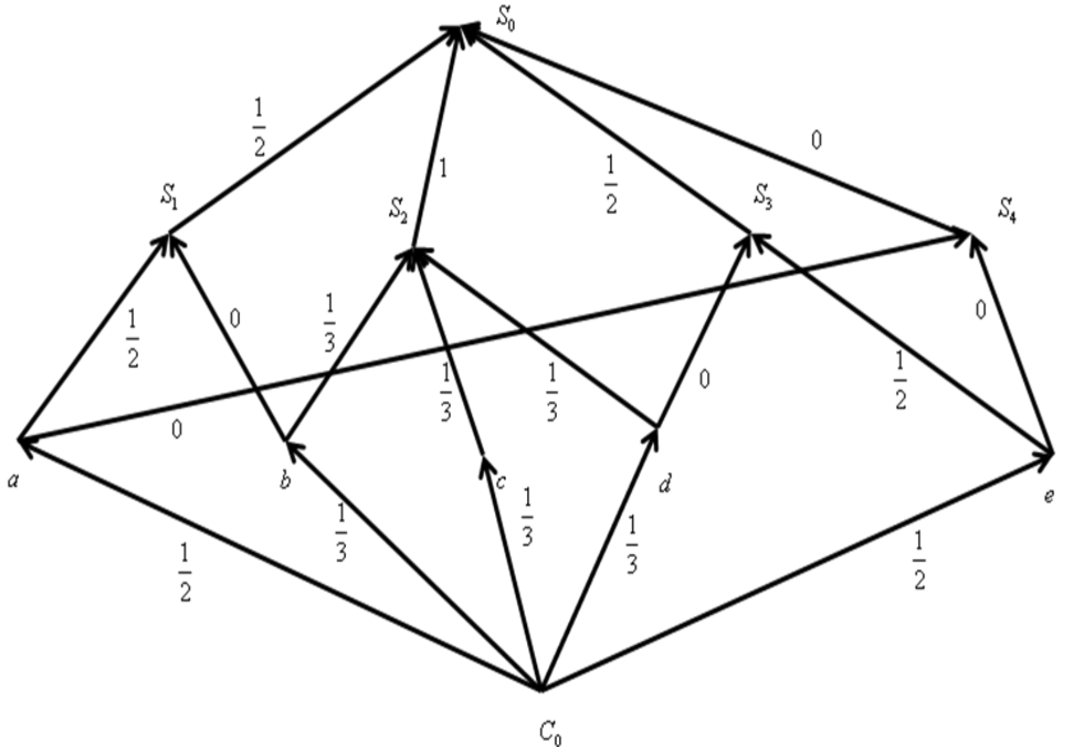

Fig-4.3 Network Constructed for Example 4.1

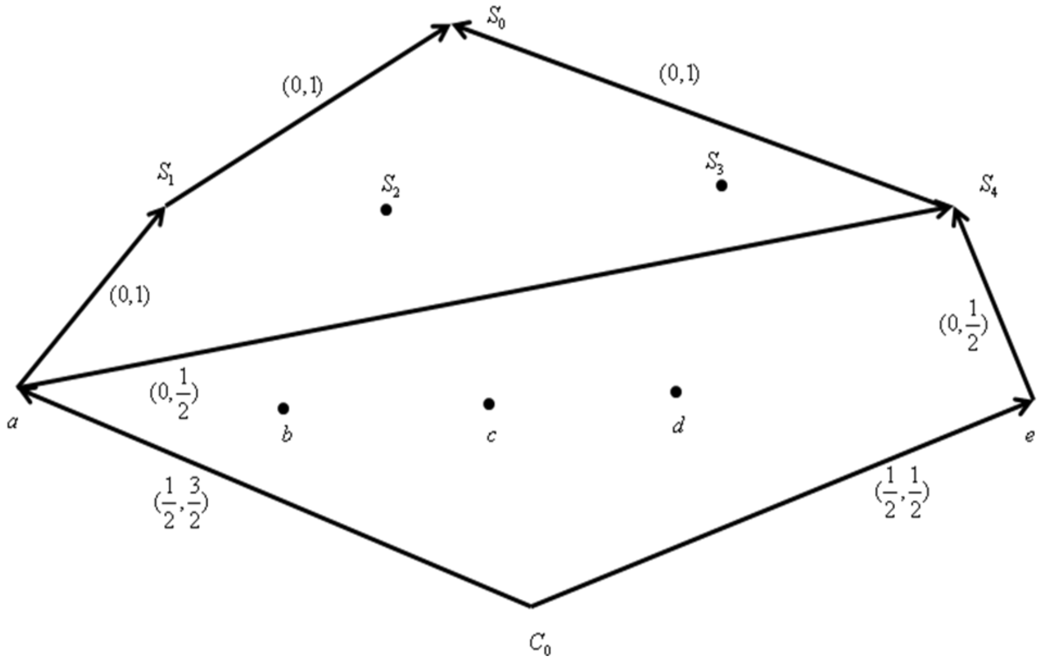

Fig-4.4 Surviving Network After Deleting Vertex S_2 and Its Neighbors

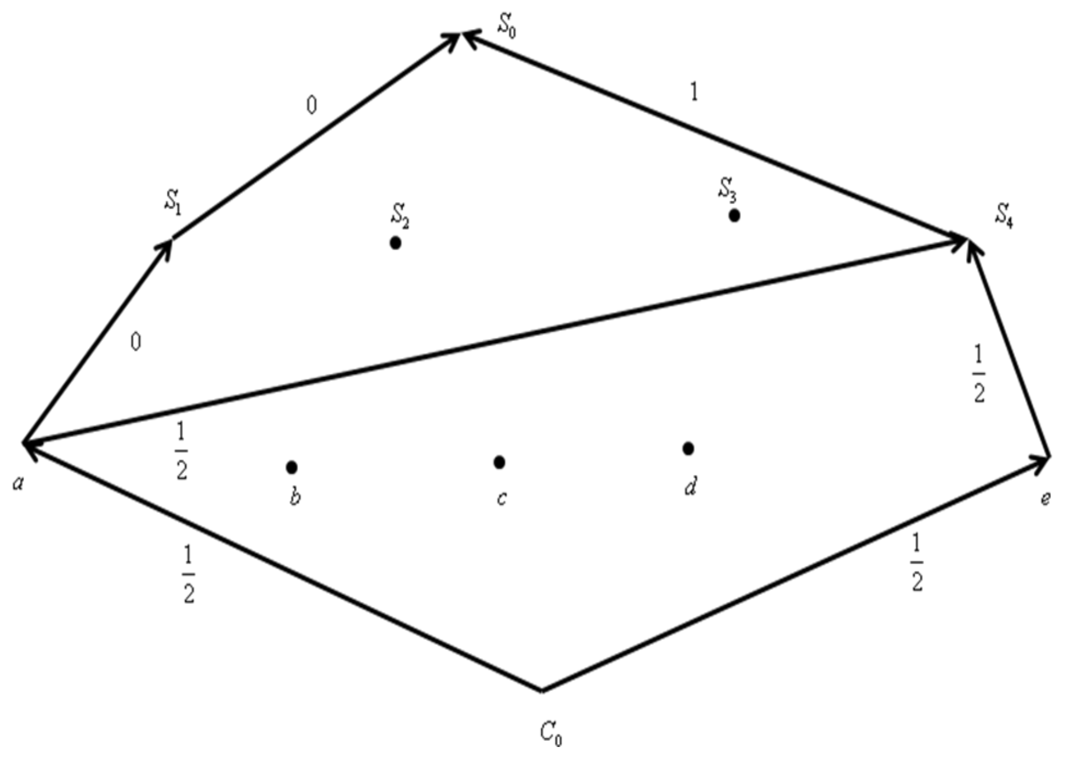

Fig-4.5 Recalculated Flow Values for Surviving Network

Therefore, during second and last iteration, algorithm selects vertex S_4 in the solution, because of maximum flow value on the arc S_4S_0, on the arcs of E_s.. Therefore, $S^* = \{S_2, S_4\}$. It can be easily observed that this is the set cover.

4.3 Performance Analysis of Minimum Flow Algorithm [27]

If every time the algorithm chooses to reduce the flow along the $[c_0c_js_is_0]$ such that c_0c_j has smaller lower capacity and the arc c_js_i has the smaller upper capacity, then this algorithm produces standard greedy algorithm.

