PRIMAL-DUAL HEURISTICS FOR SOLVING THE SET COVERING PROBLEM

by

BELMA YELBAY

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
Fall 2009

PRIMAL-DUAL HEURISTICS FOR SOLVING
THE SET COVERING PROBLEM

APPROVED BY

Assoc. Prof. Ş. İlker Birbil (Thesis Supervisor)	....
Assist. Prof. Kerem Bülbül (Thesis Co-advisor)	....
Assist. Prof. Nilay Noyan	....
Assist. Prof. Güvenç Şahin	....
Assist. Prof. Hüsnü Yenigün	....

DATE OF APPROVAL: ....

to my son

Acknowledgments

I owe my deepest gratitude to my thesis advisor Assoc. Prof. Ş. İlker Birbil for his invaluable support, supervision, advice and guidance throughout the research.

I gratefully acknowledge my thesis co-advisor Assist. Prof. Kerem Bülbül for his advice, supervision and crucial contribution to my thesis. This thesis would not have been possible without my advisors' encouragement and support.

I am indebted to all my friends from Sabanci University for their motivation and endless friendship. My special thanks go in particular to Figen for her endless patience and valuable support. Many special thanks go to İbrahim, Ceyda, Nurşen, Taner, Mahir, Duygu, Birol, Vildan, Halil, Semih, Merve, Nimet and Selin.

My parents deserve special mention for their invaluable support and gentle love. Words fail me to express my appreciation to my son whose love always encourages and motivates me. Lastly, I would like to thank my husband for his dedication, love and persistent confidence in me.

PRIMAL-DUAL HEURISTICS FOR SOLVING THE SET COVERING PROBLEM

Belma Yelbay

Industrial Engineering, Master of Science Thesis, 2009

Thesis Supervisors: Assoc. Prof. Ş. İlker Birbil
Assist. Prof. Kerem Bülbül

Keywords: combinatorial optimization, set covering, primal-dual approach, heuristics

Abstract

The set covering problem (SCP) is a well known combinatorial optimization problem applied widely in areas such as; scheduling, manufacturing, service planning, network optimization, telecommunications, and so on. It has been already shown that SCP is NP-hard in the strong sense [15]. Therefore, many heuristic and enumerative algorithms have been developed to solve SCP effectively. The primary purpose of the present study is to develop an effective heuristic for SCP. The heuristic is based on a primal-dual approach which is commonly used in the literature for approximating NP-hard optimization problems.

In this study, we present numerical results to evaluate the performance of the heuristic as well as our observations throughout the development process. Our results indicate that the heuristic is able to produce good results in terms of both solution quality and computation time. Moreover, we show that the proposed heuristic is simple, easy to implement and has a potential to solve large-scale SCPs efficiently.

KÜME ÖRTÜLEME PROBLEMLERİNİN ÇÖZÜMÜ İÇİN TEMEL-EŞLENİK SEZGİSELLER

Belma Yelbay

Endüstri Mühendisliği, Yüksek Lisans Tezi, 2009

Tez Danışmanları: Doç. Dr. Ş. İlker Birbil
Yrd. Doç. Dr. Kerem Bülbül

Anahtar Kelimeler: birleşi eniyileme, küme örtüleme, temel-eşlenik yaklaşım, sezgiseller

Özet

Küme örtüleme problemi çizelgeleme, üretim, hizmet planlama, ağ eniyilemesi, uziletişim gibi pek çok alanda uygulanan iyi bilinen bir birleşi eniyileme problemidir. Küme örtüleme probleminin NP-zor olduğu kanıtlanmış olup, problemin etkin bir şekilde çözümüne yönelik çok sayıda birerleme algoritmaları ve sezgiseller geliştirilmiştir. Bu çalışmanın temel amacı küme örtüleme problemi için etkin bir sezgisel geliştirmektir. Sezgisel temel-eşlenik yaklaşımına dayalı olup literatürde NP-zor birleşi eniyileme problemlerini yaklaşıklamak için kullanılmaktadır.

Bu çalışmada, sezgiselin başarımını değerlendirmek için sayısal çözümlemelerle birlikte sezgiselin geliştirilme aşamalarında elde ettiğimiz gözlemler sunulmaktadadır. Elde edilen sonuçlar sezgiselin çözüm kalitesi ve hesaplama zamanı açısından iyi sonuçlar verdiğini göstermektedir. Bunun yanısıra sezgiselin basit, kolay uygulanabilir, ve büyük ölçekli problemleri etkin bir şekilde çözme potansiyeli olduğunu göstermektedir.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION AND MOTIVATION

The set covering problem is a well known combinatorial optimization problem applied widely in areas such as; scheduling, manufacturing, service planning, network optimization, telecommunications, and so on. Given a collection $\mathcal{S}$ of sets over a finite universe $\mathcal{U}$, a set cover $\mathcal{C} \subseteq \mathcal{S}$ is a sub-collection of the sets whose union is $\mathcal{U}$. When each set in the collection has a cost, then the set covering problem is about finding a set cover $\mathcal{C}$ such that the cost is minimized.

The integer programming (IP) formulation of the set covering problem (SCP) is as follows:

$$\text{minimize} \quad \sum_{j \in \mathcal{S}} c_j x_j \tag{1.1}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{S}} a_{ij} x_j \geq 1, \quad i \in \mathcal{U}, \tag{1.2}$$

$$x_j \in \{0, 1\}, \quad j \in \mathcal{S}. \tag{1.3}$$

and $c_j > 0$ is the coverage cost of the j^{th} set. x_j is a binary variable which equals 1 if $j \in \mathcal{C}$, and 0 otherwise. S_j is the set of items that can be covered by set j. a_{ij} is a binary constant which equals 1 if $i \in S_j$, and 0 otherwise. Constraints (1.2) ensure that each item is covered by at least one set, and constraints (1.3) are the integrality restrictions. If the cost of coverage is the same for each set, that is $c_1 = c_2 = \cdots = c_n$ with $n = |\mathcal{S}|$ then the problem is referred to as the unicost set covering problem. Since, solving the model is hard, some algorithms use the optimal solution of the LP relaxation or its dual to find a lower bound to the optimal solution or find a near-optimal solution. In the subsequent part of the thesis, we refer to the following LP relaxation model

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j && (1.4) \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i \in \mathcal{U}, && (1.5) \\ & x_j \geq 0, \qquad j \in \mathcal{S}. && (1.6) \end{aligned}$$

The corresponding dual problem then becomes

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} y_i && (1.7) \\ \text{subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq c_j, \quad j \in \mathcal{S}, && (1.8) \\ & y_i \geq 0, \qquad i \in \mathcal{U}. && (1.9) \end{aligned}$$

and $c_j - \sum_{i=1}^{m} a_{ij} y_i$ is the value of the j^{th} dual slack variable or the reduced cost of the j^{th} the primal variable. It has been already shown that SCP is NP-hard in the strong sense [15]. Therefore, many heuristic and enumerative algorithms have been developed to solve SCP effectively.

1.1 Contributions

The primary purpose of the present study is to develop an effective heuristic for SCP. The following list shows the contributions of this study:

- In this study, we present a heuristic for SCP. The heuristic is based on a primal-dual approach which is commonly used in the literature for approximating NP-hard optimization problems.
- The heuristic has a potential to solve large-scale SCPs in a reasonable time through column generation.
- We show that despite the fact that the theoretical performance of the heuristic is poor, the empirical performance is quite well.
- The heuristic is simple, easy to implement and fast.

- We present numerical results to evaluate the performance of the heuristic as well as our observations throughout the development process.
- We show that the problem structure affects the performance of the heuristic. This provides some insight into the behavior of the primal-dual heuristic.

1.2 Outline

The thesis is structured as follows. We start with the the combinatorial optimization literature on SCP in Chapter 2. We introduce the proposed primal-dual heuristics and present our observations in Chapter 3. In Chapter 4, the computational study is given. The thesis ends with Chapter 5 which includes conclusions and directions for future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

The combinatorial optimization literature on the set covering problem is immense. Studies related to the set covering problem are divided into two major groups. Researchers in the first group try to solve stochastic/probabilistic set covering problems, whereas in the second group the focus is on deterministic set covering problems. Our goal in this chapter is to analyze the deterministic set covering problems. Thus, we only focus on the deterministic set covering literature. Methods can be analyzed under two main headings as exact algorithms and heuristics. Our algorithm falls under the second heading so the literature that is closely related to our work will be given in Section 2.2.4.

2.1 Exact Algorithms

Exact algorithms generally rely on the branch and bound method to obtain the optimal solution. As a bounding procedure, a common approach is to apply Lagrangian relaxation to the coverage constraints (1.2). After Lagrangian relaxation, the objective function of IP model (1.1-1.3)becomes

$$\max_{\substack{\lambda_i \geq 0, \\ i \in \mathcal{U}}} \left\{ \min_{\substack{x_j \in \{0,1\}, \\ j \in \mathcal{S}}} \sum_{j=1}^{n} \left(c_j - \sum_{i=1}^{m} \lambda_i a_{ij} \right) x_j + \sum_{i=1}^{m} \lambda_i \right\} , \tag{2.1}$$

where the Lagrange multiplier for the coverage constraint of item i is denoted by λ_i and $m = |\mathcal{U}|$.

It is well-known that the optimal value of (2.1) for a fixed set of $\lambda_i, i \in \mathcal{U}$, can be used as a lower bound for the optimal IP solution. This lower bound is easily calculated for a given set of λ_i, because the value of x_j depends only on the sign of the coefficient of x_j in the objective function. The value of x_j is equal to 1, if the sign of the coefficient is negative, and 0; otherwise. The objective is to find the best set of Lagrange multipliers that yield the tightest lower bound. This is achieved by changing the values of the multipliers, solving the Lagrangian relaxation model, and then updating the lower bound iteratively. In the literature, there are several methods to calculate the multipliers like the subgradient approach.

Beasley [4] uses subgradient optimization and a heuristic algorithm to give a lower and an upper bound for the SCP. At each iteration, the Lagrangian relaxation model is solved by the current set of Lagrange multipliers. The subgradient procedure is used to update the Lagrange multipliers and improve the lower bound. The value of the initial Lagrange multipliers is calculated by a dual ascent procedure, which relies on finding a feasible solution to the dual (1.7-1.9) of the linear programming relaxation of the SCP. Then, the solution obtained after the subgradient procedure is used to find the optimal integer solution of SCP by using a tree search procedure. Beasley and Jornsten [7] use the same method but improve the solution quality through Gomory f-cuts with a better branching strategy. Fisher and Kedia [14] use a primal and a dual heuristic to find an upper and a lower bound for the branch-and-bound procedure. Similarly, Balas and Carrera [2] use a primal and a dual heuristic for upper and lower bounding procedure, but they iteratively improve the bounds by fixing some of the variables at 1 and then changing the current set of Lagrange multipliers by a dynamic subgradient procedure.

