LARGE-SCALE OPTIMIZATION FOR DATA PLACEMENT PROBLEM

LAZIMA ANSARI
Bachelor of Science, Military Institute of Science and Technology, 2010

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

LARGE-SCALE OPTIMIZATION FOR DATA PLACEMENT PROBLEM

LAZIMA ANSARI

Date of Defence: August 17, 2017

Dr. Daya Gaur Supervisor	Professor	Ph.D.
Dr. Shahadat Hossain Committee Member	Professor	Ph.D.
Dr. Robert Benkoczi Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

I dedicate this thesis to my loving **parents**. I cannot express how lucky I am to have parents who love endlessly. Thank you so much for believing in me.

Abstract

Large-scale optimization of combinatorial problems is one of the most challenging areas. These problems are characterized by large sets of data (variables and constraints). In this thesis, we study large-scale optimization of the data placement problem with zero storage cost. The goal in the data placement problem is to find the placement of data objects in a set of fixed capacity caches in a network to optimize the latency of access. Data placement problem arises naturally in the design of content distribution networks. We report on an empirical study of the upper bound and the lower bound of this problem for large sized instances. We also study a semi-Lagrangean relaxation of a closely related k-median problem. In this thesis, we study the theory and practice of approximation algorithm for the data placement problem and the k-median problem.

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Daya Gaur for his guidance and support throughout the whole learning process. I would also like to thank my supervisory committee members Dr. Robert Benkoczi and Dr. Shahadat Hossain for their precious advice and inspiration.

I am thankful to all the members of optimization research group of the University of Lethbridge for their support. I am grateful to Nabi and Umair for their help. I also want to thank Anamay Sarkar with whom I worked on some important parts of my thesis.

A very special thanks goes to my husband and best friend Imtiaz, who has been a constant source of support and encouragement during all the challenges of my life. This accomplishment would not have been possible without him. Thanks for making me realize that the dreams can actually come true. Thanks for everything.

I would also like to thank my family for the support they provided me through my entire life. I must acknowledge my parents, sister, parents-in-law. Without their unconditional love, sacrifices, and prayers, I would not have finished this thesis.

I want to thank my friends Fatema, Tasnuba and Jeeshan who made my life easier in Canada.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Facility location problems have occupied a major place in Operations Research since the early 1960s. They model situations such as the placements of warehouses [35], factories, fire stations, hospitals and so on. Every location problem consists of four basic components [13].

- A set of *locations* where the facilities are placed. For each such location, we are given a cost of opening (or storing) the facilities.
- A set of *demand points* or *clients* who need certain services from the facilities and have to be assigned to a facility such that their requirements are fulfilled.
- A list of *requirements* to be met by the facilities and assignment of clients to facilities.
- A *cost function* associated with the assignment of the demand points to the facilities.

A typical objective is to select a set of facilities to open in order to optimize the given function. Various types of facility location problems can be obtained using the above mentioned features. The location problem that we study in this thesis is the data placement problem [3]. Data placement problem have been studied extensively in the areas of database management [24] and cooperative caching in networks [18, 4, 33]. This problem also arises naturally in the modeling and operation of Content Distribution Networks [10, 9, 50, 19]. In such a network, the computer systems need to optimize the distribution of Internet packets to the users by replicating and caching the data at multiple locations in the network. This reduces the load on the server and also helps to eliminate network congestion. Due to the

enormous growth of applications such as Netflix, the data placement problem has received considerable attention. Let us consider a distributed network, where some nodes serve as a cache (facility) to store the data objects (services installed in a facility), and some nodes (clients) need to access the data objects. The goal is to find an assignment of clients and objects to caches such that the total object storage cost and client access cost is minimized. The optimal placement of data objects to caches and assignment of users to cache is an NP-complete problem. We formally define the data placement problem in Chapter 4. We relate the data placement problem with the red-blue median problem [27], and the k-median problem [15].

In this thesis, we study a semi-Lagrangean relaxation for the k-median problem. We compare this semi-Lagrangean lower bound with the optimal value for large test instances [7] of k-median problem. This study was motivated by Beasley's work on computing an efficient lower bound to the k-median problem [15, 5]. We compute an upper bound and a lower bound for the data placement problem where the cost of storing objects in caches is negligible. For computing an upper bound, we use a local search approach, and for computing the lower bound, we use a Lagrangean relaxation. We perform an empirical study of the duality gap for large instances of the data placement problem.

1.1 Contributions

To the best of our knowledge, no previous empirical study of approximation algorithms for large sized instances of data placement problem has been carried out. The following are some of our specific contributions in this thesis:

- We formulate a semi-Lagrangean relaxation for the k-median problem. We compute a lower bound and report on the duality gap for a set of test instances [7].
- We develop two Lagrangean relaxations for the data placement problem by selecting a different set of constraints to relax.

- We study a local search based method to compute an upper bound and compare it to the two Lagrangean lower bounds.
- We generate large sized test instances for the data placement problem and report on an empirical study on those large instances of the problem.

Some of the results in this thesis were presented as a poster paper at the The 19th Conference on Integer Programming and Combinatorial Optimization (IPCO) held in University of Waterloo, Ontario, June 26-28, 2017.

1.2 Organization of the thesis

Including this chapter, there are four more chapters in this thesis. We start by discussing the terminology to be used in this thesis in Chapter 2. We describe the fundamentals of linear programming and the methods used to solve a linear program in this chapter. We also discuss Lagrangean relaxation and local search heuristics in Chapter 2. Local search heuristics are known to give constant factor approximations algorithms for the k-median problem [2] and the red-blue median problem [27].

In Chapter 3, we define the k-median problem. We discuss the previous research on k-median. Then, we explain the local search approach we use to compute an upper bound and a semi-Lagrangean relaxation we use to obtain a lower bound. We discuss our experimental study on the k-median problem and compare our results with the results of Beasley [15, 5].

In Chapter 4, we define the data placement problem. We discuss the related research.We explain the local search approach we use to compute an upper bound. We present two Lagrangean relaxations to compute the lower bounds. Finally we report on the empirical study of large sized instances of the problem and discuss our results.

We conclude the thesis in Chapter 5 with a summary of the findings of this research and list possibilities for future research.

Chapter 2

Preliminaries and Related Concepts

In this chapter, we discuss the terminology and concepts related to this thesis.

2.1 Optimization problem

An optimization problem is to determine the "best" solution from a set of all possible "feasible" solutions.

Definition 2.1. Optimization problem [42]
An instance of an optimization problem is a pair (F, c), where F is the domain of feasible solutions and c is the cost function, $c : F \rightarrow R$. The problem is to find a function $f \in F$ for which,

$$c(f) \leq c(y) \qquad \forall y \in F \tag{2.1}$$

Function f is called an optimal solution to the given minimization problem. The cost of the optimal solution lies between an upper bound and a lower bound. For a minimization problem, values which are larger than the cost of an optimal solution are called upper bounds and values which are smaller than the optimal solution are called lower bounds. The quality and gap between these bounds is important for the computational success of any approximation algorithm. We prefer the bounds to be as close as possible to the optimal solution.

Depending on the type of variables, optimization problems can be divided into two categories: *continuous optimization problems* which contain continuous variables and *combinatorial optimization problems* which contain discrete variables. The methods for solving

these two kinds of problems are quite different. In this thesis, we focus on combinatorial optimization problems.

2.2 Linear programming

Linear programming plays a unique role in optimization theory, and it is fundamental to the study of many combinatorial problems. A linear program (LP) is an optimization problem designed to maximize or minimize a given linear objective function by satisfying a given set of linear inequality or equality constraints. Linear programming was first introduced by Leonid Kantorovich in 1939 [47]. The basic elements of a linear program are [14]:

- Decision variables: The decision variables is a set of quantities that need to be determined in order to solve the problem. The goal is to find values of the variables that provide the best value of the objective function.
- Objective function: This is a mathematical expression that combines the variables to express the goal of a problem. We need to either maximize or minimize the objective function value.
- Constraints: Mathematical expressions that combine the variables to express limits or restrictions on the possible solutions.

A generalized form of the linear objective function is given below [53]

$$\zeta = c_1x_1 + c_2x_2 + ... + c_nx_n \tag{2.2}$$

Here ζ is the objective function value. $x_1, x_2, ..., x_n$ and $c_1, c_2, ..., c_n$ are the decision variables and their coefficients respectively. Linear programming is by far the most widely used method for constrained linear optimization. The constraints are in the form of either inequalities or equalities and they are broadly defined into two sub-categories: *technological constraints* (equation 2.3) and *non-negativity constraints* (equation 2.4). Technological

constraints usually define limitations on the decision that is made in the solution of the problem and non-negativity constraints ensure that the variables are non-negative.

$$a_{ij}x_1 + a_{ij}x_2 + ... + a_{ij}x_n \left\{ \begin{array}{l} \leq \\ = \\ \geq \end{array} \right\} b_i \qquad i = 1,2,....,m \tag{2.3}$$

$$x_j \geq 0 \qquad j = 1,2,....,n \tag{2.4}$$

In Equation (2.3), a_{ij} is the coefficient and b_i is the right side value for the i-th constraint. We combine the objective function with constraints to get the following complete linear programming model.

$$maximize \quad or \quad minimize \quad \zeta = \sum_{j=1}^{n} c_j x_j \tag{2.5}$$

$$s.t. \qquad \sum_{j=1}^{n} a_{ij}x_j \left\{ \begin{array}{l} \leq \\ = \\ \geq \end{array} \right\} b_i \qquad \forall i = 1,2,....,m \tag{2.6}$$

$$x_j \geq 0 \qquad \forall j = 1,2,....,n \tag{2.7}$$

In this LP model, the goal is to optimize the objective function such that the given constraints are satisfied. The number of decision variables is represented by n and number of constraints, is denoted by m.

There are two canonical forms taken by linear programming problems, a maximization canonical form and a minimization canonical form. [51].

Canonical form of maximization problem:

$$maximize \quad \sum_{j=1}^{n} c_j x_j \tag{2.8}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij}x_j \leq b_i \quad \forall i = 1,2,....,m \tag{2.9}$$

$$x_j \geq 0 \quad \forall j = 1,2,....,n \tag{2.10}$$

Canonical form of minimization problem:

$$minimize \quad \sum_{j=1}^{n} c_j x_j \tag{2.11}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij}x_j \geq b_i \quad \forall i = 1,2,....,m \tag{2.12}$$

$$x_j \geq 0 \quad \forall j = 1,2,....,n \tag{2.13}$$

To describe properties and algorithms for LP, it is convenient to use the standard form. A linear program in the standard form is a maximization or a minimization problem subject to linear equalities. The transformation of an LP from canonical to standard form can be easily done by using slack or surplus variables [42]. If we use the canonical form of a maximization LP using a matrix representation we get the following standard form.

Standard form of maximization problem:

$$maximize \quad c^T x \tag{2.14}$$

$$s.t. \quad Ax + s = b \tag{2.15}$$

$$x \geq 0, \quad s \geq 0 \tag{2.16}$$

Standard form of minimization problem:

$$minimize \quad c^T x \tag{2.17}$$

$$s.t. \quad Ax - s = b \tag{2.18}$$

$$x \geq 0, \quad s \geq 0 \tag{2.19}$$

Here, $c^T x$ is the objective function where x is the vector of decision variables and c is the vector of coefficients of x. $\boldsymbol{A}$ is an $m \times n$ matrix called the coefficient matrix, while b is a column vector that represents the right-hand side of the constraints. For the maximization problem, we use a slack variable s which is a vector in the constraint (Equation 2.15) to convert it to an equality constraint. On the other hand, in case of a minimization problem, we subtract a surplus vector to convert the constraint (Equation 2.18) to an equality constraint.

A set of specific values for the decision variables $(x_1, x_2, ..., x_n)$ is called a solution to the LP [53].

Feasible solution: A solution is called feasible if it satisfies all the constraints.

Optimal solution: A solution is called optimal if in addition the objective function attains the maximum (or minimum) value.

Infeasible LP: It is possible that there is no feasible solution to the given inequalities. In this case, we call the linear program infeasible [52].

Unbounded LP: If the linear program is feasible, sometimes it is possible that, for a maximization problem, there are solutions of arbitrarily large value, or for a minimization problem, there are solutions of arbitrarily small value. In this case, we say that the linear program is unbounded.

2.2.1 Simplex method

There are numerous methods available to solve a linear programming problem. The first and the most widely used method introduced by George B. Dantzig [17] is the Simplex method. It is an iterative process to find an optimal solution. In this section, we describe the steps we follow to solve a linear program using the simplex method [53]. We start with a general linear programming problem and transform it into the standard form. Let us consider the linear program described in (2.8 - 2.10). We add a slack variable s_i to each of

the inequality constraints. Thus, we get the following standard form

$$maximize \quad \zeta = \sum_{j=1}^{n} c_j x_j \tag{2.20}$$

$$s.t. \qquad s_i = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad \forall i = 1,2,....,m \tag{2.21}$$

$$x_j \geq 0 \qquad \forall j = 1,2,....,n, \quad s_i \geq 0 \tag{2.22}$$

We can consider

$$(x_1, x_2, ..., x_n, s_1, s_2, ..., s_m) = (x_1, x_2, ..., x_n, x_{n+1}, ..., x_{n+m}) \tag{2.23}$$

Thus, we can write,

$$maximize \quad \zeta = \sum_{j=1}^{n} c_j x_j \tag{2.24}$$

$$s.t. \qquad x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \qquad \forall i = 1,2,....,m \tag{2.25}$$

$$x_j \geq 0 \qquad \forall j = 1,2,....,m+n \tag{2.26}$$

The LP in the form of equations (2.24 - 2.26) is called a dictionary. The variables appearing on the left-hand side of the equality constraints are called basic variables while the ones on the right-hand side are called non-basic variables. Any solution obtained where all the non-basic variables are set to zero is called a basic feasible solution. As the simplex method progresses, it moves from one dictionary to another in its search for an optimal solution. Within each iteration of the simplex method, exactly one variable goes from nonbasic to basic, and exactly one basic variable becomes nonbasic. The variable that goes from nonbasic to basic is called the entering variable, and the variable that goes from basic to nonbasic is called the leaving variable. We examine the coefficients in the objective function and pick the entering variable such that it has the largest non-negative coefficient. Let $\mathcal{B}$ be the set of indices of the basic variables and $\mathcal{N}$ be the set of indices of the non-

basic variables. We select an entering variable from $\{j \in \mathcal{N} \mid c_j \geq 0\}$. Once the entering variable is selected, the leaving variable can be picked by finding the minimum value from $\{b_i/a_{ij} \mid i \in \mathcal{B} \text{ and } a_{ij} > 0\}$. Once the leaving and entering variables have been selected, the transition from the current dictionary to the new dictionary requires suitable row operations to achieve the interchange. This step of transition from one dictionary to other is called a pivot. At the time of performing a pivot operation we may encounter two special cases:

- Degeneracy: We say that a dictionary is degenerate if $b_i = 0$ for some i. Usually, after a few degenerate dictionaries, we reach a non-degenerate dictionary that leads towards an optimal solution. Sometimes by performing a sequence of degenerate pivots, the simplex method returns to a previously generated dictionary which causes an infinite loop. To avoid such a situation, we can use two well known pivoting methods namely the perturbation method and the Bland's rule [53].

- Unboundedness: This happens when all the ratios b_i/a_{ij} are non-positive or undefined ($a_{ij} = 0$). This means that there is no upper bound on the value of the entering variable. This gives an infinitely large value of the objective function.

2.2.2 Other methods

Some other methods for solving linear programs are:

Primal-dual method: This method was first introduced by Kuhn in 1955 [36]. Associated with every linear program is another called its dual. Thus, linear programs come in primal/dual pairs. Feasible solution for one of these two linear programs gives a bound on the optimal objective function value for the other [53].

Interior point method: In 1984, Karmarkar [32] proposed an interior point method to solve an LP. Contrary to the simplex method, it reaches the best solution by traversing the interior of the feasible region [53].

2.3 Integer programming (IP)

In a linear program, if some or all the variables are constrained to be integers, such problems are called Integer programming problems [53]. A large variety of real-life problems in practice are formulated as integer optimization problems. Even though the number of solutions is reduced when the variables are restricted to be an integer, IP problems are usually much more difficult to solve than LP problems. Depending on the types of variables there are three possible IP models.

