UNIVERSITY OF SOUTHAMPTON

FACULTY OF SOCIAL AND HUMAN SCIENCES

School of Mathematics

Models and Algorithms for Assignment and Cache Allocation Problems in Content Distribution Networks

by

Narges Haghi

Thesis for the Degree of Doctor of Philosophy

June 2016

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF SOCIAL AND HUMAN SCIENCES
SCHOOL OF MATHEMATICS

Doctor of Philosophy

Models and Algorithms for Assignment and Cache Allocation Problems in Content Distribution Networks

by Narges Haghi

This thesis considers two difficult combinatorial optimization problems for request routing and client assignment in content distribution networks. The aim is to introduce lower and upper bounds to estimate optimal solutions. Existing solution methods and techniques for similar problems have been reviewed. The first problem consists of minimizing the total network cost for request routing with no origin server by considering the delay function. The second problem is cache allocation problem. Lagrangian relaxation and two perspective cuts are used to linearize first problem and to find valid lower bounds. Two different linearization methods are implemented for cache allocation problem. Iterated Variable Neighborhood Descent and Tabu Search are two solution methods which are suggested to find best upper bounds. Different local search operators are introduced to improve objective function values as follows: swap, remove-insert, insert from origin server to a proxy, insert from one proxy to another proxy, swap between origin server and a proxy, swap between two proxies and cyclic exchange. All computational results are presented on randomly generated instances.

Contents

List of Figures

List of Tables

List of Algorithms

Declaration of Authorship

I, Narges Haghi, declare that the thesis entitled "Models and Algorithms for Assignment and Cache Allocation Problems in Content Distribution Networks" and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- none of this work has been published before submission.

Signed:..

Date:..

Acknowledgements

Thanks to Almighty God for graciously bestowing me the perseverance to undertake this research. I would like to express my deep gratitude and appreciation to my supervisor Professor Chris Potts for his invaluable advice and his enthusiasm and inspiration has always stimulated me to take steps forward. This thesis would not be possible without generous support in all aspects of my graduate study specially from last year. Also, I would like to express my appreciation to my supervisor Dr Güneş Erdoğan.

I would like to thank my family, my parents Ahmad Haghi and Ghodsi Hejazi, for supporting me spiritually throughout my life. My sincere thanks also go to all of my friends for their love, encouragement and support during my studies. A special thank to Paul and Lorraine Abeyta, their prayer for me was what sustained me thus far.

I would like to express appreciation to my beloved husband Farshad for the love, kindness and support he has shown during the past two years, it has taken me to finalize this thesis. Without him, this thesis would not exist. I would like to thank my daughter, Melika to accommodate with her patient with new situation.

To
My Dearest Husband
Farshad
&
My Lovely Daughter
Melika

Chapter 1

Introduction

1.1 Background

Over the past years, the Internet has had an important effect on the way business transactions are carried out. For most businesses, the performance of Web connections has a direct impact on their profit. For example, the lack of network management, in turn, makes it very difficult to guarantee satisfactory proper performance and to deal with related problems. The available network bandwidth and server capacity continue to be exhausted by the increasing number of users and by the accelerating growth of request for bandwidth intensive content.

The delays experienced by end users have important economic consequences, especially in electronic marketing. It is controversial as to what constitutes an acceptable waiting time for a typical Web page download (Bailey and Schaffer, 2001). A typical client is likely to abandon a Web site failing to respond to download requests in approximately 8 seconds, according to the report by Zona marketing bulletin (Zona, 2001), which states that "the amount of time taken for Web pages to load is one of the most critical factors in determining the success of a site and the satisfaction of its users." As a result, quality of Internet services as perceived by customers is becoming unpredictable and unsatisfactory. On the other hand, the result of the study by Nah (2004) suggests that the time that users are willing to wait for a Web page to be downloaded before abandoning it, is approximately

2 seconds. Selvidge (1999) believes that there is no difference in users' frustration levels between a 1 second and 20 seconds delay, but there is difference between a 1 second and 30 seconds delay.

Web page download time is affected by different items such as the structure of the Web page requested, the network traffic and the speed of the Internet connection. In this thesis, we focus on network traffic. One of the techniques that reduces the overall traffic in the network is caching. Caching can be described as holding frequently accessed content in storage centers called caches, to where future accesses to specific content are made.

The first type of caching implements caching at the client site and since these caches are usually of limited size, only a restricted amount of content can be stored. When new content needs to be stored, it must replace an existing object. This brings the need for a replacement policy to determine which objects should be replaced by new ones. Client-based caching is therefore rarely used, since it can only serve a relatively small population of clients. But the real problem with such an approach is that the content provider has limited control over the content once it has been downloaded from the origin server and placed into caches (Kangasharju et al., 2002). The second type, server-based caching, is performed by installing additional caches at various places within the server. The third type of caching is usually performed using a Web proxy located somewhere between the client site and an origin server. When a client issues a request, the proxy will intercept the request and serve the client if the requested content is located in the cache; otherwise, the request will be sent to the server and the content will be accessed from the server itself. A copy will be stored at the proxy to serve further requests. These proxies are at different locations on the network, so they can serve a large number of clients and are very effective in reducing the network traffic.

Efforts to overcome delays in Internet traffic has given way to a new emerging technology called Content Distribution Network (CDN). According to Buyya et al. (2008). CDN is an effective approach to improve Internet service quality which is used in this study and will be discussed in Chapter 3 and 4 in this thesis. It replicates the content from the place of origin to replica servers over the Internet and serves the request from a replica server close to where request originates. A CDN aims at moving content as close as possible to clients. Therefore, a CDN can significantly improve performance of a network and reduce the total cost associated with distributing content, since clients are no longer served by origin server, but from a proxy server located nearby.

A content provider who issues the content for public use, may choose to resort to a commercial CDN to have its content efficiently distributed to its users. It is often the case that there is some kind of a Service Level Agreement (SLA) between the publisher and the CDN provider, usually the hosting service, that includes some kind of a Quality of Service (QoS) requirement. QoS refers to a wide collection of networking techniques and technologies. The aim of QoS is to provide guarantees on the ability of a network to deliver predictable results. Elements of network performance within the scope of QoS often include availability, bandwidth, latency, error rate and security of the service. Therefore, in performing the distribution operations, CDN must also take into account QoS (Peng, 2004).

1.2 Contribution and Objective of Our Research

The main contribution of this thesis is to create two new models to use in a CDN, and to design methods for obtaining solutions. This has been achieved by considering two optimization models in CDN with added delay and cache allocation, respectively. Selecting the best proxy for client with lowest response time by reducing expenditure is a major consideration in CDN. Usually, from the perspective of CDN provider, reducing expenditure helps to minimize the total cost.

Two models will be used to minimize the total access cost, which is the main objective of this study. In this thesis, one of the main objectives is to examine the effect of adding delay consideration into one of the CDN problems. The first model is to use one of the CDN models together with an added delay consideration. This model is related to the data placement problem on a network with no origin server, which consists of placing objects to minimize the average access cost, subject to the availability of system resources and traffic patterns. The traffic in the new formulation will be less than or equal to the capacity of the link. Lagrangian relaxation and decomposition approaches will be considered to find lower bounds due to the large scale and complexity of the problem making exact solution methods unsuitable. Also, two complicating constraints which are related to the amount of traffic in the link and the assignment will be used to relax this problem. The problem is non-linear and can be challenging to solve. Due to the non-linearity, an integer programming formulation based on linearization through perspective cuts will be proposed by using tangents of the function as valid inequalities to provide a lower bound. In order to improve results and achieve stronger bounds, between consecutive integer points in the function, we will consider linear segments instead of tangents where the segments pass through two consecutive integer points.

The second model is cache allocation. This problem explores the benefit of collaboration of objects, which refers to the sharing objects between a network of proxy caches. In such a network, if one proxy does not have an object, a neighbouring proxy can provide service. The point of using this strategy is that if an object is demanded from the cache that does not have a copy of the required object, it can be found in a neighbour cache. Each object can be assigned to each proxy at most once, which makes it difficult to solve. This strategy results in cost reduction, because instead of going to the origin server to get the objects, they can be retrieved from neighbouring caches. A local cluster of proxies will be considered by putting an object in each proxy to avoid having to use slower origin server. Essentially all proxies will be identical, but of different sizes in general. The objective function assumes that there is no difference where to place the object as each proxy has the same cost.

For both models as discussed above, Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) will be used to find upper bounds. IVND combines the strengths of Iterated Local Search (ILS) and Variable Neighbourhood Descent (VND). In addition, IVND performs well over a range of problem types and is usually competitive with other heuristic and metaheuristics. Also, IVND is flexible and problem independent, and is easy to implement. Tabu search is another method that is competitive across a range of problem types.

For the first model, optimal solution will be found by the NEOS server and compared with integer programming techniques and Lagrangian relaxation for lower bounds and Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) for upper bounds. Also, Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) will be applied to the second model for upper bounds and linearizations for lower bounds. For the two models, the aim is to evaluate the quality of lower bounds and upper bounds by calculating the gap between them.

1.3 Overview of the Thesis

The remainder of thesis is categorized as follows: In Chapter 2, a more extensive review of Content Distribution Network (CDN) and optimization models is provided since this is the main focus of this research. This chapter also includes various concepts and different solution methodologies such as: computational complexity theory; heuristics and metaheuristics; exact solution methods and integer programming formulations based on linearization. Optimization models have been divided into two independent problems. Chapter 3 presents an optimization model related to object retrieval and request routing problem. Also, this chapter contains a problem description, a new formulation with delay function, lower bound techniques, and metaheuristics that are normally used to solve problems related to CDN, as well as computational results alongside design of test instances.

Chapter 4 deals with the second problem, which is cache allocation. This study proposes a new model by considering a local cluster of proxies and placement of each object in an appropriate proxy to avoid having to use the slower origin server. Demand is measured by expected requests and can be satisfied by any proxy that holds the object of interest. This chapter also presents two different linearizations and two metaheuristic methods. Computational results with test instances and a discussion of results are provided for the proposed approaches. Finally, concluding remarks and some future research directions are given in Chapter 5.

Chapter 2

Literature Review

2.1 Background

The term *content* refers to any kind of information that is available to the public on the Word Wide Web such as Web pages, text documents, and multimedia files. An *object* refers to a specific item of the content, such as a sound file or a text document. Three main entities are in a CDN as follows:

- **Content Provider** (or customer): origin servers where original Web objects to be distributed are stored.

- **CDN Provider**: a company or organization that provides fundamental facilities to content providers in order to maintain copies of the content. This allows the providers to deliver content with high performance and high availability.

- **End-user** (or client): the end-users who access content from the Web sites of the content provider.

CDN providers use caching or replica servers located in different areas of the world to replicate content. CDN cache servers are called edge servers or surrogates. The edge server of a CDN is called Web cluster. A CDN distributes contents to surrogates, so all of them can share the same content. When a client requests an object, it is redirected to a nearby surrogate and the requested content is delivered to the client. After that, the surrogate sends accounting information for the delivered content to the accounting system of the CDN and reports traffic and billing data. Figure 2.1 shows a CDN network where content is distributed to a set of Web servers, broadcasted over the world for delivering content to clients. Each client is served by a nearby surrogate Web server. The objective is to provide high availability of the content and quick response.

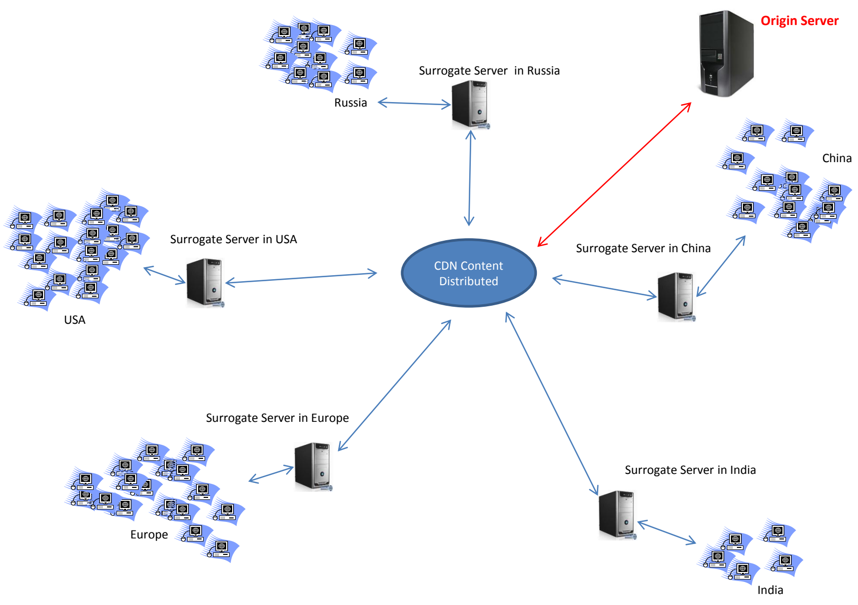
Origin Server
Surrogate Server in Russia
Russia
China
Surrogate Server in USA
USA
CDN Content Distributed
Surrogate Server in China
Surrogate Server in Europe
Europe
Surrogate Server in India
India

Figure 2.1: Model of CDN

CDN can be considered as a graph when a set of proxies with objects inside them connect together. The interconnected proxies are called vertices (nodes) and the links that connect some pairs of vertices are called edges (lines). A CDN focuses on building its network infrastructure to provide different services such as: cache management, delivery of data, storage and management of content and distribution of content among edge servers. A content provider (customer) uses a CDN provider for service and to have its content on the cache servers. Typically customers of a CDN are media and Internet advertisement companies, Internet service providers (ISPs), mobile operators and other carrier companies. Each of these customers want to deliver their contents to end users on the Internet in a reliable and timely manner.

Clients can interact with the CDN by specifying the content request through a cell phone, laptop, small phone or desktop (Pathan et al., 2008). CDN providers charge their customers according to the content delivered (i.e. *traffic*) to the end users by their edge servers. CDNs have an accounting mechanism that collects and tracks client usage information in real time according to request routing, distribution and delivery. The average cost of charging of CDN services is high. The most important factors affecting the price of CDN services include: bandwidth cost, variation of traffic distribution (pricing under different situations of traffic), size of content replicated over surrogate server and the number of edge servers.

The efficiency of distributing electronic content is affected by increasingly popularity of the World Wide Web and the high demand for electronic information. The size of delivered content and number of users grow steadily, thus resulting in increased loads for servers and network congestion, and unacceptable delays for end users often occur. Because of these reasons, distribution of electronic content has become an important problem. The main objective of CDN is to replicate content from the origin server to geographically distributed proxy servers, and then deliver requested content to clients from proxy servers rather than the origin server.

A CDN tries to improve the performance of the network by reducing the total cost related to distributing content, when the client is no longer served by the origin server, but instead is served by a nearby proxy server. Because of this reason, CDNs try to move content as close as possible to the clients. Due to the limited capacity of the proxy servers and the high cost of replication, it is not practical to replicate the whole content to each proxy server in a CDN. Therefore, to use network resources efficiently, a CDN must replicate the subsets of the content in an intelligent way. This problem is referred to as the object placement problem. Another challenge for CDN is to identify the best proxy server that should respond to a given request from a client. This problem is referred to as the client assignment problem.

2.2 Literature Review on CDN

Two problems that we are trying to solve have the characteristic of facility location and knapsack problems. The facility location problem is a critical element in strategic planning for a wide range of private or public firms. A CDN is a system of distributed networks that deliver Web pages and other Web content to a client based on the geographic locations of the client. This service is effective in speeding up the delivery of the content of Web sites with high traffic and Web sites that have global reach. If the client is close enough geographically to the CDN server, then the content will be delivered to the client faster. Therefore, a CDN has the characteristic of a facility location problem. The development of a new facility is typically costly and time-sensitive project. So, before purchasing or constructing a facility, suitable locations must be identified and appropriate facility capacity specifications must be determined (Owen and Daskin, 1998).

In a CDN, one of the models that connects the client's request for an object directly to the origin server, when the object is not available in any other proxy servers, is the client assignment problem (Thomas and VanderMeer, 2003). When the storage capacities of proxy servers are limited, there are many requests forwarded to the origin server. The client assignment mechanism is generally route requests to the best CDN serve (Barbir et al., 2003). The objective function is to minimize the total cost to retrieve all objects that are requested at a particular time.

Also, the proposed models have a knapsack problem characteristic because each proxy has a capacity and each object has a size, and we are choosing a subset of n objects and trying to fill each proxy without exceeding its capacity. All knapsack problems belong to the family of NP-hard problems, meaning that it is unlikely that anyone can devise polynomial algorithms for these problems. The outcome of several decades of research which have exposed the special structural properties of knapsack problems, make the problems easy to solve. Martello and Toth (1990) believe that all knapsack algorithms have the exponential worst-case solution time, but several large-scale instances may be solved to optimality in fractions of a second.

2.2.1 Client Assignment Problems

Different techniques for redirecting user requests to a suitable CDN server to reduce request latency and to balance load are studied in Qiu et al. (2001). The authors explore the problem of Web server replica placement and believe that the assumption of the tree topology makes it possible to obtain an optimal solution to the placement problem when the constraints are satisfied. In a more general setting it does not perform as well as the heuristics that work in general graph topologies. Moreover, its high computational complexity ($O(n^3m^2)$ for choosing m proxies among n potential sites) prevents its practical use in topologies with thousands of nodes.

Qiu et al. (2001) describe two versions of the center placement problem: one is the uncapacitated or capacitated facility location problem, and the other is the minimum K-median problem, both of which are NP-hard. They prefer to use the uncapacitated minimum K-median problem where they restrict the maximum number of replicas, but not the number of requests served by each replica. They state that this is a reasonable model because increasing the number of replica sites is significantly more difficult than increasing the capacity of a site. They also ignore the cost of placing replicas.

An assortment of algorithms such as: a tree-based algorithm, a greedy algorithm, a random algorithm and a hot-spot algorithm are proposed for solving the minimum K-median problem. A tree-based algorithm performs better than random placement, but not as well as the algorithms for general graph topologies. A greedy algorithm that places replicas based upon both a distance metric and request load, performs the best. Also, a hot-spot algorithm based upon request load, performs nearly the same. The main conclusion of their work is that a greedy algorithm for Web server, replica placement can provide content distribution networks with performance that is close to optimal.