On the other hand if the algorithm chooses reducing path arbitrarily, then solution is not always Johnson's standard greedy algorithm.

For large *mn* values, the average execution times for the Johnson's standard greedy algorithm and minimum flow algorithm of set cover heuristics become almost identical.

CHAPTER 5

CONCLUSION

Johnson showed the performance ratio for standard greedy algorithm is $H(k)$. This lies between $\ln k$ and $\ln k+1$. The complexity of standard greedy algorithm is of $O(mn)$, where m is the size of universal set(ground set or base set) and n is total number of sets. By maintaining a priority queue standard greedy algorithm can be implemented in such a way that it runs in time $O(\sum_{S_i \in S} |S_i|)$.

For 20 years this remained the bound for standard greedy set cover algorithm. Petr Slavik in 1996 [13] proved that the approximation ratio of standard greedy algorithm is exactly $\ln m - \ln \ln m + \Theta(1)$. Here m is the size of ground set or universe. Also the lower and upper bound differ by less than 1.1.

For standard greedy algorithm the upper and lower bounds are:

$$\frac{c_{greedy}}{c_{\min}} < \ln m - \ln \ln m + 0.78$$

$$\frac{c_{greedy}}{c_{\min}} > \ln m - \ln \ln m - 0.31$$

Lund and Yanakakis have established that set covering cannot be approximated with ratio c $\log_2 m$ for any c < l/4 unless all NP problems are solvable in $DTIME(m^{poly\log m})$. Feigh proved stronger result showing that for any $\varepsilon > 0$, no polynomial time algorithm can approximate set cover within (1 - ε) $\ln m$, unless

$NP\,TIME[m^{O(\log\log m)}]$. The above results rule out drastic improvement of standard greedy algorithm. But enhancements are possible. Till now best known enhancement of standard greedy is Asaf Levin algorithm. It has a performance ratio of $H(k)-\frac{196}{390}$ and the time complexity of $O(m^5n^6)$.

I have also implemented the three algorithms and created a software library that stores the code I have written.

Table-5.1 does the comparative study for different set cover algorithms presented. For set cover problem, optimal solution is possible set cover is possible for set size of 2. All the enhancements use this result to obtain improvement over standard greedy algorithm.

In nutshell, standard greedy algorithm is near optimal solution to set cover problem and Lund's result rules out any drastic improvement of standard greedy algorithm.

Table-5.1 Performance Ratio for Different Set Cover Algorithms

Name of the algorithm	Performance ratio
Johnson's Standard Greedy algorithm	$H(k)$
Goldsmidt, Hochbaum and Yu's Modified Greedy algorithm	$H(k)-\frac{1}{6}$
Halldorsson's Local optimization technique	$H(k)-\frac{11}{42}$
Dur and Furer semi local optimization	$H(k)-\frac{1}{2}$
Asaf Levin's improvement to Dur and Furer algorithm	$H(k)-\frac{196}{390}$
LP simple rounding algorithm	f
LP randomized rounding algorithm	$4\ln(4n)$
LP duality	f
LP Primal-Dual Schema	f
Network flow technique	$H(k)$

Note: $H(k)$ is the harmonic series and k is the size of the largest set.

f is the frequency of the most frequent element where frequency is the maximum number of sets that an element belongs.

n is total number of sets.

REFERENCES

[1] T. Erlebach and E. J. V. Leeuwen, “Approximating Geometric Coverage Problems,” in *Proc. Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1267-1276, Jan 2008.

[2] M. Franceschetti, M. Cook and J. Bruck, “A Geometric Theorem for Appoximate Disk Covering Algorithms,” *Technical Report ETR035,* (Caltech University, USA), January 2001.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, First edition, MIT Press and McGraw-Hill, 1990.

[4] S. A. Cook, “The Complexity of Theorem Proving Procedures,” in *Proc. Third Annual ACM Symposium on Theory of Computing*, pp. 151-158, 1971.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, First edition, W. H. Freeman and Company, San Fransisco, 1979.

[6] D. S. Johnson, “The NP-Completeness Column: On Ongoing Guide,” *Journal of Algorithms*, vol. 13, pp. 502-524, 1992.