2.2 Heuristics

Solving large SCPs optimally through exact algorithms takes excessively long time. Therefore, especially for those kind of problems, sacrificing optimality by using a heuristic rather than an exact algorithm is more preferable, because heuristic algorithms give near-optimal solutions in a reasonable time. Caprara *et al.* [11] and Grossman and Wool [17], and Gomes *et al.* [16] give lists of various heuristics and approximation algorithms, and they all test their performances on some instances. Results show that although the theoretical (worst case) performance bounds are poor, their empirical performances are quite well. In the literature, there are several approaches to develop a heuristic algorithm. Among these, we have greedy algorithms, linear programming and Lagrangian relaxations, randomized search, primal-dual methods. In the subsequent sections, we review these approaches and some of the related studies in the SCP literature.

2.2.1 Greedy Algorithms

Greedy algorithms can be used to solve large scale set covering problems very quickly but their myopic nature may easily yield suboptimal solutions. Greedy algorithms use some rules to determine the variable x_j that will be set to 1 at each iteration. For example:

- Each constraint is checked and if it is not satisfied then all the x_j variables of that constraint are set to 1.
- The variable corresponding to the set, which contains the largest number of uncovered elements, is set to 1.
- The ratio c_j/k_j is calculated for each variable, where k_j denotes the number of currently uncovered items that could be covered by set j. Then, the variables are sorted in nondecreasing order according to their c_j/k_j values and the variable having the minimum value of c_j/k_j is set to 1.

In the literature, there are similar greedy algorithms, with different selection criteria. Some examples are c_j/k_j^2, $c_j/k_j \log(1+k_j)$, $c_j^{1/2}/k_j$, and $c_j/k_j^{1/2}$ [22]. Algorithms proposed by Lan et al. [22] and Vasko and Wilson [27] use the combinations of those ratios and select one of them randomly at each iteration.

2.2.2 Linear Programming and Lagrangian Relaxations

The optimal solution of the linear programming (LP) relaxation of SCP (1.4-1.6) can be used to find a solution for SCP. Algorithms using the solution of the LP relaxation model can be summarized as follows:

- In Hochbaum's algorithm [20], the variable having value $x_j^* \geq 1/f$ is set to 1, where f denotes the maximum number of ones per row and x_j^* is the optimal solution of the LP relaxation model (1.4-1.6).
- Peleg et.al. [25] sort the x_j^* variables in a nonincreasing order. If the first unchecked variable in the list is included in an unsatisfied constraint, then the value of that variable is set to 1.
- In an alternate algorithm, Hochbaum [20] adds j^{th} variable to the cover, if $x_j^* > 0$.

An alternate method is to use the optimal solution of the dual of the LP relaxation model. Hochbaum [20] obtains an optimal solution y^* to the dual of the LP relaxation. If the constraint that corresponds to the j^{th} primal variable is tight, that is $\sum_{i=1}^{m} a_{ij} y_i^* = c_j$, then value of the j^{th} primal variable is set to 1.

Since, solving the LP model optimally takes time especially for large SCPs, the Lagrangian relaxation model can be used as an alternate method to LP relaxation model. Ceria et al. [12] use a Lagrangian-based heuristic to solve large scale set covering problems. They iteratively use a subgradient approach to calculate the set of Lagrangian multipliers λ_i and the corresponding reduced costs. Then, they reduce the problem by eliminating some columns whose reduced costs are above a given threshold. After that they add some columns to the reduced problem to guarantee that a feasible solution can be obtained by solving the reduced problem, and this new problem is called the core problem. Finally, a heuristic algorithm is used to find a feasible solution to the core problem. Caprara *et al.* [10] also use a Lagrangian based heuristic to solve large scale set covering problems. They perform several subgradient iterations to find the best set of multipliers that give the best lower bound. Then, similar to Ceria *et al.* [12], they use a greedy heuristic to find a feasible solution to the core problem. As a different approach, they use reduced costs instead of actual costs in the selection rules of the greedy algorithm, and they apply a column fixing approach based on fixing those variables that have reduced costs below a certain threshold value.

2.2.3 Search Algorithms

The main idea behind the search algorithms is to find a good solution that is not a necessarily optimal until a satisfactory improvement is obtained or a user defined time bound is elapsed. In the literature, there are many heuristic algorithms that are based on search algorithms to find near-optimal solutions especially for large-scale set covering problems. In addition to those, there are some heuristics, which use search algorithms to improve the heuristics solutions.

Jacobs and Brusco [21] use a local search heuristic based on the simulated annealing algorithm to solve large scale non-unicost SCPs. They use a greedy algorithm to generate an initial feasible solution. After constructing the initial feasible solution, some of the columns in the solution are eliminated randomly, and new columns are identified to restore feasibility. Brusco *et al.* [9] use a simulated annealing heuristic for cost and coverage correlated set covering problems (problems in which there is a correlation between the cost of a column and the number of items covered by that set). They modify the simulated annealing heuristic such that they measure the similarities between two columns by calculating the fraction of covering the same item, called coverage index, and by interchanging only the columns having similar indices. Haouari and Chaouachi [19] propose a probabilistic greedy search algorithm, which gives satisfactory results for solving large scale set covering problems. They introduce randomness and this property sometimes enables to prioritize a column with higher cost against another one with lower cost. They extend the search region by using perturbed costs instead of the original costs, and introduce a penalization procedure to expand the search towards unexplored regions. Similarly, the meta-heuristic developed by Lan et al. [22] uses randomness and penalization. The flexible structure of this algorithm gives satisfactory results for both uni-cost and non-unicost SCPs as well as multi-dimensional knapsack problems, traveling salesman problems and resource constrained project scheduling problems.

In the literature, there is a number of genetic algorithms proposed for the SCP. Beasley and Chu [5] present a genetic algorithm to solve the non-unicost SCP. Authors modify the classical genetic algorithm by using a fitness based crossover operator and a variable mutation rate. They use a heuristic to convert the solution of the genetic algorithm to a feasible solution. The proposed algorithm solves small instances optimally and generates high quality solutions for the large scale set covering problems. Lorena and Lopes [23] use a similar genetic algorithm to solve difficult SCPs by introducing a local search algorithm to find an initial feasible solution. They test the performance of the algorithm by generating hard instances such that each row has exactly three ones, and for each column pair j and k, there is only one row i such that $a_{ij} = a_{ik} = 1$. Aickelin [1] proposes a quite different algorithm. The genetic algorithm is used only to generate a permutation of rows rather than a permutation of columns and this row sequence is converted to a feasible column sequence by another procedure. Finally, the solution is improved by a post-processing procedure.

2.2.4 Primal-Dual Algorithms

The primal-dual approach is commonly used for approximating NP-hard optimization problems that can be modeled as integer programming problems. The primal and dual approaches in Section 2.2.2 are based on using the optimal solution of the primal (1.4-1.6) and dual (1.7-1.9) of the LP relaxation model. Since both approaches need to find an optimal solution to a linear programming model, the computation time of the algorithms using those approaches is high especially for large-scale problems. On the other hand, the primal-dual approach is based on finding only a feasible solution to the dual of the LP relaxation model (1.4-1.6); using this solution, an integral solution for the SCP is found. It is not necessary to solve a linear programming problem to optimality. Therefore, the performance of the algorithms using primal-dual approach is usually better than others in terms of computation time. However, it has already been proven that [18], the worst case performance of the primal-dual algorithm given by Algorithm 1 is known to be fz, where z denotes the optimal primal objective function value. Hall and Vohra [18] give a worst case instance that shows that the theoretical bound fz is attainable.

Vazirani [28] gives a list of problems, for which this approach is appropriate to apply. This list includes the metric traveling salesman problem, the Steiner tree problem, the Steiner network problem, and the set covering problem. Bar-Yehuda and Even [3] are the first researchers who consider a primal-dual approach to approximate the set covering problem with Algorithm 1.

Algorithm 1 Primal-Dual Algorithm

```
1: Initialize:
2: $y_i = 0 \ \forall i \in \mathcal{U}$
3: $\bar{c}_j = c_j - \sum_{i=1}^{m} a_{ij} y_i = c_j \ \forall j \in \mathcal{S}$
4: $J = \emptyset$                    // J will contain the set of indices picked for the cover
5: $x_j = 0 \ \forall j \in \mathcal{S}$
6: while there are uncovered items do
7:     pick an uncovered item $i \in \mathcal{U}$        // thus all $x_j$ with $i \in S_j$ must be 0
8:     pick an index $k = \arg\min_j \{\bar{c}_j | i \in S_j\}$
9:     $y_i = \bar{c}_k$
10:    for $j \in \{j | i \in S_j\}$ do
11:        $\bar{c}_j = c_j - \sum_{i=1}^{m} a_{ij} y_i$        // this will make $\bar{c}_k = 0$
12:    end for
13:    $x_k = 1$ and $J = J \cup \{k\}$
14: end while
15: return the sets $S_j$ with $j \in J$
```

The algorithm starts with a dual feasible solution by setting all of the dual variables to 0 and iterates while maintaining the dual feasibility. At each iteration, a dual constraint j with the minimum additional reduced cost $\bar{c}_k$ is selected (Algorithm 1, line 8) and one of the dual variables in that constraint is set to the value of $\bar{c}_k$ to make the constraint binding. The value of primal variable corresponding to that constraint is set to 1 and the $\bar{c}_k$ values are updated. The algorithm ends when all items are covered.

Since at least one additional item is covered at each iteration, the number of iterations can be at most m. Using this approach, all the following conditions are satisfied at each iteration:

1. $x_j \in \{0,1\},\ \forall j \in \mathcal{S}$.
2. $y_i \geq 0\ \forall i \in \mathcal{U}$ and $\bar{c}_j{=}c_j - \sum_{i=1}^{m} a_{ij}y_i \geq 0\ \forall j$ (dual feasibility).
3. $x_j > 0 \Rightarrow \bar{c}_j = 0\ \forall j \in \mathcal{S}$ (complementary slackness).
4. $y_i > 0 \Rightarrow\ \sum_{j=1}^{n} a_{ij}x_j \geq 1\ \forall i \in \mathcal{U}$,

Bertsimas and Vohra [8] propose a primal-dual algorithm that is similar to Algorithm 1. The only difference is that at each iteration, instead of selecting an uncovered element, a set k that has a minimum reduced cost, i.e., $k = \arg\min_{i \in S_j}\{\bar{c}_j\}$, is selected. Next, one of the uncovered items i that could be covered by the set k is determined. Therefore, the number of iterations can be at most n, that is x_j may be set to 1 even if this set covers no additional uncovered item. Williamson [26] uses Algorithm 1 but introduces a post-processing procedure. This procedure searches and eliminates the redundant columns in the solution as long as the elimination does not violate the primal feasibility. At each iteration of Algorithm 1, only one dual variable is increased (or saturated). As a different method, Williamson permits multiple saturation in one iteration. Melkonian [24] uses a similar algorithm to Williamson's, but he combines the *interior point method* and the primal-dual approach. This method keeps the dual solution in the interior feasible region rather than on the boundary. This can be provided by increasing value of the dual variable until the value of the dual variable is λ times the reduced cost of the corresponding constraint ($y_i = \lambda\bar{c}_k$) where $0 < \lambda < 1$. The advantage of this method to prevent the solution from getting stuck with a solution of high objective function value. Since the dual objective function value is smaller, the approximation ratio is $1/\lambda$ times worse than the original algorithm. However, it is expected to obtain a lower primal objective function by applying this method.