On the Internet, an Autonomous System (AS) is the unit of router policy, either a group of networks or a single network that is controlled by a common network administrator or a group of administrators on behalf of a single administrator entity (such as a university) that presents a common, clearly defined routing policy to the Internet. Kangasharju et al. (2002) propose a model where each Internet Autonomous System (AS) is a node with finite storage capacity for replicating objects. The average number of AS traversed is minimized when client can get the requested object from the nearest CDN server.

Kangasharju et al. (2002) formulate the problem as combinatorial optimization problem and prove that this optimization problem is NP-complete. They develop four heuristics and compare them numerically by using real Internet topology data. A CDN ensures its content servers are close to users by placing content servers and clients at the same nodes. They conclude that each client is usually served by only one proxy server for security reasons, and the result of retrieving a whole object from a single proxy server is much better than retrieving different parts of an object from multiple servers.

Thomas and VanderMeer (2003) propose a model with proxy servers and an origin server. The model considers three different costs: the cost of retrieving an object where the request is directed from a proxy which has the object, the cost of retrieving an object where the request is directed to a proxy which does not have the object, so additional cost occurs to retrieve the object from the origin server, and the cost when the request is routed to the origin server.

Demand for accessing large files is expected to require high bandwidth capacity and a high storage server and this is one of the reasons that motivates multimedia providers to optimize the delivery distance, as well as electronic content allocation. In Cidon et al. (2002), a distributed algorithm for solving electronic content allocation over a distribution (directed) tree is presented. This algorithm minimizes the overall cost for storage and communication of media provider and allows for different costs at different servers for storing the same object.

Cidon et al. (2002) assume that users access content on-demand. Generally, combining media transmissions from the same server for two different users would be impossible, even for the same program, because users may start viewing the program or access file at different times. They assume non-server nodes cannot store content. When the second user accesses the server, then a second transmission from the server must start. They use dynamic programming to solve their problem.

An analytical model for the operation of CDN is discussed in Ercetin (2002). Their aim is to minimize the user latency by intelligently distributing content and serving user requests from the most efficient sites. Their models closely resemble the current network structure where the users are selfish, non-cooperative and try to maximize their own benefit regardless of others. They consider the distributed resource allocation strategy for both content dissemination and the routing problems observed in the CDN. Their results rely on non-cooperative game theory to show that an equilibrium exists to the aforementioned system and develop a resource allocation algorithm. They identified two types of optimization problems in their model: publisher revenue optimization and surrogate revenue maximization. Finally, they discuss the joint problem of dissemination and routing, conduct numerical experiments and show that the resulting framework can give significant gains with respect to more conventional methods due to capacity.

Networks with storage servers for providing multimedia content are discussed in Xuanping et al. (2003), where the authors investigate the problem of optimally replicating objects at candidate proxies in a CDN. They assume that proxies can be chosen from a number of candidate servers distributed in the network. Each proxy in the set of candidates has a finite storage capacity for replicating objects and incurs a fixed fee for replication. The optimization problem is to find a set of proxies from a candidate set to minimize the total access cost, subject to constraints that total weight of objects placed in the proxy should not exceed the storage capacity of the proxy and the total fee charges by the proxies should not exceed a pre-specified budget. They formulate this problem as a combinatorial optimization problem and show that it is NP-complete. They develop two heuristic algorithms. To evaluate the performance of two heuristics, Xuanping et al. (2003) compare their algorithms with a random method that randomly picks up one candidate proxy and stores as many as possible objects. It is shown that both heuristics perform better than random methods when proxy storage capacities and the total budget are limited, and the relative cost is reduced as the number proxy candidates become higher. One of the heuristic algorithms performs slightly better than the other because it takes the fee charge of each proxy into account.

A model for replica placement in CDN for multimedia applications is studied in Yang and Fei (2003), which is the allocation of objects in a network of proxies that collaborate to serve requests from clients. They propose a new model for replica placement and perform theoretical analysis on the cost of distributing multimedia files over CDNs. Their results indicates that deploying as many replicas as possible is not always a good strategy. Because of that, they chose finite capacities and a finite number of proxies in their model as is common in multimedia environments. Two well-known graph problems have been used to model the replica placement problem, the first one is the K-median problem and other one is the capacitated facility location problem. They propose several replica placement algorithms that can determine the optimal number of replicas which are selected from a given set of potential sites.

A resource allocation problem in a graph, and the joint optimization of capacity allocation decisions and object placement decisions with a single capacity constraint is studied in Laoutaris et al. (2004). The solution of the problem is based on a multi-commodity generalization of the single commodity K-median problem. The authors describe a two-step exact solution algorithm: a decomposition step is first responsible for the solution of a series of K-median problems relating to the original problem and a composition step then selects of a number of optimal solution components from the first step. The composition step is a packing problem that selects among the optimally solved K-median solutions. When combined together, these steps yield an optimal solution for the original capacity allocation problem. They formulate specific instances for their problem and propose a heuristic algorithm for them.

The problem of replica placement and client request routing is addressed in Almeida et al. (2004). They model the Internet as a graph where each node represents either a router or an Autonomous System (AS) and a link. The objective is one of minimizing the total delivery cost, which includes network and server delivery costs. They show that the variety of realistic scenarios can be solved exactly using available optimization software. Almeida et al. (2004) develop both exact and approximation methods for determining the number and placement of replicas, and the routing of requests as to minimize the total delivery cost. Therefore, six heuristics are described that are able to find near-optimal solutions at lower computational cost than the exact algorithm. The best algorithms use a greedy min-cost Traveling Salesman Problem (TSP) placement heuristic, and either shortest path routing or a greedy min-cost heuristic for routing client requests. Exact and approximate methods are considered for determining the number and placement of replicas, as well as the routing of requests and multi-cast streams, to minimize the total delivery cost.

A new technique for the problem of designing a CDN is studied in Bektaş et al. (2007), which is the technology used for efficiently distributed electronic content among an existing IP network. Their design consists of the number and placement of proxy servers on a given set of potential nodes and replicating content on the proxy servers, and finally routing the requests for content to a suitable proxy server such that the total cost of distribution is minimized. Bektaş et al. (2007) also describe a fast and efficient greedy heuristic algorithm that can be used to obtain near-optimal solutions for the problem is introduced. Their approach considers a similar problem with respect to the objective function but differs from the related studies considering the situation in which requested objects are not found in any proxy server. The proposed model includes a quadratic objective function, which is linearized, and the linearized model is solved by a Benders decomposition algorithm. Bektaş et al. (2007) provide computational experiments that are based on randomly generated data. Internet topologies suggest that the proposed algorithm is superior to CPLEX and is very efficient especially when the number of clients is large and the number of objects is relatively small. The algorithm is also a demonstration of an exact solution technique that can be used for similar integer models with quadratic objective functions. Bektaş et al. (2007) believe that an exact solution approach is useful for relatively small networks, but a faster heuristic algorithm is required for very large networks, such as the Internet itself.

Two exact algorithms for the joint problem of object placement and request routing in a CDN, one based on Benders decomposition and the other based on Lagrangian relaxation and decomposition, are described in Bektaş et al. (2008). The proposed models capture the fundamental characteristics of the problem, and the necessity to support a certain level of Quality of Service (QoS). A non-linear integer programming formulation is described for the problem which is linearized in three different ways. Computational experiments are performed using randomly generated data, where some real-life aspects of a content distribution environment also incorporated in the data generation process. A powerful heuristic is described for realistic instances. The proposed heuristic algorithm is a two-level implementation of the well-known simulated annealing metaheuristic introduced by Kirkpatrick et al. (1983), and the algorithm is simple, easy to implement, and able to produce good-quality solutions within reasonable computing times.

Luss (2010) presents a model for content distribution network, works on Video-On-Demand (VOD) services and formulates an integer programming model and develops a dynamic programming formulation. He considers a tree network when each node has demands for multiple different VOD program families, where all similar programs are considered as a program family. This model has a different cost for servers, cost for assigning program families to servers and cost of link bandwidths used to broadcast programs. The demand for specific program is assigned to the nearest server that has this program along the route that connect the requesting node to the root of the tree network. Luss uses a dynamic programming method that determines optimal server locations and optimal program assignments for minimizing the sum of all these three costs. His algorithm starts from the end nodes of the tree network and determines optimal solution of the subtrees. Eventually by reaching to the root node, optimal solution for the entire network are achieved. This model has flexibility for large service providers that may have overlapping, by adding a new layer of decision to the tree where demands will be assigned.

Neves et al. (2010) propose exact and heuristic approaches to solve replica placement and request routing problem in CDN. The objective is to find the best servers to hold the replicas and to handle requests, so the traffic cost in the network is minimized without violating server and Quality of Service (QoS) requirements. QoS constraints are given by a required minimal bandwidth and the maximum delay that the client's request has until being served. Their assumption are: all contents are positioned in the origin server and a server is allowed to serve client's request partially or totally, and the request may also be served by several servers partially at the same time or served in several periods of time. The problem of finding the best set of servers to place content replicas over the CDN is NP-complete. Two heuristics are proposed for this problem and an analysis of them indicates that the gap between these two is caused by the natural difference between exact and heuristics methods.

Deane et al. (2012) develop a multi-objective CDN model for optimal cost and performance. The purpose of considering both cost and performance as objectives is to avoid the requirement to create bounds, and the decision maker is totally flexible to compromise between cost and performance. They assume the proxy servers are fixed and each node in the network is an Internet Autonomous System (AS) or proxy server that represents a server location; thus the origin server would be a possible AS. The cost includes the cost of object storage and the cost of object transfer. They calculate storage cost based on object collection size by using a cost per gigabyte stored, and object transfer cost is a bandwidth usage cost that is charged based on gigabytes of data transferred. They utilize a case example to illustrate the feasibility of solving their model for a CDN of reasonable size. Their model finds optimal strategies for distributing content objects among various proxy sites to address multiple objectives, assuming that the decisions on which objects to cache and when to cache them have been made. Integer programming is used to solve their problem.

Coutinho et al. (2012) design and implement an algorithm for a request routing problem as a transportation problem to find the best server for each customer's request in CDN, in order to minimize the overall cost. A distributed algorithm is proposed to solve their problem. The algorithm is composed of two independent parts: a distributed heuristic that finds an initial solution for the problem and a distributed transportation simplex algorithm, also not requiring global information about the content requests. They compare their algorithm with a sequential version of transportation simplex and with an auction-based distributed algorithm and they find that their algorithm has a performance similar to its sequential counterpart, despite not requiring global information about the content requests.

Drwal and Jozefczyk (2013) formulate a data placement problem as quadratic binary programming problem. They consider a model as a directed graph, and the vertices represent access routers of local area networks and edges represent bidirectional communication links. Users of the local area network announce their demands to the Internet Service Providers (ISPs) that manage the routers, as the ISP is responsible for serving the users in local networks behind each routers. Each access router is equipped with a cache server that can store copies of objects and each cache memory is connected to the router via a very fast transmission bus. If an object is stored inside a cache server, it can be served directly to the users; otherwise, it can be accessed by a node from any cache server storing that object. In computer system design, the overall performance of the system is determined by processing time (CPU time), memory and communication bandwidth. From the system's operator point of view, optimal design can be stated as follows: given a user's demand and limited resources, allocate the resources to the demand to minimize the total cost of service. They consider three types of costs that are all expressed in time units. Each of the costs is related to the service latency on a router and related to a cache server, and the total cost is the sum of the costs carried by all access routers. They propose heuristic methods and two decompositions-based solution approaches: the Lagrangian relaxation and randomized rounding leading to two different solution approaches. The purpose of decomposition is a reduction of complexity of the original problem that allows for applying specialized algorithms for solving smaller subproblems.

Sen et al. (2014) introduce a new model for the Data Partitioning Problem (DPP) and Segment Allocation Problem (SAP). The DPP is a decision problem that decides how the files are clustered into segments, and the SAP is a decision problem that identifies potential sites, locates the segments, and routes the requests in such a way that cost is minimized. They consider files as units of allocation in a cost optimization problem that solves three problems: locations of servers, file placement and user assignment. They build a data partitioning method using facility location models and show that the method partitions the database within a reasonable limit of error. After formulating the problem as an MILP exact solution approaches are used to solve large-scale problems. A Bender's decomposition algorithm is also developed and the superiority of this decomposition approach is shown through computational results.

2.2.2 Cache Allocation Problems

The problem of placing caches in a network is studied in Krishnan et al. (2000), where the problem is formulated as a location problem both for general caches and for Transparent En-Route Caches ($TERC$). In $TERC$, caches are only located along routes from clients to servers, where clients can detect them easily. An en-route cache blocks any request that passes through and then forwards the request towards the server along the regular routing path. Krishnan et al. (2000) present a computationally efficient dynamic programming algorithm for the problem defined on a tree network with a single source. They observe that a small number of $TERC$, when placed at optimal locations, are sufficient to reduce the network traffic significantly. These optimal locations are not specifically at the edge of individual networks, where providers currently intend to place them. They also compare their exact algorithm against a greedy algorithm and report on their relative performance. The effectiveness of a proxy is primarily determined by locality, which is largely influenced by the location of the Web proxy. Putting a Web proxy in the wrong place is not only costly, but also does little improve the system performance.

Li et al. (1999) investigate the optimal placement of Web proxies in the Internet. They focus on two major factors: the overall traffic and access latency, and the objective is to minimize the overall latency of searching, subject to the system resources and traffic pattern. They formulate the optimal placement problem into dynamic programming and obtain an optimal solution for tree topology in $O(n^3m^2)$ time, where n is potential sites and m is multiple Web proxies.

Woeginger (2000) reformulate the dynamic program of Li et al. (1999), and prove that one of the underlying cost functions in this problem carries a Monge structure. He finds a faster algorithm compare to Li et al. (1999) according to time complexity. Monge structure is used in a variety of applications such as transportation problems, search problems, traveling salesman problems, etc. Usually, this additional structure helps to find polynomial time solution algorithms or at least helps to speed up algorithms.

The optimal placement of a limited number of Web proxies in an environment where a Web site is replicated is discussed in Jia et al. (2001). They optimize two important objectives of proxy placement: minimization of overall cost for all clients to access the Web server in the system, and minimization of longest delay for any client to access the Web server in the system. Both problems are formulated using dynamic programming through which optimal solutions can be obtained in polynomial time. Proxies are considered as transparent en-route proxies, which are only located along the routes from clients to a Web server and are transparent to the clients. If a client's request meets a proxy on its way to the server and the requested data is available at this proxy, it will be served by the proxy; otherwise, it has to go all the way to the server to be served, even though the client is close to the proxy. Simulations are conducted to demonstrate the performance of their proposed algorithms and it is demonstrated that optimal algorithm can significantly reduce the overall access cost and improve the response time for client to access Web servers.

Placement problems in proxy servers with transparent data replication is considered in Xu et al. (2002). The study presents a hybrid transparent replication model which combines the advantages of the en-route replication model with the hierarchical replication model. The placement issues result in replication of proxies and data, with the objective of minimizing the total data transfer cost, given that a maximum number of proxies are allowed. First, they present optimal solutions for a single object in a tree network, with or without constraints on the number of replicas. Xu et al. (2002) split the problem into two sub-problems: the problem of proxy placement in the network and the problem of replica placement on the installed proxies. For proxy placement, two schemes are proposed for the replication proxy placement problem. For replica placement, an exact algorithm is presented that runs in linear time. Simulation experiments are conducted over a wide range of system parameters so that the results would be applicable for a wide variety of settings.

Laoutaris et al. (2005) study the position of proxies in CDN, and emphasise that the most suitable proxies are transparent to clients since they are easier to manage and require no cooperation from clients. The storage capacity allocation problem is addressed for CDN, which takes into account the optimal location and capacity of the proxies and the objects that should be placed in each of them. All relevant sub-problems are combined, concerning node location, node sizes, and object placement, and solved jointly in a single optimization algorithm. A linear-time efficient heuristic algorithm is proposed.

Two techniques to decrease delay, as experienced by clients, are caching and replication. Bakiras and Loukopoulos (2005) investigate the potential performance when CDN server is considered as replicator and proxy server at the same time. They develop an analytical model to describe the benefit of each technique and propose a greedy algorithm to solve the combined problem of caching and replica placement. They consider geographically distributed servers each with a total storage capacity, and the communication cost between two servers is the cost of the shortest path between two nodes which is symmetric. They consider several Web sites that have subscribed to the CDN provider's hosting service and each Web site consists of different objects and the popularity of these objects follows the Zipf-like distribution. According to their replication policy, there is one primary copy for each Web site in the network. Every server stores two records: the primary Web site and the nearest server which holds the replica. So the storage capacity for each server can be considered as replication and caching. Whenever a client request the Web site, in terms of the network distance, will reply if the requested object is neither replicated nor cached locally, the client request will be redirected to the appropriate server. Their simulation results indicate that there is much room for performance improvement compared to using a replication or caching mechanism individually.

Sun et al. (2011) propose an optimization model with server storage capacity constraints for the replica placement problems. They divide replica placement problems into two sub-problems: the number of each replica and the identity of servers to store the replicas. They consider the total storage capacity limit of all CDN servers, in order to record the number of replicas. After the number of replicas has been fixed, the replica placement problem is transformed into P-median problems.

Jabraili et al. (2013) propose a model to find an optimal placement of objects within replica servers, so users are able to receive contents from the nearest replica server. They use a heuristic approach that considers the object frequency, object latency and object load in order to decide the replica server in which the content is to be placed. They solve replication problem and develop an analytic simulation environment to test the proposed scheme which can be used independently. The results of simulation show that their method significantly improves the response time of requests.