- Pure IP: All the variables take integer values.
- Mixed IP: Only some of the variables are restricted to integer values.
- Binary IP: All the variables are binary (restricted to the values 0 or 1).

The canonical form of an Integer program (maximization) is:

$$maximize \quad \sum_{j=1}^{n} c_j x_j \tag{2.27}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad \forall i = 1,2,....,m \tag{2.28}$$

$$x_j \in \mathbb{Z} \quad \forall j = 1,2,....,n \tag{2.29}$$

2.4 Integer programming solution methods

Even though a bounded IP has only a finite number of feasible solution, the integer nature of the variables makes it difficult to formulate an efficient algorithm. Solution methods for IP can be categorized as:

- Optimal or exact methods
- Heuristic methods

An optimal algorithm guarantees to find the optimal solution. A heuristic algorithm finds a feasible solution which is close to the optimal solution. Heuristic or approximate methods are needed because exact methods cannot solve even moderately sized problems.

Exact solution strategies: The development of exact optimization methods for IP optimization problems during the last 50 years was very successful [25]. Some of the approaches are described briefly.

Enumerative approaches: The simplest approach of solving a pure integer-programming problem is to enumerate all the possibilities [29]. However, due to the combinatorial explosion resulting from the size of the problem, only relatively small-sized instances can be solved within a reasonable computational time limit. The most commonly used enumerative approach is called branch and bound. This algorithm was introduced by Land and Doig [37]. This approach recursively splits the search space into smaller spaces, and tries to minimize the objective function value on these smaller spaces; this splitting is called branching. The bounding refers to figure out the possible solutions by comparison to a known upper or lower bound on the solution value.

Cutting plane algorithms: Cutting plane methods solve the linear relaxation of a given integer program. Here the objective is to find a linear inequality that reduces the space of feasible solutions while assuring that all the feasible integer points satisfy the inequality [25]. Cutting planes are added successively until an integer solution is found.

Relaxation and Decomposition Methods: There are three basic approaches to relax an IP problem: Linear Programming (LP) relaxation, Combinatorial relaxation and Lagrangean relaxation. The first two approaches extend the feasible domain. In the third approach, a set of constraints is included in the objective function. This approach is described in detail in Section 2.5. Branch and Bound is typically used with cuts and relaxations.

Heuristic: The integer programming problems belong to the class of NP-hard optimization problems. For large sized problems, exact methods do not work. In that case, we use heuristic algorithms, to obtain good feasible solutions. Some of the widely used methods

are described briefly.

Local search based metaheuristics: The research in heuristics began with the concepts of local search [29]. Starting with a feasible solution, this method yields better solutions by iteratively changing the current solution. This method is described in detail in Section 2.6.

Constructive algorithms: Constructive algorithms for finding a feasible solution pick the best single move without any look-ahead. It always makes the choice that seems to be the best at that moment. These algorithms are among the fastest approximate algorithms, but they often achieve low-quality solutions which are far from the optimal solution.

2.5 Lagrangean relaxation

An important observation is that many hard integer programming problems can be viewed as an easy problem complicated by a relatively small number of side constraints [20]. These problems are difficult to solve using classical exact optimization methods. For such types of problems, we can obtain a lower bound (for minimization problem) comparatively quickly by applying Lagrangean relaxation. Held and Karp [28] pioneered the use of this technique for travelling salesman problem. In this section, we illustrate the Lagrangean Relaxation (LR) method for solving a relaxation of the IP.

2.5.1 Basic Formulation

Lagrangean relaxation is generally used in combinatorial optimization to find a lower bound for a minimization problem. For several problems, Lagrangean relaxation gives very good lower bounds at reasonable computational cost. For a given optimization problem, there can exist multiple different Lagrangean relaxations. At first, we take an integer programming formulation of a problem. Then the general steps in solving the problem using Lagrangean relaxation are:

- Select some constraints from the formulation to move into the objective function using Lagrange multipliers.

- Solve the resulting integer program either optimally or heuristically.

Let us consider the following integer program with complicating constraints A,

$$Z = min \quad c^T x \tag{2.30}$$

$$s.t. \quad Ax \geq b \tag{2.31}$$

$$Bx \geq d \tag{2.32}$$

$$x \in \{0,1\} \tag{2.33}$$

We introduce a dual variable for every constraint we are going to relax. Let $\lambda \geq 0$ be the Lagrange multiplier which is the vector of dual variables that will be attached to constraints A. If we consider a relaxation of this problem with respect to constraints (2.31), then the Lagrangean program $\boldsymbol{P'}$ is:

$$Z_{LR_1} = min \quad c^T x + \lambda^T (b - Ax) \tag{2.34}$$

$$s.t. \quad Bx \geq d \tag{2.35}$$

$$x \in \{0,1\} \tag{2.36}$$

In this relaxation, we add the term $\lambda^T (b - Ax)$ to the objective function. Here $\lambda \geq 0$ and $(b - Ax) \leq 0$ for a feasible solution, which makes the term $\lambda^T (b - Ax) \leq 0$. Thus we are adding a non-positive term with the objective function. Thus $Z_{LR_1} \leq Z$. We need to find a λ such that Z_{LR1} is maximum possible. This is achieved using the subgradient method explained below.

Alternatively, we can also select constraint set (2.32) to relax. Then the Lagrangean relaxation is:

$$Z_{LR_2} = min \quad c^T x + \lambda^T (d - Bx) \tag{2.37}$$

$$s.t. \quad Ax \geq b \tag{2.38}$$

$$x \in \{0,1\} \tag{2.39}$$

In this case, $Z_{LR_2} \leq Z$. The program above is called the Lagrangean lower bound program (LLBP).

There are two main concerns when using Lagrangean relaxation, which are:

- Selecting the constraint set to relax.
- Determining the values for the Lagrange multipliers.

Selecting constraint set to relax

There are many possible Lagrangean formulations depending on which constraint set we relax. Selecting an appropriate constraint set for relaxation is a strategic issue. Typically we choose to move the set of "complicating" constraints to the objective function.

Determining the Lagrange multiplier value

We try to find multipliers that gives us the possible maximum lower bound value. Thus, for ***P´***, we try to find multipliers such that,

$$\max \quad \lambda \geq 0 \quad \left\{ \begin{array}{r} \min \quad c^T x + \lambda^T (b - Ax) \\ s.t. \quad Bx \geq d \\ x \in \{0,1\} \end{array} \right\}$$

This program is called the Lagrangean dual program. There are two main approaches to decide the value of the Lagrange multipliers.

- Subgradient optimization
- Multiplier adjustment

In this thesis, we compute a lower bound to the k-median problem and the data placement problem using subgradient optimization. In the next section, we discuss the method briefly.

2.5.2 Subgradient optimization

Subgradient optimization is an iterative procedure which attempts to maximize the lower bound for a minimization problem. It starts with an initial set of multipliers and updates the value of multipliers in a systematic way. Algorithm 1 describes the basic steps [6] of the subgradient optimization procedure used by several people. The notations used in this algorithm are described in Table 2.1. In this subgradient iterative procedure, m is the number of relaxed constraints and n is the number of decision variables.

Table 2.1: Notations used in subgradient procedure

Symbols	**Description**
Z_{max}	Maximum lower bound found for a problem.
Z_{UB}	Upper bound for a problem (best feasible solution).
Z_{LB}	Lower bound for a problem.
π	A parameter.
G	Subgradient for the relaxed constraints.
T	Step size.
λ	Lagrange multiplier.

Algorithm 1: Subgradient iterative procedure

Input : A Lagrangean lower bound program $\boldsymbol{P}'$ and Z_{UB}.
Output: Maximum lower bound Z_{max}.

1 Initialize π, λ, and Z_{max}.
2 Solve the Lagrangean lower bound problem using the current set of multipliers and get a solution Z_{LB}, which is the lower bound of the problem.
3 Update the value of $Z_{max} = max(Z_{max}, Z_{LB})$
4 Evaluate the subgradient value G for the relaxed constraints using the formula: $G_i = (b_i - \sum_{j=1}^{n} a_{ij}x_j)$, $i = 1,2,..,m$ (considering $\boldsymbol{P}'$)
5 Define step size T by : $T = \frac{\pi(Z_{UB}-Z_{LB})}{\sum_{i=1}^{m}(G_i)^2}$
6 Update λ using : $\lambda_i = max(0, \lambda_i + TG_i)$, $i = 1,2,..,m$ and go to step (2) to recalculate the value of the lower bound with new set of multipliers.

The terminating condition for this iterative procedure can be set in various different ways as follows:

• We can limit the number of iterations.

• We can start with any value of the scalar parameter $\pi > 0$ then reduce the value of π systematically and terminate when the value of π is sufficiently small.

2.5.3 Semi-Lagrangean relaxation (SLR)

Semi-Lagrangean relaxation is a modified form of the Lagrangean relaxation method. The method of Semi-Lagrangean relaxation was introduced by C. Beltran et al. in the year 2006 [11]. The method has been used with success to solve large instances of combinatorial problems, but the relaxed problem is more difficult to solve than in the case of the standard Lagrangean relaxation [12] as original complicating constraints are still part of the relaxation.

Problem formulation

Let $S \subset X \cap \mathbb{N}^n$, where X is a polyhedral set, $0 \in S$. Consider the primal problem:

$$Z^* = \min_x \quad c^T x \tag{2.40}$$

$$s.t \quad Ax = b \tag{2.41}$$

$$x \in S \tag{2.42}$$

In the standard Lagrangean relaxation, if we relax the equality constraint we obtain the following problem:

$$\mathcal{L}_{LR}(\lambda) = \min_x \quad c^T x + \lambda^T (b - Ax) \tag{2.43}$$

$$s.t \quad x \in S \tag{2.44}$$

The Lagrangean dual problem is :

$$Z_{LR} = \max_{\lambda \in \mathbb{R}^m} \mathcal{L}_{LR}(\lambda) \tag{2.45}$$

$$s.t \quad x \in S \tag{2.46}$$

The solution to the Lagrangean dual yields a lower bound for the original problem.

$$Z_{LR} \leq Z^* \tag{2.47}$$

In the semi-Lagrangean relaxation, we relax the equality constraint but at the same time keep a weaker form of the equality constraint in the subproblem.

$\mathcal{L}_{SLR}(\lambda)$ is the semi-Lagrangean LB defined as,

$$\mathcal{L}_{SLR}(\lambda) = \min_{x} \quad c^T x + \lambda^T (b - Ax) \tag{2.48}$$

$$s.t \quad Ax \leq b \tag{2.49}$$

$$x \in S \tag{2.50}$$

The Lagrangean dual problem is :

$$Z_{SLR} = \max_{\lambda \in \mathbb{R}^m} \mathcal{L}_{SLR}(\lambda) \tag{2.51}$$

$$s.t \quad Ax \leq b \tag{2.52}$$

$$x \in S \tag{2.53}$$

Significance of SLR

If we contrast the Lagrangean and the semi-Lagrangean relaxation we see that semi-Lagrangean relaxation is more constrained. Thus, the lower bound given by SLR is better (no worse) than the standard Lagrangean relaxation.

$$Z_{LR} \leq Z_{SLR} \leq Z^* \tag{2.54}$$

Although the semi-Lagrangean relaxation is more powerful than the standard Lagrangean relaxation, solving Z_{SLR} is much more complex than solving Z_{LR}.

2.6 Local search heuristic

Local search is a widely used and general approach to solve hard optimization problems. Computational studies of local search algorithms have been extensively reported in the literature for various combinatorial optimization problems [31, 30]. Empirically, local search heuristics appear to converge rather quickly, within a low-order polynomial time [41]. Local search based approximation algorithms are known for uncapacitated facility location problem, k-median problem [2], red-blue median problem [27] etc.

2.6.1 General algorithm

In this subsection, we describe a general scheme for local search [42].
Given is an instance (F, c) of an optimization problem, where F is the set of feasible solution and c is the cost function. A good local search heuristic involves choosing an appropriate neighbourhood. We choose a neighbourhood

$$N : F \to 2^F \tag{2.55}$$

In order to find a neighbourhood, we need to explore at most 2^F feasible solutions.
At point $t \in F$ we improve the current solution as follows

$$\text{improve}(t) = \begin{cases} \text{any } s \in N(t) \text{ with } c(s) < c(t) \\ \text{“no” otherwise.} \end{cases}$$

Finding efficient neighbourhood functions that lead to high quality local optima can be viewed as one of the challenges of local search [1].
Algorithm 2 describes the general local search heuristic. We start with an initial feasible solution and search for a better solution in neighbourhood. If such a solution is found, it replaces the current solution, and the search continues. As long as the solution can be

Algorithm 2: General local search algorithm

```
1 t ← Some initial solution in F.
2 while improve(t) ≠ "no" do
3 |   t ← improve(t).
4 end
5 return t
```

improved we iterate. We stop when we reach a local optimum. A large neighbourhood would provide a better local optima but searching a large neighbourhood requires more time.

The terms discussed in this chapter will be used in the following chapters. In the next chapter we explain the k-median problem.

Chapter 3

k-median problem

The problem of locating facilities in a manner so that they can effectively serve a set of clients has been the subject of a lot of research [2]. There are many interesting variants of facility location problems. One of the widely studied facility location problem is the k-median problem. This chapter discusses in detail the k-median problem along with the local search method and a semi-Lagrangean relaxation. At first, the k-median problem formulation is presented in Section 3.1. The related works are described in Section 3.2, computation of upper bound and lower bound is discussed in Sections 3.3 and 3.4 respectively. Finally, the experiments and the results are described in Section 3.5.

3.1 Problem definition

The k-median problem has been extensively studied for decades, in computer science and in operations research. The problem is to locate k facilities on a network such that the sum of all the distances from each client to its nearest facility is minimized [15]. In k-median problem, we are given a set of clients D, a set of facilities F and a positive integer k, $(0 < k \leq \mid F \mid)$ that is an upper bound on the number of facilities that can be opened. There is a cost c_{ij} of assigning client $j \in D$ to facility $i \in F$. The goal is to open at most k facilities and assign clients to open facilities such that the total assignment cost is minimized. We consider the clients and facilities are in metric space and the assignment cost c_{ij} is the distance between client j and facility i.

The k-median problem can be formulated as an integer linear program [54]. Let us consider

two binary variables y_i and x_{ij} such that,

$$y_i = \begin{cases} 1 & \text{if facility } i \text{ is open} \\ 0 & \text{Otherwise.} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if client } j \text{ is assigned to facility } i \\ 0 & \text{Otherwise.} \end{cases}$$

Then the integer linear program of k-median problem is as follows:

$$\textit{minimize} \quad \sum_{i \in F} \sum_{j \in D} c_{ij} x_{ij} \tag{3.1}$$

$$s.t. \quad \sum_{i \in F} x_{ij} = 1, \qquad \forall j \in D \tag{3.2}$$

$$x_{ij} \leq y_i, \qquad \forall i \in F, j \in D \tag{3.3}$$

$$\sum_{i \in F} y_i \leq k \tag{3.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in F, j \in D \tag{3.5}$$

$$y_i \in \{0,1\} \qquad \forall i \in F \tag{3.6}$$

(3.1) is the objective function, where the goal is to minimize the total assignment cost $\sum_{i \in F} \sum_{j \in D} c_{ij}$. Constraint (3.2) indicates each client must be assigned to exactly one facility. Equation (3.3) implies, a client j can be assigned to facility i only if that facility is open. Constraint (3.4) indicates, at most k facilities can be opened. (3.5) & (3.6) are integrality constraints.