CHAPTER 3

PROPOSED PRIMAL-DUAL HEURISTICS

Our main heuristic resembles the primal-dual heuristic described in the previous chapter (Algorithm 1). As an alternate approach, we consider a dual variable selection method and expect that this method decreases the primal objective function value when compared to the solution found by Algorithm 1.

3.1 Main Algorithm

We know that the following two properties are satisfied by the Algorithm 1:

- Property 1. $\mathcal{C} = \{j \in J | c_j = \sum_{i=1}^{m} a_{ij} y_i\}$
- Property 2. $\sum_{j=1}^{n} c_j x_j = \sum_{j \in \mathcal{C}} c_j = \sum_{j \in \mathcal{C}} \sum_{i=1}^{m} a_{ij} y_i$

Property 2 shows the relationship between the value of the primal objective function and the dual variables. Since the value of the objective function depends on the number of times an item appears in the sets in $\mathcal{C}$, we want to avoid including an item in several sets that are selected. Hence, items or dual variables are sorted in nondecreasing order according to the number of sets in which they appear. The item that appears in the minimum number of sets is selected first in step 7 of Algorithm 1. In this way, we expect to decrease the primal objective function value. The pseudocode of our algorithm is given in Algorithm 2.

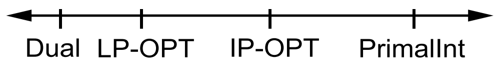
We first start with conducting a preliminary computational study to observe the performance of the main algorithm. The algorithm is set to solve 45 non-unicost and 5 unicost benchmark instances from Beasley's OR Library [6] with sizes ranging from 500 variables (sets) and 50 constraints (items) to 4000 variables to 400 constraints. Algorithm 2 provides a feasible solution to the dual of the LP relaxation of SCP, and an integer feasible solution to SCP. The associated objective function values are denoted as "Dual" and "PrimalInt" in Table 3.1. Columns 2-4 in Table 3.1 show the percentage

Algorithm 2 Main Algorithm

```
1: Initialize:
2: $y_i = 0 \ \forall i \in \mathcal{U}$
3: $\mathcal{U}^* = \mathcal{U}$                    // $\mathcal{U}^*$ is the set of currently uncovered items
4: $\bar{c}_j = c_j - \sum_{i=1}^{m} a_{ij} y_i = c_j \ \forall j \in \mathcal{S}$
5: $J = \emptyset$                    // $J$ will contain the set of indices picked for the cover
6: $x_j = 0 \ \forall j \in \mathcal{S}$
7: for $i = 1$ to $m$ do
8:     calculate $count_i$                    // total number of sets that cover item $i$
9: end for
10: sort the indices $i$ in nondecreasing order of their respective $count_i$ values, and put
    them into a list
11: while there are uncovered items do
12:     pick the first uncovered item in the list
13:     pick the smallest index $k = \arg\min_j \{\bar{c}_j | i \in S_j\}$
14:     $y_i = \bar{c}_k$
15:     for $j \in \{j | i \in S_j\}$ do
16:         $\bar{c}_j = c_j - \sum_{i=1}^{m} a_{ij} y_i$                    // this will make $\bar{c}_k = 0$
17:     end for
18:     $x_k = 1$, $J = J \cup \{k\}$ and $\mathcal{U}^* = \mathcal{U}^* \setminus S_k$
19:     remove all items in set $S_k$ from the list
20: end while
21: return the subsets $S_j$ with $j \in J$
```

gaps between the LP relaxation optimal value (LPOPT) and the dual objective function value (Dual), IP optimal value (IPOPT) and LPOPT, the primal objective function value (PrimalInt) obtained by the algorithm and IPOPT, respectively. Figure 3.1 shows the relative positions of the Dual, LPOPT, IPOPT, PrimalInt on the real axis. We expect that IP solution of the algorithm is close to IPOPT.

Figure 3.1: Positions of primal, dual objective function values and IP, LP optimal objective function values on the real axis



Algorithm 2 gives a set cover $\mathcal{C}$ as an output but this set cover may not be a minimal set cover. Thus, post-processing may be needed to check if the number of sets in $\mathcal{C}$ may be decreased without violating the coverage constraints. The largest decrease in objective function value after any kind of post-processing, over the selected columns found by the algorithm, cannot be better than solving SCP optimally. The last column in Table 3.1 shows the percentage gap between this best result we can get after any kind of post-processing (PostProc) and IPOPT. Table 3.1 shows the statistics for 45 non-unicost problems. The results show that by using this algorithm, the gap is 18.91% on average.

On the other hand, this gap is calculated as 16.03% for the non-unicost instances. When the instances are examined individually, it is observed that for the instances that IPOPT and LPOPT are close, this percentage gap decreases considerably. For example, there are three instances that IPOPT and LPOPT are equal and the gap between PrimalInt and IPOPT for these cases are 6.29%, 7.44%, and 7.55%, respectively.

Table 3.1: Percentage gaps for benchmark problems

Statistics	LPOPT-Dual	IPOPT-LPOPT	PrimalInt-IPOPT	PostProc-IPOPT
Avg	20.14%	2.68%	45.56%	18.91%
Median	18.49%	1.41%	44.29%	17.35%
Min	8.40%	0.00%	21.16%	6.25%
Max	43.95%	10.08%	89.66%	46.38%

In addition to the standard benchmark instances, we have generated 320 cost and coverage correlated SCP instances with sizes ranging from 1560 variables and 40 constraints to 9900 variables to 100 constraints to test the performance of the algorithm. Items are the points which are located in two dimensional space. Sets are defined as the combination of some points in the plane such that one point is the center and the others are located around the center. The cost of a set is determined by the farthest point from the center, and all the points in the set are covered when that set is selected. That is, the cost of coverage c_j is given by $\max_{k \in S_j}\{d_{ik}^{\alpha}\}$ where α is a scalar, i is the center point and d_{ik} is the distance between items i and k. These instances are solved for different α values and for two different distance metrics, Euclidean (E) and Manhattan (M), to see the effects of these parameters on the solution quality. Table 3.2 gives the summary of the results. The results show that the algorithm gives better results for this type of problems compared to Beasley's instances, especially for large α values. In addition, the percentage reduction in PrimalInt for (E) and (M) type instances is higher than standard benchmark instances. When we compare the performance for Euclidean and Manhattan distance metrics, it can be seen that algorithm performs slightly better for Euclidean type instances.

Table 3.2: Average percentage gaps for instances having Euclidean and Manhattan distances

E/M	α	LPOPT-Dual	IPOPT-LPOPT	PrimalInt-IPOPT	PostProc-IPOPT
E	2	2.84%	0.03%	104.26 %	14.55 %
E	3	1.17%	0.02%	65.82 %	9.70 %
M	2	3.85%	0.11%	95.94 %	16.14 %
M	3	1.49%	0.02%	61.29 %	9.81 %

After obtaining these results, we have investigated the behavior of the algorithm to figure out the underlying reasons behind the relative success of the algorithm for the instances having Euclidean and Manhattan distances. The main characteristic of these problem instances is that the cost of coverage is assumed to be monotonically increasing with the distance. That is, it costs more to cover points within a greater coverage radius. Empirical analysis showed that IPOPT and LPOPT are close to each other and the items/points are prone to be clustered for this kind of problems. The proposed algorithm tries to cluster the points in such a way that items that are located farther are selected first. In addition, while the dual feasibility is maintained at each iteration, a set with the lowest cost is selected among the sets that cover the same items. Consequently, this behavior of the algorithm reduces the coverage cost.

3.2 Methods for Improving The Main Algorithm

In this section, we describe the methods to increase the solution quality of the algorithm and compare the resulting variants with the original algorithm.

3.2.1 Changing Primal Variable Selection Rules

Our first attempts are intended to decrease the primal objective function value to decrease the gap between this value and the optimal IP objective function value. The main algorithm always selects the set having the smallest index as a tie breaker. For example, if there are several candidate primal variables each of which attains the minimum remaining slack value ($\bar{c_j}$) in the corresponding dual constraint, then the set with the smallest index is selected. Changing the set selection rule may change the primal objective function value. Thus, the following alternate rules were tested on standard benchmark instances.

- **Random Selection:** With this rule, when a tie occurs (more than one dual constraint may be tight in one iteration), the set is selected randomly among all alternatives.
- **Maximum Element Coverage:** With this rule, the set that could cover more within the set of currently uncovered items is selected. Ties are broken by the smallest index.
- **Minimum Additional Cost:** We know by Property 2 that $\sum_{j=1}^{n} c_j x_j = \sum_{j \in \mathcal{C}} \sum_{i=1}^{m} a_{ij} y_i$. Therefore, at each iteration the primal objective function value increases by the

sum of the dual variables in the selected set. With this rule, for each of the alternate sets, the sum of the dual variables is calculated and then, the set having the smallest value is selected.

The effect of these rules on the primal objective function value is summarized in Table 3.3. The figures in the table show the average percentage changes after applying these selection rules to standard non-unicost benchmark instances (compare against the second row of Table 3.1).

It is observed that only the random rule leads to a decrease on average, and this decrease is just 0.33%. We have observed that this minor improvement is obtained because of the low frequency of tie occurrences at each iteration. The average number of primal variables that tie at one iteration is only 1.32 per instance.

Table 3.3: Percentage changes after applying the selection rules for standard non-unicost benchmark instances

	Set Selection Rule		
Statistics	Random Selection	Max. Element Coverage	Min. Additional Cost
Average	-0.33%	2.24%	0.00%

3.2.2 Generating Dual Variable Sequence by a Greedy Approach

Our next attempt is to change the dual variable selection by incorporating the Greedy Approach discussed in Section 2.2.1 to our algorithm. The idea behind the greedy algorithm is to select the set that minimizes the cost per additional element covered at each iteration. At each iteration, $f(c_j, k_j) = c_j/k_j$ is calculated, where k_j is the number of uncovered items that could be covered by set j, and then the set which minimizes $f(c_j, k_j)$ is selected. In a similar way, the algorithm is modified so that at the first iteration, for each item i, the value $f(\sum_{i \in S_j} c_j, \sum_{i \in S_j} k_j)$ is calculated once (Algorithm 2 line 7), and then items are ordered according to $f(\sum_{i \in S_j} c_j, \sum_{i \in S_j} k_j)$ values in nondecreasing order (Algorithm 2 line 10). Algorithm after this modification is given as Algorithm 3. After applying this rule, we compare our results with Algorithm 2 and observe that the dual objective function value decreases by 7.85%, whereas, the primal objective function value increases by 2.83% on average.