A new hybrid heuristic for replica placement and request distribution in CDN is studied in Neves et al. (2014). In their problem, every request must be handled fully, servers bandwidth and storage capacity must be satisfied and at least one replica for each content exists in each period (one period is necessary to download a content). The servers are allowed to handle requests if, and only if, they have a copy of desired contents. A new hybrid heuristic is proposed that uses a network traffic model for solving the request distribution problem and a formulation, based on a generalized assignment problem is provided for solving replica placement problem. The results show that the hybrid network heuristic works better with less computational time compare to solving the mathematical formulation of the problem.

Few strategies to improve the cache management architecture by considering browser with proxy caches server, where the browser cache acts as proxy server by sharing its content through hybrid architecture is proposed by Imtiaz et al. (2014). Their main aim of the new architecture is to distribute the load of proxy server, to minimize the respond time, to reduce Web traffic and reduce the amount of bandwidth used by a client, this is also client that can save money if he is paying for the traffic. Two basic caching architecture that use multiple caching servers to work together are: Distributed caching and Hierarchical caching. Distributed caching spreads the cached Web objects across two or more caching servers, where all these servers are all on the same level of the network so caching is more efficient in terms of disk space usage. However, hierarchical caching works differently; caching servers are placed at different levels on the network, so that caching is more efficient in terms of bandwidth usage. Imtiaz et al. (2014) combine these two caching architectures (distributed caching and hierarchical caching) to create a hybrid caching architecture which improves the performance and efficiency of the network.

2.3 Optimization Models for Assignment and cache Allocation Problems

The CDN architecture is described in Figure 2.1, with the origin server and five proxy servers with different clients for each proxy server, where each client is connected to a single proxy server. The problem consists of deciding on the following issues: the assignment of a client to a single or multiple proxy server, the location and number of proxy servers to be installed in a CDN on a given set of potential sites, and the objects to be located in each proxy server. The perspective of the cost minimization models is studied for the CDN provider because their aim is to deliver content to content providers with high performance and high availability at the lowest expense. When the client requests an object, the request is served by the associated proxy, if the object is already stored there; otherwise, other proxies are not contacted and the request is forwarded to the origin server, from where the client is able to access the request from the origin server via the path from the corresponding proxy server. This strategy is regarded as non-cooperative caching. The second strategy allows a client to connect to more than one proxy server to fetch a requested object. This strategy is named as cooperative caching. The notations used in formulating our models are presented below:

Sets	**Notation**
Set of clients	I
Set of proxies	J
Set of objects	K
Set of proxies with origin server	$J_0 = J \bigcup \{0\}$
Set of nodes $I \cup J$	V
Set of links $E = \{(i, j) : i \in I, j \in J\}$	E
Complete network $G = (V, E)$	G
Parameters	**Notation**
Cost for transferring an unit size object over link $(i, j) \in E$,	c_{ij}
Capacity of the potential proxy $j \in J$	s_j
Integer size of each object $k \in K$	b_k
Probability of client $i \in I$ requesting object $k \in K$ over a given time interval	λ_{ik}
Expected demand for object $k \in K$ at proxy $j \in J$	e_{jk}
Cost of transferring an object from the proxy (locally)	c_l
Cost of transferring an object from other proxies in the network	c_n
Cost of transferring an object from origin server	c_o
Variables	
x_{ijk} equal to 1 if client $i \in I$ requests object $k \in K$ that is served by proxy $j \in J$; and 0 otherwise	
z_{jk} equal to 1 if proxy $j \in J$ holds object $k \in K$; and 0 otherwise	
m_k equal to 1 if object $k \in K$ is acquired from the origin server; and 0 otherwise	

2.3.1 Object Retrieval and Request Routing

First we start with one of the basic models, the objective of the model being to minimize the total access cost (the cost for a client $i \in I$ to access object $k \in K$ from proxy $j \in J$ is $b_k \lambda_{ik} c_{ij}$). Baev and Rajaraman (2001) consider the data placement of cooperative caching that determines a static placement of replicated objects between nodes in the network with a given access pattern and try to minimize the average access cost taken over all nodes and all objects. Their problem formulation is most suitable for applications where the objects are read-only because there is a different mechanism to maintain consistency among the replicas. Baev and Rajaraman (2001) assume any request at a node is satisfied by a copy of the requested object which is nearest to the node. They consider set of nodes, which means each node can act as both a client and a proxy server and each proxy server $j \in J$ has a capacity s_j. Each object $k \in K$ has size b_k and the sum of all objects stored in any proxy $j \in J$ must be at most the capacity of the proxy s_j.

The probability of client $i \in I$ requesting object $k \in K$ over a given time interval is λ_{ik}, and the cost of transferring an object over link (i, j) is c_{ij} per unit size. A binary variable x_{ijk} indicates if client $i \in I$ is assigned to access a copy of object $k \in K$ stored in proxy j. Also, another binary variable z_{jk} denotes if object $k \in K$ is held in proxy $j \in J$. Baev and Rajaraman (2001) proposed the mathematical model as below:

$$\min \quad \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} b_k \lambda_{ik} c_{ij} x_{ijk} \tag{2.3.1}$$

$$\begin{aligned}
\text{s.t.} \quad & \sum_{j \in J} x_{ijk} = 1 \qquad \forall i \in I, k \in K, & (2.3.2)\\
& \sum_{k \in K} b_k z_{jk} \leq s_j \qquad \forall j \in J, & (2.3.3)\\
& x_{ijk} \leq z_{jk} \qquad \forall i \in I, j \in J, k \in K, & (2.3.4)\\
& z_{jk} \in \{0, 1\} \qquad \forall j \in J, k \in K, & (2.3.5)\\
& x_{ijk} \in \{0, 1\} \qquad \forall i \in I, j \in J, k \in K. & (2.3.6)
\end{aligned}$$

The objective function expresses the expected total cost of serving requests for all proxies and objects. Constraint (2.3.2) indicates that each client and object pair must be assigned to exactly one proxy server. Constraint (2.3.3) relates to the limited capacity s_j of each proxy $j \in J$. Constraint (2.3.4) implies that an assignment to a proxy can only be made if that specific proxy is holding the requested object.

2.3.2 Objects in a Network of Caches

The process of replicating Web content that can be easily accessed by users is called caching according to Tawarmalani et al. (2009), which is a process to reduce the overall traffic in the network. Caching holds an accessed content in storage centers called caches, which is the process of replicating Web contents in locations where they can be easily accessed by end-users. There are three different kinds of caching: client-based caching, server-based caching and proxy-based caching.

The first type implements caching at the client site and only serves the requests made from this specific location. In these caches, only a restricted amount of content can be stored because of capacity limits. Therefore, when the new content needs to be stored, it must replace existing content. The problem with this approach is that the content provider has limited control over the content once it has been downloaded from the origin server and placed into caches.

The second approach is performed by installing additional caches in different places within the server. This type of caching helps to share the load on the server and reduce the incoming traffic to the server to a small amount.

The third type uses a web proxy located somewhere between the client site (such as a university or a company) and the origin server. When the client requests an object, if the requested object is located in the cache, the proxy intercepts the request and serves the client. Otherwise, the request will be sent to the origin server and the copy will be stored at the proxy to serve further requests. All proxies in this approach are located at different places on the network. Therefore, a large number of clients can be served while network traffic is reduced.

The key motivation to use proxy-based caching is that, when a requested object is not held by a cache, a neighbour cache that holds the object can deliver the object faster than the origin server. Therefore, cooperative caches are designed as follows: if the requested object is not held in local cache, the object could be found in neighbour caches; failing that, the request is forwarded to the origin server. Tawarmalani et al. (2009) analyze the allocation of objects in a network of caches that collaborate to service requests from customers.

The binary variable z_{jk} indicates if object $k \in K$ is held in proxy $j \in J$. The authors allocate the objects to the caches to minimize the network cost. The model is named *Cache Allocation Problem* as follows:

$$\min \quad \sum_{j \in J} \left\{ (c_l - c_n) \sum_{k \in K} e_{jk} z_{jk} + c_n \sum_{k \in K} e_{jk} + (c_o - c_n) \sum_{k \in K} e_{jk} \prod_{j \in J} (1 - z_{jk}) \right\} \tag{2.3.7}$$

$$\text{s.t.} \quad \sum_{k \in K} z_{jk} \leq s_j \qquad \forall j \in J, \tag{2.3.8}$$

$$z_{jk} \in \{0, 1\} \qquad \forall j \in J, k \in K. \tag{2.3.9}$$

Constraint (2.3.8) imposes a capacity of s_j for each proxy $j \in J$. In this model, the cost of serving the object locally from the proxy where it is requested is c_l, from the origin server is c_o and from other neighbouring proxies in the network is c_n. The object is obtained from neighbour only when it is not held locally and at least one of the neighbors holds that object, and the object is obtained from origin server when it is not accessible either locally or from the neighbors. It is possible that $c_n > c_o$ when the caches access the objects from the origin server rather than from the network. To allow for this case, c_n may be replaced by $\min\{c_n, c_o\}$.

First of all, Tawarmalani et al. (2009) consider a unit size for each object with fixed capacity for each proxy, and assume that a reasonable estimate of demand for each object is available. They look at centralizing scenarios linearized by the model using the techniques of Glover and Woolsey (1974), and provide a transportation algorithm for the problem. Secondly, they discuss the generalization of their results to objects with arbitrary sizes. In terms of complexity, the problem becomes much harder. The objects are not allowed to be split and are stored in individual parts in different proxies. The reason is that, if the object can be divided into packets for transmission, to access the object from one source or different sources, then reassembly of the object may be required, so the cost of combining different parts of an object may be too high to be ignored.

2.4 Solution Methodologies

An optimization problem is designated as combinatorial if a finite number of feasible solutions exist or continuous if it has an infinite number of feasible solutions. The objective of an optimization problem is to find a feasible solution that maximizes or minimizes the value of a given objective function. In this section, we introduce some of the methods which can be used to solve combinatorial optimization problems. To find near-optimal solutions, heuristic algorithms are recommended, and for optimal solutions exact methods are considered. The theoretical complexity of a combinatorial optimization problem indicates if it is appropriate to apply exact methods or not.

2.4.1 Computational Complexity Theory

One of the major research areas of computer science is computational complexity theory, which focuses on classifying computational problems according to their difficulty and relates to finding a set of problems having the same complexity class as each other. According to Tovey (2002), NP (Non-deterministic Polynomial) is one of the complexity classes of decision problems, for which the feasibility of a solution can be checked in polynomial time.

Garey and Johnson (2002a) believe that two different problems are related to each other with respect to their difficulty. The principle technique is that of reducing one to the other by giving a constructive transformation, for which any instance of the first problem is assigned into an equivalent instance of the second problem. That means any algorithm which solves the second problem is being converted to the corresponding algorithm that solves the first problem. This reduction process is fundamental to NP-completeness.

The foundation for the theory of NP-completeness is due to Cook (1971). He emphasizes the polynomial time reducibility, that is the reductions for required transformation can be done by a polynomial time algorithm. Also, he focuses on the class NP of decision problems that can be solved in polynomial time by a non-deterministic computer which has ability to pursue an unbounded number of independent computational operations is parallel. Cook proves that, for one particular problem in NP called the satisfiability problem, every other problem in NP can be polynomially reduced to it. This means that if the satisfiability problem can be solved in polynomial time, then every other problem in NP can be solved polynomially and if any problem in NP is intractable, then satisfiability problem is also intractable. Finally, he suggests that the satisfiability problem are considered as the hardest members in NP.

Afterwards, Karp (1972) presents the collection of Cook results and proves that many well-known combinatorial problems are as hard as the satisfiability problem. Also, a wide variety of other problems have been proved equivalent in difficulty to these problems with the same class, all these problems are categorized as hardest problems in NP and have been given a name: the class of NP-complete problems. There is no known polynomial time algorithm already for any NP-complete problem and there is no proof that a polynomial time algorithm exists or not for any of them, so this issue is a open question in complexity theory.

The complexity class P is formed of these problems which are solvable in polynomial time and is contained in NP. The relationship between P and NP (P $\subseteq$ NP) is fundamental for NP-completeness theory. Every decision problem in P can be solved by a polynomial time deterministic algorithm and can be solved by a polynomial time non-deterministic algorithm. This problem is categorized as NP-hard since it is as hard as NP-complete problem. Therefore, if decision problem is NP-complete, then the corresponding optimization problem is NP-hard.

2.4.2 Exact Solution Methods

Exact solution methods are techniques that are used to solve difficult problems to optimality. For NP-hard problems, they are mainly based on principle of enumeration, which gives an exact solution for the problem. The performance of these methods is often measured by size of problems that can be solved in reasonable computation time. Lower bounds and upper bounds are often used to restrict search as much as possible and are derived by applying relaxation and computing solutions using heuristics. Relaxation is an approximation of a difficult problem that involves removing some of a problem complexity so that becomes easier to solve. One of the methods that we use to find lower bounds is Lagrangian relaxation and another that is useful for non-linear problems is to linearize the non-linear functions.

2.4.2.1 Branch and Bound

Branch and bound is a structured search of the space of all feasible solutions and is used for solving integer and discrete optimization problems (Land and Doig, 1960). A branch an bound algorithm consists of a systematic enumeration of candidate solutions, in which successive configurations of an instance are considered as the best solution. The set of candidate solutions are considered as a rooted tree with the full set of root. The algorithm explores branches of this tree, that represent subsets of solution set. The space of all feasible solutions is repeatedly partitioned into smaller and smaller subsets and attempts are made to omit a node based on a lower bound of the objective function, which is called bounding. In each partitioning, the lower bound for a subset is checked as upper and lower estimated bounds on optimal solution, which could be the value of an initial feasible solution found using a heuristic approach.

For brevity, we describe branch and bound for minimization problems. If a cost criteria is available, the node to be expanded next, is one with the best cost within the queue (branch). In such a case, the cost function may also be used to discard from queue nodes that can be determined to be expensive (bound). If lower bound is more than the best known objective value, the node is pruned, because it cannot produce a better solution, than the best one found by the algorithm so far. The partitioning continues until a feasible solution is found (Lawler and Wood, 1966). Before branch and bound, to find upper bound heuristic can be applied, that is considered as initial upper bound, or upper bound can be calculated in every node of search tree. Tightness and easy calculation of lower bound in addition to quality of initial solution affect the efficiency of this method. The main disadvantage of branch and bound is usually its heavy computational requirements because of large number of nodes.

2.4.2.2 Bin Packing

Characteristic of bin-packing application is to pack a collection of objects into different regions, so the amount of wasted space is minimized (Coffman J. et al., 1984). Basic idea is making efficient use of time or space (or both). The advantage of this method is to have guaranteed packing performance bounds. In computational complexity theory, bin packing is a combinatorial NP-hard problem, so optimal solutions to very large instances can be produced with sophisticated algorithms (Garey and Johnson, 2002b).

Bin-packing mostly concerns on polynomial-time approximation algorithms such as first-fit, first-fit decreasing and full-bin packing. In first-fit algorithm when items place in order they come in first bin in which they fit. In this algorithm items are fitting neatly and efficiently inside a large bin. This method is quick and easy to perform but does not usually leads to an optimal solution. In first-fit decreasing algorithm when an alternative strategy first orders the objects from largest to smallest, then places sequentially in first bin in which they fit. This algorithm is fast and easy to perform and better solution from first algorithm but does not always guarantees an optimal solution. Third algorithm is full-bin packing by arranging objects into size of container and try to use the least possible numbers of container. This algorithm usually gets an optimal solution but can be difficult to perform when numbers awkward. All of these algorithms mostly focus on the quality of solution rather than optimal solutions (Korf, 2002).

2.4.2.3 Lagrangian Relaxation

Lagrangian relaxation is one of the basic tools in constrained optimization problems to deal with difficult constraints. A general constrained optimization problem is as below (Nedić, 2011):

$$\min f(x) \tag{2.4.1}$$

$$\text{s.t.} \quad g_j(x) \leq 0 \qquad j = 1, ..., m, \tag{2.4.2}$$

$$h_i(x) = 0 \qquad i = 1, ..., r, \tag{2.4.3}$$

$$x \in X \tag{2.4.4}$$

Where f:$R^n \rightarrow R$, g_j:$R^n \rightarrow R$ for $j = 1, ..., m$, h_i:$R^n \rightarrow R$ for $i = 1, ..., n$ and $X \subseteq R^n$. Two set of constraints $g_j(x) \leq 0$ and $h_i(x) = 0$ are brought into the objective function by assigning a non-negative scalar μ_j to the inequality constraint $g_j(x) \leq 0$ and assigning an unrestricted scalar δ_i to the constraint $h_i(x) = 0$. Then new objective function for problem is defined as:

$$L(\mu, \delta) = \min f(x) + \sum_{j=1}^{m} \mu_j g_j(x) + \sum_{i=1}^{r} \delta_i h_i(x), \tag{2.4.5}$$

where μ=$(\mu_1, ..., \mu_m)$ and δ=$(\delta_1, ..., \delta_r)$ and constraints (2.4.2) and (2.4.3) are removed. Normally, μ_j and δ_i are referred to as Lagrangian multiplies and the minimization in (2.4.5) is Lagrangian problem. The value of $L(\mu, \delta)$ is a lower bound on optimal solution value of original problem for any given δ and $\mu \geq 0$. The problem max $L(\mu, \delta)$ for $\mu \geq 0$ and any δ is known as dual problem.

Subgradient optimization is a method to heuristically solve a Lagrangian dual problem. It iteratively adjusts Lagrangian multipliers to find values that produce the best or nearly the best lower bound. It relies on an upper bound for optimal objective value of original problem. An upper bound could be calculated by finding a feasible solution with a heuristic for original problem. Given initial values μ^0 and δ^0, sequences $\{\mu^k\}$ and $\{\delta^k\}$ are generated as below:

$$\mu^{k+1} = \mu^k + S_\mu g_j(x^k), \delta^{k+1} = \delta^k + S_\delta h_i(x^k), \tag{2.4.6}$$

where x^k is an optimal solution of $L(\mu^k, \delta^k)$ in (2.4.5). Also, $S_\mu = \lambda_k \frac{UB-LB}{||g_j(x^n)||^2}$ and $S_\delta = \lambda_k \frac{UB-LB}{||h_i(x^n)||^2}$ are positive scalar step sizes and λ_k is a scalar satisfying $0 < \lambda \leq 2$ (Fisher, 2004). UB is an upper bound is obtained by applying heuristic and LB=$L(\mu^k, \delta^k)$ is a lower bound of (2.4.5). Often the sequence λ_k is determined by setting $\lambda_0 = 2$ and whenever objective function has failed to increase for fixed number of iterations, halving λ_k. Justification of these formulas is given in Held et al. (1974).