3.2 Related research

There is a vast amount of research and empirical studies on the k-median problem both in computer science and operations research. In the year 1982, Christofides and Beasley presented two lower bounds for the k-median problem based on two separate Lagrangean

relaxations in conjunction with subgradient optimization [15]. They formulated the k-median problem as a zero-one program as follows:

Let V be the vertex set and d_{ij} the non-negative cost of allocating vertex j to vertex i.

$$a_{ij} = \begin{cases} 0 & \text{if vertex } j \text{ cannot be allocated to vertex } i \\ 1 & \text{Otherwise.} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if vertex } j \text{ is allocated to vertex } i \\ 0 & \text{Otherwise.} \end{cases}$$

The problem is :

$$minimize \quad \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij} \tag{3.7}$$

$$s.t. \sum_{i \in V} x_{ij} = 1 \qquad \forall j \in V \tag{3.8}$$

$$\sum_{j \in V, i \neq j} a_{ij} x_{ij} \leq n_i x_{ii}, \qquad \forall i \in V \tag{3.9}$$

$$\sum_{i \in V} x_{ii} = k \tag{3.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in V, j \in V \tag{3.11}$$

In equation (3.9),

$$n_i = \sum_{j \in V, i \neq j} a_{ij} \qquad \forall i \in V \tag{3.12}$$

The two Lagrangean relaxations considered by the authors were,

1. LR1 : relaxation of the constraint $\sum_{i \in V} x_{ij} = 1, \qquad \forall j \in V$
2. LR2 : relaxation of the constraint $\sum_{j \in V, i \neq j} a_{ij} x_{ij} \leq n_i x_{ii}, \qquad \forall i \in V$

For both relaxations LR1 and LR2, they start with an initial multiplier λ and progressively update the value of λ by using the subgradient optimization technique. They use penalty tests on the lower bound and a heuristically determined upper bound for the problem which

results in a large reduction in the problem size. For the case when the subgradient procedure does not optimally solve the dual problem, a reduction in the problem size is obtained because the penalty tests will fix certain vertices as medians/non-medians, these removes a large percentage of possible allocation. They used the depth first tree search procedure on the reduced problem, to obtain an optimal solution. They tested their procedure for arbitrary number of medians up to 200 vertices. In their experiments, relaxation LR1 was found to be superior to relaxation LR2 and very few of the instances required branching for relaxation LR1. Later in 1985, Beasley showed that it is possible to enhance their algorithm to optimally solve instances up to 900 vertices [5]. In this work, the vector processing capability of a super computer Cray-1S was used and some algorithmic enhancement was done for relaxation LR1. Crowder et al. [16] reported that it is beneficial to abandon the tree search after a certain stage and restart the problem. Beasley [5], followed a similar strategy to restart the problem. In 1993, Beasley presented a framework for developing Lagrangean heuristics based upon Lagrangean relaxation and subgradient optimization with respect to location problems [8]. The basic idea was that the information contained in the Lagrangean lower bound problem at each subgradient iteration can be used to construct a feasible solution. The best feasible solution found is a heuristic solution to the original problem. Beltran et al. studied a modified Lagrangean relaxation to generate lower bound for the k-median problem [11]. For an instance I, let $global(I)$ denote the global optimum and $local(I)$ be the loally optimum solution provided by a certain local search heuristic. Then the supremum of the ratio $global(I)/local(I)$, is called the locality gap [2]. In 2004, Vijay Arya et al. [2] analyzed the local search heuristic for the k-median problem and they proved the local search heuristic with a swap operation has locality gap of 5. We describe the local search approach and the proof due to [26], as it form the basis of the work in subsequent chapter on the data placement problem.

3.3 Computing the upper bound

In this section, we describe a technique we use to compute the upper bound of the k-median problem. First, we describe a local search heuristic to compute the upper bound and then we discuss the analysis.

3.3.1 Local search method

Local search heuristics are very popular for hard combinatorial optimization problems. The main idea of local search is that it is often possible to find an acceptable solution to a problem by frequently improving the given solution locally. Different types of possible local changes/moves represent various heuristics. In the local search heuristic for computing the upper bound of the k-median problem, we use a local move called swap. In a swap move, we can open a new facility which is currently closed and close a currently open facility simultaneously.

Swap move

If F is the set of facilities and $S \subseteq F$ is the set of currently open facilities, we can define the swap operation as :

$Swap(i,i') := S - i + i'$, where $i \in S$ and $i' \notin S$.

In the swap operation, we close currently open facility i and open a new facility i' which is currently closed. All the clients $j \in D$ are re-assigned to the nearest open facility.

Algorithm 3 describes the steps of the local search heuristic to compute the upper bound of the k-median problem. We start with a feasible solution $S \subseteq F$ by randomly opening k facilities from the given set of facilities F. We repeatedly swap a median from S for an element in $F - S$ which minimizes the cost. We re-assign the clients to the nearest open facility in S. By this way we continue to perform the swap operation until there is an improvement in the total assignment cost.

Algorithm 3: Local search for k-median

Input : Set of facilities F, set of clients D, positive integer k, cost matrix C.
Output: Set of open facilities S.

$S \leftarrow$ an arbitrary feasible solution, where $S \subseteq F$ and $|S| = k$
Assign all the clients to their nearest open facility.
Calculate the total assignment cost $C_{total} = \sum_{i \in F} \sum_{j \in D} c_{ij}$.
while $\exists\, Op(S) = S - s + s'$ *such that* $s \in S$ *and* $s' \notin S$ *and* $Cost(Op(S)) < C_{total}$ **do**
    $S \leftarrow Op(S)$.
    $C_{total} \leftarrow Cost(Op(S))$.
end
return S

3.3.2 A simpler analysis of local search method for the k-median problem

In 2008, Gupta and Tangwongsan [26] gave a simpler analysis of the local search algorithm (Algorithm 3) for the k-median problem. We describe their proof in this section. Table 3.1 describes the notations used in the analysis.

Table 3.1: Notations used in the analysis of local search method for the k-median problem

Symbols	**Description**
F	Local optimal set of facilities.
F^*	Optimal set of facilities.
D	Set of clients.
$kmed(F) = \sum_{j \in D} d(j,F)$	k-median cost which is the sum of the distance from each client j to facility F given that, F has at-most k facilities.
$\eta : F^* \to F$	Mapping of each optimal facility f^* to the closest facility $\eta(f^*) \in F$ such that $d(f^*, \eta(f^*)) \leq d(f^*, f)$ for all $f \in F$.
$R \subseteq F$	All facilities that have at-most 1 facility in F^* mapped to it by the map η.
$\varphi : D \to F$	Functions mapping each client to the closest facility in F.
$\varphi^* : D \to F^*$	Functions mapping each client to the closest facility in F^*.
$O_j = d(j,F^*) = d(j,\varphi^*(j))$	Client j's cost in the optimal solution.
$A_j = d(j,F) = d(j,\varphi(j))$	Client j's cost in the local optimal solution.
$N^*(f^*) = \{j \mid \varphi^*(j) = f^*\}$	Set of clients assigned to f^* in the optimal solution.
$N(f) = \{j \mid \varphi(j) = f\}$	Set of clients assigned to f in the local optimal solution.

Let us assume that $|F| = |F^*| = k$. We define a set of k pairs $P = (r, f^*) \subseteq R \times F^*$ such that,

• Each $f^* \in F^*$ appears in exactly one pair (r, f^*).

• If $\eta^{-1}(r) = f^*$ then r appears only once in P as the tuple (r, f^*).

• If $\eta^{-1}(r) = \emptyset$ then r appears in at most two tuples in P.

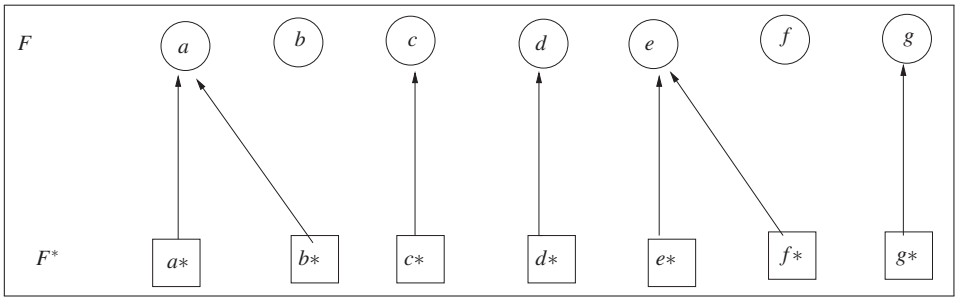

Figure 3.1: An example mapping $\eta : F^* \rightarrow F$

Procedure for construction of the k-pairs with an example:

For each $r \in R$ (with in-degree 1), construct the pair $(f, \eta^{-1}(r))$. Such pairs in Figure 3.1 are: $\{(c,c^*),(d,d^*),(g,g^*)\}$.

Let the optimal facilities that are already matched be denoted by F_1^*. In this example, F_1^* = $\{c^*, d^*, g^*\}$.

Let R_0 be the set of facilities in F with in-degree zero. Here, $R_0 = \{b, f\}$.

A simple averaging argument shows that the number of unmatched optimal facilities $|F^* \setminus F_1^*| \leq 2|R_0|$. In Figure 3.1, $F^* \setminus F_1^*$ = $\{a^*, b^*, e^*, f^*\}$. $|F^* \setminus F_1^*| = 4$ and $|R_0| = 2$. Thus we get, $|F^* \setminus F_1^*| = 2|R_0|$.

Now, we arbitrarily create pairs by matching each node in R_0 to at most two pairs in $F^* \setminus F_1^*$, so that the above conditions are satisfied. For example, we can construct the following pairs : $\{(b,a^*),(b,b^*),(f,e^*),(f,f^*)\}$.

Finally, $P = \{(c,c^*),(d,d^*),(g,g^*),(b,a^*),(b,b^*),(f,e^*),(f,f^*)\}$.

When each facility in F is closest to exactly one facility in F^* and far away from all other facilities in F^*, opening facility $f^* \in F^*$ and closing the matched facility $f \in F$ can be

handled by re-assigning all the clients served by facility f to the facility f^*. When a facility $f \in F$ is closest to several facilities in F^*, closing f and opening only one of several facilities in F^* might still cost too much. This is the reason for creating the pairs as above.

Lemma 3.1. *For each swap* $(r, f^*) \in P$,

$$kmed(F + f^* - r) - kmed(F) \leq \sum_{j \in N^*(f^*)} (O_j - A_j) + \sum_{j \in N(r)} 2O_j \tag{3.13}$$

Proof. Left hand side is the change in the cost, where the initial cost was $kmed(F)$ and the new cost is $kmed(F + f^* - r)$. In the right hand side, we consider the following possibly suboptimal candidate assignment of the clients:

• Map each client in $N^*(f^*)$ to f^*.

• For each client $j \in N(r) \setminus N^*(f^*)$ we reassign as in figure 3.2.

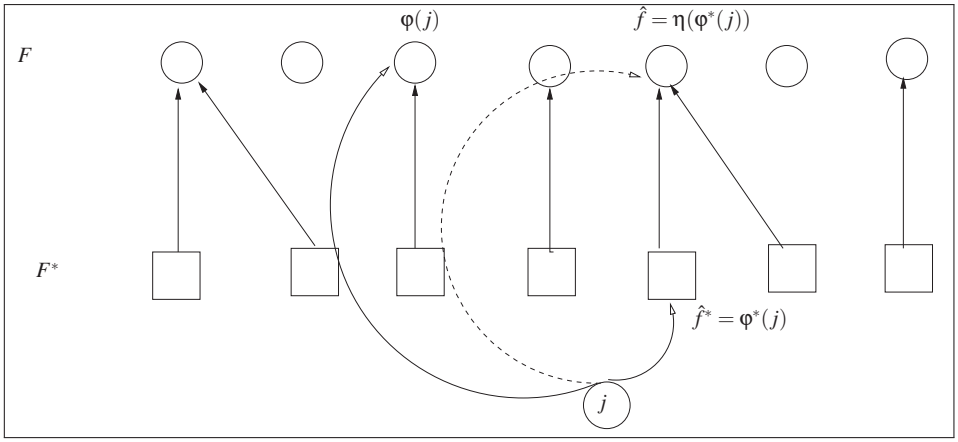

Figure 3.2: Assignment of clients

Let the facility $\hat{f}^* = \varphi^*(j)$: assign j to $\hat{f} = \eta(\hat{f}^*)$, the closest facility in F to $\hat{f}^*$. This is a valid new assignment ($\hat{f} \neq r$) because $\hat{f}$ is not closed. All the other clients in $D \setminus (N(r) \bigcup N^*(f^*))$ are assigned as they were in φ.

For any client $j \in N^*(f^*)$, the change in the cost is exactly $O_j - A_j$. Summing over all clients we get: $\sum_{j \in N^*(f^*)} (O_j - A_j)$.

For any client $j \in N(r) \setminus N^*(f^*)$ change in the cost is,

$d(j,\hat{f}) \leq d(j,\hat{f}^*) + d(\hat{f}^*,\hat{f})$ (By the triangle inequality)

$d(j,\hat{f}) - d(j,r) \leq d(j,\hat{f}^*) + d(\hat{f}^*,\hat{f}) - d(j,r)$ (Subtracting $d(j,r)$ from both sides)

$\leq d(j,\hat{f}^*) + d(\hat{f}^*,r) - d(j,r)$ ($\hat{f}$ is the closest vertex in F from $\hat{f}^*$, so $d(\hat{f}^*,r) \geq d(\hat{f}^*,\hat{f})$)

$\leq d(j,\hat{f}^*) + d(j,\hat{f}^*)$ (By the triangle inequality, $d(\hat{f}^*,r) \leq d(j,\hat{f}^*) + d(j,r)$)

$\leq 2O_j$

Summing up the total change for all these clients is at most $\sum_{j \in N(r) \setminus N^*(f^*)} 2O_j$.

Now, $\sum_{j \in N(r) \setminus N^*(f^*)} 2O_j \leq \sum_{j \in N(r)} 2O_j$

Thus we get, $kmed(F + f^* - r) - kmed(F) \leq \sum_{j \in N^*(f^*)} (O_j - A_j) + \sum_{j \in N(r)} 2O_j$

□

Theorem 3.2. *At a local minimum F, the cost $kmed(F) \leq 5\ kmed(F^*)$.*

Proof. The right hand side of the inequality (3.13) must be non-negative, otherwise we have a move that decreases the cost. Summing (3.13) over all the tuples in P along with the fact that, each $f^* \in F^*$ appears exactly once and each $r \in R \subseteq F$ appears at most twice gives us,

$(O - A) + 2.(2.O) \geq 0$

$5.O - A \geq 0$

$A \leq 5.O$

Which proves that, cost $kmed(F) \leq 5\ kmed(F^*)$. □

3.4 Computing the lower bound

In this section, we describe the semi-Lagrangean relaxation we use to compute the lower bound for the k-median problem. Semi Lagrangean relaxation is a modified Lagrangean relaxation, that closes the integrality gap for any combinatorial problem with equality constraint. Beltran et al. first introduced the concept of semi-Lagrangean relaxation [11] for k-median problem. In order to strengthen the standard Lagrangean relaxation, they relaxed the equality constraints that ensures that each client must be assigned to exactly one median. At the same time, that equality constrains were considered as "less than or equal"

inequality. They called this formulation, a semi-Lagrangean relaxation. At first, we describe the semi-Lagrangean formulation and then we discuss the algorithms we use to solve the Lagrangean lower bound problem.

3.4.1 Semi-Lagrangean relaxation for k-median

In order to determine the lower bound, we formulate the semi-Lagrangean relaxation from the original integer linear formulation discussed in Section 3.1. We relax the equality constraints (3.2) as well as constraints in (3.3). Thus, the semi-Lagrangean relaxation is given by:

$$min \sum_{i\in F}\sum_{j\in D} c_{ij}x_{ij} + \sum_{j\in D}\lambda_j(1-\sum_{i\in F}x_{ij}) + \sum_{j\in D}u_j(\sum_{i\in F}x_{ij}-1) + \sum_{i\in F}\sum_{j\in D}v_{ij}(x_{ij}-y_i) \quad (3.14)$$

$$s.t. \sum_{i\in F} x_{ij} \leq 1, \qquad \forall j \in D \quad (3.15)$$

$$\sum_{i\in F} y_i \leq k \quad (3.16)$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in F, j \in D \quad (3.17)$$

$$y_i \in \{0,1\} \qquad \forall i \in F \quad (3.18)$$

In the objective function (3.14), λ, u and v are non-negative Lagrange multipliers. Multipliers λ and u are associated with the equality constraints and brought into the objective function. We also keep a weaker form of the equality constraints in (3.15). Multiplier v is attached with the constraints (3.3). Equation (3.14) can be re-written as :

$$min \sum_{i\in F}\sum_{j\in D} x_{ij}(c_{ij}-\lambda_j+u_j+v_{ij}) - \sum_{i\in F}\sum_{j\in D} v_{ij}y_i + \sum_{j\in D}(\lambda_j-u_j) \quad (3.19)$$

Now the goal is to minimize this objective function subject to (3.15) - (3.18). At first, we start with a non-negative set of multipliers and calculate the first term of the objective function. Since this is a minimization problem, we want the negative terms in this function

to be as large as possible. We set $x_{ij} = 1$ if its smallest coefficient value in (3.19) is non-positive.