Algorithm 3 Modified Algorithm (Greedy Approach)

```
1: Initialize:
2: $y_i = 0 \ \forall i \in \mathcal{U}$
3: $\mathcal{U}^* = \mathcal{U}$    // $\mathcal{U}^*$ is the set of currently uncovered items
4: $\bar{c}_j = c_j - \sum_{i=1}^{m} a_{ij} y_i = c_j \ \forall j \in \mathcal{S}$
5: $J = \emptyset$    // $J$ will contain the set of indices picked for the cover
6: $x_j = 0 \ \forall j \in \mathcal{S}$
7: for $i = 1$ to $m$ do
8:     calculate $f(\sum_{i \in S_j} c_j, \sum_{i \in S_j} k_j)$    // expected cost per additional item
9: end for
10: sort the indices $i$ in nondecreasing order of their respective $f(\sum_{i \in S_j} c_j, \sum_{i \in S_j} k_j)$ values,
    and put them into a list
11: while there are uncovered items do
12:     pick the first uncovered item in the list
13:     pick the smallest index $k = \arg\min_j \{\bar{c}_j | i \in S_j\}$
14:     $y_i = \bar{c}_k$
15:     for $j \in \{j | i \in S_j\}$ do
16:         $\bar{c_j} = c_j - \sum_{i=1}^{m} a_{ij} y_i$    // this will make $\bar{c}_k = 0$
17:     end for
18:     $x_k = 1$, $J = J \cup \{k\}$ and $\mathcal{U}^* = \mathcal{U}^* \setminus S_k$
19:     remove all items in set $S_k$ from the list
20: end while
21: return the subsets $S_j$ with $j \in J$
```

3.2.3 Changing Dual Variable Sequence Dynamically

Our next method is related to changing the dual variable selection order as in Section 3.2.2. Pseudocode of the algorithm using this method is given as Algorithm 4. Here, items or dual variables are sorted in nondecreasing order according to the number of sets in which they appear. Algorithm 2 is modified so that, rather than selecting the first uncovered item from the original list, items are selected from a candidate list (CL) based on a priority rule. First r uncovered elements in the original sequence are kept in a list (CL), where r is a parameter that is determined by the user. At each iteration, we try each uncovered element $i \in CL$, determine the associated dual variable y_i, and then select the dual variable i by $\{i = \arg\max_{i \in CL} \{y_i\}\}$. At each iteration, new uncovered items in the original sequence are added to CL such that the size of CL is always r as long as the number of uncovered items is greater than r. Table 3.4 shows the percentage changes in the primal (P Change) and the dual (D Change) objective function values between Algorithm 2 and Algorithm 4. Performance of the algorithm is tested on the non-unicost standard benchmark instances when the number of items in CL is 10, 20, and 40. The results show that, using this method, minor improvements may be obtained in both the dual and the primal objective function value. Moreover, percentage decrease in the primal objective function value changes with different r values. Table 3.4 shows that $r = 20$ seems to be a good balance for standard non-unicost benchmark instances with the number of items ranging from 200 and 400.

Table 3.4: Percentage changes in primal and dual objective function values with dynamic sequence for standard benchmark instances

r=10		r=20		r=40	
P Change	D Change	P Change	D Change	P Change	D Change
-2.96%	0.19%	-2.72%	1.35%	0.78%	-2.89%

Algorithm 4 Modified Algorithm (Dynamic Sequence)

```
1: Initialize:
2: $y_i = 0 \ \forall i \in \mathcal{U}$
3: $\mathcal{U}^* = \mathcal{U}$    // $\mathcal{U}^*$ is the set of currently uncovered items
4: $\bar{c}_j = c_j - \sum_{i=1}^m a_{ij} y_i = c_j \ \forall j \in \mathcal{S}$
5: $J = \emptyset$    // $J$ will contain the set of indices picked for the cover
6: $CL = \emptyset$
7: $x_j = 0 \ \forall j \in \mathcal{S}$
8: for $i = 1$ to $m$ do
9:     calculate $count_i$    // total number of sets that cover item $i$
10: end for
11: sort the indices $i$ in nondecreasing order of their respective $count_i$ values, and put
    them into a list
12: while there are uncovered items do
13:     pick the first $\min\{|L|, r - |CL|\}$ uncovered items in the list and put it into
        $CL$
14:     pick item $i = \arg\max_{i \in CL}\{y_i\}$
15:     pick the smallest index $k = \arg\min_j\{\bar{c}_j | i \in S_j\}$
16:     $y_i = \bar{c}_k$
17:     for $j \in \{j | i \in S_j\}$ do
18:         $\bar{c}_j = c_j - \sum_{i=1}^m a_{ij} y_i$    // this will make $\bar{c}_k = 0$
19:     end for
20:     $x_k = 1$, $J = J \cup \{k\}$ and $\mathcal{U}^* = \mathcal{U}^* \setminus S_k$
21:     remove all items in set $S_k$ and $CL$ from the list
22: end while
23: return the subsets $S_j$ with $j \in J$
```

3.2.4 Finding Complementary Primal Solution

The experiments performed so far indicate that the dual objective function value is closer to the optimal IP objective function value. Thus, we reckon that if we can find a primal solution whose objective function value is closer to the dual objective function value, then we may find an integer primal solution that is closer to the optimal.

Simplex Point of View

We first observe that at each iteration of our algorithm, a dual basic feasible solution can be obtained for

$$maximize \quad ye \tag{3.1}$$

$$\text{subject to}$$

$$yA + sI = c, \tag{3.2}$$

$$y, s \geq 0, \tag{3.3}$$

where A is an $n \times m$ 0-1 matrix, $e = (1, 1, \ldots, 1)$, I is the $n \times n$ identity matrix, and s is a row vector that represents the slack variables. This gives us the opportunity to obtain the complementary primal solution, which would have exactly the same solution as the dual solution. The proposed primal-dual algorithm follows the following simplex iterations for m steps or less.

- At the first iteration of the Simplex, B and A_B^{-1} are given where B is current set of basic variables and A_B^{-1} is the inverse of the current basis. At the first iteration of the main algorithm, $y_i = 0$, $\forall i \in \mathcal{U}$ (Algorithm 2 line 2). All y variables are non-basic and all slack variables are basic at iteration 0. Therefore, $A_B^{-1} = I$ and $B = \{s_1, \cdots, s_n\}$.

- The current basic feasible solution of Simplex is $\bar{x}_B = A_B^{-1}b$. The value of all non-basic variables are zero. On the other hand, main algorithm computes them as follows: If $i \in B$, then $y_i = \bar{c}_k$, otherwise $y_i = 0$ (Algorithm 2 line 14).

- Simplex selects the variable t within the set of nonbasic variables, where $\bar{c}_t < 0$, and $N = \{1, \cdots, n\} \setminus B$. The main algorithm selects the variable based on the dual selection sequence. Since, the objective function value increases or stays the same at each iteration, $\bar{c}_i \leq 0$.

- Simplex selects the leaving variable by applying minimum ratio test. However, algorithm uses a selection rule which maintains the feasibility in (Algorithm 2 line 13). Since $y_i = \bar{c}_k$ (Algorithm 2 line 14), $s_k = 0$, and k^{th} slack variable leaves the basis.

- Simplex follows these iterations until an optimal solution is obtained. However, algorithm iterates until $\sum_{j=1}^{n} a_{ij}x_j \geq 1$, $\forall i \in \mathcal{U}$. At each iteration, at least one of the item is covered, so the number of iteration can be at most m.

It is clear that algorithm and simplex iterations are equivalent. Therefore, we can conclude that algorithm gives a dual basic feasible solution at each iteration. The main idea at this point is to obtain the complementary primal solution by using the dual basic feasible solution. Clearly, unless we attain optimality for the LP relaxation, this complementary primal solution is infeasible. The source of the infeasibility is the violation of the non-negativity constraints for the variables x or some of the coverage constraints. If this primal infeasibility could be repaired and the integrality be imposed without increasing the primal objective function value considerably, then the solution of the algorithm might get closer to the IP optimal solution. The algorithm after the modification is given as Algorithm 5.

At each iteration, a dual variable i ($i \notin S_j$ for all $j \in \mathcal{C}$) enters the basis and y_i is set to a non-negative value. The slack variable corresponding to the set covering item i leaves the basis. Entering and leaving dual variables are determined as in the main algorithm (Algorithm 2). In the main algorithm, a dual variable i enters the basis, if $y_i \geq 0$ and $i \notin S_j$ for all $j \in \mathcal{C}$. The value of x_k corresponding to the k^{th} binding constraint covering item i is set to 1. However, in the modified algorithm, the value of x_k is the k^{th} element of the row vector $c_B B^{-1}$ where c_B is a row vector including the objective function coefficients of the dual basic variables (Algorithm 5, line 29). After finding the complementary solution, if the solution is primal feasible and integral, then the complementary primal solution is optimal. If some of the primal variables are negative or some of the coverage constraints are violated, then the infeasibility must be repaired. After some empirical analysis, the following rule was determined as a way of restoring primal feasibility. If $x_j > 0$, then the value of x_j is set to 1, otherwise it is set to 0 (Algorithm 5, lines 30-36). If there are some uncovered items left (Algorithm 5, line 39), additional sets are selected by using a greedy algorithm mentioned in Section 2.2.1. Using this variant of our algorithm, we have solved 45 benchmark instances and 160 randomly generated (Euclidean and Manhattan) instances. Computational results can be seen in Table 3.5. "PrimalInt-IPOPT (old)" is the IP gap when the original algorithm is applied to those instances and "PrimalInt-IPOPT (new)" is the IP gap when the new method is applied. It is observed that, the new method provides 5% decrease in the solution of the main algorithm.