Lagrangian relaxation is the most successful, when Lagrangian sub-problem can be solved efficiently. Lagrangian relaxation has been used by Geoffrion (1974) in his highly successful work to develop theory and explore benefit of Lagrangian relaxation in branch and bound and also by Fisher (2004) in his algorithms for a number of important problems in areas of routing, location, scheduling, assignment and set covering.

2.4.3 Integer Programming Formulation Based on Linearization

A common solution method for non-linear integer and mixed integer programming problems is linearization. Let $f(x)$ be a non-linear and convex function and define non-linear integer programming problem $P(1)$ as below:

$$P(1) \qquad \min \qquad f(x) \tag{2.4.7}$$

$$\text{s.t.} \quad Ax \geq b, \tag{2.4.8}$$

$$x \in N. \tag{2.4.9}$$

In (2.4.8) A is $m \times n$ matrix, x is $m \times 1$, and b is column vector. By applying linearization approach of Erdoğan et al. (2010) and constructing following valid inequalities, we can solve function P(1) through P(2), where w is an auxiliary variable.

$$P(2) \qquad \max \qquad w \tag{2.4.10}$$

$$\text{s.t.} \quad Ax \geq b, \tag{2.4.11}$$

$$w \geq f'(x^*)x + f(x^*) - f'(x^*)x^* \qquad \forall x^* : Ax^* \geq b, x^* \in N, \tag{2.4.12}$$

$$x \in N. \tag{2.4.13}$$

Where all constraints are linear. The tangent point of approximating line is $(x^*, f(x^*))$ and slope of that line will be $f'(x^*)$. The main point is finding tangent line of the point and then using that line to provide a lower bound for value of function. The main reason for using this technique, called perspective cut, for our problem is non-linearity of problem due to delay function. We can apply linearization constraints at only integer values of x.

2.4.3.1 Relaxation by Linearization Methods for Mixed 0-1 Polynomial Problem

In real-world, there are many problems such as job scheduling and network design which are formulated as mixed 0-1 problems. Non-linear models should be transformed into linear form by using auxiliary constraints and additional 0-1 variables. Many efforts are made to develop and improve previous methods based on reducing the number of extra variables or auxiliary constraints. Some improvement are provided by Watters (1967) who discuss the method that reduce 0-1 polynomial formulation to 0-1 linear formulation. Also for Multi-dimensional Nonlinear Knapsack Problem (MNKP) with a single constraint by considering the fact that the problem is separable is studied in Chen et al. (2006), when

Li et al. (2009) develop a convergent Lagrangian and cut method for solving MNKP.

Kettani and Oral (1990) develop a linearization technique, which has been accepted as the most efficient linearization method. This technique requires the least number of additional continuous variables and auxiliary constraints. However, for mixed integer problems with higher order polynomials terms, it is difficult to extend. A new general method for linearizing different orders of mixed integer problems not previously solved by Kettani and Oral (1990), is proposed by Chang (2000). This model can be applied to polynomial mixed-integer terms that appear in objective function or in constraints. Most techniques for solving polynomial programming problems can only find locally optimal solutions. Li and Chang (1998) propose a method that finds approximately globally optimal solutions for practical polynomial problems. Later, Chang and Chang (2000) modify Li and Chang (1998)'s model by reducing number of variables and auxiliary constraints. After testing the method for different numbers of variables, they conclude that this model uses less CPU time and has fewer iterations compare with Li and Chang (1998) model, while finding same optimal solutions.

Adams and Sherali (1990) proposed a method for solving quadratic problems, which finds lower bounds with a proposed branch and bound scheme which is proven to converge to a globally optimal solution. After that Sherali and Tuncbilek (1992) use a similar approach to solve polynomial programming problems only consisting of continuous variables. The quality of local optimum for both of them is dependent on selected initial solutions. On the other hand, finding the global optimum for many optimization problems is much more important rather than listing different local optima. Both of proposed methods have restrictions when solving practical polynomial problems where it is required to find global optimal solution. For example Adams and Sherali (1990) can only solve problems that have quadratic cross-product terms while Sandgren (1990) can only solve continuous polynomial problems and it is very difficult to use these two methods for solving general polynomial problems since their methods need to generate a huge numbers of constraints to find solution.

To overcome these difficulties, Li (1994) proposed a global approach for solving mixed 0-1 non-linear problems that include convex or separable continuous functions and used a branch and bound method to obtain global optima. As discussed above, method of Li (1994) finds the global optima for general polynomial problems and method proposed by Ghezavati and Saidi-Mehrabad (2011) tries to reduce number of constraints to decrease computation required to find global optima. A computational study by Taha (1972) conjectures that neither non-linear approach nor transformed linear approach has an advantage over the other one, sometimes one of them works better but other times another one works very well.

It is clear that linear 0-1 methods are easier to program and many more techniques are available for linear problems compared with non-linear problems. Glover and Woolsey (1973) have proposed different rules for achieving more economical linear representations of polynomial programming problems that makes possible to replace polynomial cross product terms by continuous variables rather than integer variables.

2.4.4 Heuristics and Metaheuristics

Heuristics methods are used to find approximate solution for NP-hard problems when the exact methods require too much computational time (Polya, 2008). Many optimization problems are NP-hard and heuristic solution methods are required to find satisfactory solutions. Heuristics often get complex and involve many parameters. Due to the dependency on parameters, their effectiveness may increase or decrease. Heuristics are broadly classified as constructive or improvement heuristics.

Constructive heuristics build solutions from their constituting elements rather than improve complete solutions with one element being added to a partial solution at a time. Improvement heuristics start with an initial solution and by perturbation improve the solution. Perturbation is the key for improvement method. General frameworks for building heuristics are called metaheuristics which are solution methods that facilitate an interaction between local improvement procedures and higher level strategies to create a method that can escape from local optima (Glover and Kochenberger, 2003).

2.4.4.1 Local Search

A widely used heuristic is local search, which is often referred to as descent. Local search has become popular to solve complicated real-world optimization problems. It is very important to define a suitable neighbourhood in local search because it determines the search mechanism. A neighbourhood defines the solution search space and demonstrates how attempt are made to reach the best solution.

Local search methods for constraint satisfaction and optimization problems proceed by performing a sequence of local search moves on an initial solution, each improving the value of objective function, until a local optimum is found. In each iteration, a newly found solution is compared with current solution. If it is better, the current solution should be replaced by new solution until no further improvement is found.

Algorithm 1 solves a minimization problem where F is a real-valued function (Burke and Kendall, 2005). In an initialization step, an initial solution is found possibly by applying a constructive heuristic. The main part of the algorithm finds the best solution in neighbourhood and compares the solution with previous solution. If new solution is better, it is considered as the best-known solution and searching proceeds from this new solution; otherwise, the procedure terminates.

Algorithm 1 Steps of Local Search (Descent)

```
Initialization:
Given an initial solution $x$ and a neighbourhood structure $N$
Local search:
Find the best solution $x'$ in the neighbourhood $N(x)$
if $F(x') < F(x)$ then
    Set $x \leftarrow x'$
    Go to Local search
else
    Stop
end
```

Local search often terminates with a local optimum that is only of moderate quality.

One way of trying to avoid this, is to apply Variable Neighbourhood Descent (VND) which performs several local search (descent) with different neighbourhoods until a local optimum for all considered neighbourhoods is achieved.

2.4.4.2 Variable Neighbourhood Descent

Mladenovic and Hansen (1997) propose a new optimization technique is called Variable Neighbourhood Search (VNS). The basic idea of VNS is to define different neighbourhoods to explore solution space and reduce risk of becoming trapped in a local optimum. Hertz and Mittaz (2001) describe another form of VNS which performs several local search (Descent) with different neighbourhoods until a local optimum for all considered neighbourhoods is achieved.

Algorithm 2 describes the steps of VND by considering an initial solution x and different neighbourhoods (Burke and Kendall, 2005). The exploration strategy of neighbourhoods is the key in this algorithm. The aim is to find a direction of steepest descent from x, within a neighbourhood N_k, and move to minimum of $F(x)$ in neighbourhood along that direction; if there is no direction of descent, VND stops; otherwise it is iterated. In other words, we start with the first neighbourhood and compare the result with initial solution. If there is not any improvement, we switch to another neighbourhood; otherwise, we stay with same neighbourhood until no improvement is obtained.

Algorithm 2 Steps of Basic VND

```
Initialization:
Given an initial solution $x$ and neighbourhood structures $N_k = 1, 2, ..., k_{\max}$
Set $k \leftarrow 1$
while $k \leq k_{\max}$ do
    Exploration of neighbourhood: find the best neighbour $x' \in N_k(x)$
    if $F(x') < F(x)$ then
        Set $x \leftarrow x'$ and $k \leftarrow 1$
    else
        Set $k \leftarrow k + 1$
    end
end
```

This particular version of VNS is called Variable Neighbourhood Descent (VND), which is an enhanced local improvement strategy that is usually used as a component in other heuristics (Hu and Raidl, 2006). In VND, changes of neighbourhood are performed in a deterministic way. Several different neighbourhoods are explored with the process iterating over each neighbourhood while improvements are found (applying local search until a local optimum in each neighbourhood is achieved). Only strictly better solutions are accepted after each neighbourhood search (Ruiz and Naderi, 2009).

The process of changing neighbourhood in case of no improvement diversifies the search especially by following a dynamic neighbourhood strategy. If a poor-quality solution is found in one neighbourhood, it could be lead to a better by searching in another neighbourhood. Moreover, a solution that is locally optimal with respect to one neighbourhood is probably not a local optimum in another neighbourhood, so the search strategy performs differently on different neighbourhoods (Blum and Roli, 2003). VND will produces a local optimum with respect to all of neighbourhoods considered, but each neighbourhood may have various local optima that are better than the one found with VND. Thus, a method is needed to explore several local optima within a neighbourhood. One such method is called Iterated Local Search (ILS), that is explained in next section.

2.4.4.3 Iterated Local Search

Iterated Local Search (ILS) is a powerful metaheuristic algorithm and simple and easy to implement. It applies local search to an initial solution until it finds a local optimum, then shakes the local optimum and restarts local search again from that point. Blum and Roli (2003) explain the importance of shaking: too small shaking might not enable an escape from local optimum already found; on the other hand, too large shaking would produce an algorithm similar to random restart local search. Sometimes shaking involves doing n random moves in the local search neighbourhood, where n is determined by experimentation.

Algorithm 3 explains Iterated Local Search (ILS). Given an initial solution x, local search is applied to find the best current solution x'. Then shaking is applied to x', which leads to an intermediate solution x''. Next, local search is applied to x'' and the solution x_{best} is reached. If x_{best} is feasible, it becomes the next element of the walk in x''; otherwise, one returns to x'. This Iterated Local Search (ILS) procedure leads to find a reasonable local optimum as long as shaking is neither too small nor too large (Lourenço et al., 2003).

Algorithm 3 Iterated Local Search

```
Initialization:
Set computation time limit
Given an initial solution $x$, and a neighbourhood structure $N$
Apply local search method to find the best solution $x' \in N(x)$
while computation time < computation time limit do
    By shaking solution $x'$, generate $x''$
    Apply local search with starting solution $x''$ and find the best solution $x_{best}$
    if $F(x_{best}) < F(x')$ then
        Set $x' \leftarrow x_{best}$
    end
    Determine computation time
end
```

Generally, an ILS walk will not be reversible, but this aspect of procedure does not prevent ILS from being very effective in practice. Shaking may take form of performing n random moves in neighbourhood N. Therefore, ILS has only one parameter n that requires setting. However, this may also be a disadvantage in that nature of shaking is such that some parts of the solution space are not explored. Thus, a more elaborate shaking scheme may provide better-quality solutions. Iterated Variable Neighbourhood Descent (IVND), which is explain in next section, partly solves this problem.

2.4.4.4 Iterated Variable Neighbourhood Descent

Iterated Variable Neighbourhood Descent (IVND) combines the strengths of ILS and VND (Lourenço et al., 2003). Firstly, an initial solution x is built. Then an attempt is made to improve the solution x by a VND procedure which finds a local optimum solution x''. Once a local optimum is found, IVND performs shaking. Generate a point x', by picking a random k and a random neighbour, $x' \in N_k(x'')$. A new VND starts and produces a new local optimum. The process of shaking and applying VND is repeated until a computational time limit is reached. The framework of the proposed algorithm for IVND is given in Algorithm 4.

Algorithm 4 Iterated Variable Neighbourhood Descent

```
Initialization:
Set *computation time* limit
Given an initial solution $x$ and set of shaking neighbourhood structures $N_k$ for $k$= 1,2,...,$k_{\max}$
Apply VND method to find the best solution $x''$
Start:
while computation time < computation time limit do
    Shaking: generate a point $x'$ by picking a random $k \in \{1, ..., k_{\max}\}$ and a random neighbour from $x' \in N_k(x'')$
    Apply VND with starting solution $x'$ and find the best solution $x''$
    if $F(x'') < F(x')$ then
        Set $x' \leftarrow x''$
    end
    Determine computation time
end
```

IVND performs well and is competitive relative to other heuristics and metaheuristics. Additionally, IVND is flexible and problem independent, as well as easy to implement compared with some other heuristic methods.

2.4.4.5 Tabu Search

Tabu search (TS), created by Glover (1986), is the most cited metaheuristic. It provides very good quality solutions for solving combinatorial optimization problems, designed to guide other methods to escape the trap of local optimality by using memory structures that characterize the visited solutions. TS begins in same way as local search, proceeding from one solution to another until a chosen termination criterion is met. TS permits moves that worsen from the current objective function value but the moves are chosen from a modified version of neighbourhood N_k where some moves are forbidden.

Short-term and long-term memory structures in TS are responsible for modifying $N_k(x)$. For strategies based on a short-term aspect, $N_{k'}(x)$ is a subset of $N_k(x)$ and tabu classification serves to identify elements of $N_k(x)$ without considering $N_k(x')$. In TS strategies based on a longer-term aspect, $N_k(x')$ may be expanded to include solutions that are not found in $N_k(x)$, such as solutions found and evaluated previously, or identified as high-quality neighbours of these previous solutions, so TS is viewed as a dynamic neighbourhood method. This means that the neighbourhood of $N_k(x)$ is not a static set, but rather is a set that can change according to the history of the search.

TS uses attributive memory, that records information about solution properties (attributes) that change when moving from one solution to another. Recency-based memory and frequency-based are the most common attributive memory. Recency-based memory keeps track of solutions attributes that have changed during the recent past, while frequency-based memory consists of ratios about the number of iterations for which a certain attribute has or has not changed (Glover and Laguna, 1999). TS explores the neighbourhood of each solution as the search progresses. The solutions concluded in a neighbourhood are determined by the use of memory structure, which is known as a tabu list. The tabu list is the short-term memory containing attributes of a set of solutions that have been visited in the past and usually a fixed limited quantity of information about these solutions is recorded.

There are several possibilities for recording information. One could record complete solutions, but this requires a lot of storage and is very expensive to check whether a move is tabu or not. The tabu list helps to keep track of all recently visited solutions and prevent any moves towards them. The most common tabu list is implemented on circular lists of fixed length (Gendreau and Potvin, 2010). In a TS method, tabu lists are sometimes too powerful and may prohibit attractive moves, even when there is no danger of cycling. It is necessary to use an algorithmic device that will allow cancellation of tabu status. This is called an aspiration criterion. When a solution with an objective function value that is better than current best-known solution (while new solution has not been previously visited), the move is allowed even if it is tabu (Gendreau and Potvin, 2010). Algorithm 5 describes the basic TS by implementing a best improvement local search. This version of TS allows the use of multiple neighbourhoods.

Algorithm 5 Tabu Search

```
Initialization:
Set *computation time* limit
Given an initial solution $x$ and set of neighbourhood structures $N_k$ for $k$= 1,2,...,$k_{\max}$
Set $x'' \leftarrow x$
Set Tabu list $\leftarrow \emptyset$
while computation time < computation time limit do
    $X$=$\emptyset$
    $x_0 = x$
    while $x_0 = x$ do
        Apply a local search method and find the best neighbour $x' \in N_1(x) \bigcup ... \bigcup N_{k_{\max}}(x) \setminus X$
        if $x'$ is not tabu, or $x'$ is tabu but satisfies the aspiration criterion then
            Set $x \leftarrow x'$
            if $F(x) < F(x'')$ then
                $x'' \leftarrow x$
            end
        else
            $X = X \bigcup \{x'\}$
        end
    end
    Update Tabu list and aspiration criterion
    Determine computation time
end
```

We can add more algorithmic devices to TS to make it more effective. If all the design choices are suitably chosen, a good solution will usually result. TS has a similar framework with local search but has two differences as follows: some neighbours are not allow to be accepted because they are tabu and if the best neighbour is worse, we still may accept that move.

2.5 Discussion

This chapter presented background materials require to understand a Content Distribution Network (CDN). We show a diagram of CDN network, where a content is distributed to a set of Web server and broadcast over the world for delivering content to clients. Client assignment and cache allocation are considered as two design models in CDN and literature review is written according to these two designs.

Object retrieval and request routing is one of the optimization models for client assignment problem, which is related to data placement of cooperative caching. This model determines a placement of replicated objects between nodes in the network with a given access pattern and try to minimize the average access cost. The objective function expresses total cost of serving requests for all proxies and objects (Baev and Rajaraman, 2001). This model does not take into account any degradation (the quality of service), when the usage of the system is close to the capacity. In Chapter 3, we propose a new model which overcomes the disadvantage of previous models by introducing delay in formulation. The objective is to minimize overall latency of searching the target Web server, subject to the availability of system resources and the traffic pattern.