We consider the sum of the negative term $\sum_{i \in F} \sum_{j \in D} v_{ij} y_i$ in the equation and attach this sum with constraint (3.16) and we get the following subproblem:

$$max \sum_{i \in F} \sum_{j \in D} v_{ij} y_i \qquad \forall i \in F, j \in D \tag{3.20}$$

$$s.t \quad \sum_{i \in F} y_i \leq k \tag{3.21}$$

$$y_i \in \{0,1\} \qquad \forall i \in F \tag{3.22}$$

We have a total of n facilities (items) from which we can select at-most k facilities (items). Each facility (item) i has a cost (profit) v_{ij}, $\forall i \in F, j \in D$. We have to pick the facilities (items) such that the total cost of selected facilities (items) is maximum. Here y_i is a binary variable that is one when i^{th} facility (item) is selected and zero otherwise. Then this subproblem is exactly the 0-1 knapsack problem where each item has unit size.

3.4.2 The Algorithm

Solving the 0-1 knapsack subproblem

In the classic 0-1 knapsack problem, we are given n items and a knapsack of capacity c. If each item j has a profit p_j and size w_j then the problem is to select a subset of n items such that the total size does not exceed the capacity c of knapsack and the total profit is maximum [39].

Now, in our subproblem given by (3.20), (3.21), and (3.22), we assume each item has a unit size ($w_j = 1$). We solve this 0-1 knapsack problem optimally using a greedy approach.

In Algorithm 4, we sort the items by non-increasing profit value. Then we fill the knapsack by selecting items from the sorted list until the total number of items is exactly equal to k. This greedy approach of solving knapsack problem gives us the optimal value when the items are of uniform sizes.

Algorithm 4: Knapsack Greedy algorithm

Input : Non-negative cost (profit) v_{ij}, $\forall i \in F$ and $\forall j \in D$ and number of facilities (items) to select, k.

Output: Maximum cost, val and facilities (items) selected, y.

```
y ← 0
val ← 0
foreach i = 1,....,| F | do
    foreach j = 1,....,| D | do
        S_i ← Sum(v_ij)
    end
end
T ← Sort(S).
idx ← Index of sorted elements of S
foreach l = 1,....,k do
    val ← val + S_l
    y(idx(l)) ← 1
end
```

We get optimal solution for this semi Lagrangean formulation of the problem because in the objective function (3.19), we are setting $x_{ij} = 1$ only for the smallest non-positive coefficient associated with it and we are also solving the 0-1 knapsack sub-problem optimally.

Subgradient optimization

In order to maximize the lower bound we use an iterative procedure. We start with non-negative set of multipliers and update the value of multipliers in a systematic way.

Table 2.1 in Chapter 2 describes the notations used in subgradient procedure. In Algorithm 5, we initialize the value $\pi = 7$, Lagrange multipliers λ and $u = \min_i c_{ij} \quad \forall j \in D$, $v = 0$ and $Z_{max} = -\infty$. Then we find an upper bound on the problem by solving either an integer linear program where possible or by using a local search method. To calculate the lower bound, we evaluate the term $(c_{ij} - \lambda_j + u_j + v_{ij})$ in equation (3.19) and if the value is less than or equal to 0 then we set $x_{ij} = 1$ such that $\sum_{i \in F} x_{ij} \leq 1, \forall_{j \in D}$. We obtain the maximum value for the negative term in equation (3.19) by solving the 0-1 knapsack problem as described in Algorithm 4. Finally, after computing the term $\sum_{j \in D} (\lambda_j - u_j)$, we calculate the value Z_{LB} which is the lower bound for the problem. Next, we update the value of Z_{max} which is the

Algorithm 5: Subgradient optimization procedure

Input : A Lagrangean lower bound program.
Output: Maximum lower bound Z_{max}.

1 Initialize the values of : π, λ, u, v and Z_{max}.
2 Find an upper bound Z_{UB} by solving the k-median problem using Integer linear program or by the local search method described in Section 3.3.1 .
3 Compute the Lagrangean lower bound using the initial set of multipliers and by solving the 0-1 knapsack sub-problem described in 3.4.2. Get a solution Z_{LB}, which is the lower bound of the problem.
4 Update the value of $Z_{max} = max(Z_{max}, Z_{LB})$
5 If the value of new lower bound is not better than the previous lower bound for consecutive 30 iterations then $\pi = \pi/2$
6 Evaluate the subgradient value G.
7 Define the step size $T = \frac{\pi * 1.05 * (Z_{UB} - Z_{LB})}{|G|^2}$.
8 Update the lagrange multiplier values and go to step (3) to re-calculate the value of lower bound with new set of multipliers.
9 Repeat this process for a fixed number of iterations.

best lower bound found so far. In our next step, we determine the subgradient value $G = [G_1$ G_2 $G_3]$ by combining the following three subgradient vectors :

$$G_1 = (1 - \sum_{i \in F} x_{ij}), \quad \forall j \in D \tag{3.23}$$

$$G_2 = (\sum_{i \in F} x_{ij} - 1), \quad \forall j \in D \tag{3.24}$$

$$G_3 = x_{ij} - y_i, \quad \forall i \in F, j \in D \tag{3.25}$$

Using the value of G, we calculate the step-size T. This step size also depends upon the gap between the current lower bound and the upper bound, and the user defined parameter π. While calculating the step size, we use a multiplicative factor of 1.05 which ensures that T does not become very small as the gap between the upper bound and lower bound closes. After calculating the step size, we update the multiplier values as :

$$\lambda_i = max(0, \lambda_i + TG_1(i)) \qquad i = 1, ..., \mid D \mid \tag{3.26}$$

$$u_j = max(0, u_j + TG_2(j)) \qquad j = 1, ..., \mid D \mid \tag{3.27}$$

$$v_{ij} = max(0, v_{ij} + TG_3(ij)) \qquad i = 1, ..., \mid F \mid, j = 1, ..., \mid D \mid \tag{3.28}$$

We repeat the subgradient process with the new set of multipliers. Thus, we get a new value for the lower bound. If this new lower bound is better than the previous value we update the value of Z_{max} with the new lower bound. We continue for upto 500 iterations.

3.5 Experiments and Results

The semi-Lagrangean relaxation for the k-median problem described in section 3.4.1 is tested on 40 different test instances from Operations Research (OR) library [7] with up to 900 vertices. The format of these data files is: number of vertices, number of edges, k, for each edge: the end vertex and the cost of the edge. In these test instances, the number of vertices n vary from 100 to 900 in steps of 100. Values of k is selected as 5, 10, $n/10$, $n/5$ and $n/3$. For each distinct pair of values (n,k), a new network is randomly generated with $n^2/50$ edges. Each edge cost is an integer uniformly generated in $[1:100]$ and the transitive closure of the cost matrix is computed using the Floyd Warshal algorithm [21] to ensure that the cost matrix is triangular. To generate the complete allocation cost matrix before passing it to the Floyd's algorithm we follow the steps described in Algorithm 6.

Algorithm 6: Generate allocation cost matrix

Input : Number of vertices n.
Output: Symmetric allocation cost matrix c

1 Set $c(i, j) = \infty$ for $i = 1, ..., n$ and $j = 1, ..., n$.
2 Set $c(i, i) = 0$ for $i = 1, ..., n$
3 Read each edge line from the data file in turn: if three numbers in the line are i, j, k.
4 Set $c(i, j) = k$ and $c(j, i) = k$.

In this algorithm, we initialize every possible edge with a large cost (∞). Then we assign a value of zero to all diagonal elements. We read each edge cost value in turn from the input file. Finally we subject the cost matrix c to Floyd's algorithm to get a symmetric cost matrix.

In Table 3.2, we give the optimal value, lower bound, duality gap and the time to compute the lower bound for 40 different size test instances. We obtain the optimal value for the test files from the OR library [7]. We compute the lower bound using semi-Lagrangean relaxation in conjunction with subgradient optimization method. The duality gap [5] is given by the formula:

$$\frac{OPT - LB_{max}}{OPT} \times 100\% \tag{3.29}$$

Here, OPT is the optimal value for the problem and LB_{max} is the maximum lower bound for the problem. From the experimental results we can see that the optimal value and lower bound values are relatively close to each other.

Table 3.2: Experimental results for k-median problem

Test file	**Size**	**OPT**	**Lower bound**	**Duality gap**	**Time(seconds)**
1	Facility/client=100 k=5	5819	5816	0.05	395.36
2	Facility/client=100 k=10	4093	4059.2	0.83	393.69
3	Facility/client=100 k=10	4250	4235	0.35	391.32
4	Facility/client=100 k=20	3034	3034	-	390.37
5	Facility/client=100 k=33	1355	1342	0.96	392.78
6	Facility/client=200 k=5	7824	7754	0.89	1562.9
7	Facility/client=200 k=10	5631	5609	0.39	1549.9
8	Facility/client=200 k=20	4445	4408	0.83	1548.9
9	Facility/client=200 k=40	2734	2688	1.68	1551.8
10	Facility/client=200 k=67	1255	1211	3.5	1557.5
11	Facility/client=300 k=5	7696	7644	0.67	3491.8
12	Facility/client=300 k=10	6634	6598	0.54	3494.2

Table 3.2: Experimental results for k-median problem

Test file	Size	OPT	Lower bound	Duality gap	Time(seconds)
13	Facility/client=300 k=30	4374	4345	0.66	3490.2
14	Facility/client=300 k=60	2968	2897	2.39	3492.5
15	Facility/client=300 k=100	1729	1637	5.32	3493.7
16	Facility/client=400 k=5	8162	8078.2	1.02	6241
17	Facility/client=400 k=10	6999	6931	0.97	6222.9
18	Facility/client=400 k=40	4809	4731	1.62	6263.9
19	Facility/client=400 k=80	2845	2765.2	2.8	6269.3
20	Facility/client=400 k=133	1789	1638	8.44	6247.8
21	Facility/client=500 k=5	9138	9128	0.11	9778.9
22	Facility/client=500 k=10	8579	8502	0.89	9859.2
23	Facility/client=500 k=50	4619	4534.3	1.83	9851.5
24	Facility/client=500 k=100	2961	2835.4	4.24	9622.5
25	Facility/client=500 k=167	1828	1619	11.43	9911.8
26	Facility/client=600 k=5	9917	9812	1.05	13860
27	Facility/client=600 k=10	8307	8265	0.5	14004
28	Facility/client=600 k=60	4498	4388	2.45	14132
29	Facility/client=600 k=120	3033	2863	5.61	13799
30	Facility/client=600 k=200	1989	1760.3	11.49	14563
31	Facility/client=700 k=5	10086	9974	1.11	19192
32	Facility/client=700 k=10	9297	9235.3	0.66	19194
33	Facility/client=700 k=70	4700	4561.1	2.96	19386

Table 3.2: Experimental results for k-median problem

Test file	Size	OPT	Lower bound	Duality gap	Time(seconds)
34	Facility/client=700 k=140	3013	2821	6.37	19270
35	Facility/client=800 k=5	10400	10252	1.42	25381
36	Facility/client=800 k=10	9934	9785	1.5	25283
37	Facility/client=800 k=80	5057	4894	3.22	25215
38	Facility/client=900 k=5	11060	10876	1.66	32111
39	Facility/client=900 k=10	9423	9275	1.57	32073
40	Facility/client=900 k=90	5128	4944.4	3.58	31969

We implemented the subgradient method (Algorithm 5) for computing Lagrangean lower bound in Octave 4.0.2. The machine has an Intel Xeon processor with a clock speed of 3.40 GHz and 8GB of RAM running CentOS. The local search approach (Algorithm 3) is implemented in C++.

3.6 Discussion

- We used the test instances by Beasley [7] to experimentally study the computation of lower bound. The relaxations LR1 and LR2 as described in Section 3.2 were tested for a maximum of 200 vertices [15]. The relaxation LR2 was able to handle up to 100 vertices with 5 medians, and the maximum duality gap was 7.396 [15]. In our experiment, we can see the duality gap for the same instance is 0.05 (test file 1 in Table 3.2). Beasley enhanced relaxation LR1 and solved the k-median problem for up to 900 vertices [5]. If we compare their results with ours then we see that, they received an optimal value for 8 test instances out of 40 instances where we get optimal value for 1 instance. While comparing the duality gap we see, the gap we get is very close to their duality gap for 22 test instances out of 40 instances. They used a

super computer Cray-1S to perform their calculation. We can solve the instances on a laptop given Moore's law. Beltran et al. [11] tested their semi-Lagrangean relaxation by using data from the traveling salesman problem library [46] to define k-median instances. The objective of their numerical experiments was to study the influence of using a good starting point to maximize the dual function, the solution quality of semi-Lagrangean solution and study its performance. In their experiment, at first they formulated the standard Lagrangean relaxation of the k- median problem by relaxing constraints (3.2) and (3.4) while considering (3.4) as equality constraint. Then they formulated the semi-Lagrangean relaxation by keeping a weak form of constraints (3.2) and (3.4) in the problem. After solving the Lagrangean relaxation dual problem, if the optimal solution to the dual problem is also feasible for the original ILP of k-median problem, then they used the associated Lagrange multiplier for this dual solution as the starting point for solving the semi Lagrangean dual problem. By this way, they studied how to find a good starting point to maximize the dual function. So a direct comparison to the work of Beltran [11] is not possible.

- Our work differs from Beasley's [15, 5] on the following aspects. Our problem formulation for k-median problem is different. We formulate the constraints indicating a client j can be assigned to facility i only if that facility is open by using, $x_{ij} \leq y_i, \forall i \in F, j \in D$ whereas Beasley uses $\sum_{j \in V, i \neq j} a_{ij} x_{ij} \leq n_i x_{ii}, \forall i \in V$, where $n_i = \sum_{j \in V, i \neq j} a_{ij}, \forall i \in V$. The constraints set we select to relax are also different from them. We relax the equality constraints (3.2) and constraints (3.3) while keeping a weaker form of (3.2) but for LR1 they relaxed only equality constraint (3.8) and for LR2 constraint (3.9) is relaxed. Beltran et al. [11] relaxed only the equality constraint in their semi-Lagrangean formulation. The solution strategy they used is also different as the subproblem is different. In [15], Beasley compared two different Lagrangean relaxations (LR1 and LR2) by developing penalty tests for reducing the problem size and by incorporating the lower bounds in a tree search procedure. In the

enhanced LR1 algorithm [5] if, at any stage of the tree search, they find an improved feasible solution then they restart the problem from scratch with a new initial tree node.

- In the standard Lagrangean relaxation, we relax the linear equality constraints and solve a dual problem. On the other hand, in semi-Lagrangean relaxation, we relax the equality constraint but keep a weaker form of that equality constraint in the subproblem. Thus the semi-Lagrangean relaxation is more constrained and stronger than standard Lagrangean formulation and that is the main advantage of using semi-Lagrangean relaxation.

Chapter 4

Data placement problem

This chapter discusses in detail the data placement problem. We examine a local search method and a Lagrangean relaxation. At first, a formulation is presented in Section 4.1. The related research is described in Section 4.2. Computation of upper bound and lower bound is discussed in Sections 4.3 and 4.4 respectively. Finally, the experiments and the results are described in Section 4.5.

4.1 Problem Definition

Let us consider a distributed network of caches that have some storage capacity and a set of clients who need to access certain data objects from those caches. A way to improve the performance of such a network is cooperative caching. This cooperation can reduce the average access cost and can improve the space utilization. A vital problem in such a cooperative system is to determine the placement of the data objects to caches such that the average access cost is minimized. The goal is to place data objects in fixed capacity caches in a network to optimize the storage and the access cost where each client has a demand for a specific object. Thus, we can consider the following mathematical formulation of the problem.

Given a set of caches F, a set of data objects O and a set of clients D. Each cache $i \in F$ has a capacity u_i, that limits the total number of data objects that can be stored in the cache. Each client $j \in D$ has demand d_j for a specific data object $o(j) \in O$ and has to be assigned to a cache that stores the object. Storing an object o in cache i incurs a storage cost of f_i^o,

and assigning client j to cache i incurs an access cost of $d_j c_{ij}$ proportional to the distance c_{ij} between i and j. The goal is to place the data objects in caches that satisfies the cache capacities, and compute an assignment of clients to caches, so as to minimize the total storage and client access cost [3].