After obtaining the first results, the effects of the dual variable selection are analyzed. We have used the method explained in Section 3.2.3 to the same 205 instances,

Algorithm 5 Modified Algorithm (Complementary Primal Solution)

```
1: Initialize:
2: $y_i = 0 \ \forall i \in \mathcal{U}$
3: $\mathcal{U}^* = \mathcal{U}$                          // $\mathcal{U}^*$ is the set of currently uncovered items
4: $A_B^{-1} = I_{n \times n}$
5: $c_B = (0, \cdots, 0)$
6: $\bar{c}_j = c_j - \sum_{i=1}^{m} a_{ij} y_i = c_j \ \forall j \in \mathcal{S}$
7: $J = \emptyset$                  // $J$ will contain the set of indices picked for the cover
8: $x_j = 0 \ \forall j \in \mathcal{S}$
9: for $i = 1$ to $m$ do
10:     calculate $count_i$                    // total number of sets that cover item $i$
11: end for
12: sort the indices $i$ in nondecreasing order of their respective $count_i$ values, and put
    them into a list
13: while there are uncovered items do
14:     pick the first uncovered item in the list
15:     pick the smallest index $k = \arg\min_j \{\bar{c}_j | i \in S_j\}$
16:     $y_i = \bar{c}_k$
17:     $c_B(k) = 1$
18:     update $A_B^{-1}$
19:     for $j \in \{j | i \in S_j\}$ do
20:         $\bar{c_j} = c_j - \sum_{i=1}^{m} a_{ij} y_i$                    // this will make $\bar{c}_k = 0$
21:     end for
22:     $x_k = 1,\ J = J \cup \{k\}$ and $\mathcal{U}^* = \mathcal{U}^* \setminus S_k$
23:     remove all items in set $S_k$ from the list
24: end while
25: calculate $x = c_b A_B^{-1}$
26: for $i = 1$ to $m$ do
27:     if $x_j > 0$ then
28:         $x_j = 1$
29:     else
30:         $x_j = 0$
31:     end if
32: end for
33: $\mathcal{U}^* = \mathcal{U}^* \setminus \left( \bigcup_{k=1,\, x_k=1}^{n} S_k \right)$
34: if $\mathcal{U}^* \neq \emptyset$ then
35:     Use greedy algorithm until $\mathcal{U}^* = \emptyset$
36: end if
37: return the subsets $S_j$ with $j \in J$
```

Table 3.5: Percentage changes in IP gap with complementary primal solution approach for benchmark and random instances

	PrimalInt-IPOPT (old)	PrimalInt-IPOPT (new)
Benchmark	17.82%	12.35%
Euclidean	11.52%	7.19 %
Manhattan	13.45%	7.78%
All	13.65%	8.55%

where r is taken as 10% of the number of items. This modification decreases the primal objective function value in some of the instances, but it increases the IP optimality gap on average. When this modification is applied to Algorithm 2 and Algorithm 5, PrimalInt-IPOPT values are calculated as 17.79% and 10.50% on average. Table 3.5 shows that this gap is 13.65% and 8.55% before the modification. In Section 3.2.3, we have concluded that this method improves the solution quality for fixed r (see Table 3.4) for standard benchmark instances. Then, we test the algorithm on both the standard benchmark and random instances with $r = m/10$ and observe that this method on this set of instances gives poor results.

After finding a complementary primal solution, we know that the source of primal infeasibility is the basic variables having negative values or not satisfying the coverage constraints. An alternate modification is checking the value of the original basic primal variables at each iteration, and then moving the dual variable to the end of the dual selection sequence whenever it causes at least one of the original primal basic variables to become negative. If all of the remaining dual variables in the list cause infeasibility, then dual variables are selected from the list as usual. After this modification, the solutions obtained by the modified algorithm have improved for some problems, but on average this modification has not performed well. When this modification is applied to Algorithm 2 and Algorithm 5, PrimalInt-IPOPT values are calculated as to 13.37% and 8.54%. Table 3.5 shows that this gap is 13.65% and 8.55% before the modification. This modification changes the dual variable selection order as well, and we know that changing the sequence also changes the solution. Therefore, we cannot conclusively state whether the lack of improvement is due to the modified dual variable sequence or over method of repairing infeasibility in the primal.

We have tried different methods to improve the performance of the algorithm and we have seen that finding the complementary solution method improves the performance of the algorithm in terms of solution quality. Since updating the dual basis at each iteration, increases the solution time, performance of the Algorithm 5 is worse than

Algorithm 2. We know that Algorithm 2 and Algorithm 5 gives the same set of original basic variables as a solution. However, all x_j values are set to 1 in the main algorithm and they can take any value in Algorithm 5. We have performed some analysis on the instances and solutions to develop a better infeasibility repairing method. However, the value or sign of the basic variables have not give insight to develop a better method. One of the reason behind Algorithm 5 is that some of the primal basic variables in the solution are dropped, and new solution is found by interchanging the columns. Thus, we can conjecture that we may obtain a similar improvement by applying a neighborhood search to the primal solution at the end of Algorithm 2.

3.3 Effects of Dual Sequence Selection

In this section, we analyze the effect of the dual variable selection sequence on the solution. First, a computational study is performed to see whether we can obtain zero dual-LP-OPT gap when all possible permutations of the dual variables are tried. We have generated 100 problems with 6 items and 50 sets. This yields 6!=720 different orders. These problems are solved by using Algorithm 5 and then by Algorithm 2. Only for 12 problems, we have not obtained a zero Dual-LPOPT gap. We make the following observations:

- When Dual-LPOPT gap is zero, then IPOPT is equal to LPOPT for all instances.
- For those problems, where IPOPT is equal to LPOPT, there is always a sequence that gives an optimal IP solution.
- Optimal integer solutions are obtained by using both methods in 98 out of 100 problems. One of the remaining two problems is solved optimally by the algorithm using the complementary primal method, and the other is solved optimally by the main algorithm.
- Small Dual-LPOPT gap does not necessarily imply that the algorithm would find a good integer solution. We have observed that in some instances, the algorithm finds good integer solutions, despite large Dual-LPOPT gap.

The most important conclusion regarding these results is that the dual variable selection sequence is a critical part of the algorithm and it is possible to decrease the value of the primal objective function by changing this sequence.

Clearly, the complexity of the algorithm increases when the number of items increases. Therefore, it is not practical to evaluate all possible permutations of dual variables. We can obtain $m(m-1)/2$ different sequences by carrying out a single pairwise interchange of the dual variables. In this case, the complexity of searching all pairwise interchanges increases polynomially with the problem size. To analyze the effects of the pairwise interchange, 100 instances with size 30 items and 300 sets are created randomly. The IP-LP gap for these problems is ranging from 0% to 15.5%. These instances are solved once for each sequence, by Algorithm 2 and Algorithm 5. In 97 out of 100 problems, there is a sequence such that both algorithm found an optimal solution, whereas in 2 out of the remaining three problems, there is a sequence such that either Algorithm 2 or Algorithm 5 finds the optimal solution. In only one problem, optimal solution could not be found by any of the algorithms and the IP optimality gap was 3.51% for both problems. These results emphasize the importance of the dual variable selection sequence. It seems possible to obtain near optimal solutions by a pairwise interchange instead of considering all possible sequences.

We have shown that changing dual sequence plays a critical role to find a good solution. Thus, through a well designed neighborhood search procedure on the dual sequence it is possible to improve the performance of the algorithm in terms of both time and solution quality. Since, the dual sequence partially changes through neighborhood search, we can find the new solution by modifying the dual solution at the previous iteration. This feature of the algorithm may improve the efficiency of the neighborhood search. Similarly, algorithm has a potential to solve large scale SCP with column generation method effectively. At each iteration, new columns and new rows are introduced to the primal and the dual problem respectively. With a new row in the dual problem, some of the $count_i$ values increase by one whereas some of them do not change. That is, new rows that are added to the problem result in minor changes in the dual sequence. With a well designed algorithm and data structures, it may be possible to find a new solution, without re-applying the algorithm from scratch.

3.4 Some Approaches to Improve Computational Speed

In addition to the attempts to increase the solution quality, we have also tried pre-processing and post-processing procedures to decrease the computation time of the algorithm. There are different methods in the literature for pre-processing. In this research, the method proposed by Beasley [4] is used. He introduces a computationally efficient method that is based on column domination. Any column j whose rows can be covered by other columns for a cost less than c_j is deleted from the problem. Formally, any column j for which $c_j > \sum_{i=1}^{m} d_i a_{ij}$ $j = 1, \cdots, n$ is deleted from the problem where $d_i = \min_{j \in \mathcal{S}} \{c_j | a_{ij} = 1\}$.

A post processing procedure is the elimination of the redundant columns in the solution. The common method in the literature is to order the sets in the cover in nonincreasing order of respective costs c_j, and eliminate the set whose elimination does not violate the coverage constraints. In this research, the same method used in the post-processing procedure. The pseudocode of the proposed heuristic after including the pre-processing and post-processing procedures is given in Algorithm 6.

Algorithm 6 Proposed Primal-Dual Heuristics

```
perform pre-processing procedure to the original problem
$\mathcal{S} = \mathcal{S} \setminus j$ if column $j$ is deleted from the problem
Initialize:
$y_i = 0 \; \forall i \in \mathcal{U}$
$\mathcal{U}^* = \mathcal{U}$                       // $\mathcal{U}^*$ is the set of currently uncovered items
$n = |\mathcal{S}|$
$\bar{c_j} = c_j - \sum_{i=1}^{m} a_{ij} y_i = c_j \; \forall j \in \mathcal{S}$
$J = \emptyset$                   // $J$ will contain the set of indices picked for the cover
$x_j = 0 \; \forall j \in \mathcal{S}$
for $i = 1$ to $m$ do
    calculate $count_i$              // total number of sets that cover item $i$
end for
sort the indices $i$ in nondecreasing order of their respective $count_i$
while there are uncovered elements do
    pick the first uncovered element in the list
    pick an index $k = \arg\min_{i \in S_j} \{\bar{c_j}\}$
    $y_i = \bar{c_k}$
    for $j = 1$ to $n$ do
        $\bar{c_j} = c_j - \sum_{i=1}^{m} a_{ij} y_i$          // this will make $\bar{c_k} = 0$
    end for
    $x_k = 1$, $J = J \cup \{k\}$ and $\mathcal{U}^* = \mathcal{U}^* \setminus S_k$
    remove all items in set $S_k$ from the list
end while
perform post-processing procedure to the original problem
return the subsets $S_j$ with $j \in J$
```

CHAPTER 4

COMPUTATIONAL STUDY

In this chapter, we evaluate the performance of primal-dual heuristic (PDH) against six algorithms on standard benchmark instances [6] and 320 randomly generated instances described in section 3.1. In this study, Intel(R) Celeron(R) 1.6 GHz computer is used to obtain all computational studies. All of the algorithms are implemented in C++ and optimal LP and IP solutions are solved by ILOG IBM CPLEX 12.1.