Another optimization model is objects in a network of caches (Tawarmalani et al., 2009), related to cache allocation problem. The problem analyzes the allocation of objects in a network of caches that collaborate to service requests from customers. Problem is proxy-based caching, because when a requested object is not held by a proxy, a neighbouring proxy that holds that object can deliver the object faster than origin server. Therefore, customer can access to the requested object in a short time. In original problem there is a demand for each object and proxy, but in our propose formulation, we introduce a demand that we can assign to any proxy that holds the object of interest. From the formulation, it is clear that each object can be allocated at most once, that means we cannot put all the objects in the proxy because solution will be dominated. We consider set of proxies, origin server and objects with different sizes. We are looking at simpler model of cache allocation problem which have local cluster of proxies and trying to put an object in each proxy to avoid having to use slower origin server. Essentially all proxies are identical, that might be in different sizes in general. However, for computational testing, we consider all proxies to have the same size. In objective function there is no difference to put the object in which proxy; the cost will be the same.

Solution methodologies are categorized as follows: complexity theory, heuristic and meta-heuristic methods (Local Search (LS), variable Neighbourhood Descent (VND), Iterated Local Search (ILS), Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS)) are categorized according to their advantages and explained alongside of their algorithms also, exact solution methods (Branch and Bound, Bin Packing and Lagrangian Relaxation) and integer programming formulation are explained. In exact solution methods branch an bound, Lagrangian relaxation and linearization methods are explained and mainly are used in this thesis. In next chapter, we will propose a new mathematical formulation for delay function, and start with static data placement problem on a network with no origin server which consists of placing objects to minimize the total expected access cost. The objective is to minimize overall latency of searching Web server, subject to the availability of system resources and the traffic pattern. Lagrangian relaxation and two other linearization will be used to find lower bounds. In metaheuristic, constructive heuristic is introduced to find an initial solution for this problem. Variable Neighbourhood (VND), Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) with two neighbourhoods are suggested to solve the problem.

Chapter 3

Lower Bounding Techniques and Metaheuristics for Object Retrieval and Request Routing Problem

3.1 Introduction

As discussed in the previous chapter, Model (2.3.1) is considered as one of the cost minimization models for the CDN provider. The main aim of a CDN provider is to deliver content to content providers with high performance and high availability at lowest expense. The performance of system tends to degrade when the flow gets closer to the capacity. Sometimes delay is caused by switching between two nodes, when size of object being transferred is large. Therefore, the overall traffic and delay will be the key performance factors of a CDN provider.

We propose a new model which overcomes the disadvantage of previous models by introducing delay in formulation. This model is related to data placement of cooperative caching, which determines a placement of replicated objects between nodes in a network having a given access pattern and aims to minimize the average access cost. The objective is to minimize overall latency of searching the target Web server, subject to the availability of system resources and the traffic pattern.

Dynamically selecting the best proxy for a client with lowest respond time is another scheme that can be considered in CDN, which reduces expenses. From the perspective of the CDN provider, wishing to reduce expenditure is expressed as minimizing total cost, by knowing that the major factor that causes the cost to increase or decrease is a delay function.

We introduce a model with a delay function $\frac{Df_{ij}}{Q_{ij}-f_{ij}}$ associated with the delay of a message carried over the link $(i,j) \in E$, where Q_{ij} is the capacity of the link, D is the unit delay cost of the message that has delay in the network, and f_{ij} is the amount of traffic carried on the link (i,j) per unit of time (Gavish, 1991). A higher investment in the network is the reason that leads to have lower queuing delays and costs. Thus our aim is to create a new design to minimize the total cost of serving requests for all proxies and objects.

Lower and upper bounds are two aspects that are considered in this chapter. Different techniques can be used to find lower bounds for non-linear problems such as Lagrangian relaxation and non-linear programming relaxation. Because the size of our problem is large and constraints are complex, Lagrangian relaxation combined with a decomposition approach is one of the best options to find a lower bound.

Another method is an integer programming formulation based on linearization through perspective cuts by using tangents of the function as valid inequalities to provide a lower bound using auxiliary variables that represent the non-linear terms. Then, to have stronger bounds, a linear segment is used between consecutive integer points in the function rather than a tangent.

Another section of this chapter is about metaheuristics, which are approximate methods, designed for complex optimization problems where exact optimization methods are not efficient (Osman and Laporte, 1996). A metaheuristic guides the heuristic algorithm to efficiently produces high-quality solutions. Metaheuristics may consider a single solution or a collection of solutions at each iteration. There are typically two components in metaheuristics, diversification and intensification. Diversification refers to the exploration of all the areas of the search space, and intensification refers to exploitation which means attempting to find the best solution in an area of the solution space (Blum and Roli, 2003). We propose a heuristic method to solve large problems which are outside the scope of exact methods. The results from applying a heuristic are upper bounds that can be compared with lower bounds from Lagrangian relaxation and linearizations. Through that we can assess the quality of the heuristics that we use.

Section 3.1 is general introduction about the chapter. In Section 3.2, a problem definition for a new formulation with a delay function is proposed. Section 3.3 contains lower bound techniques: Lagrangian relaxation and decomposition, and also an overview of integer programming formulations based on linearization with two different approaches. Section 3.4 describes metaheuristics that search two different neighbourhoods. Also, a constructive heuristic algorithm to find an initial solution for the problem is proposed. Moreover, Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are discussed in this section. In Section 3.5, computational results obtained by implementing Lagrangian relaxation and two different linearizations, and computational results for Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are reported. Grouping of our results is by size, namely small and large instances, and with respect to the number of clients, proxies and objects. Finally, a discussion of the findings in this chapter are given in Section 3.6.

3.2 Problem Definition and Formulation

We consider the complete network $G = (V, E)$, where $V = I \cup J$ is set of nodes and E is set of links. A set of clients is represented by $I \subset V$, and set of proxies by $J \subset V$. Problem (2.3.1) from the literature review is considered. In this model, when traffic gets close to capacity, the performance of the system degrades and causes delay.

According to Gavish (1991), a delay function $\frac{Df_{ij}}{Q_{ij}-f_{ij}}$ is associated with delay when messages are carried over a link $(i,j) \in E$, where Q_{ij} is capacity of the link, D is the unit delay cost of message that has delay in the network and f_{ij} is amount of traffic carried on the link (i,j) per unit of time, and all these are major factors that cause the cost to increase or decrease. Bear in mind that throughout this thesis, the total delay for a client to access an object cannot exceed the QoS requirement.We propose a new mathematical formulation for delay function, and start with static data placement problem on a network with no origin server which consists of placing objects to minimize the total expected access cost (the cost for a client $i \in I$ to access object $k \in K$ from proxy $j \in J$ is $b_k\lambda_{ik}c_{ij}$). The size of all objects that are contained in the proxy $j \in J$ which are requested by client $i \in I$ is considered to be the amount of traffic in the link ($f_{ij} = \sum_{k\in K} b_k x_{ijk}$ for $\forall i \in I, j \in J$). The traffic in the new formulation is constrained to be the capacity of a link. We assume that the capacity Q_{ij} and the size b_k of each object k are integer. A binary variable x_{ijk} indicates if client $i \in I$ is assigned to access a copy of object $k \in K$ stored in proxy j. Also, another binary variable z_{jk} denotes if object $k \in K$ is held in proxy $j \in J$. A new mathematical model for this problem is given below.

$$\min \quad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} b_k\lambda_{ik}c_{ij}x_{ijk} + \sum_{i\in I}\sum_{j\in J}\frac{Df_{ij}}{Q_{ij}-f_{ij}} \tag{3.2.1}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ijk} = 1 \qquad \forall i \in I, k \in K, \tag{3.2.2}$$

$$\sum_{k\in K} b_k z_{jk} \le s_j \qquad \forall j \in J, \tag{3.2.3}$$

$$f_{ij} = \sum_{k\in K} b_k x_{ijk} \qquad \forall i \in I, j \in J, \tag{3.2.4}$$

$$x_{ijk} \le z_{jk} \qquad \forall i \in I, j \in J, k \in K, \tag{3.2.5}$$

$$f_{ij} \le Q_{ij} - 1 \qquad \forall i \in I, j \in J, \tag{3.2.6}$$

$$z_{jk} \in \{0,1\} \qquad \forall j \in J, k \in K, \tag{3.2.7}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i \in I, j \in J, k \in K. \tag{3.2.8}$$

The new formulation presented above is a non-linear integer programming model. The objective function has two parts: first term is the cost for clients to access objects from a proxy and second term is the delay function.

Constraint (3.2.2) indicates that each client and object pair must be assigned to exactly one proxy server. Constraint (3.2.3) relates to the limited capacity of each proxy $j \in J$. Constraint (3.2.4) refers to the fact that amount of traffic in the link (i, j) must be equal to the size of all objects that are contained in proxy $j \in J$ which are requested by client $i \in I$. Constraint (3.2.5) implies that an assignment to a proxy can only be made if that specific proxy is holding requested object. Constraint (3.2.6) implies that the amount of traffic carried on the link is less than the capacity of the link by assuming that the capacity and the size of each object are integer (because size of each object b_k, amount of traffic f_{ij} and capacity of the link Q_{ij} are integer, so we can write the constraint as $f_{ij} \leq Q_{ij} - 1$). Constraints (3.2.7) and (3.2.8) denote the integrality restriction of the decision variables.

3.3 Lower Bound Techniques

In this section, two different methods are explained. Because the size of the proposed problem is large and constraints are complicated, Lagrangian relaxation is an appropriate technique to find lower bounds. The second technique is linearization through perspective cuts by using tangents of the function as valid inequalities to provide lower bounds for the auxiliary variables that represent the non-linear term. Also to improve the bounds, linear segment between two consecutive integer points are recommended instead of tangents.

3.3.1 Lagrangian relaxation and Decomposition

Lagrangian relaxation and decomposition approaches are widely used to address problems with large-sized formulations with complex constraints. For this reason, the formulation should be partitioned into sub-problems with smaller size that can be solved easily, (Fisher, 2004). The lower bound is sometimes exact and an optimal solution can be found. The general idea behind Lagrangian relaxation is to remove constraints that are interfering by incorporating a violation of that constraints in the objective function, hopefully producing a problem that is easy to solve. The aim is to find decompositions where the non-linearities present in the problem are easy to solve (Bektaş et al., 2009).

3.3.1.1 Lagrangian Relaxation

For Lagrangian relaxation, we choose Constraints (3.2.4) and (3.2.5) and introduce multipliers α_{ij} and β_{ijk}, where α_{ij} is unrestricted and $\beta_{ijk} \geq 0$. The resulting formulation is reported below:

$$\begin{aligned}\min \quad & \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} b_k\lambda_{ik}c_{ij}x_{ijk} + \sum_{i\in I}\sum_{j\in J}\frac{Df_{ij}}{Q_{ij}-f_{ij}} + \\ & \sum_{i\in I}\sum_{j\in J}\alpha_{ij}(f_{ij} - \sum_{k\in K} b_k x_{ijk}) + \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}\beta_{ijk}(x_{ijk} - z_{jk})\end{aligned} \tag{3.3.1}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ijk} = 1 \qquad \forall i \in I, k \in K, \tag{3.3.2}$$

$$\sum_{k\in K} b_k z_{jk} \leq s_j \qquad \forall j \in J, \tag{3.3.3}$$

$$f_{ij} \leq Q_{ij} - 1 \qquad \forall i \in I, j \in J, \tag{3.3.4}$$

$$z_{jk} \in \{0,1\} \qquad \forall j \in J, k \in K, \tag{3.3.5}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i \in I, j \in J, k \in K. \tag{3.3.6}$$

The relaxed problem decomposes into three sub-problems with respect to x, f and z under the name of LRX, LRF and LRZ respectively.

$$(LRX) \qquad \min \quad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}(b_k\lambda_{ik}c_{ij} - \alpha_{ij}b_k + \beta_{ijk})x_{ijk} \tag{3.3.7}$$

$$\begin{aligned}\text{s.t.} \quad & \sum_{j\in J} x_{ijk} = 1 \qquad \forall i \in I, k \in K, \\ & x_{ijk} \in \{0,1\} \qquad \forall i \in I, j \in J, k \in K.\end{aligned}$$

$$(LRF) \qquad \min \qquad \sum_{i\in I}\sum_{j\in J}(\frac{D}{Q_{ij}-f_{ij}}+\alpha_{ij})f_{ij} \tag{3.3.8}$$

$$\text{s.t.} \quad 0 \le f_{ij} \le Q_{ij}-1.$$

$$(LRZ) \qquad \max \qquad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}\beta_{ijk}z_{jk} \tag{3.3.9}$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{k\in K} b_k z_{jk} \le s_j \qquad \forall i \in I, j \in J, \\ & z_{jk} \in \{0,1\} \qquad \forall j \in J, k \in K. \end{aligned}$$

Note that LRZ is written as a maximization problem because the origin of $\beta_{ijk}z_{jk}$ in (3.3.1) is negative. In the next sub-section, we show how each problem can be solved efficiently.

3.3.1.2 Solving LRX

After relaxation, LRX reduces to an assignment problem. Let $\widehat{C}_{ijk}$=$(b_k\lambda_{ik}c_{ij}-\alpha_{ij}b_k+\beta_{ijk})$:

$$\min \qquad \sum_{i\in I}\sum_{j\in J}\sum_{k\in K}\widehat{C}_{ijk}x_{ijk} \tag{3.3.10}$$

$$\text{s.t.} \quad \sum_{j\in J} x_{ijk} = 1 \qquad \forall i \in I, k \in K, \tag{3.3.11}$$

Problem (3.3.10) decomposes into a sub-problem for each pair i and k. For given i and k, let $\widehat{x}_j = x_{ijk}$ and $\widehat{C}_{ijk} = \widehat{C}_j$ and the sub-problem becomes:

$$\min \qquad \sum_{j\in J}\widehat{C}_j\widehat{x}_j \tag{3.3.12}$$

$$\text{s.t.} \quad \sum_{j \in J} \widehat{x}_j = 1, \tag{3.3.13}$$

$$\widehat{x}_j \in \{0,1\} \qquad \forall j \in J. \tag{3.3.14}$$

Therefore, the optimal solution for LRX is as follow:

$$\hat{x}_j = \begin{cases} 1 & \text{for} \quad j = j^*, \\ 0 & \text{for} \quad j \neq j^*. \end{cases} \tag{3.3.15}$$

Algorithm 6 explains the step-by-step the solution process for this assignment problem.

Algorithm 6 Solving the Assignment Problem (LRX)

```
Initialization :
Set $\widehat{C}_{ijk} = (b_k\lambda_{ik}c_{ij} - \alpha_{ij}b_k + \beta_{ijk}) \quad \forall i \in I, j \in J, k \in K$
for $i \in I$ do
    for $k \in K$ do
        Find $j^*$ such that $\widehat{C}_{ij^*k}= \min_{j\in J} \widehat{C}_{ijk}$
        Set $x_{ij^*k}$ =1, and $x_{ijk}$=0 $\forall j \in J\backslash\{j^*\}$
    end
end
```

3.3.1.3 Solving LRF

The optimal solution for the sub-problem LRF is calculated as follows:
Let $F(f_{ij}) = \frac{Df_{ij}}{Q_{ij}-f_{ij}} + \alpha_{ij}f_{ij}$

Proposition 3.1: The value f^*_{ij} that minimizes $F(f_{ij})$ is given by:

$$f^*_{ij} = \begin{cases} \max\{\mathrm{Q_{ij}} + \frac{\sqrt{\text{-D}\alpha_{\mathrm{ij}}\mathrm{Q_{ij}}}}{\alpha_{\mathrm{ij}}}, 0\} & \text{if } \alpha < 0 \\ 0 & \text{if } \alpha \geq 0 \end{cases}$$

Proof. The derivative of $F(f_{ij})$ is $\frac{dF}{df_{ij}} = \frac{DQ_{ij}}{(Q_{ij}-f_{ij})^2} + \alpha_{ij}$. Setting $\frac{dF}{df_{ij}} = 0$, yields to the quadratic equation $\alpha_{ij}f^2_{ij} - 2\alpha_{ij}Q_{ij}f_{ij} + (\alpha_{ij}Q^2_{ij} + DQ_{ij}) = 0$. Using the discriminant $(-4D\alpha_{ij}Q_{ij})$ to find the roots, results in $R_1 = Q_{ij} - \frac{\sqrt{-D\alpha_{ij}Q_{ij}}}{\alpha_{ij}}$ and $R_2 = Q_{ij} + \frac{\sqrt{-D\alpha_{ij}Q_{ij}}}{\alpha_{ij}}$. If $\alpha_{ij} \geq 0$, $F(f_{ij})$ is monotonically increasing and is therefore minimized in the range $0 \leq f^*_{ij} \leq Q_{ij} - 1$, with $f^*_{ij} = 0$. Alternatively suppose that $\alpha_{ij} < 0$, then $R_1 =$

$Q_{ij} - \frac{\sqrt{-D\alpha_{ij}Q_{ij}}}{\alpha_{ij}} > Q_{ij}$ and the resulting solution is infeasible. If $R_2 \geq 0$, then $f_{ij} = R_2$ minimizes $F(f_{ij})$; otherwise, $f_{ij} = 0$ minimizes $F(f_{ij})$. $\square$

3.3.1.4 Solving LRZ

LRZ is a collection of knapsack problems, one for each $j \in J$. Knapsack problems are commonly tackled using dynamic programming. Dynamic programming is a technique which appears to be a useful tool in many areas of operational research. Dynamic programming can be useful, because an optimal solution of our problem is a combination of optimal solutions of sub-problems according to Kellerer et al. (2004a). In Algorithm 7, we add objects iteratively to problem as is usual in dynamic programming. The basic idea of dynamic programming applied to the knapsack problem.

Suppose that an optimal solution of knapsack problem for a subset of all objects and capacities are already computed, and now we want to add one object to this subset and check if the optimal solution needs to be changed. This procedure of adding an object is iterated until all objects are considered and a final optimal solution is found. We introduce the following sub-problem that contains set of objects $\{1, ..., k\}$ for $k = 1, 2, ..., |K|$ and knapsack capacity $r \leq s_j$ for $r = 0, 1, ..., s_j$. We initialize by setting $z_0(r) = 0$ for $j = 0, 1, ..., s_j$ consider a given $j \in J$.