More precisely, we want to determine a set of objects $O(i) \subseteq O$ to place in each cache $i \in F$ satisfying $\mid O(i) \mid \leq u_i$, and assign each client j to a cache $i(j)$ that stores object $o(j), (i.e., o(j) \in O(i(j)))$ so as to minimize $\sum_{i \in F} \sum_{o \in O(i)} f_i^o + \sum_{j \in D} d_j c_{i(j)j}$.

Let us define two binary variables x_{ij} and y_i^o as:

$$x_{ij} = \begin{cases} 1 & \text{if client } j \text{ is assigned to cache } i \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_i^o = \begin{cases} 1 & \text{if object } o \text{ is stored in cache } i \\ 0 & \text{Otherwise.} \end{cases}$$

The integer linear program (ILP) for the data-placement can be written as:

$$minimize \sum_{i \in F} \sum_{o \in O} f_i^o y_i^o + \sum_{i \in F} \sum_{j \in D} d_j c_{ij} x_{ij} \tag{4.1}$$

$$s.t. \sum_{i \in F} x_{ij} = 1, \qquad \forall j \in D \tag{4.2}$$

$$x_{ij} \leq y_i^{o(j)}, \qquad \forall i \in F, j \in D \tag{4.3}$$

$$\sum_{o \in O} y_i^o \leq u_i, \qquad \forall i \in F \tag{4.4}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in F, j \in D \tag{4.5}$$

$$y_i^o \in \{0,1\} \qquad \forall i \in F, o \in O \tag{4.6}$$

Equation (4.1) is the objective function to minimize the total storage and client assignment cost. The first constraint (4.2) indicates that each client must be assigned to exactly one cache. Equation (4.3) states that a client j can be assigned to a cache only if the cache contains the object $o(j)$. Equation (4.4) is the capacity constraint. Equations (4.5) and (4.6) are the integrality constraints.

4.2 Related research

The problem of data management in a distributed network has been extensively studied. Bartal et al. [4] studied the competitive analysis of algorithms for data management in a distributed environment . They examined competitive algorithms to minimize the communication cost over arbitrary sequences of reads and writes and also defined the constrained file allocation problem to be the solution to many individual file allocation problems. Various studies have incorporated routing information into the caching problem. Maggs et al. [38] considered the problem of placing and accessing shared objects that are read and written from the nodes in a network. They introduced new static and dynamic data management strategies for tree connected networks and for the Internet. In 2001, Meyerson et al. [40] considered the natural theoretical problems of assigning pages to caches and determining the optimal cache locations on the web. They considered a generalization of the data placement problem called the page placement problem where each cache has a client capacity that limits the number of clients that can be assigned to a cache. They gave a constant factor approximation algorithm, but with a logarithmic violation of the client capacity and the object capacity. Shmoys et al. [48] introduced a problem closely related to the data placement problem which is motivated by applications in facility location. In their study, each facility i has a opening cost f_i and service installation cost f_i^l for every facility-service pair (i,l). They provided a primal-dual 6 factor approximation algorithm under the assumption that, if i comes before i' in ordering then for every service type l, $f_i^l \leq f_{i'}^l$. In 2004, Ravi and Sinha [45] proposed approximation algorithms for multicommodity facility location (MCFL) problem which is an extension of the facility location problem where different clients have demand for different goods. They proved the hardness of the multicommodity facility location by reducing from the set cover problem. They provided an $O(\log t)$ approximation algorithm for the t-MCFL problem where the maximum number of allowable commodities in any facility's configuration t equals the total number of commodities k. Qiu et al. [44] studied the problem of web server replica placement. They developed several

placement algorithms that use workload information to determine the placement decisions. Data placement problem is a generalization of uncapacitated facility location (UFL) problem. There is a vast amount of literature that deals with designing approximation algorithms for UFL. The first constant factor approximation for UFL was obtained via LP rounding by Shmoys, Tardos and Aardal [49]. Closely related, k-median problem is discussed in chapter 3. In 2008, Baev et al. [3] presented a 10-approximation algorithm for the data placement problem. Their algorithm is based on rounding an optimal solution to a natural LP relaxation. Data placement problem is also related to the red-blue median problem [27] which is a generalization of the k-median problem [15]. In the red-blue median problem, we are given a set of red facilities, a set of blue facilities, a set of clients in a metric space and two integers k_r, $k_b \geq 0$. The problem is to open at most k_r red facilities and at most k_b blue facilities so as to minimize the sum of distances from the clients to their respective closest open facilities [27]. The data placement problem can be motivated from a red-blue median perspective. In the data placement problem, let us assume that there are only two objects o_1 and o_2 in the object set O. We can consider o_1 as red facility and o_2 as blue facility. Each client requests either object o_1 or object o_2. The problem is to place objects in caches and assign a client j to its nearest cache that contains the requisite object. We assume that the placement cost is zero. If the number of clients requesting red facility (o_1) is k_r and number of clients requesting blue facility (o_2) is k_b, then we can relate the data placement problem with the red blue median problem. Krishnaswamy et al. [34] studied a generalization of budgeted red-blue median known as matroid median, where a matroid structure is given over the set of facilities and we can only open a set of facilities if they form an independent set in the matroid. They obtain a constant-factor approximation for matroid median by rounding an LP relaxation. The study of budgeted red-blue median from the perspective of approximation algorithms was initiated by Hajiaghayi, Khandekar, and Kortsarz [27]. In 2016, Zachary and Yifeng [22] showed that a multiple-swap local search heuristic gives a $(5+\varepsilon)$-approximation for budgeted red-blue median for any constant $\varepsilon > 0$.

4.3 Computing the upper bound

In this section, we describe a local search heuristic we use to compute the upper bound for the data placement problem. Motivation for this is the success of local search for the budgeted red-blue median problem [22].

4.3.1 Local search method

The data placement problem is a generalization of the uncapacitated facility location problem. In the facility location problem, we are given a set of facilities with facility opening cost and a set of clients. We want to open some facilities and assign clients to open facilities such that the sum of facility opening cost and client assignment cost is minimized. Local search heuristics can be used to solve hard combinatorial problems. They start with a feasible solution and try to improve the solution repeatedly by applying some local changes. We use the following local search for UFL as a subroutine in our local search. The local moves for UFL we use are: *add*, *delete* and *swap*. If, S is the current set of facilities we can define the local moves as follows.

i) *Add move* : Opens one additional facility. Add operation is defined as : $S = S + s'$, where $s' \notin S$. We add a new facility s' which is currently not in S.

ii) *Delete move* : Closes one facility that is currently open. This operation is defined as: $S = S - s$, where $s \in S$. We close the facility s which is currently open.

iii) *Swap move* : Opens one new facility and closes a facility that is currently open simultaneously. Swap operation is defined as: $S = S - s + s'$, where $s' \notin S$ and $s \in S$. We close an open facility s, and at the same time we open s' which is currently closed.

For the data placement problem, we can consider caches as facilities, objects as the services installed in those facilities. A restriction that is imposed in the data placement problem is on the capacity of caches which is the number of services that can be installed in a facility. The data placement problem for a single object is an uncapacitated facility location (UFL)

instance. Thus data placement problem is a generalization of the UFL problem. In our approach for computing an upper bound to the data placement problem, we generate some UFL instances as a subproblem and solve those instances by using the local search heuristic for the UFL problem (Algorithm 8). Algorithm 7 describes the local search heuristic we use to compute the upper bound for the data placement problem.

Algorithm 7: *Data_placement_localSearch*

Input : Set of caches F, Set of clients D, Set of requested objects $o : D \rightarrow O$ where o is a function from clients to objects, Cost matrix C, Demand $d : D \rightarrow N$

Output: Minimum total cost.

```
1  Place all the objects, randomly in different caches.
2  Assign all the clients to the nearest cache with the requested object. η : D → F.
3  Calculate the assignment cost, C_total = Σ_{i∈F} Σ_{j∈D} d_j c_ij given η.
4  while true do
5      foreach object l do
6          pick l and remove it from all the caches.
7          no_facility ← number of available empty caches.
8          clients ← set of clients requesting object l.
9          η_new ← UFL_localSearch(no_facility, clients, C, d)
10         Calculate the new assignment cost, C_new from mapping η_new.
11     end
12     if η = η_new then
13         break.
14     η ← η_new only if the cost is improving.
15 end
16 return C_new.
```

At first, we open only the requested objects randomly in different cache locations and assign clients to their nearest cache that contains the requisite object. Mapping η gives us information about which client is assigned to which cache. Using the mapping η, we calculate the total assignment cost C_{total}. From the list of requested objects $r_{obj} \subseteq O$, we pick an object l and remove it from all possible cache locations. Now, we consider empty cache locations as number of facilities and the clients requesting object l as the total number of clients and consider this subproblem as a UFL instance. Then we pass this information to the local search procedure for solving an uncapacitated facility location problem described

in Algorithm 8.

Algorithm 8: *UFL_localSearch*

Input : Set of facilities K, Set of clients L, Cost matrix C, Demand d.
Output: Mapping $\eta : L \rightarrow K$.

$S \leftarrow$ an arbitrary feasible solution.
Assign all the clients to the nearest open facility.
Calculate the total assignment cost $Cost(S) = \sum_{i \in S} \sum_{j:\eta(i)=j} d_j c_{ij}$.
while *true* **do**
  if $\exists\ op(S) = S + s'$, *such that* $s' \notin S$ & $Cost(op(S)) < Cost(S)$ **then**
    $S \leftarrow op(S)$.
    $Cost(S) = Cost(op(S))$.
  else if $\exists\ op(S) = S - s$, *such that* $s \in S$ & $Cost(op(S)) < Cost(S)$ **then**
    $S \leftarrow op(S)$.
    $Cost(S) = Cost(op(S))$.
  else if $\exists\ op(S) = S - s + s'$, *such that* $s \in S, s' \notin S$ &$Cost(op(S)) < Cost(S)$ **then**
    $S \leftarrow op(S)$.
    $Cost(S) = Cost(op(S))$.
  else
    Compute $\eta : L \rightarrow S$.
    return η.
  end
end

In this algorithm, the input is a set of facilities K, a set of clients L, the cost of assigning clients to the facilities and demand d for each client. We start with a feasible set of facilities S and assign all clients to their nearest open facility. Then we calculate the assignment cost. We check if addition of a new facility to set S reduces the total assignment cost. If it does, we perform the local move *add*. If not, we check whether deleting a facility from S reduces the cost. If the *delete* operation does not help to reduce the cost, we check the *swap* operation. In this way, we continue to perform the local changes until there is an improvement in cost. Finally, we return the assignment of clients to the facilities to the data placement problem. This UFL local search gives us a sub-optimal placement of objects into caches. We perform this local search for all objects in set r_{obj} to solve the UFL instances as in Algorithm 7. After performing local improvements (if any) for all the objects in set r_{obj}, we get a new mapping η_{new} and from this mapping we calculate the assignment cost

by $\sum_{i \in F} \sum_{j \in D} d_j c_{ij}$. If the new mapping η_{new} is exactly same as the previous mapping η, we stop the local search. Otherwise, we iterate the whole process again.

According to Williamson and Shmoys [54], if the cost of the solution in the natural local search for UFL improves by one with each local move, then the algorithm could take time exponential in size of the input. If we want to speed up the algorithm rather than just requiring the decrease in cost, we can consider the cost decrease in each iteration by some factor. Let, $\varepsilon > 0$ be a constant, $n = \mid F \mid$ be the number of facilities, $m = \mid D \mid$ be the number of clients and $p(n,m)$ is a polynomial in n and m. According to Arya et al. [2], any operation op in the local search is considered admissible for S if $cost(op(S)) \leq (1 - \varepsilon/p(m,n))cost(S)$. Then there will be at most a polynomial number of operations to be checked for admissibility and during each admissible operation, the cost of the current solution will also decrease by a factor of at least $\varepsilon/p(m,n)$. Then the algorithm terminates in polynomial time.

In 2004, Arya et al. [2] studied the local search heuristics for facility location problems. They used a careful "coupling" argument to show that local optima had cost at most constant times the global optimum. In most of the local search proofs, they showed that since a carefully chosen set of local moves are non-improving, we can assume some relationship between our cost and the optimal cost. In such case, the set of local moves has to be carefully defined.

4.3.2 A simpler analysis of local search method for the uncapacitated facility location problem

In 2008, Gupta and Tangwongsan [26] gave a simpler analysis of the local search algorithm (Algorithm 8) for the uncapacitated faciility location problem. we describe their proof in this section. Table 4.1 describes the notations used in the analysis.

Lemma 4.1. *(Connection Cost): At a local optimum, the fact that "open new facility"*

Table 4.1: Notations used in the analysis of local search method for the uncapacitated facility location problem

Symbols	Description
F	Local optimal set of facilities.
F^*	Optimal set of facilities.
C	Set of clients.
$fac(f)$	Opening cost for facility f.
$fac(F)$	$\sum_{f\in F} fac(f)$.
O_j	Connection cost in an optimal solution for client j.
A_j	Connection cost in local-optimal solution for client j.
$N(f)$	Set of clients assigned to facility f in a local optimum solution.
$N^*(f^*)$	Set of clients assigned to facility f^* in an optimal solution..
φ	$\varphi : V \to F$, mapping client $j \in V$ to closest open facility $f \in F$.
φ^*	$\varphi^* : V \to F^*$, mapping client $j \in V$ to closest open facility $f^* \in F^*$.
$\eta : F^* \to F$	Maps each optimal facility f^* to a closest facility $\eta(f^*) \in F$ such that $d(f^*, \eta(f^*)) \leq d(f^*, f)$ for all $f \in F$.
opt	$fac(F^*) + \sum_{j\in C} O_j$
alg	$fac(F) + \sum_{j\in C} A_j$

moves are non-improving implies that the connection cost:

$$\sum_{j\in C} A_j \leq fac(F^*) + \sum_{j\in C} O_j. \tag{4.7}$$

Proof. At a local optimum, if we open each facility in F^* then the change in the cost is non-negative for opening each $f \in F^*$ that is not open. Here the change in cost is:

$$fac(f^*) + \sum_{j|\varphi^*(j)=f^*} (O_j - A_j) \geq 0 \tag{4.8}$$

Summing up (4.8) over $f^* \in F^*$ and C we get,

$$fac(F^*) + \sum_{j\in C} O_j - \sum_{j\in C} A_j \geq 0 \tag{4.9}$$

$$\sum_{j\in C} A_j \leq fac(F^*) + \sum_{j\in C} O_j \tag{4.10}$$

□

Lemma 4.2. *(Facility Cost): The facility opening cost* $fac(F) \leq fac(F^*) + 2\sum_{j \in C} O_j$ *if* F *is locally optimal.*

Proof. A facility $f \in F$ is called good if $\eta^{-1}(f) = \emptyset$ (it has in-degree 0) and bad otherwise.

Case 1: If facility f is good, we can consider closing the facility and re-assigning the clients assigned to f to $\hat{f} = \eta(\phi^*(j))$. This is a valid assignment because $\hat{f} \neq f$.

The reassignment cost is calculated as shown in Figure 4.1. Suppose currently the client j

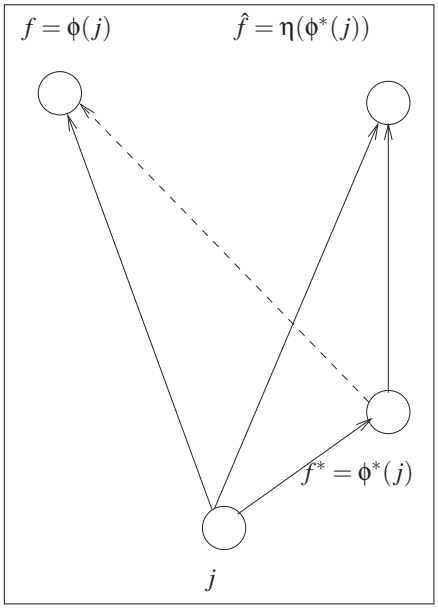

Figure 4.1: Assignment of client to facility

is assigned to facility $\varphi(j)$. If we optimally assign j, it would have been assigned to $\varphi^*(j)$ which is not open. Let us assign j to $\eta(\varphi^*(j))$ which is the closest facility to the optimal one $\varphi^*(j)$. By the triangle inequality, $d(j,\hat{f}) \leq d(j,f^*) + d(f^*,\hat{f})$.