Algorithms are as follows:

- PR: LP rounding algorithm by Hochbaum [20] uses optimal solution of the primal LP model ($x_j^* > 0 \Rightarrow x_j \rightarrow 1$).
- PR2: LP rounding algorithm by Hochbaum [20] uses optimal solution of the primal LP model ($x_j^* > 1/f \Rightarrow x_j \rightarrow 1$).
- DR: LP rounding algorithm by Hochbaum [20] uses optimal solution of the dual LP model. ($\sum_{i=1}^{m} a_{ij} y_i^* = c_j \Rightarrow x_j \rightarrow 1$)
- GR: Randomized greedy algorithm by Lan et al. [22]. Columns are selected based on the following priority rules c_j/k_j, c_j/k_j^2, $c_j^{1/2}/k_j$, $c_j/k_j^{1/2}$ and rule is determined randomly at each iteration.
- BE: Primal-dual approximation algorithm (Algorithm 1). Items/dual variables are selected randomly among the currently uncovered items.
- PDH2: Primal-dual heuristic using complementary primal solution (Algorithm 5).

Table 4.1 provides a summary of comparison on the solution quality for all benchmark algorithms. Performance of the PDH is worse than the algorithms using the optimal solution of the LP model. However, there are some instances that are highlighted in the table in which the objective function values are the same or better than

PR2. In addition, PDH2 dominates PDH for all instances in terms of solution quality. Last 5 instances are the uni-cost SCP instances and GR dominates other algorithms.

Figures in this section are called performance profiles [13] which show the fraction of problems for which the algorithm is within a factor of the best CPU time, or the best solution. Figure 4.1 compares the performance of the algorithms on non-unicost benchmark instances in terms of solution quality. It is shown in the figure that algorithm PR is the best of all in 90% of the instances. Solutions found by PDH is at most 1.6 times worse than the best solution and in 90% of the instances, performance of the algorithm is at most 1.2 times worse than the best solution. PDH2 outperforms PDH for all instances, and in approximately 5% of the instances, PDH2 dominates all algorithms.

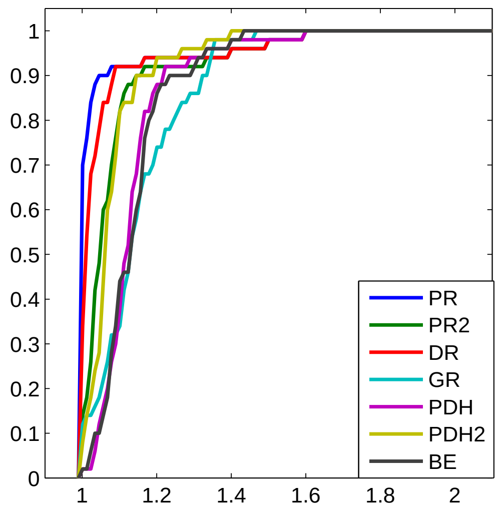


Figure 4.1: Performance profile for the algorithms on standard benchmark instances in terms of solution quality

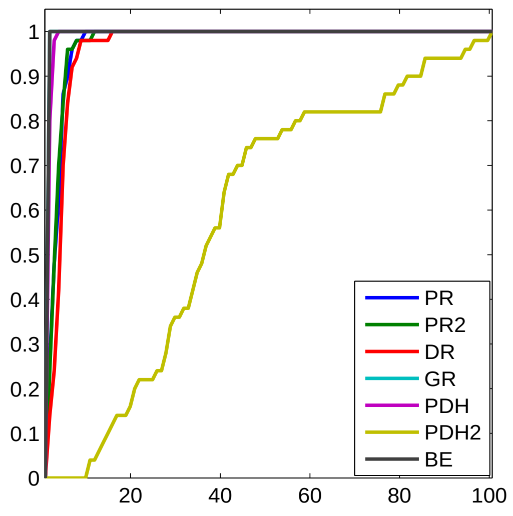
Table 4.2 reports the computation time of the algorithms on benchmark instances. The same pre-processing and post-processing procedures are applied to each instance and computation times are recorded. Figure 4.2 compares the performance of benchmark algorithms. Computation time of PDH2 is higher than other algorithms. Therefore, two sub-figures are used to compare the relative performance of each algorithm in terms of computation time. It shows that since GR and PDH do not solve LP optimally, they outperform the others. In approximately 90% of the instances, algorithms solving LP optimally are at most 6 times slower than the fastest algorithm. Since finding a complementary primal solution takes time, PDH2 is the worst among all in terms of computation time. In approximately 80% of the instances, computation time

Table 4.1: Summarized results for the solution quality for benchmark instances

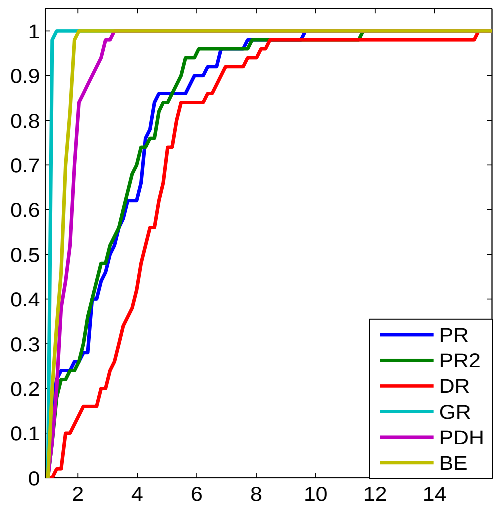
Instance	Row	Column	IP OPT	PR	PR2	DR	GR	PDH	PDH2	BE
1	200	1000	429	429	449	448	459	462	450	472
2	200	1000	512	512	559	556	595	592	548	595
3	200	1000	516	516	539	528	569	609	591	598
4	200	1000	494	495	533	501	567	557	546	583
5	200	1000	512	512	527	517	584	567	540	544
6	200	1000	560	568	608	572	731	649	623	659
7	200	1000	430	430	444	436	523	447	439	463
8	200	1000	492	504	509	502	659	568	535	532
9	200	1000	641	688	697	688	858	706	689	735
10	200	1000	514	516	572	518	596	533	531	603
11	200	2000	253	255	279	260	295	279	270	253
12	200	2000	302	330	326	342	363	375	352	354
13	200	2000	226	226	245	243	256	242	228	242
14	200	2000	242	252	252	252	271	269	264	285
15	200	2000	211	211	227	217	223	238	241	245
16	200	2000	213	213	239	217	252	243	216	241
17	200	2000	293	308	323	323	331	337	328	333
18	200	2000	288	298	312	301	326	321	297	339
19	200	2000	279	279	317	292	345	308	306	362
20	200	2000	265	265	274	287	359	289	278	302
21	200	1000	138	154	152	152	175	172	174	164
22	200	1000	146	156	160	156	187	180	166	176
23	200	1000	145	154	160	154	184	178	168	190
24	200	1000	131	138	140	136	145	141	136	136
25	200	1000	161	182	192	182	176	215	192	186
26	300	3000	253	260	263	263	271	308	275	298
27	300	3000	252	268	278	268	361	285	275	277
28	300	3000	232	243	251	245	276	255	245	265
29	300	3000	234	239	258	240	268	266	259	276
30	300	3000	236	243	245	243	297	270	259	264
31	300	3000	69	77	83	80	87	80	81	78
32	300	3000	76	84	88	84	84	102	92	90
33	300	3000	80	87	85	86	89	92	89	97
34	300	3000	79	88	90	88	96	85	88	79
35	300	3000	72	74	81	74	99	95	93	72
36	400	4000	227	239	234	238	314	245	240	227
37	400	4000	219	233	240	235	257	264	246	240
38	400	4000	243	266	269	265	293	283	282	243
39	400	4000	219	235	243	235	264	265	246	265
40	400	4000	215	223	228	224	325	244	237	215
41	400	4000	60	63	63	64	83	70	69	76
42	400	4000	66	72	76	72	74	80	77	86
43	400	4000	72	79	83	79	87	92	86	80
44	400	4000	62	71	66	71	83	74	69	71
45	400	4000	61	65	72	65	68	78	78	77
46	50	500	5	9	8	9	6	8	8	8
47	50	500	5	7	9	7	6	7	6	7
48	50	500	5	7	8	7	5	8	7	6
49	50	500	5	8	7	8	5	7	6	7
50	50	500	5	6	7	6	6	7	6	7

of PDH2 is at most 80 times worse than the fastest algorithm.

Figure 4.2: Performance profile for the algorithms on standard benchmark instances in terms of computation time



(a) PDH2 is included



(b) PDH2 is excluded

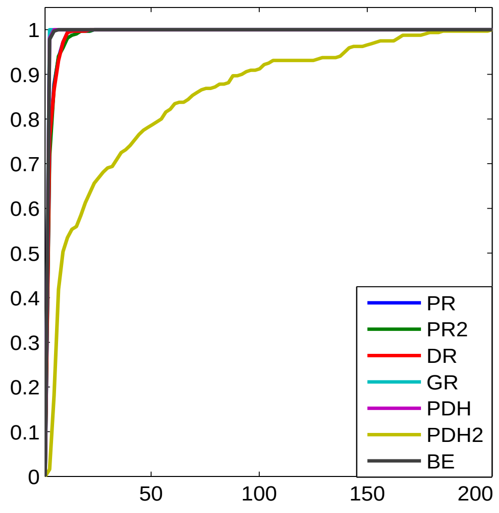
We also analyze whether the running time of the CPLEX decreases or not when the solution of the algorithm is given as an initial integer solution. Column CPLEX(i.bas.) and CPLEX denote the running time of the CPLEX with/without any initial solution, respectively. There are some instances in which solution time decreases with if an initial solution is given. These are highlighted in the table. However, the initial integer solution does not increase the efficiency of CPLEX on average.

Figures 4.3 and 4.4 give the performance of the algorithm on the randomly generated instances in terms of solution quality and computation time. Figure 4.4 shows that relative performance of PDH and PDH2 are better when compared to standard benchmark instances. Conversely, GR is the worst among all algorithms on randomly generated instances in terms of solution quality. Figure 4.3 shows that relative performances of the algorithms are almost the same except PDH2. Although, there are some instances which the computation time is approximately 180 times slower than the fastest algorithm, in 80% of the instances, it is 50 times slower. This number is 80 for the standard benchmark instances.

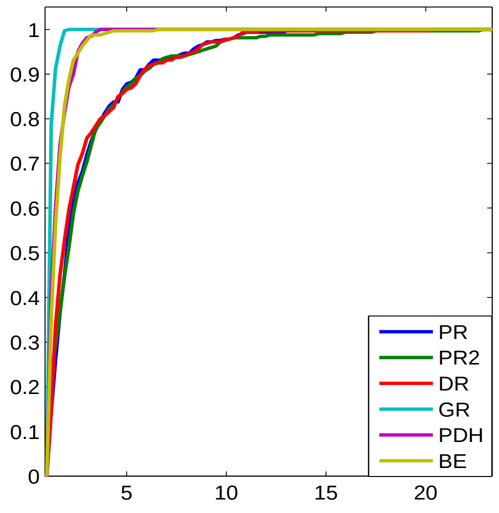
Table 4.3 compares performances for the algorithms on all instances in terms of solution quality. Instances are grouped based on their types benchmark (B) or random (R) and their sizes. It is seen that the performance of all algorithms except GR is relatively worse for B6 group which includes unicost instances. The solution quality of the PDH is clearly better for random instances when compared to standard benchmark instances. The columns PDH and BE show that the dual variable selection sequence of Algorithm 2 increases the performance for only random instances. Using complementary primal solution by PDH2 decreases IP gap.