The optimal solution value is $z_k(r)$ for objects $k \in K$ and capacity s_j when b_k is the weight of the object k and v_{jk} is the profit of object k where $v_{jk} = \sum_{i \in I} \beta_{ijk} z_{jk}$. If $z_{k-1}(r)$ is known for all capacity values $r = 0, 1, ..., s_j$, then additional item k can be considered by applying the following recursive formula (Kellerer et al., 2004b).

$$z_k(r) = \begin{cases} z_{k-1}(r) & \text{if} \quad b_k > s_j \\ \max z_{k-1}(r), z_{k-1}(s_j - b_k) + v_{jk} & \text{if} \quad b_k \leq s_j \end{cases} \tag{3.3.16}$$

For $r = 0, 1, ..., s_j$ and $k = 1, 2, ..., |K|$. If the considered object is too large for the knapsack ($s_j < b_k$) then the optimal value does not change, but if object k fits into the knapsack ($s_j \geq b_k$) there are two possibilities: either object k is not packed into the knapsack so the optimal value is unchanged as before or object k is added to the knapsack and this will affect the optimal solution value and reduce the remaining capacity of the knapsack (the appointed capacity will be $z_{k-1}(s_j - b_k)$). The optimal solution value is given by $\max r = 0, 1, ..., s_j, Z_{|K|}(r)$, and the corresponding obtained solution is found by backtracking.

Algorithm 7 Dynamic Programming (LRZ)

```
Initialization:
for $j \in J$ do
    $s_j \leftarrow$ Capacity of knapsack
    Set $v_{jk} \leftarrow \sum_{i \in I} \beta_{ijk} z_{jk} \qquad \forall k \in K$
    for $r$=0 to $s_j$ do
        Set $z_{j0}(r) \leftarrow 0$
    end
    for $k$=1 to $K$ do
        for $r$=0 to $b_{k-1}$ do
            $z_{jk}(r) = z_{jk-1}(r)$
        end
        for $r$=$b_k$ to $s_j$ do
            if $z_{jk-1}(r - b_k) + v_{jk} > z_{jk-1}(r)$ then
                $z_{jk}(r) = z_{jk-1}(r - b_k) + v_{jk}$
            else
                $z_{jk}(r) = z_{jk-1}(r)$
            end
        end
    end
    Find the optimal solution value $\max_{r=0,...,s_j} z_{jk}(r)$, and identify the corresponding values $z_{jk}, \forall k \in K$
end
```

3.3.2 Solving the Relaxed Problem

After solving each sub-problem and finding different values for x_{ijk}, f_{ij} and z_{jk} we substitute them to find each sub-gradient so that α_{ij} and β_{ijk} can be updated. As we have two different relaxed constraints, we initialized a sub-gradient for each. Thus, $\delta_{ij} = f_{ij} - \sum_{k \in K} b_k x_{ijk}$ is the sub-gradient for multiplier α_{ij} and $\gamma_{ijk} = x_{ijk} - z_{jk}$ is the sub-gradient for multiplier β_{ijk}.

To upgrade multipliers, we need to calculate step lengths for updating α_{ij} and β_{ijk} . The step length for α_{ij} is computed as $s_\alpha = \lambda_h \frac{UB-LB}{\|\delta\|}$ and for β_{ijk} is computed as

$s_\beta = \lambda_h \frac{UB-LB}{\|\gamma\|}$, where the norm of sub-gradient δ is $\|\delta\|$, the norm of sub-gradient γ is $\|\gamma\|$, UB is an upper bound, LB is the current lower bound and λ_h is a convergence parameter that satisfied $0 < \lambda_h \leq 2$.

We set the convergence parameter $\lambda_h = 1$ initially and divided by 0.5 if there is not any improvement in the best known lower bound for five consecutive iterations. After finding step lengths, we next update multipliers α_{ij} and β_{ijk} using: $\alpha_{ij}^{n+1} = \alpha_{ij}^n + \delta s_\alpha$ and $\beta_{ijk}^{n+1} = \beta_{ijk}^n + \gamma s_\beta$, where n is the iteration number. Algorithm 8 explains step-by-step solution process for this relaxed problem.

Algorithm 8 Sub-gradient Optimisation Algorithm

```
Initialization:
Set *iteration limit*
Set $\alpha_{ij}^1 \leftarrow 0$
Set $\beta_{ijk}^1 \leftarrow 0$
Set $n \leftarrow 1$
while $n <$ *iteration limit* do
    Solve (3.3.1)
    $\delta_{ij} \leftarrow f_{ij} - \sum_{k \in K} b_k x_{ijk} \quad \forall i \in I, j \in J$
    $\gamma_{ijk} \leftarrow x_{ijk} - z_{jk} \quad \forall i \in I, j \in J, k \in K$
    Update $s_\alpha$ and $s_\beta$
    Update multiplier as $\alpha_{ij}^{n+1} = \alpha_{ij}^n + \delta s_\alpha$
    Update multiplier as $\beta_{ijk}^{n+1} = \beta_{ijk}^n + \gamma s_\beta$
    Set $n \longleftarrow n + 1$
end
```

3.3.3 Linearization Schemes

In Model (3.2.1) the term $\sum_{i \in I} \sum_{j \in J} \frac{D f_{ij}}{Q_{ij} - f_{ij}}$ is convex. So, two different linearization techniques are proposed in this section to solve this model. In the first scheme perspective cuts are considered by using tangents of the function as valid inequalities to provide lower bounds for the auxiliary variables that represent the non-linear term. In second scheme to improve the bounds, linear segment between two consecutive integer points are recommended instead of tangents.

3.3.3.1 Scheme I

We focus on non-polynomial function with all integer decision variables, only if the size of object (b_k) is integer. To find a lower bound through perspective cuts, tangents of the function are used as valid inequalities to provide a lower bound for the auxiliary variables representing the non-linear terms (Erdoğan, 2012). As part of the linearization scheme, we define w_{ij} as an auxiliary variable and q_{ij} as an integer point. We define function $g_{ij}(f) = \frac{Df_{ij}}{Q_{ij}-f_{ij}}$ that linearization constraints are established. We construct the following valid inequalities by applying linearization approach of Erdoğan et al. (2010):

$$w_{ij} \geq \frac{dg_{ij}(q_{ij})}{df_{ij}} f_{ij} + g_{ij}(q_{ij}) - \frac{dg_{ij}(q_{ij})}{df_{ij}} q_{ij}, \qquad \forall q_{ij} \in [0, ..., Q_{ij}-1], \forall i \in I, j \in J. \tag{3.3.17}$$

The linearized version of (3.2.1) is given as below:

$$\min \sum_{i\in I} \sum_{j\in J} \sum_{k\in K} b_k \lambda_{ik} c_{ij} x_{ijk} + \sum_{i\in I} \sum_{j\in J} w_{ij} \tag{3.3.18}$$

$$\text{s.t.} \qquad w_{ij} \geq \frac{DQ_{ij}}{(Q_{ij}-q_{ij})^2}(f_{ij}-q_{ij}) + \frac{Dq_{ij}}{Q_{ij}-q_{ij}}, \forall q_{ij} \in [0, ..., Q_{ij}-1], \forall i \in I, j \in J. \tag{3.3.19}$$

3.3.3.2 Scheme II

To improve the result and find better lower bound, another linearization is considered. So, instead of tangent, the linear segment between two consecutive integer points is considered to yield stronger bounds. Suppose that the coordinates of these points are (q_{ij}, $h(q_{ij})$) and ($q_{ij}+1$, $h(q_{ij}+1)$). The line passing through these two integer points is used as the linearization constraint. We define $h(q_{ij})$ as:

$h(q_{ij}) = \frac{Dq_{ij}}{Q_{ij}-q_{ij}} \qquad \forall q_{ij} \in [0, ..., Q_{ij}-1], \forall i \in I, j \in J$

The slope for the segment which passes through two points q_{ij} and $q_{ij}+1$ is:

$\frac{h(q_{ij}+1)-h(q_{ij})}{q_{ij}+1-q_{ij}} = h(q_{ij}+1) - h(q_{ij}),$

So a new inequality for auxiliary variable w_{ij} is:

$$w_{ij} \geq (h(q_{ij}+1) - h(q_{ij}))f_{ij} + h(q_{ij}) - (h(q_{ij}+1) - h(q_{ij}))q_{ij}, \quad \forall q_{ij} \in [0, ..., Q_{ij}-2], \forall i \in I, j \in J. \tag{3.3.20}$$

After simplifying (3.3.20), we have a new equation as below:

$$w_{ij} \geq (h(q_{ij}+1) - h(q_{ij}))\,(f_{ij} - q_{ij}) + h(q_{ij}), \forall q_{ij} \in [0, ..., Q_{ij}-2], \forall i \in I, j \in J. \tag{3.3.21}$$

The new version of linearization with new constraint is given as below:

$$\min \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} b_k \lambda_{ik} c_{ij} x_{ijk} + \sum_{i \in I} \sum_{j \in J} w_{ij} \tag{3.3.22}$$

$$\text{s.t.} \qquad w_{ij} \geq \frac{DQ_{ij}}{(Q_{ij} - q_{ij} - 1)(Q_{ij} - q_{ij})} f_{ij} - \frac{Dq_{ij}^2 + Dq_{ij}}{(Q_{ij} - q_{ij} - 1)(Q_{ij} - q_{ij})}, \quad \forall q_{ij} \in [0, ..., Q_{ij}-2], \forall i \in I, j \in J, \tag{3.3.23}$$

3.3.4 Comparison Scheme I & Scheme II

Proposition 1. Linearization scheme II improves compared to linearization scheme I.

Proof. Linearization constraint from scheme I as below:

$$w_{ij} \geq \frac{dg_{ij}(q_{ij})}{df} f_{ij} + g_{ij}(q_{ij}) - \frac{dg_{ij}(q_{ij})}{df} q_{ij}, \qquad \forall q_{ij} \in [0, ..., Q_{ij}-1], \forall i \in I, j \in J. \tag{3.3.24}$$

Linearization constraint from scheme II as below:

$$w_{ij} \geq (h(q_{ij}+1) - h(q_{ij}))\,(f_{ij} - q_{ij}) + h(q_{ij}),$$
$$\forall q_{ij} \in [0, ..., Q_{ij}-2], \forall i \in I, j \in J.$$

The aim is to prove linearization constraint from scheme II dominates the linearization constraint of scheme I. We use Constraints (3.3.19) from scheme I and (3.3.23) from scheme II and we want to establish as below:

$$\frac{DQ_{ij}}{(Q_{ij}-q_{ij})^2}(f_{ij}-q_{ij})+\frac{Dq_{ij}}{Q_{ij}-q_{ij}} \leq$$
$$\frac{DQ_{ij}}{(Q_{ij}-q_{ij}-1)(Q_{ij}-q_{ij})}f_{ij}-\frac{Dq_{ij}^2+Dq_{ij}}{(Q_{ij}-q_{ij}-1)(Q_{ij}-q_{ij})},$$

$$\Rightarrow \left(\frac{DQ_{ij}}{(Q_{ij}-q_{ij}-1)(Q_{ij}-q_{ij})}-\frac{DQ_{ij}}{(Q_{ij}-q_{ij})^2}\right)(f_{ij}-q_{ij}) \leq 0,$$

$$\Rightarrow (f_{ij}-q_{ij})\frac{DQ_{ij}}{(Q_{ij}-q_{ij}-1)(Q_{ij}-q_{ij})^2} \leq 0.$$

D, Q_{ij} and $(Q_{ij}-q_{ij})^2$ are positive. We need to prove $(f_{ij}-q_{ij}) \leq 0$ and $(Q_{ij}-q_{ij}-1) \geq 0$. If $(f_{ij}-q_{ij}) \leq 0$ then $f_{ij} \leq q_{ij}$ which is correct, according to the constraint $f_{ij} \leq Q_{ij}-1$. If $(Q_{ij}-q_{ij}-1) \geq 0$, then $Q_{ij} \geq q_{ij}+1$ which is right because $q_{ij} \in [0, ..., Q_{ij}-1]$. $\square$

3.4 Metaheuristics

Optimization algorithms can be broadly categorized into two different levels: exact algorithm and heuristics. Exact algorithms guarantee to find an optimal solution, but in heuristics there is no guarantee and sometimes the solution is worse than optimal solution. Exact algorithms not only have to allocate this solution in the solution space but also have to prove that the solution is optimal (Sörensen, 2013). For this purpose, an exact algorithm must consider implicitly or explicitly every single solution in the solution space. In this section, two different local search neighbourhoods are considered: swap and remove-insert. Also, a constructive heuristic is introduced to find an initial solution. Moreover, Variable Neighbourhood Descent (VND), Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are introduced to find upper bounds. The main objective function (3.2.1) is defined as F for all the algorithms in this chapter. These are three associated constraints as below.

- Capacity: means placing objects in each proxy and checking the capacity of each proxy according to $\sum_{k\in K} b_k z_{jk} \leq s_j \quad \forall j \in J$,
- Assignment: indicates that each client must be assigned to exactly one proxy server,

due to $\sum_{j \in J} x_{ijk} = 1 \qquad \forall i \in I, k \in K,$

- Flow: checks the capacity of the link for all the objects that can pass through the link by checking the constraint $f_{ij} \leq Q_{ij} - 1 \qquad \forall i \in I, j \in J$.

If placement and flow constraints are violated, the assignments should be changed. When placement, assignment and flow are satisfied, solution is found. In the Algorithm 9 the method of computing the value of x_{ijk} is explained.

Algorithm 9 The Method of Computing the Value of x_{ijk}

```
$F(O) \leftarrow$ Objective function value of problem (3.2.1)
Initialization:
Set $\sum_{j \in J} x_{ijk} = 1 \qquad \forall i \in I, k \in K$
if $x_{ijk} \leftarrow 1$ then
    Set $z_{jk} \leftarrow 1$
else
    Set $z_{jk} \leftarrow 0$
end
for $i$=1 to $|I|$ do
    for $j$=1 to $|J|$ do
        for $k$=1 to $|K|$ do
            Calculate $f_{ij} = \sum_{k \in K} b_k x_{ijk}$
            if $b_k z_{jk} \leq s_j$ and $f_{ij} \leq Q_{ij} - 1$ violated then
                Back to initialization and do another assignment
            else
                Calculate $F(O)$
            end
        end
    end
end
```

Algorithm 9 shows the way of computing x_{ijk}. Randomly pick one of the x_{ijk} and try random solution, if it works we keep x_{ijk}; otherwise, we change another randomization. If we cannot find any solution with this assignment, the new assignment should be applied.

3.4.1 Neighbourhoods

Local search for combinatorial optimization is performed by evaluating changes in a solution by performing moves in a given neighbourhood. When the value of objective function cannot be improved, then the solution is identified as a local optimum. Some heuristics or metaheuristics try to avoid being trapped in the local optimum by changing neighbourhood in the search (Mladenović and Hansen, 1997). Swap and remove-insert are two different neighbourhoods employed in this chapter. The main reason to use these two neighbourhoods is to explore the search space in order to improve our solutions and find near-optimal solutions. An effective method is obtained by changing these neighbourhoods within the local search algorithm, choosing one of the neighbourhoods until no more improvement is obtained and then switching to the other neighbourhood. We repeat this procedure until no more improvement is found. In both algorithms F_{best} is the value of the best solution, and initially set to infinity.

3.4.1.1 Swap

Swap changes the placement of an object in a proxy with placement of another object in a different proxy, while respecting the capacities of both proxies. We look at all possible pair of proxies with any pair of objects that is already in that proxies and swap them, after checking the capacity of each proxy.

Algorithm 10 Swap Algorithm

```
$F_{best} = \infty$
$F(O) \leftarrow$ Objective function value (3.2.1) for sets $O_j$ $(j \in J)$
Initialization:
Set $O_j \leftarrow$ Set of objects in proxy $j$
Set $\bar{O}_j \leftarrow$ Temporary set of objects in proxy $j$
Set $O_j^* \leftarrow$ Best set of objects in proxy $j$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k \qquad \forall j \in J$
for $j, j' \in J \qquad j < j'$ do
    for $k \in O_j$ do
        for $k' \in O_{j'}$ do
            $\bar{O}_j \leftarrow (O_j \backslash \{k\}) \bigcup \{k'\}$
            $\bar{O}_{j'} \leftarrow (O_{j'} \backslash \{k'\}) \bigcup \{k\}$
            $\bar{B}_j = B_j - b_k + b_{k'}$
            $\bar{B}_{j'} = B_{j'} - b_{k'} + b_k$
            if $\bar{B}_j \leq s_j; \quad and \quad \bar{B}_{j'} \leq s_{j'}; \quad and \quad F(\bar{O}) < F_{best}$; then
                Set $F_{best} \leftarrow F(\bar{O})$
                $O_j^* = \bar{O}_j \qquad O_{j'}^* = \bar{O}_{j'}$
                $O_{\hat{j}}^* = O_{\hat{j}} \qquad \hat{j} \in J \backslash \{j, j'\}$
            end
        end
    end
end
```

Algorithm 10 explains the swap neighbourhood. All objects in the proxy j are considered as a set (O_j). Before an object is removed from a proxy, capacity needs to be checked by considering the size of new object. If the size of objects are less than the residual capacities of the proxies then swap can be performed; otherwise, we should swap another pair of objects.

3.4.1.2 Remove-Insert

One of the local search operators is remove-insert. By inserting a new object, residual capacity of proxy is compared with size of new object. The action can be completed if capacity allows this move; otherwise, another object should be considered.