Now subtracting $d(j,f)$ from both sides of the equation, we get

$d(j,\hat{f}) - d(j,f) \leq d(j,f^*) + d(f^*,\hat{f}) - d(j,f)$

Since $\hat{f}$ is the closest facility to f^*, so $d(f^*,f) \geq d(f^*,\hat{f})$. We get,

$d(j,\hat{f}) - d(j,f) \leq d(j,f^*) + d(f^*,f) - d(j,f)$

By the triangle inequality, $d(f^*,f) \leq d(j,f^*) + d(j,f)$. So we get,

$d(j,\hat{f}) - d(j,f) \leq d(j,f^*) + d(j,f^*)$

$d(j,\hat{f}) - d(j,f) \leq 2d(j,f^*)$. Thus,

$$d(j,\hat{f}) - d(j,f) \leq 2O_j \tag{4.11}$$

For all good facilities $f \in F$, we close f and open $\hat{f}$. The change in the cost is non-negative because it is local optimum,
$-fac(f) + d(j,\hat{f}) - d(j,f) \geq 0$. From equation (4.11) we get,

$$-fac(f) + \sum_{j \in N(f)} 2O_j \geq 0 \tag{4.12}$$

Case 2: Facility f is bad. Let P_f^* be the set $\eta^{-1}(f) = \{f_0^*, f_1^*, ..., f_t^*\}$ where $t \geq 0$, and let f_0^* be the closest one to f. We then consider t possible moves of opening facility $f_i^* \in P_f^* \backslash \{f_0^*\}$ and assign any client $j \in N^*(f_i^*) \cap N(f)$ to f_i^*. The local optimality ensures that,

$$fac(f_i^*) + \sum_{j \in N^*(f_i^*) \cap N(f)} (O_j - A_j) \geq 0 \tag{4.13}$$

Moreover, consider the move of opening f_0^* and closing f:

• Any client $j \in N(f)$ with $\varphi^*(j) \notin P_f^*$ is assigned to a facility $\eta(\varphi^*(j)) \neq f$. This is a valid assignment because $\eta(\varphi^*(j)) \neq f$ $(\varphi^*(j) \notin P_f^*)$. Equation (4.11) implies that the increase in the connection cost for such j is at most $2O_j$.

• Any client $j \in N(f)$ with $\varphi^*(j) = f_i^* \in P_f^*$ for some $(i \in \{0,1,...,t\})$ is assigned to f_0^*. Now the change in the connection cost is $d(j,f_0^*) - d(j,f)$. The local optimality implies,
$fac(f_0^*) - fac(f) + \sum_{j \in N(f) \wedge \varphi^*(j) \notin P_f^*} 2O_j + \sum_{i=0}^{t} \sum_{j \in N(f) \cap N^*(f_i^*)} (d(j,f_0^*) - d(j,f)) \geq 0.$

$$fac(f_0^*) - fac(f) + \sum_{j \in N(f) \wedge \varphi^*(j) \notin P_f^*} 2O_j + \sum_{i=0}^{t} \sum_{j \in N(f) \cap N^*(f_i^*)} (d(j,f_0^*) - A_j) \geq 0 \tag{4.14}$$

Summing (4.14) over the t inequalities (one for each $i \in \{0,1,...,t\}$) we get,
$fac(f_0^*) + fac(f_i^*) - fac(f) + \sum_{j \in N(f) \wedge \varphi^*(j) \notin P_f^*} 2O_j + \sum_{i=0}^{t} \sum_{j \in N(f) \cap N^*(f_i^*)} (d(j,f_0^*) + O_j -$

$2A_j) \geq 0$.

From the fact $fac(f_0^*) + fac(f_i^*) = fac(P_f^*)$ we get,

$$fac(P_f^*) - fac(f) + \sum_{j \in N(f) \wedge \varphi^*(j) \notin P_f^*} 2O_j + \sum_{i=0}^{t} \sum_{j \in N(f) \cap N^*(f_i^*)} (d(j, f_0^*) + O_j - 2A_j) \geq 0 \tag{4.15}$$

Consider the rightmost sum, $\sum_{i=0}^{t} \sum_{j \in N(f) \cap N^*(f_i^*)} (d(j, f_0^*) + O_j - 2A_j)$,

when $i = 0$, we get $O_j + O_j - 2A_j = 2(O_j - A_j) \leq 2O_j$.

when $i \neq 0$, the rightmost sum is $d(j, f_0^*) + O_j - 2A_j$.

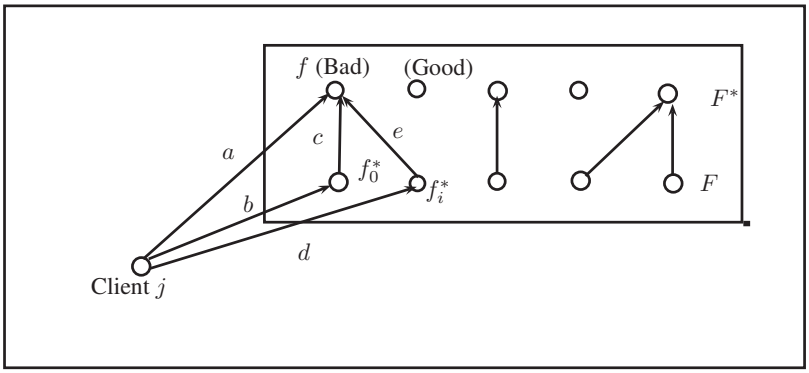

Figure 4.2: Assignment of client to facility

Consider Figure 4.2, from triangle inequality, $b \leq a + c$.

Subtracting $(d - 2a)$ from both sides we get, $b + (d - 2a) \leq a + c + (d - 2a)$ or,

$b + (d - 2a) \leq d + c - a$.

We know, f_0^* is the closest one to f, so $c \leq e$. Replacing c by e we get, $b + (d - 2a) \leq$

$d + e - a$.

Again by triangle inequality we get, $e \leq a + d$ or $e - a \leq d$. So we get,

$b + (d - 2a) \leq 2d$ or $b + (d - 2a) \leq 2O_j$. or,

$d(j, f_0^*) + d(j, f_i^*) - 2d(j, f) \leq 2O_j$.

$d(j, f_0^*) + O_j - 2A_j \leq 2O_j$.

Equation (4.15) can be written as,

$$fac(P_f^*) - fac(f) + \sum_{j \in N(f)} 2O_j \geq 0 \tag{4.16}$$

Summing up equation (4.12) over all good facilities $f \in F$ we get,

$$- fac(F) + \sum_{j \in N(f)} 2O_j \tag{4.17}$$

Summing equation (4.16) over all bad facilities $f \in F$ we get,

$$fac(F^*) + \sum_{j \in N(f)} 2O_j \geq 0 \tag{4.18}$$

From (4.17) and (4.18) we get,

$fac(F^*) - fac(F) + \sum_{j \in C} 2O_j \geq 0$

or $fac(F) \leq fac(F^*) + \sum_{j \in C} 2O_j$

□

Theorem 4.3. *At a local optimum, $UFL(F) \leq 3\ UFL(F^*)$.*

Proof. From lemmas 4.1 and 4.2 we get,

$\sum_{j \in C} A_j + fac(F) \leq fac(F^*) + \sum_{j \in C} O_j + fac(F^*) + 2.\sum_{j \in C} O_j$

$UFL(F) \leq 2fac(F^*) + 3.\sum_{j \in C} O_j$

$UFL(F) \leq 3(fac(F^*) + \sum_{j \in C} O_j)$

$UFL(F) \leq 3\ UFL(F^*)$

Thus, for an uncapacitated facility location instance, the cost of any local optimal solution is at most three times the cost of a global optimal solution.

□

4.4 Computing the lower bound

In this Section, we describe the Lagrangean relaxations we use to compute a lower bound on the cost of the optimal solution for the data placement problem. At first, we describe two Lagrangean formulations of the problem by selecting a different set of constraints to relax. Then we discuss the algorithms we use to compute the lower bound.
In our study of the Lagrangean relaxation method, we consider the restriction that the placement cost is zero, which means $f_i^o = 0, \forall i \in F$ and $\forall o \in O$. Thus the linear program of the data placement problem with zero placement cost is:

$$minimize \sum_{i \in F} \sum_{j \in D} d_j c_{ij} x_{ij} \tag{4.19}$$

$$s.t. \sum_{i \in F} x_{ij} = 1, \qquad \forall j \in D \tag{4.20}$$

$$x_{ij} \leq y_i^{o(j)}, \qquad \forall i \in F, j \in D \tag{4.21}$$

$$\sum_{o \in O} y_i^o \leq u_i, \qquad \forall i \in F \tag{4.22}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in F, j \in D \tag{4.23}$$

$$y_i^o \in \{0,1\} \qquad \forall i \in F, o \in O \tag{4.24}$$

4.4.1 Lagrangean relaxation 1 (LR1-DP)

In our first Lagrangean relaxation, we relax the equality constraint (4.20) as well as constraint (4.21). Thus we obtain the following problem:

$$min \sum_{i \in F} \sum_{j \in D} d_j c_{ij} x_{ij} + \sum_{j \in D} \lambda_j (1 - \sum_{i \in F} x_{ij}) + \sum_{j \in D} u_j (\sum_{i \in F} x_{ij} - 1) + \sum_{i \in F} \sum_{j \in D} v_{ij} (x_{ij} - y_i^{o(j)}) \tag{4.25}$$

$$s.t. \sum_{o \in O} y_i^o \leq u_i, \qquad \forall i \in F \tag{4.26}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in F, j \in D \tag{4.27}$$

$$y_i^o \in \{0,1\} \qquad \forall i \in F, o \in O \tag{4.28}$$

In the objective function (4.25), λ, u and v are non-negative Lagrange multipliers. Multipliers λ and u are associated with the equality constraints. Multiplier v is attached with constraints (4.21). We rewrite the objective function (4.25) as:

$$min \sum_{i \in F} \sum_{j \in D} x_{ij}(d_j c_{ij} - \lambda_j + u_j + v_{ij}) - \sum_{i \in F} \sum_{j \in D} v_{ij} y_i^{o(j)} + \sum_{j \in D} (\lambda_j - u_j) \tag{4.29}$$

Now the goal is to minimize this objective function subject to (4.26) - (4.28). We start with a non-negative set of multipliers and calculate the first term in (4.29). We set $x_{ij} = 1$ if the coefficient for the first term in (4.29) is non-positive. Since this is a minimization problem, we want the negative terms in this equation to be as large as possible. In order to maximize the negative sum (second term), we solve the following ILP:

$$maximize \sum_{i \in F} \sum_{j \in D} v_{ij} y_i^{o(j)} \tag{4.30}$$

$$s.t. \sum_{o \in O} y_i^o \leq u_i, \qquad \forall i \in F \tag{4.31}$$

$$y_i^o \in \{0,1\} \qquad \forall i \in F, o \in O \tag{4.32}$$

Subgradient optimization

In order to maximize the lower bound we use an iterative procedure. We start with non-negative set of multipliers and update the value of multipliers in an efficient way. We use the notation from table 2.1 to describe the subgradient procedure. In Algorithm 9, we initialize $\pi = 2$, λ, $u = \min_i c_{ij} \quad \forall j \in D$, $v = 0$ and $Z_{max} = -\infty$. We obtain an upper bound on the data placement problem by solving either an integer linear program (where possible) or by using the local search method described in Algorithm 7. To calculate the lower bound, we evaluate the term $(d_j c_{ij} - \lambda_j + u_j + v_{ij})$ in equation (4.29) and if the value is less than or equal to 0 then we set $x_{ij} = 1$. We obtain the maximum value for the second negative term in equation (4.29) by solving the sub problem described by the ILP in (4.30)-(4.32).

Algorithm 9: Subgradient optimization procedure [6]

Input : A Lagrangean lower bound program.
Output: Maximum lower bound Z_{max}.

1 Initialize the values of : π, λ, u, v and Z_{max}.
2 Find an upper bound Z_{UB} by solving the data placement problem using Integer linear program or by the local search method described in Algorithm 7 .
3 Compute the Lagrangean lower bound using the initial set of multipliers and by solving the sub-problem defined by the ILP in (4.30)-(4.32). Get a solution Z_{LB}, which is the lower bound of the problem.
4 Update the value of $Z_{max} = max(Z_{max}, Z_{LB})$
5 Evaluate the subgradient value G.
6 Define the step size $T = \frac{\pi * (Z_{UB} - Z_{LB})}{|G|^2}$.
7 Update the lagrange multiplier values and go to step (3) to re-calculate the value of lower bound with new set of multipliers.
8 Repeat this process for a fixed number of iterations.

Finally, after computing the term $\sum_{j \in D} (\lambda_j - u_j)$, we calculate Z_{LB} which is the lower bound for the problem. Next, we update the value of Z_{max}, which is the best lower bound found so far. Next, we determine the subgradient value $G = [G_1\ G_2\ G_3]$ by combining the following three subgradient vectors :

$$G_1 = (1 - \sum_{i \in F} x_{ij}), \quad \forall j \in D \tag{4.33}$$

$$G_2 = (\sum_{i \in F} x_{ij} - 1), \quad \forall j \in D \tag{4.34}$$

$$G_3 = x_{ij} - y_i^{o(j)}, \quad \forall i \in F, j \in D \tag{4.35}$$

Using the value of G, we calculate the step-size T. After calculating the step size, we update the multiplier values as :

$$\lambda_j = \max(0, \lambda_j + TG_1(j)) \qquad j = 1, ..., \mid D \mid \tag{4.36}$$

$$u_j = \max(0, u_j + TG_2(j)) \qquad j = 1, ..., \mid D \mid \tag{4.37}$$

$$v_{ij} = \max(0, v_{ij} + TG_3(ij)) \qquad i = 1, ..., \mid F \mid, j = 1, ..., \mid D \mid \tag{4.38}$$

We repeat the subgradient process with the new set of multipliers. Thus we get a new value for the lower bound. If this new lower bound is better than the previous value we update the value of Z_{max} with the new lower bound. We continue for up to 500 iterations.

4.4.2 Lagrangean relaxation 2 (LR2-DP)

In the second Lagrangean relaxation, we consider the following modified data placement problem : constraint (4.20) has been relaxed to an inequality and placement costs are zero.

$$minimize \sum_{i\in F}\sum_{j\in D} d_j c_{ij} x_{ij} \tag{4.39}$$

$$s.t. \sum_{i\in F} x_{ij} \geq 1, \qquad \forall j \in D \tag{4.40}$$

$$x_{ij} \leq y_i^{o(j)}, \qquad \forall i \in F, j \in D \tag{4.41}$$

$$\sum_{o\in O} y_i^o \leq u_i, \qquad \forall i \in F \tag{4.42}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in F, j \in D \tag{4.43}$$

$$y_i^o \in \{0,1\} \qquad \forall i \in F, o \in O \tag{4.44}$$

We relax constraints (4.40), (4.41) and (4.42). Thus we get the following problem:

$$min \sum_{i\in F}\sum_{j\in D} d_j c_{ij} x_{ij} + \sum_{j\in D} \lambda_j (1 - \sum_{i\in F} x_{ij}) + \sum_{i\in F}\sum_{j\in D} v_{ij}(x_{ij} - y_i^{o(j)}) + \sum_{i\in F} b_i (\sum_{o\in O} y_i^o - u_i) \tag{4.45}$$

$$s.t. \quad x_{ij} \in \{0,1\} \qquad \forall i \in F, j \in D \tag{4.46}$$

$$y_i^o \in \{0,1\} \qquad \forall i \in F, o \in O \tag{4.47}$$

In equation (4.45), λ, v and b are non-negative Lagrange multipliers. Multiplier λ is associated with constraints (4.40), v is attached with constraints (4.41), b is associated with

(4.42). Equation (4.45) can be rewritten as:

$$min \sum_{i\in F}\sum_{j\in D} x_{ij}(d_j c_{ij} - \lambda_j + v_{ij}) + \sum_{o\in O}\sum_{i\in F} y_i^o (b_i - \sum_{j\in D|o=o(j)} v_{ij}) - \sum_{i\in F} b_i u_i + \sum_{j\in D} \lambda_j \quad (4.48)$$

Now the goal is to find an integral solution that minimizes this objective function. We start with a non-negative set of multipliers and calculate the first term in the equation. We set $x_{ij} = 1$ if the coefficient is non-positive. Similarly, we calculate the second term in the equation and set $y_i^o = 1$, if the value of the term associated with y_i^o is non-positive.