Table 4.4 compares performances for the algorithms on all instances in terms of computation time. Computation time of PDH is less than the heuristics solving LP optimally. Since the most of the columns are deleted through pre-processing procedures for highly cost and coverage correlated random instances, the size of the instances decreases considerably. Therefore, computation times are very close for all algorithms except PDH2. Although solution quality of PDH2 is better than PDH, computation time is the highest among all algorithms.

Figure 4.3: Performance profile for the algorithms on randomly generated instances in terms of computation time



(a) PDH2 is included



(b) PDH2 is excluded

Table 4.2: Comparisons of computation time for benchmark instances(in seconds)

Instance	CPLEX	CPLEX(i.bas.)	PR	PR2	DR	GR	PDH	PDH2	BE
1	0.59	0.69	0.54	0.48	0.593	0.1	0.09	0.874	0.093
2	0.4	0.43	0.18	0.19	0.172	0.04	0.06	1.779	0.047
3	0.34	0.45	0.17	0.17	0.156	0.05	0.13	1.576	0.047
4	0.34	0.53	0.39	0.24	0.312	0.04	0.08	1.419	0.063
5	0.52	0.56	0.17	0.17	0.219	0.05	0.07	1.497	0.078
6	0.49	0.75	0.19	0.19	0.203	0.05	0.06	1.638	0.078
7	0.3	0.34	0.16	0.18	0.14	0.05	0.07	1.107	0.063
8	0.59	0.72	0.18	0.21	0.203	0.04	0.11	1.669	0.063
9	0.55	0.56	0.19	0.5	0.203	0.04	0.1	1.966	0.062
10	0.29	0.44	0.13	0.14	0.203	0.04	0.06	1.482	0.062
11	0.35	0.77	0.24	0.22	0.219	0.04	0.05	1.638	0.047
12	0.72	0.62	0.32	0.3	0.39	0.06	0.07	2.277	0.063
13	0.49	0.5	0.34	0.21	0.327	0.05	0.07	1.451	0.063
14	0.64	0.61	0.34	0.25	0.421	0.05	0.07	1.685	0.062
15	0.33	0.36	0.21	0.23	0.25	0.05	0.08	1.466	0.062
16	0.3	0.44	0.38	0.28	0.218	0.05	0.06	1.357	0.062
17	0.55	0.99	0.24	0.23	0.234	0.06	0.11	1.653	0.063
18	0.53	0.63	0.26	0.44	0.265	0.06	0.09	2.013	0.078
19	0.39	0.4	0.22	0.23	0.405	0.05	0.08	1.716	0.078
20	0.52	0.58	0.22	0.23	0.265	0.05	0.07	1.997	0.078
21	0.76	0.71	0.14	0.17	0.25	0.13	0.15	1.716	0.14
22	0.83	0.81	0.16	0.16	0.375	0.14	0.18	2.153	0.172
23	0.65	0.74	0.16	0.16	0.203	0.14	0.17	1.685	0.156
24	0.43	0.47	0.22	0.15	0.172	0.12	0.2	1.154	0.156
25	0.68	0.96	0.17	0.19	0.187	0.15	0.3	2.278	0.172
26	1.16	1.31	0.29	0.22	0.25	0.07	0.12	5.647	0.125
27	1.01	0.86	0.19	0.34	0.608	0.15	0.16	6.13	0.156
28	1.63	0.85	0.65	0.4	0.952	0.2	0.19	5.414	0.25
29	1.23	0.82	0.36	0.33	0.608	0.15	0.19	6.91	0.156
30	1.79	0.61	0.4	0.34	0.499	0.19	0.17	6.209	0.234
31	1.89	1.97	0.24	0.65	0.437	0.25	0.3	6.458	0.358
32	3.61	3.6	0.3	0.31	0.546	0.14	0.29	6.162	0.265
33	2.03	1.94	0.42	0.34	0.608	0.15	0.47	7.972	0.266
34	5.51	5.31	0.4	0.38	0.452	0.15	0.28	6.583	0.25
35	2.25	2.19	0.36	0.35	2.262	0.15	0.26	5.85	0.265
36	1.35	1.02	0.41	0.42	0.733	0.12	0.21	9.875	0.172
37	1.54	1.46	0.39	0.46	0.593	0.12	0.23	11.357	0.187
38	4.02	3.89	0.39	0.43	0.764	0.16	0.3	13.572	0.187
39	2.82	2.55	0.49	0.42	0.624	0.12	0.23	11.576	0.187
40	1.42	1.3	0.33	0.31	0.624	0.12	0.23	11.122	0.203
41	3.47	3.19	0.57	0.38	0.858	0.23	0.67	18.969	0.436
42	10	9.79	0.33	0.32	0.671	0.34	0.51	24.494	0.562
43	7.43	8.04	1.03	1.12	1.279	0.32	0.81	24.615	0.546
44	15.42	14.7	0.75	0.78	1.482	0.32	0.62	17.807	0.546
45	2.51	3.2	0.31	0.87	1.42	0.29	0.53	16.823	0.546
46	0.97	0.85	0.11	0.11	0.124	0.08	0.15	1.795	0.156
47	1.18	0.93	0.11	0.1	0.156	0.11	0.3	2.005	0.156
48	1.3	1.08	0.1	0.1	0.359	0.11	0.18	1.645	0.156
49	1.15	1.17	0.38	0.11	0.296	0.09	0.19	1.823	0.141
50	1.09	1.32	0.12	0.12	0.249	0.08	0.16	2.1	0.156

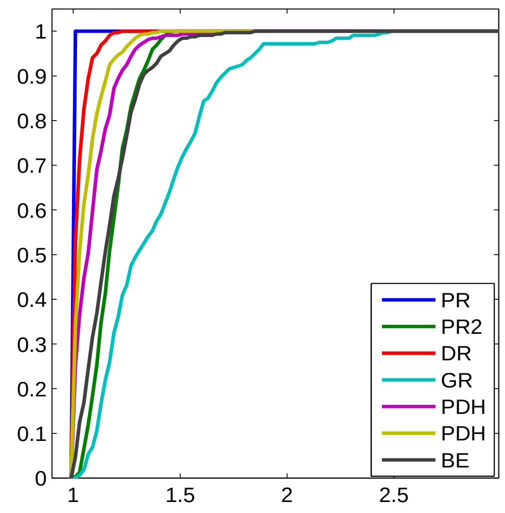


Figure 4.4: Performance profile for the algorithms on randomly generated instances in terms of solution quality

Table 4.3: Comparisons of solution quality and computation time for all instances

Instance Group		Average IP gap						
Instance	Size	PR	PR2	DR	GR	PDH	PDH2	BE
B.1	200x1000	1.18%	6.44%	3.14%	19.84%	11.40%	7.53%	13.19%
B.2	200x2000	2.28%	8.61%	6.01%	17.15%	12.55%	7.85%	17.93%
B.3	200x1000	8.61%	11.24%	8.01%	20.29%	22.37%	13.16%	17.73%
B.4	300x3000	6.40%	10.53%	6.96%	21.77%	17.75%	13.41%	19.01%
B.5	400x4000	7.71%	10.03%	7.93%	26.50%	19.23%	14.69%	18.10%
B.6	50x500	48.00%	56.00%	48.00%	12.00%	48.00%	32%	40%
R.1	40x1560	0.00%	17.12%	1.68%	32.25%	7.94%	4.15%	14.79%
R.2	60x3540	0.12%	17.52%	2.73%	39.38%	9.90%	6.87%	15.97%
R.3	80x6320	0.03%	18.18%	3.08%	40.58%	8.21%	6.28%	19.46%
R.4	100x9900	0.22%	19.53%	2.76%	38.69%	10.36%	6.22%	18.74%

Table 4.4: Comparisons of solution quality and computation time for all instances

Instance Group		Average running time in seconds								
Instance	Size	PR	PR2	DR	GR	PDH	PDH2	BE	CPLEX	CPLEX(i.bas.)
B.1	200x1000	0.23	0.25	0.24	0.05	0.08	1.5	0.07	0.44	0.55
B.2	200x2000	0.28	0.26	0.30	0.05	0.07	1.73	0.07	0.48	0.59
B.3	200x1000	0.17	0.17	0.24	0.14	0.20	1.82	0.16	0.67	0.74
B.4	300x3000	0.36	0.36	0.72	0.16	0.24	6.33	0.23	2.21	1.95
B.5	400x4000	0.50	0.55	0.90	0.21	0.43	16.02	0.36	5.00	4.91
B.6	50x500	0.16	0.11	0.24	0.09	0.19	1.87	0.15	1.14	1.07
R.1	40x1560	0.21	0.20	0.19	0.06	0.08	0.73	0.07	0.33	0.34
R.2	60x3540	0.26	0.26	0.28	0.13	0.22	3.9	0.22	0.36	0.44
R.3	80x6320	0.37	0.35	0.36	0.29	0.41	15.04	0.42	0.36	0.56
R.4	100x9900	0.55	0.54	0.53	0.40	0.55	25.26	0.55	0.49	0.83

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this study, we develop a heuristic that uses a primal-dual approach for solving SCP. The heuristic finds a feasible solution to the dual of the LP relaxation of SCP. Then, this dual feasible solution is used to obtain an integer solution for the SCP problem. The proposed heuristics shows several interesting properties: it is simple, quite fast, easy to implement, and robust.

We have implemented seven different heuristics. The results are compared on standard benchmark, Euclidean, and Manhattan type instances. The results exhibit that the proposed heuristic is competitive in terms of the computation time and it is possible to improve the solution quality by using the complementary primal solution. Also, we discuss that the solution quality has a potential to be improved through a well designed neighborhood search algorithm.

Our computational study on the Euclidean and Manhattan type instances shows that our primal-dual heuristic performs quite well. Based on our empirical evidence, we conjecture that for Euclidean problems, we may find an approximation bound on the worst case performance of the proposed primal-dual algorithm and expect that this bound is lower than fz. In this section, we give a preliminary analysis about this bound.