Algorithm 11 Remove-Insert Algorithm

```
$F_{best} = \infty$
$F(O) \leftarrow$ Objective function value (3.2.1) for sets $O_j$ $(j \in J)$
Initialization:
Set $O_j \leftarrow$ Set of objects in proxy $j$
Set $\bar{O}_j \leftarrow$ Temporary set of objects in proxy $j$
Set $O^*_j \leftarrow$ Best set of objects in proxy $j$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k \qquad \forall j \in J$
for $j \in J$ do
    for $k \in O_j$ do
        $\bar{B}_j = B_j - b_k$
        $\bar{O}_j = (O_j \backslash \{k\})$
        for $k' \in K \backslash \{O_j\}$ do
            if $\bar{B}_j + b_{k'} \leq s_j$ then
                $\bar{O}_j = \bar{O}_j \bigcup \{k\}$
                if $F(\bar{O}) < F_{best}$ then
                    Set $O^*_j \leftarrow \bar{O}_j$
                    Set $O^*_{\hat{j}} \leftarrow O_{\hat{j}} \qquad \hat{j} \in J \backslash \{j\}$
                    $F_{best} = F(\bar{O})$
                end
            end
        end
    end
end
```

Algorithm 11 explains remove-insert procedure clearly when all objects in the proxy j are considered as a set (O_j). After checking capacity of proxy, object k is removed from proxy j, then object k' is inserted in proxy j. Note that object k' was not in the proxy

j. If the solution of temporary set of objects in proxy j is better than F_{best}, then we can consider solution of temporary set as the best found solution so far and continue.

3.4.2 Constructive Heuristic Algorithm to Find Initial Solutions

Algorithm 12 explains how a random initial solution can be generated. Two different phases are considered in the algorithm. In the first phase, objects from origin server, one by one will be allocated in one of the proxies randomly. This allocation depends on the size of each object and capacity of receiving proxy. In the second phase, to maximize the usage of remaining capacity of each proxy, we pick random object from origin server and allocate to the random proxy. This phase has limited flexibility of matching objects and proxies.

Algorithm 12 Constructive Heuristic Algorithm to Find Initial Solution

```
$F(O) \leftarrow$ Objective function value of problem (3.2.1) for set $O_j$ $(j \in J)$
Initialization:
Set $O_j \leftarrow \emptyset$    $\forall j \in J$
Set $B_j \leftarrow 0$    $\forall j \in J$
for $k \in K$ do
    Set $\bar{J} \leftarrow J$
    while $k \notin O_1 \bigcup ... \bigcup O_{|J|}$ do
        Randomly pick a proxy $j \in \bar{J}$
        Set $\bar{J} \leftarrow \bar{J} \backslash \{j\}$
        if $b_k \leq s_j - B_j$ then
            $O_j = O_j \bigcup \{k\}$
            $B_j = B_j + b_k$
        end
    end
end
for $j \in J$ do
    Set $\bar{K} = K \backslash O_j$
    while $\bar{K} \neq \emptyset$ do
        Randomly pick an object $k \in \bar{K}$
        Set $\bar{K} \leftarrow \bar{K} \backslash \{k\}$
        if $b_k \leq s_j - B_j$ then
            $O_j \leftarrow O_j \bigcup \{k\}$
            $B_j \leftarrow B_j + b_k$
        end
    end
end
Calculate $F(O)$
```

Note that each object can be repeatedly assumed to different proxies as long as repetition is not permitted within the same proxy. Finally, each object will find the space allocated in a proxy. All the objects in each proxy are considered as a set (O_j) which is empty at starting point.

3.4.3 Variable Neighbourhood Descent (VND)

Algorithm 13 describes the steps of Variable Neighbourhood Descent (VND). Two neighbourhood structures are considered: swap neighbourhood (N_1) and remove-insert neighbourhood (N_2). An initial solution O is built by the constructive heuristic method from Algorithm 12.

Algorithm 13 Variable Neighbourhood Descent (VND)

```
$F(O) \leftarrow$ Objective function value of problem (3.2.1) for sets $O_j$ from Algorithm 12
Initialization:
Given an initial solution $O$ and a neighbourhood structure $N_1$ and $N_2$, where $N_1$=swap, $N_2$=remove-insert
$k_{\max}$=2
Set $k \leftarrow 1$
while $k \leq k_{\max}$ do
    Find the best neighbour $\bar{O} \in N_k(O)$
    if $F(\bar{O}) < F(O)$ then
        Set $O \leftarrow \bar{O}$ and $k \leftarrow 1$
    else
        Set $k \leftarrow k + 1$
    end
end
```

This algorithm considers the first neighbourhood. If the new solution is better than the previous solution, then this solution is accepted as the best known solution, and the search continues with the same neighbourhood; otherwise, change to the next neighbourhood and attempt to find an improvement.

3.4.4 Iterated Variable Neighbourhood Descent (IVND)

Iterated Variable Neighbourhood Descent (IVND) is a metaheuristic for solving optimization problem and integrates the strengths of ILS and VND (Lourenço et al., 2003). We first attempt to determine the best number of shaking. It is essential to note that n represents the number of shaking. For example, if $n = 4$, represents a move that relocate one, two, three and four objects at a time respectively. Note that the solution of each n is obtained based on the objective function value. More details regarding to n are explained later. An initial solution O is built by a constructive heuristic method. Variable Neighbourhood Descent (VND) is applied to an initial solution until it finds a local optimum solution O'. Once a local optimum is found, IVND performs shaking. Generate a point O'' , by picking a random n and swap neighbourhood. Just to remind that only swap neighbourhood is considered as a shaking neighbourhood. The process of shaking and applying VND is repeated until a computational time limit is reached. The framework of the proposed algorithm for IVND is given in Algorithm 14. Stopping criterion is usually is maximum computing time since the last improvement, or maximum number of iteration. In Algorithm 14, the limit for computational time is considered for stopping condition.

Algorithm 14 Iterated Variable Neighbourhood Descent (IVND)

```
Initialization:
Given an initial solution $O$ and a shaking neighbourhood structure $N = N_{Swap}$
Apply Algorithm 13 to find the best solution $O''$
Set $O \leftarrow O'$
while computation time < computation time limit do
    Shaking: Generate a point $O'$ by applying $n$ random moves from neighbourhood $N$ in succession to $O''$
    Generate a solution $O''$ by applying Algorithm 13 to $O'$
    if $F(O'') < F(O')$ then
        Set $O' \leftarrow O''$
    end
    Determine computation time
end
```

Bear in mind that different moves have different complexity. If moves involve too many elementary changes, the resulting heuristic may be very slow and often takes more time than an exact method.

3.4.5 Tabu Search

Tabu Search (TS) is another metaheuristic that provides very good quality solutions for solving optimization problems, designed to guide other methods to escape the trap of local optimality by using memory structures based on visited solutions, (Glover, 1986). In TS, instead of recording full solutions, attribute memory structures are used. Usually attribute memory are based on recording information about solution properties (attribute) that change in moving from one solution to another. If a potential solution has been previously visited within a certain period or if it has violated a rule, it is marked as "tabu" (forbidden), so the algorithm does not consider that solution again. The stopping criterion is usually is maximum computing time since the last improvement, or maximum number of iteration.

TS uses neighbourhood search procedure to iteratively moves from one potential solution to an improved solution in the same neighbourhood until a stopping criterion has been satisfied. TS explores the neighbourhood of each solution as the search progress. The solutions admitted to the new neighbourhood are determined by the tabu list. The tabu list is a short-term set of solutions that have changed by moving from one solution to another. In Algorithm 15, an initial solution is the current best known solution from Algorithm 12. Swap (Algorithm 10) and remove-insert (Algorithm 11) are considered as two different neighbourhoods to find the best non-tabu solutions.

Algorithm 15 Tabu Search

```
Initialization:
Set *computation time limit*
Given an initial solution $O$ and set of neighbourhood structures $N_k$, with $N_1 = N_{Swap}, N_2 = N_{Remove}$
$k_{\max}$=2
Set $\hat{O} \leftarrow O$
Set Tabu list $\leftarrow \emptyset$
while *computation time < computation time limit* do
    X=$\emptyset$
    $O_0$=O
    while $O_0$=$O$ do
        Apply VND method and find the best neighbour $\bar{O} \in N_1(O) \bigcup ... \bigcup N_{k_{\max}}(O) \setminus X$
        if $\bar{O}$ *is not tabu, or* $\bar{O}$ *is tabu but satisfies the aspiration criterion* then
            Set $O \leftarrow \bar{O}$
            if $F(O) < F(\hat{O})$ then
                $\hat{O} \leftarrow O$
            end
        else
            $X = X \bigcup \{\bar{O}\}$
        end
    end
    Update Tabu list and aspiration criterion
    Determine computation time
end
```

In this algorithm, we consider an aspiration criterion because TS method is too restrictive, and may prohibit attractive moves, even when there is no danger of cycling. So it is necessary to use an algorithmic device that allows one to cancel tabu statues. In aspiration criteria, when the solution value is better than the current best-known solution, the move is allowed (Gendreau and Potvin, 2010). There are three main approaches to define tabu list size: fixed to a predetermined value, randomly chosen from a specific range, or dynamically changing by adjusting its value, (Salhi, 2002). Later in this chapter, we will find the appropriate tabu list size for our experiments.

3.5 Computational Results

In this section, we first present the results of computational experiments comparing lower bounds and upper bounds. The first method for lower bound is Lagrangian relaxation by relaxing problem (3.2.1), and another two methods are integer programming formulations based on two different linearizations. Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are used to calculate upper bounds. All algorithms were coded in Visual C++ 2012 and C and run on a PC with Intel(R) Core(TM), 3.10GHz and 4GB RAM. IBM ILOG CPLEX Optimization Studio (often informally referred to simply as CPLEX) is an optimization software package, that is used for solving linearization scheme I and II to find lower bounds, that is implemented through the C programming language. The quality of lower bounds is evaluated on some small scale problem using optimal solution obtain with the package BONMIN, a powerful non-linear programming solver for integer non-linear programming problem to solve problem (3.2.1) to optimality (available at http://www.neos-server.org/neos/).

3.5.1 Design of Instances

For the experiments, all the instances are categorized as small and large instances. All the computational experiments in this chapter are performed using randomly generated data and follow Bektaş et al. (2008) to generate all test instances. Our problem has a multi-objective function with two terms, where both contribute significantly to the overall cost. We try to set up a problem where the two terms are balanced. According to Huang and Abdelzaher (2005) “ average network latency of downloading a file is roughly proportional to its size when file size is between 1KB and 100KB ". Therefore, we generate the size of each object (b_k) to be an integer random variable in the interval [1,100]. The capacity of each proxy is $s_j = \delta \sum_{k \in K} b_k$, when b_k is the size of each object and δ is a coefficient which is assigned with different values of 0.8 and 0.9 of total size of all objects. The capacity of link (i, j) is $Q_{ij} = \delta' \sum_{k \in K} b_k$ when δ' is a coefficient that is assigned with four different values, namely 0.6, 0.7, 0.8 and 0.9 of the total size of all objects $\sum_{k \in K} b_k$. Upon on our experiments, δ cannot be less than 0.8 and δ' may not be less than 0.6 according to capacity restriction. The network cost of transferring a unit size object c_{ij} is a random number in [20,60], and finally the probability of requesting an object λ_{ik} is a random number in [0,1]. For both small and large instances the parameters that are explained above are common parameters that we use in both set of instances .

The first category of small instances is generated with following specifications: the number of clients ($|I|$) is 2, the number of potential proxy server ($|J|$) is 3 and the number of objects ($|K|$) is 5. The small category of instances defines capacity of proxies to be $s_j = s$ where s_j is in range of [15,30], unit delay cost in range of [250,1000] and capacity of the link in range of [10,25]. There are sixteen different combination according to the different size of proxy and different unit delay cost (D) that can be seen in Table 3.2.

In large instances, in order to achieve more realistic numbers of clients, proxies and objects, we have used long computation times. Therefore, this category has the following specifications: number of clients ($|I|$) are chosen as 10, 50 or 100, the number of potential proxies ($|J|$) as 5, 25 or 50 and the number of objects ($|K|$) as 40, 200 or 400. There are twenty seven different combinations of these dimension parameters as may be seen in Table 3.4.

According to Formula (3.2.1), we decided to calculate each component of cost separately to check if they are approximately the same size. Table B.1 in Appendix B, clearly shows the computational results for each part of the objective function separately. The values of part one of our objective function $(\sum_{i\in I}\sum_{j\in J}\sum_{k\in K} b_k\lambda_{ik}c_{ij}x_{ijk})$ is in one of the columns under the name of network transfer cost, and the value of second part $(\sum_{i\in I}\sum_{j\in J}\frac{Df_{ij}}{Q_{ij}-f_{ij}})$ is in another column of the table with the name of delay cost. Delay cost has a bigger value than the other cost, and when number of objects increases, the delay cost follows accordingly. Also, more objects means more access needed for each object from different proxies, which increases the traffic. In order to clarify why delay cost is bigger, we use different value of unit delay cost of message delay in the network (D), that balances two terms of problem, that is randomly chosen between 250 and 1000 for small instances and is calculated for large instances as the summation of total cost of all links divided by number of links as follows: $\sum_{i\in I}\sum_{j\in J}\sum_{k\in K}\frac{b_k\lambda_{ik}c_{ij}}{|I||J|}$. Figure 3.1 is the graph that use to secure the best calculation of the Lagrangian relaxation bound in terms of setting the maximum number of iterations.

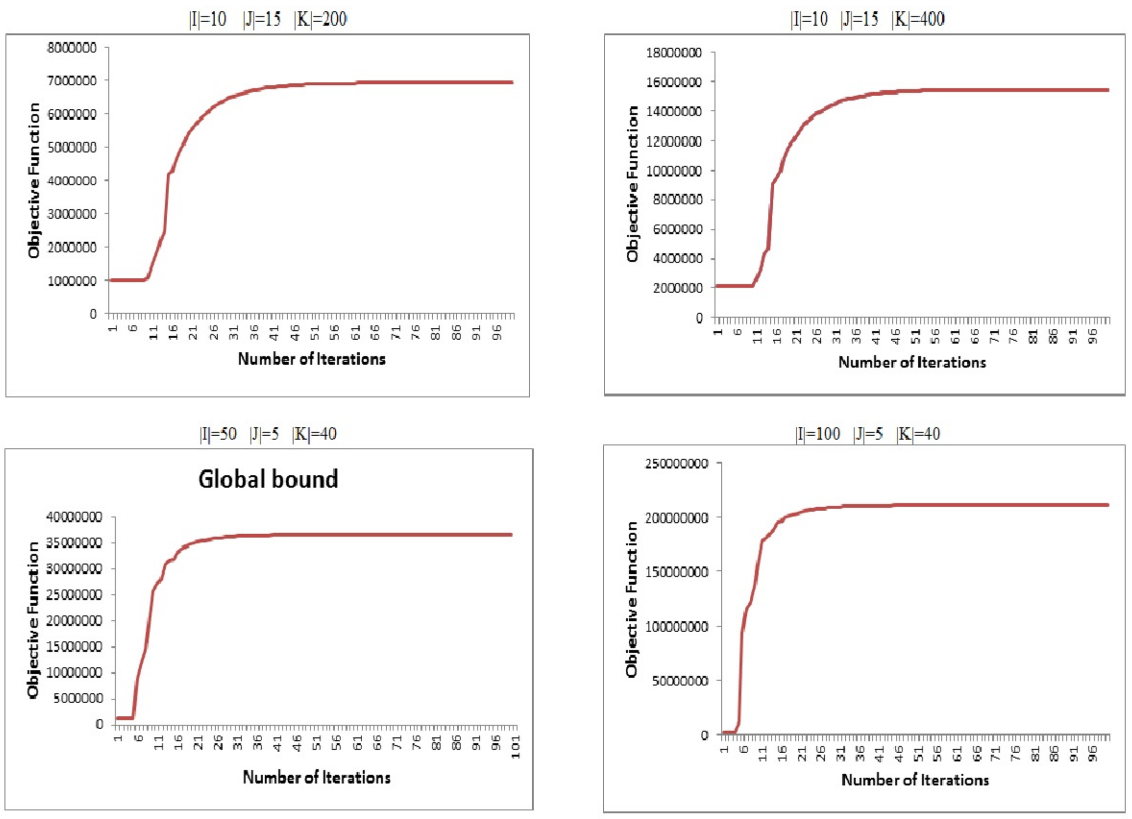

Figure 3.1: Lower Bound for Four Different Instances Respect to Iterations

Each graph is a sample taken out of twenty seven instances and sample values of δ and

δ'. In each graph n is the number of iteration in the horizontal axis and global bound is the best found lower bound for that specific iteration, what is considered in vertical axis. In each graph the number of clients |I|, the number of potential proxy servers |J| and the number of objects |K| are shown. All details are in Appendix B (B.2 - B.5). All of the four plots show that after 50 iterations, there is "no" or "very little" improvement. Therefore, we implement Lagrangian relaxation for 100 iterations for each instance to get the best lower bound from Lagrangian relaxation in a reasonable computation time. Note that for computational results, when we use gap, we are calculating relative percentage gap.

3.5.2 Computational Results for Small Instances

We provide the results of computational experiments to demonstrate the quality of the proposed methods for finding a lower bound. Table 3.1 provides a list of acronyms for Table 3.2.

Instance	Ins.
Unit delay Cost	D
Lagrangian Relaxation	LR
Linear Programming Relaxation scheme I	LPR1
Linear Programming Relaxation scheme II	LPR2
NEOS server	Neos
Gap between LR and LPR1	$gap_{LR-LPR1}$
Gap between LR and LPR2	$gap_{LR-LPR2}$
Gap between NEOS server and LR	gap_{N-LR}
Gap between NEOS server and LPR1	gap_{N-LPR1}
Gap between NEOS server and LPR2	gap_{N-LPR2}
LR Computation Time (second)	T_{LR}
LPR1 Computation Time(second)	T_{LPR1}
LPR2 Computation Time (second)	T_{LPR2}

Table 3.1: List of Acronyms for Table 3.2

Table 3.2 is based on small instances when $|I|$=2, $|J|$=3 and $|K|$=5 to obtain optimality.