[7] D. B. West, *Introduction to Graph Theory*, Second Edition, Prentice Hall, 2001.

[8] D. S. Johnson, “Approximation Algorithms for Combinatorial Problems.” *Journal of Computer Systems Science,* vol. 9, pp 256 –278, 1974.

[9] L. Lovasz, “On the Ratio of Optimal Integral and Fractional Covers,” *Discrete Mathematics*, vol. 13, pp. 383-390, 1975.

[10] V. Chvatal, “A Greedy-heuristic for the Set Covering Problem,” *Mathematics of Operations Research*, vol. 4, pp. 233-235, 1979.

[11] N. E. Young, “Greedy Set-Cover Algorithms: 1974-1979, Chvatal, Johnson, Lovasz, Stein,” *Encyclopedia of Algorithms*, Springer, 2008.

[12] V. T. Paschos, “A Survey of Approximately Optimal Solutions to Some Covering and Packing Problems,” *ACM Computing Surveys*, vol. 29(2), pp. 171-209, 1997.

[13] C. Lund and M. Yannakakis, “On the Hardness of Approximating Minimization Problems,” *ACM Journal*, vol. 41(5), pp. 960-981, 1994.

[14] U. FEIGE, “A Threshold of ln n for Approximating Set Cover,” *Journal of the ACM*, vol. 45,(4), , pp. 634 –652, 1998.

[15] Petr Slavik, “A Tight Analysis of the Greedy Algorithm,” in *Proc. Twenty-eighth Annual ACM Symposium on Theory of Computing*, pp. 435– 441, 1996.

[16] D. P. Williamson, “Lecture Notes on Approximation Algorithms,” *IBM Research Report RC 21273*, 1999.

[17] O. Goldschmidt, D. S. Hochbaum and G. Yu, “A Modified Greedy Heuristic for the Set Covering Problem with Improved Worst Case Bound,” *Information Processing Letters*, vol. 48, pp. 305-310, 1993.

[18] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, First edition, Holt, Rinehart and Winston, New York, 1976.

[19] S. Micali and V. V. Vazirani, “An O($\sqrt{|V|}\ |E|$) Algorithm for Maximum Matching in General Graphs,” in *Proc. IEEE Annual Symposium on Foundations of Computer Science*, pp. 17-27, 1980.

[20] S. Even and O. Kariv, “An O(n2.5) Algorithm for Maximum Matching in Graphs,” in *Proc. Sixteenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 100-112, 1975.

[21] Z. Galil, “Efficient Algorithms for Finding Maximum Matching in Graphs,” *Computing Surveys*, vol. 18(1), pp. 23-38, 1986.

[22] M. M. Halldorsson, “Approximating Discrete Collections via Local Improvements,” in *Proc. Symposium on Discrete Algorithms*, pp. 160–169, 1995.

[23] R. Duh and M. Furer, “Approximation of k-Set Cover by Semi Local Optimization,” in *Proc. Twenty-ninth Annual ACM Symposium on Theory of computing*, pp. 256-264, 1997.

[24] A. Levin, “Approximating the Unweighted k-Set Cover Problem: Greedy Meets Local Search,” *WAOA LNCS 4368*, pp. 290-301, 2006.

[25] R. Hassin and A. Levin, “A Better than Greedy Approximation Algorithm for the Minimum Set Cover Problem,” *SIAM Journal on Computing*, vol. 35(1), pp. 189-200, 2005.

[26] V. V. Vazirani, *Approximation Algorithms*, First edition, Springer, 2001.

[27] M. Afif, M. Hifi, V. T. Paschos and V. Zissimopoulos, “A New Efficient Heuristic for the Minimum Set Covering Problem,” *The Journal of the Operational Research Society*, vol. 46(10), pp. 1260 -1268, 1995.

[28] F. D. Croce and V. T. Paschos, “Computing Optimal Solutions for the Min 3-Set Covering Problem,” in *Proc. ISAAC, LNCS 3827*, pp. 685-692, 2005.

[29] F. D. Croce, B. Escoffier and V. T. Paschos, “Improved Worst Case Complexity for the Min 3-Set Covering Problem,” *Operations Research Letters*, pp. 205-210, 2007.