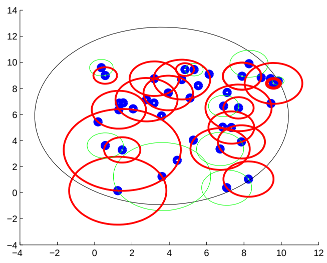
Let $e(\mathcal{Q})$ be the eccentricity of any set of points $\mathcal{Q} \subseteq \mathcal{U}$ as $e(\mathcal{Q}) = \min_{x \in \mathcal{Q}} \max_{y \in \mathcal{Q}} \; dist(x, y)$, where $dist(x, y)$ is the Euclidean distance between any two points x and y. Then, any solution which is given by the proposed heuristic always satisfies $c_j \leq e(\mathcal{U})^2$ given that $j \in \mathcal{C}$, where $e(\mathcal{U})$ is the radius of the smallest circle covering all items and this circle represents a constraint that covers all items with a cost of $e(\mathcal{U})^2$. We know that the algorithm always guarantees dual feasibility at each iteration. Therefore, $\bar{c_j}$ value has to be greater than 0 for the set j, if $c_j > e(\mathcal{U})^2$.

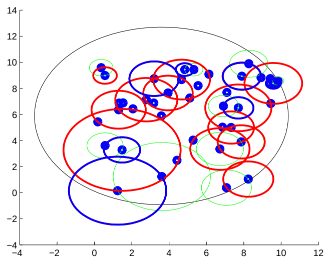
Figures 5.1 are given below to give an idea how some bad instances look like geometrically. We can reduce the value of the objective function by eliminating subsets in the solution. Figures 5.1 shows the solutions of the algorithm before and after post-processing procedure respectively where ρ denotes the ratio of primal objective function value over IP-OPT. In these figures, the red circles represent the constraints selected by the algorithm, the blue circles represent the sets that are eliminated by the post processing procedure and finally, the green circles represent the tight constraints that are not selected by the algorithm.

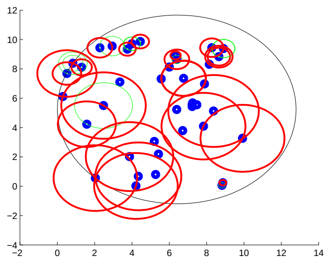
When there are nested circles in the solution, the value of the primal objective function is high compared to the optimal solution However, post-processing procedure eliminates the subsets and reduces the value of the objective function for those kind of instances. In addition, it is observed that when the number constraints that are tight in the solution is high, then the value of the objective function is high. The reason is that it is possible to increase the value of the primal objective function without increasing the sum of the dual variables. If the items are located in such a way that the distance among the adjacent items is equal, then for those instances the performance of the algorithm is low. We believe that those kind of instances are the worst case instances and we can find the approximation ratio by analyzing those instances. You see two different geometry for the locations of the items in the plane Figure 5.2. For the first geometry, the approximation ratio of the instances is 1, and 1.7143 when the number of items is equal to 7, and 31. For the second geometry, you can see the approximation ratio of two instances with m=25, 36 and 49. These figures show that, if the locations of the items are on the grid points, then the approximation ratio is at most 2.375. It can be seen that, it is not possible to decrease the objective function through post-processing procedure in those instances. We believe that these instances are the worst case instances, but then a new question arises: What is an upper bound on the approximation ratio?

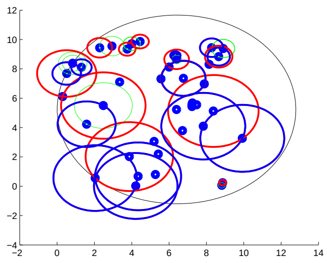
We are planning to complete this preliminary analysis by a theoretical proof in the near future. In addition, as a future work, we can improve the performance of the heuristic by developing a well designed neighborhood search algorithm. Finally, we can modify the algorithm such that when a new dual sequence is given, it can modify a solution without resolving the problem from the beginning.

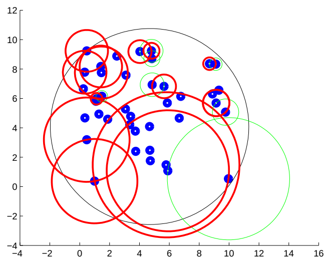
Figure 5.1: Solution before and after post processing

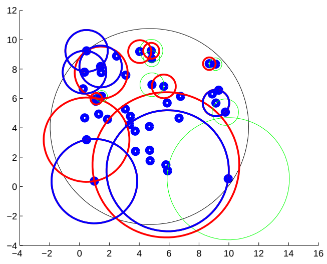

(a) Instance 10 $\rho = 2.00$

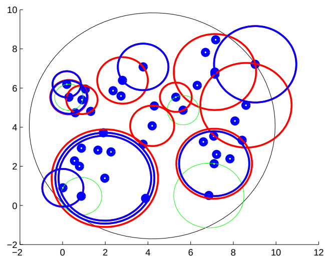

(b) Instance 10 $\rho = 1.48$


(c) Instance 19 $\rho = 2.80$


(d) Instance 19 $\rho = 1.30$


(e) Instance 3 $\rho = 2.2102$


(f) Instance 3 $\rho = 1.2035$


(g) Instance 20 $\rho = 2.48$

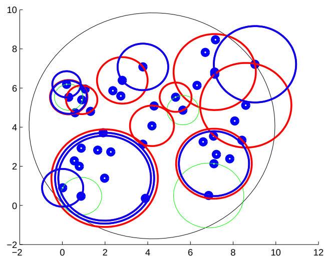
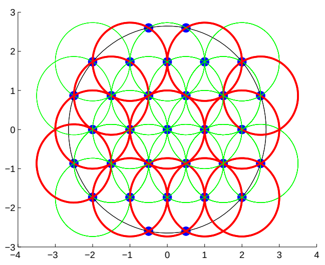
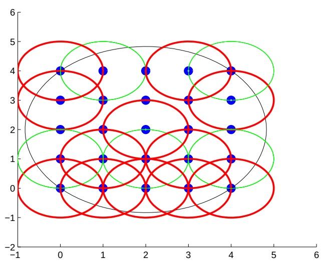

(h) Instance 20 $\rho = 1.30$

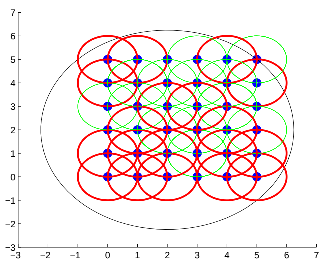
Figure 5.2: Set of possible worst case instances



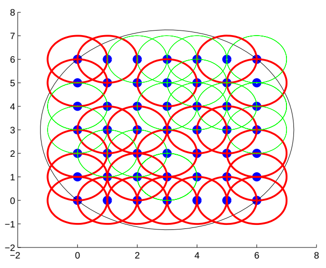
(a) $\rho = 1.71$



(b) $\rho = 1.71$



(c) $\rho = 2.37$



(d) $\rho = 1.92$

Bibliography

[1] Aickelin U., An indirect genetic algorithm for set covering problems, Journal of the Operational Research, 53 (10), 1118-1126, 2002.

[2] Balas E., Carrera M.C., A dynamic subgradient-based branch-and-bound procedure for set covering, Operations Research, 44(6), 1996.

[3] Bar-Yehuda R., Even S., A linear-time approximation algortihm for the weighted vertex cover problem, Journal of Algorithms, 2, 198-203, 1981.

[4] Beasley J.E., An algorithm for set covering problem, European Journal of Operational Research, 31, 85-93, 1987.

[5] Beasley J. E., Chu P. C., A genetic algorithm for the set covering problem, European Journal of Operational Research, 94, 392-404, 1996.

[6] Beasley J. E., A Lagrangian heuristic for set covering problems, Naval Research Logistics, 37,151-164.

[7] Beasley J.E, Jornsten K., Enhancing an algorithm for set covering problems, European Journal of Operational Research, 58, 293-300, 1992.

[8] Bertsimas D., Vohra R., Rounding algorithms for covering problems, Mathematical Programming, 80, 63-89, 1998.

[9] Brusco M. J., Jacobs L. W., Thompson G. M., A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set-covering problems, Annals of Operations Research, 86, 611-627, 1999.

[10] Caprara A., Fischetti M., Toth P., A heuristic method for the set covering problem, Operations Research, 47, 5, 730-743.

[11] Caprara A., Toth, P., Fischetti, M., Algorithms for the set covering problem, Annals of Operations Research 98, 353-371, 2000.

[12] Ceria S., Nobili, P., Sassano A., A Lagrangian-based heuristic for large-scale set covering problems, Mathematical Programmimg, 81, 215-228, 1998.

[13] Dolan E. D., More J. J., Benchmarking optimization software with performance profiles, Mathematical Programming, 91:201213, 2002.

[14] Fisher M.L, Kedia P., Optimal solution of set covering/partitioning problems using dual heuristics, Management Science, 36(6), 1990.

[15] Garey, M.R., Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[16] Gomes F. C., Meneses C. N., Pardalos P. M., Viana G. V. R., Experimental analysis of approximation algorithms for the vertex cover and set covering problems, Computers and Operations Research, 33, 3520-3534, 2006.

[17] Grossman T., Wool A., Computational experience with aprroximation algorithms for the set covering problem, European Journal of Operational Research, 101, 81-92, 1997.

[18] Hall N. G., Vohra R. V., Pareto optimality and a class of set covering heuristics, Annals of Operations Research, 43, 279-284, 1993.

[19] Haouari M., Chaouachi J. S., A probabilistic greedy search algortihm for combinatorial optimization with application to the set covering problem, Journal of the Operational Research Society, 53, 792-799, 2002.

[20] Hochbaum D.S., Approximation algorithms for the set covering and vertex cover problems, SIAM Journal on Computing, 11, 555-556, 1982.

[21] Jacobs L. W., Brusco M. J, A local search heuristic for large set-covering problems, Naval Research Logistics, 42, 1129-1140.

[22] Lan G., DePuy G.W., Whitehouse G.E, An effective and simple heuristic for the set covering problem, European Journal of Operational Research, 176, 1387-1403, 2007.

[23] Lorena LAN, Lopes L. S., Genetic algorithms applied to computationally difficult set covering problems, Journal of Operational Research Society, 48, 440-445, 1997.

[24] Melkonian V., New primal-dual algorithms for Steiner tree problems, Computers and Operations Research, 34, 2147-2167, 2007.

[25] Peleg D., Schechtman G, Wool A, Approximating bounded 0-1 integer linear programs, Proceedings 2nd Israel Symposium, Theory of Computing Systems, Netenya, Israel, 69-77, 1993.

[26] Williamson D.P., The primal-dual method for approximation algorithms, Mathematical Programming, 91, 447-478, 2002.

[27] Vasko F.J , Wilson G. R., An efficient heuristic for large set covering problems, Naval Research Logistics Quarterly, 31, 163-171, 1984.

[28] Vazirani V. V., Primal-dual schema based approximation algorithms, Theoretical Aspects of Computer Science, LNCS 2292, 198-207, 2002.