Subgradient optimization

In order to maximize the lower bound we use subgradient optimization procedure. We start with a non-negative set of multipliers and update the multipliers in a systematic way. To solve the second Lagrangean relaxation, we use the subgradient procedure described in Algorithm 9 except that in step 3, we do not solve any subproblem, because we relaxed all the constraints in this formulation. For this formulation, we initialize $\pi = 2$, $\lambda = \min_i c_{ij} \quad \forall j \in D$, $v = 0$, $b = 0$ and $Z_{max} = -\infty$. We find an upper bound on the problem (4.39), by solving either an integer linear program where possible or by using a local search method described in Algorithm 7. To calculate the lower bound, we evaluate the term $(d_j c_{ij} - \lambda_j + v_{ij})$ in equation (4.48) and if the value is less than or equal to 0 then we set $x_{ij} = 1$. We calculate the term $(b_i - \sum_{j\in D|o=o(j)} v_{ij})$ and set $y_i^o = 1$ if the value is less than or equal to 0. Finally, after computing $\sum_{i\in F} b_i u_i$ and $\sum_{j\in D} \lambda_j$, we calculate Z_{LB} which is the lower bound for the problem. Next, we update the value of Z_{max} which is the best lower bound found so far. In our next step, we determine the subgradient value $G = [G_1\ G_2\ G_3]$ by combining the following three subgradient vectors :

$$G_1 = (1 - \sum_{i\in F} x_{ij}), \quad \forall j \in D \quad (4.49)$$

$$G_2 = (x_{ij} - y_i^{o(j)}), \quad \forall i \in F, j \in D \quad (4.50)$$

$$G_3 = (\sum_{o \in O} y_i^o - u_i), \quad \forall i \in F \tag{4.51}$$

Using the value of G, we calculate the step-size $T = \frac{\pi * (Z_{UB} - Z_{LB})}{|G|^2}$. We update the multipliers as :

$$\lambda_j = max(0, \lambda_j + TG_1(j)) \quad j = 1, ..., \mid D \mid \tag{4.52}$$

$$v_{ij} = max(0, v_{ij} + TG_2(ij)) \quad i = 1, ..., \mid F \mid, j = 1, ..., \mid D \mid \tag{4.53}$$

$$b_i = max(0, b_i + TG_3(i)) \quad i = 1, ..., \mid F \mid \tag{4.54}$$

As earlier, we repeat the subgradient process with the new set of multipliers. Thus we get a new value for the lower bound. If this new lower bound is better than the previous value we update the value of Z_{max} with the new lower bound. We continue for up to 500 iterations.

4.5 Experiments and Results

4.5.1 Generation of test instances

We generated the test instances to compare the upper bound computed using local search and the lower bound computed using Lagrangean relaxation. For generating the cost matrix, we use union-find data structure [43] to create random trees. An edge in the tree represents the connection between a client and a cache. After generating the tree, we assign a cost to each edge which represents the cost of assigning a client to a cache. Union-find is a data structure that keeps track of a set of elements partitioned into some disjoint subsets. Each set (tree) is identified by a representative (root) n, which is usually a member of that set. Every member (node) of a set has an unique identity, a parent, and a rank value. The disjoint set data structure allows for three operations: *MakeSet*, *Find* and *Union* [23].

- *MakeSet*: The *MakeSet* operation creates a singleton set by adding a new member having a unique id, the rank value is set to zero and parent of the element is itself. The pseudocode for *Makeset* operation is in Algorithm 10.

Algorithm 10: $MakeSet(n,S)$

```
1 if n ∉ S then
2 |   S ← S ∪ {n}
3 |   rank(n) ← 0
4 |   parent(n) ← n
```

- *Find*: The *Find* operation follows a chain of parent indices from a node upwards through the tree until it reaches the root (element which is its own parent). This element is the representative member of the set. The pseudocode for *Find* operation is given in Algorithm 11.

Algorithm 11: $Find(n)$

```
1 if parent(n) ≠ n then
2 |   parent(n) ← Find(parent(n))
3 return parent(n)
```

- *Union*: *Union* operation creates a new set by combining two existing sets and changes the parent (root) of one set (tree) to another. In union by rank method, we maintain an integer rank value for each node or element in a set. The set with the higher rank value in the root node becomes the root of the newly merged set. If both sets have the same rank value in the root node, then any one of them can be the new root node, the rank value increased by one. The pseudocode for union by rank is given in Algorithm 12.

Algorithm 13 describes the generation of random trees with a random cost on each edge. We pass the cost matrix to Algorithm 6 (in Chapter 3) to compute the transitive closure. The transitive closure of the cost matrix is computed using the Floyd Warshal algorithm [21]. This ensures that the cost matrix is triangular. For each client, the requested object from the object set is generated uniformly as random and the demand is also generated randomly in the range $[1-10]$.

Algorithm 12: $Union(m,n)$

```
1  m_root ← Find(m)
2  n_root ← Find(n)
3  if m_root = n_root then
4      return
5  if rank(m_root) < rank(n_root) then
6      parent(m_root) ← n_root
7  else if rank(m_root) > rank(n_root) then
8      parent(n_root) ← m_root
9  else
10     parent(n_root) ← m_root
11     rank(m_root) ← rank(m_root) + 1
```

Algorithm 13: Generate random tree

```
1  S ← ϕ
2  Initialize no_nodes in the tree.
3  foreach node n do
4      MakeSet(n, S)
5  while no_edges < no_nodes do
6      m ← random value from 1 − no_nodes
7      n ← random value from 1 − no_nodes
8      if m ≠ n then
9          Union(m, n)
10         no_edges ← no_edges + 1
11 foreach edge of the generated tree do
12     Assign a random cost from range 1 − 100
```

4.5.2 Results

In Table 4.2, we give the upper bound, the time to compute the upper bound, the lower bound, the time to compute the lower bound and the duality gap for different size large instances of data placement problem. The test instances are generated using Algorithm 13. We compute an upper bound by solving the integer linear program of data placement problem where possible or by solving the local search heuristic described in Algorithm 7, as large instances cannot be solved optimally using ILP solvers. The upper bound for the first two test instances in Table 4.2 is computed using ILP and the rest are computed using

local search. We compute the lower bound using the two Lagrangean relaxations studied in Section 4.4.1 (LR1-DP) and 4.4.2 (LR2-DP).

Table 4.2: Experimental results for the data placement problem

Problem Size	UB	Time	LR1-DP	Time	Duality gap	LR2-DP	Time	Duality gap
Cache=100 Client=120	4087	5	3120	501.99	23.66	3036.7	412.98	25.69
Cache=100 Client=150	16265	80	14911	633.45	8.32	14708	522.73	9.57
Cache=200 Client=300	33695	1.07	26554	2509.1	21.19	25707	2988.5	23.7
Cache=300 Client=500	76135	4.57	60080	6305.4	21.08	58206	11500	23.55
Cache=300 Client=600	113805	5.15	91890	7870.5	19.25	89711	14481	21.17
Cache=400 Client=600	72807	11.95	53077	9914.7	27.09	50196	23570	31.05
Cache=500 Client=700	72129	25.64	47652	14435	33.94	43983	51189	39.02
Cache=600 Client=800	61292	33.72	38992	19844	36.38	35199	94498	42.57
Cache=700 Client=1000	104080	58.85	68864	28993	33.83	62390	168550	40.05
Cache=800 Client=1200	146969	80.09	97098	37982	33.93	88525	276460	39.76
Cache=900 Client=1400	170662	130.92	113540	53670	33.47	103870	624610	39.13
Cache=1000 Client=1500	188204	180.21	117150	64267	37.75	106470	901050	43.42

The duality gap [5] is given by the formula:

$$\frac{UB - LB_{max}}{UB} \times 100\% \tag{4.55}$$

Here UB is the upper bound for the problem and LB_{max} is the maximum lower bound for the problem. In Table 4.2, every test instance has a total of two objects and the cache capacity is one for every cache. From the experimental results we can see that the upper bound and the lower bound values are relatively close to each other for large sized instances of the problem. The largest instance tested contains 1000 caches, 1500 clients, 2 objects and cache capacity of one. This problem includes 1502000 variables and 1502500 constraints.

After 500 iterations of subgradient method we get the duality gap 37.75% using Lagrangean relaxation LR1-DP. The time columns in Table 4.2 represents time in seconds. We can compute the upper bound within a very small amount of time. When computing the lower bound, the time increases substantially as the problem size increases.

We implemented the subgradient method (Algorithm 9) for computing Lagrangean lower bound in Octave 4.0.2. The test machine has an Intel Xeon processor with a clock speed of 3.40 GHz and 8GB of RAM running CentOS. The local search approach in Algorithm 7 is implemented in C++. The ILP for data placement problem is implemented using GLPK (GNU Linear Programming Kit) library for Octave.

We varied the number of caches from 50 to 350 and the duality gap is shown in Figure 4.3.

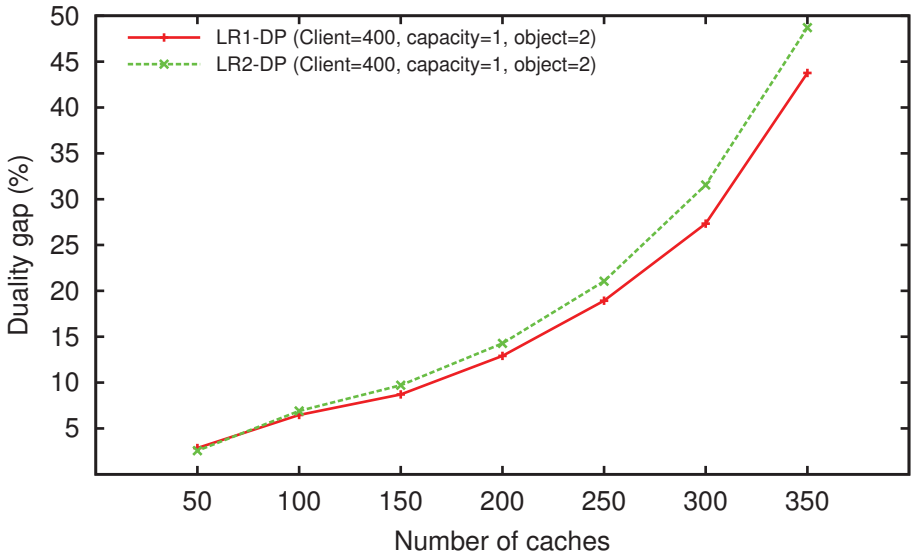

Figure 4.3: A comparison of duality gap for two Lagrangean relaxations while varying the number of caches

There are 400 clients, the capacity of each cache is 1, and there are two objects. We see from the graph that the duality gap increases with the number of caches for both LR1-DP and LR2-DP. We get a lower duality gap for LR1-DP than for LR2-DP. In Figure 4.4, we plot the time needed to solve the instance while we vary the number of caches. Here we see LR2-DP takes more time than LR1-DP.

In Figure 4.5, we vary the cache capacity from 1 to 5. There are 100 caches, 150 clients

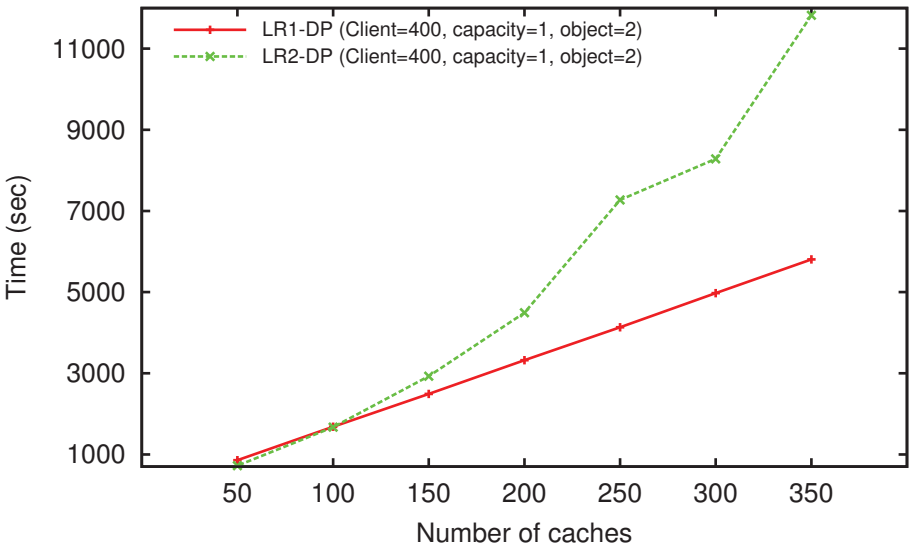

Figure 4.4: A comparison of time for two Lagrangean relaxations while varying the number of caches

and 5 objects. In this case, as we see from the graph, the duality gap decreases as the cache capacity increases. We get a lower duality gap for LR1-DP than for LR2-DP. Figure 4.6 is a plot of the time needed to solve the instance while varying the cache capacity. Here, we see LR2-DP takes less time than LR1-DP.

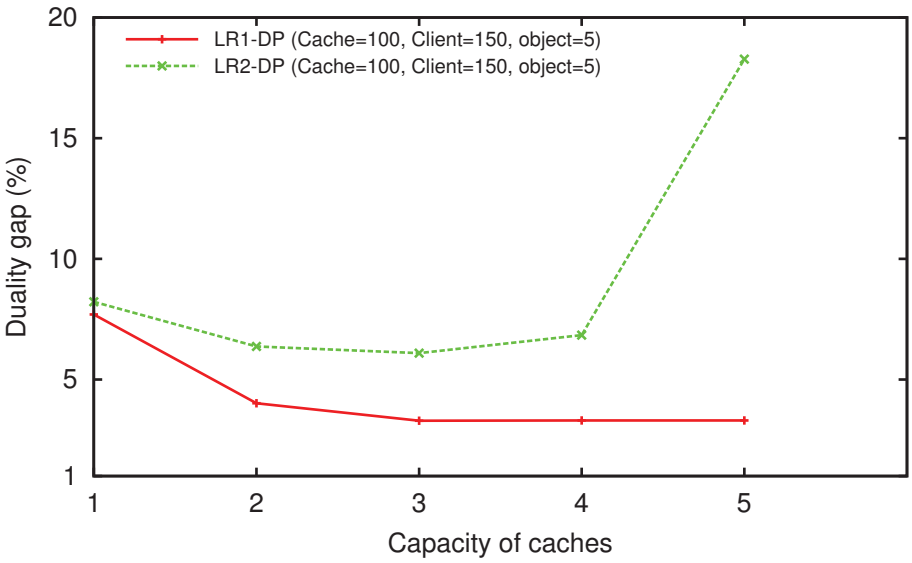

Figure 4.5: A comparison of duality gap for two Lagrangean relaxations while varying the cache capacity

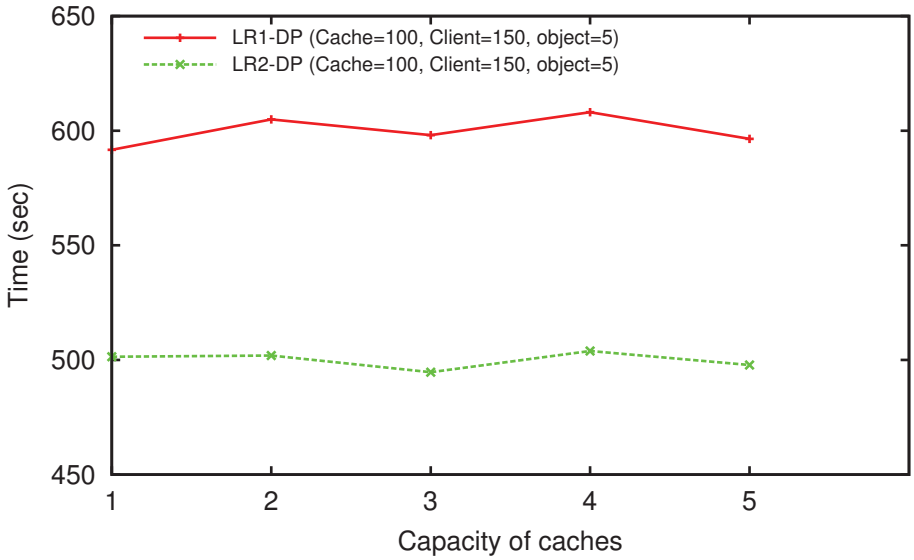

Figure 4.6: A comparison of time for two Lagrangean relaxations while varying the cache capacity

In Figure 4.7, we vary the number of clients from 550 to 950. There are 500 caches, each of capacity one and two objects. We can see from the graph that the duality gap decreases as the number of client increases. We get a lower duality gap for LR1-DP than for LR2-DP. In Figure 4.8, we plot the time needed to solve the instance while varying the

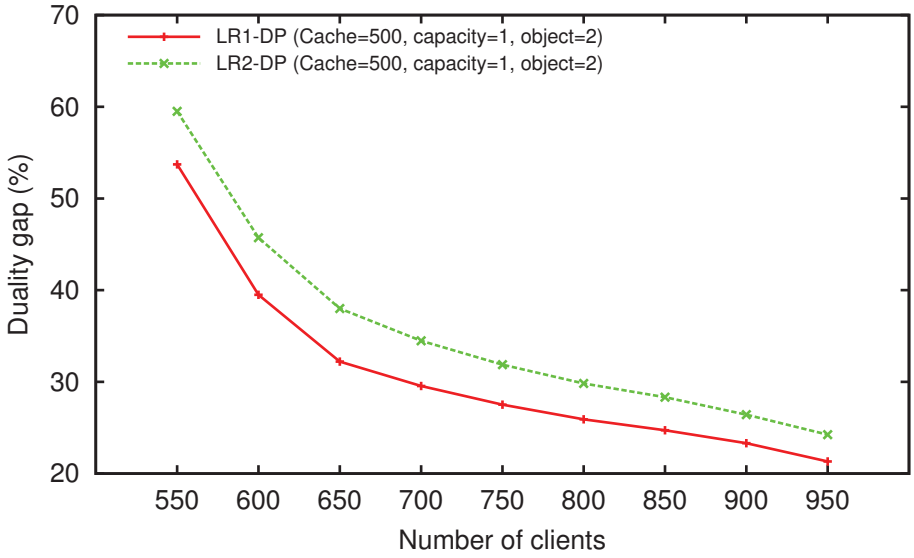

Figure 4.7: A comparison of duality gap for two Lagrangean relaxations while varying the number of clients

number of clients. Here we see LR1-DP takes less time than LR2-DP.

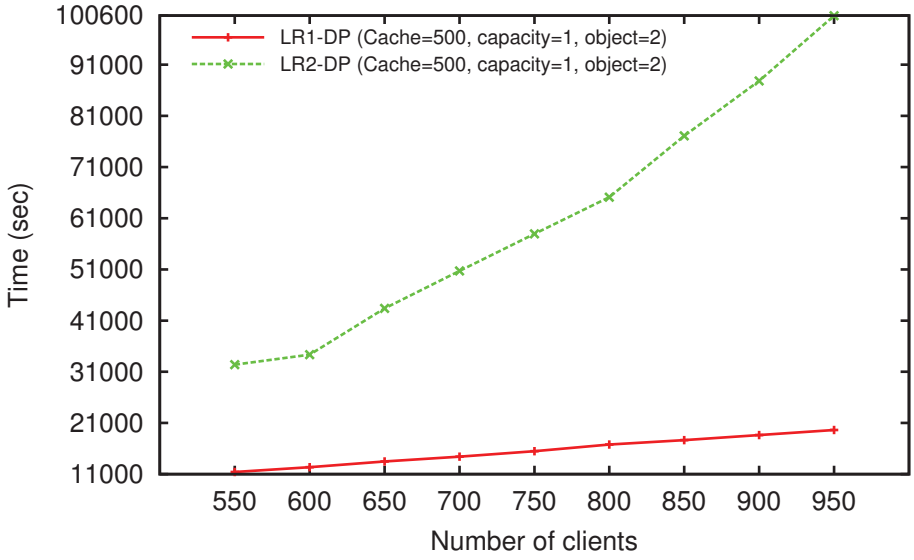

Figure 4.8: A comparison of time for two Lagrangean relaxations while varying the number of clients

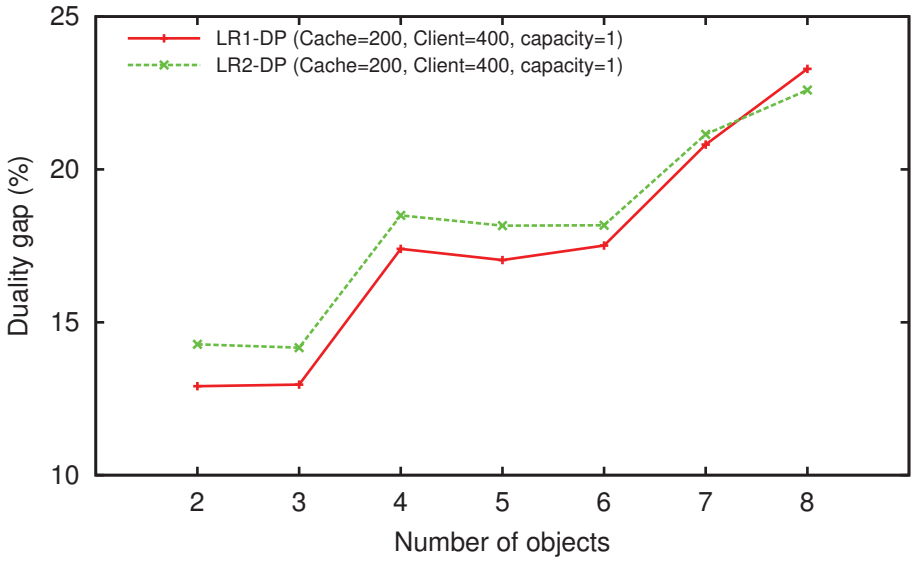

Figure 4.9: A comparison of duality gap for two Lagrangean relaxations while varying the number of objects

In Figure 4.9, we vary the number of objects from 2 to 8 for 200 caches, 400 clients and cache capacity of 1. In this case, we can see from the graph that the duality gap increases as the number of objects increases. We get lower duality gap for LR1-DP than for LR2-DP. In

Figure 4.10, we plot the time needed to solve the same instance while varying the number of objects. Here, LR2-DP takes more time than LR1-DP.

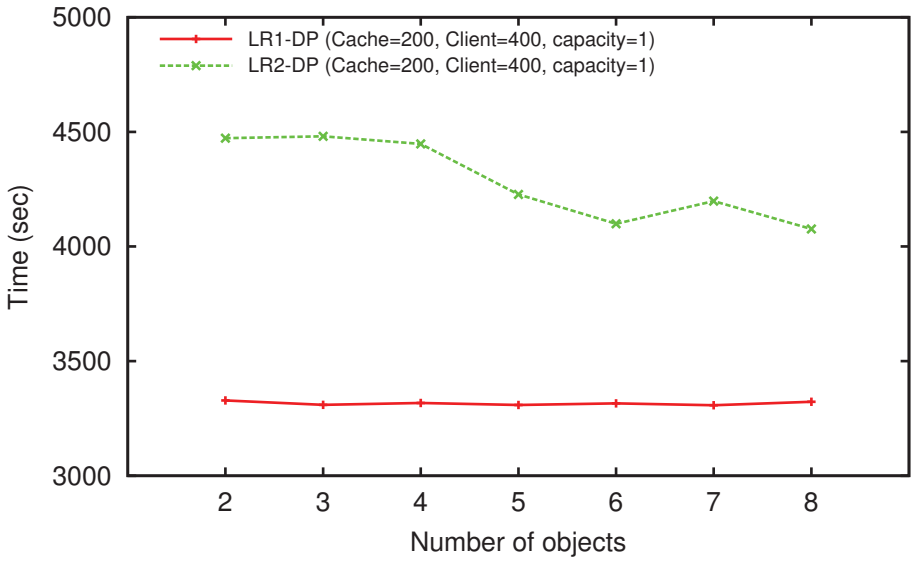

Figure 4.10: A comparison of time for two Lagrangean relaxations while varying the number of objects

4.6 Discussion

- The data placement problem with zero storage cost is an interesting problem to consider because we can relate this problem to the red-blue median problem [27] which is a generalization of the well-studied k-median problem described in Chapter 3. In the red-blue median problem, facilities are partitioned into two sets (red and blue) while clients have no types associated with them and can be served by any open facility. For the data placement problem, if we consider only two objects in the object set $O = \{o_1$ (red), o_2 (blue)$\}$ then clients can have demand for each object type. In this case, a client can only be served by the requisite object. We have a constant factor approximation algorithm for the red-blue median problem based on simple local search heuristic [27] and multiple swap heuristic [22]. A special case of the red blue median problem where all the facilities have the same color ($k_r = 0$ or $k_b = 0$) is a well studied k-median problem.

- From the experimental study, we infer that the upper bound can be computed within a very small amount of time. The largest instance needed only 180 seconds (Table 4.2) to compute an upper bound using the local search method. On the other hand, computing a lower bound using Lagrangean relaxation takes an enormous amount of time as the problem size increases. We see LR1-DP gives a smaller duality gap than LR2-DP for most of the test instances.

- The running time of the subgradient method can be reduced using a parallel implementation and it can be studied further. As a future research direction, our work can be extended to non-zero storage costs and an experimental study would be natural.

Chapter 5

Conclusion and Future works

This chapter summarizes the results in the thesis. We discuss our findings, and also outline directions for future research.

5.1 Summary

In this thesis, we studied a large-scale version of the data placement problem with zero storage cost. We computed an upper bound for the problem using local search heuristic. We also computed a lower bound using two Lagrangean relaxations. We conducted an empirical study for large sized instances of this problem. Our experimental study shows that the upper bound and lower bound values are relatively close to each other. Stated otherwise local search gives very good solutions reasonably fast. Lagrangean relaxations can be used to provide certificates of closeness.

5.2 Future work

There are many possibilities for extending the work in this thesis. It would be interesting to investigate the following aspects of the data placement problem.

- In this thesis, we studied the data placement problem with the restriction on placement costs. Our work can be extended naturally to non-zero storage costs and an experimental study would be natural.

- We use an iterative subgradient optimization method to solve the Lagrangean relaxation. Our implementation of the method is sequential. The running time of the

subgradient method can be reduced using a parallel implementation, and this can be an interesting extension to our work.

Bibliography

[1] E. H. L. Aarts and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 1997.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on computing*, 33(3):544–562, 2004.

[3] I. Baev, R. Rajaraman, and C. Swamy. Approximation algorithms for data placement problems. *SIAM Journal on Computing*, 38(4):1411–1429, 2008.

[4] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995.

[5] J. E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21(2):270–273, 1985.

[6] J. E. Beasley. A lagrangian heuristic for set-covering problems. *Naval Research Logistics (NRL)*, 37(1):151–164, 1990.

[7] J. E. Beasley. "OR-Library". `http://www.http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, 1990.

[8] J. E. Beasley. Lagrangean heuristics for location problems. *European Journal of Operational Research*, 65(3):383–399, 1993.

[9] T. Bektaş, J. F. Cordeau, E. Erkut, and G. Laporte. Exact algorithms for the joint object placement and request routing problem in content distribution networks. *Computers & Operations Research*, 35(12):3860–3884, 2008.

[10] T. Bektas, O. Oguz, and I. Ouveysi. Designing cost-effective content distribution networks. *Computers & Operations Research*, 34(8):2436–2449, 2007.

[11] C. Beltran, C. Tadonki, and J. P. Vial. Solving the p-median problem with a semi-lagrangian relaxation. *Computational Optimization and Applications*, 35(2):239–260, 2006.

[12] C. Beltran, J. P. Vial, and A. Alonso-Ayuso. Semi-lagrangian relaxation applied to the uncapacitated facility location problem. *Computational Optimization and Applications*, 51(1):387–409, 2012.

[13] A. Bumb. *Approximation algorithms for facility location problems*. PhD thesis, Netherlands: University of Twente, 2002.

[14] J. W. Chinneck. Practical optimization: a gentle introduction. *Systems and Computer Engineering), Carleton University*, 2006.

[15] N. Christofides and J. E. Beasley. A tree search algorithm for the p-median problem. *European Journal of Operational Research*, 10(2):196–204, 1982.

[16] H. Crowder, E. L. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[17] G. Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[18] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys (CSUR)*, 14(2):287–313, 1982.

[19] M. Drwal and J. Jozefczyk. Decomposition algorithms for data placement problem based on lagrangian relaxation and randomized rounding. *Annals of Operations Research*, 222(1):261–277, 2014.

[20] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, 27(1):1–18, 1981.

[21] R. W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[22] Z. Friggstad and Y. Zhang. Tight Analysis of a Multiple-Swap Heurstic for Budgeted Red-Blue Median. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 75:1–75:13, Dagstuhl, Germany, 2016.

[23] Z. Galil and G. F. Italiano. Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys (CSUR)*, 23(3):319–344, 1991.

[24] B. Gavish and M. W. Suh. Configuration of fully replicated distributed database system over wide area networks. *Annals of Operations Research*, 36(1):167–191, 1992.

[25] K. Genova and V. Guliashki. Linear integer programming methods and approaches–a survey. *Journal of Cybernetics and Information Technologies*, 11(1), 2011.

[26] A. Gupta and K. Tangwongsan. Simpler analyses of local search algorithms for facility location. *arXiv preprint arXiv:0809.2554*, 2008.

[27] M. Hajiaghayi, R. Khandekar, and G. Kortsarz. Local search algorithms for the red-blue median problem. *Algorithmica*, 63(4):795–814, 2012.

[28] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[29] K. L. Hoffman. Combinatorial optimization: Current successes and directions for the future. *Journal of computational and applied mathematics*, 124(1):341–360, 2000.

[30] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part i, graph partitioning. *Operations research*, 37(6):865–892, 1989.

[31] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310, 1997.

[32] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[33] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Placement algorithms for hierarchical cooperative caching. *Journal of Algorithms*, 38(1):260–302, 2001.

[34] R. Krishnaswamy, A. Kumar, V. Nagarajan, Y. Sabharwal, and B. Saha. The matroid median problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1117–1130. SIAM, 2011.

[35] A. A. Kuehn and M. J. Hamburger. A heuristic program for locating warehouses. *Management science*, 9(4):643–666, 1963.

[36] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[37] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[38] B. M. Maggs, F. M. auf der Heide, B. Vocking, and M. Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 284–293. IEEE, 1997.

[39] S. Martello, D. Pisinger, and P. Toth. New trends in exact algorithms for the 0–1 knapsack problem. *European Journal of Operational Research*, 123(2):325–332, 2000.

[40] A. Meyerson, K. Munagala, and S. Plotkin. Web caching using access statistics. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 354–363. Society for Industrial and Applied Mathematics, 2001.

[41] J. B. Orlin, A. P. Punnen, and A. S. Schulz. Approximate local search in combinatorial optimization. *SIAM Journal on Computing*, 33(5):1201–1214, 2004.

[42] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[43] M. M. A. Patwary, J. Blair, and F. Manne. Experiments on union-find algorithms for the disjoint-set data structure. In *International Symposium on Experimental Algorithms*, pages 411–423. Springer, 2010.

[44] L. Qiu, V. N Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1587–1596. IEEE, 2001.

[45] R. Ravi and A. Sinha. Multicommodity facility location. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 342–349. Society for Industrial and Applied Mathematics, 2004.

[46] G. Reinelt. "tsplib". `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`, 2001.

[47] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[48] D. B. Shmoys, C. Swamy, and R. Levi. Facility location with service installation costs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1088–1097. Society for Industrial and Applied Mathematics, 2004.

[49] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274. ACM, 1997.

[50] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. V. Steen. Replication for web hosting systems. *ACM Computing Surveys (CSUR)*, 36(3):291–334, 2004.

[51] J. K. Strayer. *Linear programming and its applications*. Springer-Verlag New York, 1st edition, 1989.

[52] L. Trevisan. Combinatorial optimization: Exact and approximate algorithms. *Stanford University*, 2011.

[53] R. J. Vanderbei. *Linear programming: Foundations and extensions*. Springer US, USA, 3rd edition, 2008.

[54] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.