As instances are small, the NEOS would be suitable for getting optimality for problem (3.2.1). Lagrangian Relaxation (LR) is used to find lower bounds, also Linear Programming Relaxation (LPR1 and LPR2) are two methods that use to linearize the problem. In this table sixteen instances are categorized based on capacity of each proxy (s_j) and unit delay cost (D), where (s_j) will be allocated by four different numbers of 15, 20, 25, 30, and D will be 250, 500, 700, 1000 (initial experiments show that the value of D gave the suitable balance between different components of objective function) and at the end, capacity of the link (Q_{ij}) is a random numbers between 10 to 25.

Ins.	s_j	D	$gap_{LR-LPR1}$	$gap_{LR-LPR2}$	gap_{N-LR}	gap_{N-LPR1}	gap_{N-LPR2}	T_{LR}	T_{LPR1}	T_{LPR2}
1	15	250	-7.02%	-7.02%	37.16%	32.75%	32.75%	1.78	0.06	0.05
2	15	500	-5.74%	-5.74%	36.67%	33.04%	33.04%	1.81	0.06	0.06
3	15	700	1.96%	-6.43%	36.44%	37.68%	32.35%	2.05	0.06	0.05
4	15	1000	2.21%	-6.20%	37.00%	38.39%	33.09%	1.68	0.06	0.05
5	20	250	2.94%	2.94%	20.26%	22.60%	22.60%	2.47	0.06	0.07
6	20	500	3.05%	3.05%	20.29%	22.73%	22.73%	3.53	0.06	0.06
7	20	700	10.05%	2.94%	20.43%	28.43%	22.76%	2.00	0.09	0.08
8	20	1000	10.18%	3.05%	20.36%	28.47%	22.79%	2.56	0.08	0.08
9	25	250	2.94%	2.94%	20.26%	22.60%	22.60%	2.47	0.05	0.04
10	25	500	3.05%	3.05%	20.29%	22.73%	22.73%	3.53	0.05	0.05
11	25	700	2.94%	2.94%	20.43%	22.76%	22.76%	2.00	0.03	0.03
12	25	1000	3.05%	3.05%	20.36%	22.79%	22.79%	2.56	0.03	0.03
13	30	250	9.88%	9.88%	14.12%	22.60%	22.60%	1.95	0.01	0.01
14	30	500	10.07%	10.07%	14.07%	22.73%	22.73%	1.71	0.02	0.01
15	30	700	10.10%	10.10%	14.09%	22.76%	22.76%	1.73	0.03	0.02
16	30	1000	10.08%	10.08%	14.13%	22.79%	22.79%	1.67	0.03	0.03
		Average	4.36%	2.42%	22.90%	26.61%	25.24%	2.22	0.05	0.05

Table 3.2: Computational Results for Small Instances

We are looking at the size of gaps between LR and LPR1 ($\frac{LR-LPR1}{LR}100\%$) also LR and LPR2 ($\frac{LR-LPR2}{LR}100\%$). The negative results are due to the situation when LPR is bigger than LR, which is not accountable for all analysis. The minimum gap between LR and LPR1 is reached when s_j equals to 15 and D is in range of [250,1000]. Also, the minimum gap between LR and LPR2 occurs when s_j is in range of [15,25], but for any value of D. When s_j=30, the worst situation occurs for both (LR & LPR1) and (LR & LPR2), but the value of D has no influence on the gap. In a simple conclusion, s_j=15 will provides the least value of 2% and most desirable gap. The comparison of gap value between NEOS and LR ($\frac{NEOS-LR}{NEOS}100\%$) shows when s_j=15, the gap value is high and for s_j=30 the gaps decrease.

The value of D for different instances has a consistency, and does not affect the gap results. For gap between NEOS and LPR1 ($\frac{NEOS-LPR1}{NEOS}100\%$) and gap between NEOS and LPR2 ($\frac{NEOS-LPR2}{NEOS}100\%$), the same pattern as (NEOS-LR) is achieved, and mostly the results are the same except four results from LPR2 that have smaller average gap. Simple conclusion for the three columns, shows if we want smaller gap, we should have highest possible value for s_j. The last three columns are used to highlight the time taken to compute each of these bounds. A quick glance, shows LR takes far longer time to compute for each of sixteen individual instances, considering LPR1 and LPR2 have quicker time of computation for each similar instance. Therefore, when all parameters and instances fall in right place, linear programming relaxation is a better method rather than Lagrangian relaxation when computation time is limited. There are more details can be found in Appendix A (A.1 - A.2).

3.5.3 Computational Results for Large Instances

3.5.3.1 Lower Bound Results

Two different linearization schemes are proposed to linearize model (3.2.1), as explained before. Also, each linearization solves individually through MIP (Mixed Integer Programming) and Linear Programming Relaxation (LPR). In Table 3.3 shows the summary results when $|K|$=40 and different variations between δ and δ' are kept with the same pattern as explained before. gap_1 is a gap percentage between MIP1 and LPR1 and gap_2 is between MIP2 and LPR2 and are calculated as: $gap_1 = \frac{MIP1-LPR1}{MIP1}100\%$, and $gap_2 = \frac{MIP2-LPR2}{MIP2}100\%$ for each instance.

Ins.	|I|	|J|	|K|	δ	δ'	gap_1	T_{MIP1}	gap_2	T_{MIP2}
1	10	5	40	0.9	0.8	0.02%	7200.07	0.02%	7200.04
2	10	10	40	0.9	0.8	0.03%	7200.29	0.02%	7200.38
3	10	15	40	0.9	0.8	0.07%	7211.08	0.07%	7212.97
4	50	5	40	0.9	0.8	0.07%	7264.53	0.14%	7234.19
5	50	10	40	0.9	0.8	0.16%	7197.25	0.18%	7197.26
6	50	15	40	0.9	0.8	0.25%	7219.51	0.31%	7389.49
7	100	5	40	0.9	0.8	0.11%	7216.98	0.13%	7222.52
8	100	10	40	0.9	0.8	0.28%	7241.80	0.30%	7231.05
9	100	15	40	0.9	0.8	1.58%	7234.14	0.49%	7559.48
10	10	5	40	0.9	0.9	0.02%	7200.01	0.02%	7200.04
11	10	10	40	0.9	0.9	0.03%	72001.24	0.03%	7200.38
12	10	15	40	0.9	0.9	0.06%	7200.76	0.06%	7198.87
13	50	5	40	0.9	0.9	0.04%	7207.67	0.04%	7200.71
14	50	10	40	0.9	0.9	0.13%	7327.34	0.13%	7248.84
15	50	15	40	0.9	0.9	0.23%	7707.82	0.25%	7233.77
16	100	5	40	0.9	0.9	0.07%	7196.08	0.08%	7186.37
17	100	10	40	0.9	0.9	0.26%	7268.40	0.24%	7344.85
18	100	15	40	0.9	0.9	0.43%	7274.95	0.41%	7442.48
					AVG	0.21%		0.16%	

Table 3.3: Gaps Between Mixed Integer Programming and Linear Programming Relaxation

T_{MIP1} and T_{MIP2} are computational time for running MIP1 and MIP2. At the end of the Table 3.3, AVG is the average gap value for each column. The results from table shows that linearization scheme II is working better compare with linearization scheme I. As the table shows, none of the instances, are solved to optimality. More results can be found in Appendix B (Tables B.6 - B.13).

For the next step, we use CPLEX for linearization scheme I and linearization scheme II through LPR for all twenty seven instances. So, CPLEX is required to be scaled to avoid huge values bigger than 10^9. To clarify scaling down, we describe that in main formulation (3.2.1) the terms in objective function had achieved values bigger than 10^9, so they were divided by 10^6 to restrict the size of numbers in CPLEX. Still by implementing scaling, we could not find any solution for LPR, when the number of clients, proxies and objects increase. Some dashes can been seen in results on intersection of gap column and instance row, when we compare Lagrangian relaxation with LPR. These dashes are due to incapability of CPLEX in finding a lower bound. All the results for comparing Lagrangian relaxation with LPR1 and Lagrangian relaxation with LPR2 appear in "Lagrangian Relaxation and Linear Programming Relaxation", Appendix B (Tables B.15 - B.30).

3.5.4 Upper Bound Results

In this section we compare the computational results for IVND and TS. Referring to our twenty seven instances of $|I|$, $|J|$ and $|K|$ and relevant δ and δ' for each instance, we took a sample of δ=0.9 and δ'=0.9 to help us find out the best number for the size of shaking. We start with n = 2,3,...,10 and run IVND against different values of n.

We choose same instances that we are working for all the experiments in this chapter. We are looking to find the minimum of IVND and relevant number of n for each twenty seven instances. The three graphs in Appendix B show the trend of each minimum amount. At the end, the value 7 is selected as the size of shaking. All the details are in Appendix B (B.31 - B.33).

Based on initial experiments, we found out that reasonable results could be obtained with modest computation time of up to 20 minutes . So, in order to establish a suitable computational time for different combination of instances of each $|I|$, $|J|$ and $|K|$, we use four different instances out of twenty seven, and run each one for four different time limits of 5, 10, 15 and 20 minutes. Table B.34 shows there is no any significant difference on the results of computation if we run for 10, 15 and 20 minutes. Thus, we conclude that 5 minutes is sufficient to find solutions of good quality.

Also, we are looking for the appropriate tabu list size. Different computations for four instances with several numbers of tabu list size are conducted. We set size of tabu list equals to 10, 15, 20, 25 and 30 in steps of 5 if an improvement can be found. Otherwise, the experiment stops after several setting numbers which show no further improvements. We use the same stopping condition as we use for our algorithms to provide a basis for comparison. The details results are given in Appendix B (B.35 - B.36).

From tables, it is observed that when tabu list size greater than 20, there is no more improvement. Therefore, we stop the experiment. Note here that different variations between δ and δ' are kept with the same pattern that will be explained in computational results section. The gap is calculated by $\frac{IVND-TS}{IVND}$ 100%. Because we are evaluating upper bounds and in most of the instances the gap value is positive, it is simply concluded that TS is a better method to find upper bounds for this formulation. It is only a few average value we can see a significant rise of the gap value and consequently the average for that particular row. In this particular instances, IVND is considerably larger than TS.

Another perspective of the average results for each column shows smallest average values when δ=0.9. The highest average values occur when δ=0.8 and δ'=0.6, which means if we use the minimum capacity of the link and proxy, then the complexity and gap value will be the biggest value. Table 3.4 lists gaps between IVND and TS as below:

Ins.	|I|	|J|	|K|	δ'=0.6 δ=0.8	δ'=0.7 δ=0.8	δ'=0.8 δ=0.8	δ'=0.9 δ=0.8	δ'=0.6 δ=0.9	δ'=0.7 δ=0.9	δ'=0.8 δ=0.9	δ'=0.9 δ=0.9	Average
1	10	5	40	0.02%	7.20%	0.02%	0.02%	0.08%	0.09%	0.10%	-0.01%	0.94%
2	10	5	200	0.05%	0.08%	0.00%	0.00%	0.07%	1.39%	1.54%	0.11%	0.40%
3	10	5	400	0.00%	5.05%	0.04%	0.00%	0.03%	0.00%	0.00%	-0.17%	0.62%
4	10	10	40	0.00%	0.00%	2.82%	6.28%	7.91%	0.00%	5.91%	1.13%	3.01%
5	10	10	200	0.00%	0.00%	0.89%	0.77%	0.00%	0.99%	1.11%	0.55%	0.54%
6	10	10	400	0.76%	0.00%	0.67%	0.72%	0.00%	0.16%	0.84%	0.15%	0.41%
7	10	15	40	0.00%	0.00%	9.83%	0.00%	9.41%	0.00%	0.00%	3.26%	2.81%
8	10	15	200	1.02%	0.00%	0.00%	1.24%	0.00%	0.00%	0.00%	0.87%	0.39%
9	10	15	400	-0.55%	0.00%	0.60%	0.00%	0.00%	0.72%	0.25%	-0.07%	0.12%
10	50	5	40	15.10%	3.48%	2.61%	1.11%	4.06%	6.84%	1.13%	5.50%	4.98%
11	50	5	200	30.38%	29.45%	26.38%	25.38%	16.32%	14.57%	13.23%	13.42%	21.14%
12	50	5	400	1.10%	0.56%	0.93%	0.89%	0.70%	0.60%	0.60%	0.52%	0.74%
13	50	10	40	5.72%	1.30%	7.21%	3.78%	3.55%	0.00%	0.70%	2.15%	3.05%
14	50	10	200	6.18%	0.00%	0.00%	1.17%	0.00%	0.48%	0.81%	1.06%	1.21%
15	50	10	400	0.00%	0.22%	0.84%	0.00%	0.85%	0.00%	0.40%	0.34%	0.33%
16	50	15	40	0.00%	0.00%	0.00%	4.10%	0.00%	6.43%	4.50%	3.14%	2.27%
17	50	15	200	0.00%	0.00%	0.35%	1.33%	0.00%	0.71%	0.17%	0.05%	0.33%
18	50	15	400	92.31%	22.36%	22.24%	21.92%	11.18%	10.94%	11.19%	11.01%	25.39%
19	100	5	40	5.00%	13.03%	3.08%	1.81%	5.91%	9.92%	8.35%	7.21%	6.79%
20	100	5	200	31.80%	28.36%	28.19%	25.86%	15.89%	14.15%	13.73%	13.62%	21.45%
21	100	5	400	0.36%	-4.23%	0.00%	0.00%	0.26%	0.00%	0.00%	0.03%	-0.45%
22	100	10	40	9.39%	8.06%	2.95%	8.18%	5.08%	4.26%	8.44%	3.30%	6.21%
23	100	10	200	0.93%	0.00%	0.22%	0.66%	1.43%	1.47%	0.23%	1.27%	0.78%
24	100	10	400	0.07%	0.01%	0.01%	0.57%	0.01%	0.01%	0.01%	11.30%	1.50%
25	100	15	40	8.50%	0.43%	8.90%	0.93%	7.09%	1.74%	6.08%	3.49%	4.64%
26	100	15	200	0.56%	1.38%	1.52%	0.53%	1.74%	0.04%	0.58%	0.31%	0.83%
27	100	15	400	0.15%	0.46%	0.14%	0.14%	-0.04%	0.01%	0.87%	0.48%	0.28%
			Average	7.74%	4.34%	4.46%	3.98%	3.39%	2.80%	2.99%	3.11%	4.10%

Table 3.4: Gaps Between Iterated Variable Neighbourhood Descent and Tabu Search

In other word, TS would be defiantly the best method when we are working with minimum capacity of the link and proxy. The reason is TS provides the smaller upper bounds which are close to optimality. The overall average in gap Table 3.4 is 4.10% which is positive and shows that IVND provides a large upper bound compared to TS. Also, we compare Lagrangian relaxation with TS and IVND for each δ and δ' that are found in sixteen different tables. All the tables can be seen in Appendix B (B.54 - B.61) and (B.64 - B.71).

3.5.5 Comparison Lower Bounds and Upper Bounds

CPLEX could not find a solution value for number of instances, so we were not able to compare all three methods of lower bounds together. Therefore, in Table 3.5 we found the best lower bounds and compare with the best upper bounds which is given by Tabu Search (TS). In this table, smallest average gap is, when both capacity of proxy and capacity of the link have the maximum size.

Gap column values are the average percentage difference between tabu search and related the best lower bounds and calculated as: gap=$\frac{TS-bestlowerbound}{TS}$100%, respectively for each instance in each table. In Table 3.5 it is clear that when we have maximum size of proxy (δ=0.9) and link (δ'=0.9) upper bounds and lower bounds are close to each other. The maximum capacity of proxy (δ=0.9) and minimum capacity of link (δ'=0.6) gives the largest average gap value. When number of objects increase with number of proxies, average gap value is higher, especially when number of clients is 50 or 100. When number of objects is 200 and number of proxies is 5, the average gap values are smallest. If we divide these eight columns of results into two categories according to capacity of proxy 0.8 and 0.9, we observe that the minimum average gap of 17.53% and 15.69% both belongs to δ'=0.9. This means minimum average of gap occurs when we have maximum capacity of link and capacity of proxy does not have any affect. The average gap for the table is 18.57% which shows upper bounds and lower bounds are not too far from each other but are not close either. All tables with details are in Appendix B (B.45 - B.52). Table 3.5 lists gaps between IVND and TS as below:

Ins.	|I|	|J|	|K|	δ'=0.6 δ=0.8	δ'=0.7 δ=0.8	δ'=0.8 δ=0.8	δ'=0.9 δ=0.8	δ'=0.6 δ=0.9	δ'=0.7 δ=0.9	δ'=0.8 δ=0.9	δ'=0.9 δ=0.9	Average
1	10	5	40	16.76%	11.87%	9.56%	8.34%	12.70%	9.71%	8.27%	7.57%	10.60%
2	10	5	200	3.60%	0.00%	0.00%	0.00%	0.01%	0.01%	0.01%	0.01%	0.45%
3	10	5	400	0.82%	0.75%	9.86%	9.57%	9.91%	9.46%	9.10%	9.72%	7.40%
4	10	10	40	10.06%	8.36%	7.50%	7.09%	8.68%	7.58%	7.09%	6.95%	7.92%
5	10	10	200	25.36%	7.85%	22.15%	20.31%	25.71%	22.89%	20.85%	19.30%	20.55%
6	10	10	400	14.54%	13.44%	12.79%	12.29%	14.29%	13.21%	12.84%	12.48%	13.24%
7	10	15	40	10.91%	9.20%	8.18%	7.57%	41.03%	8.28%	7.57%	7.22%	12.49%
8	10	15	200	10.10%	38.23%	37.29%	6.36%	38.90%	30.16%	6.36%	3.00%	21.30%
9	10	15	400	43.00%	40.89%	39.36%	37.23%	40.70%	38.55%	36.76%	5.47%	35.24%
10	50	5	40	3.98%	12.36%	8.50%	7.50%	13.75%	9.78%	7.49%	6.13%	8.68%
11	50	5	200	11.27%	11.01%	10.71%	10.45%	0.95%	1.72%	2.08%	2.54%	6.34%
12	50	5	400	11.48%	10.94%	10.67%	10.57%	15.42%	15.10%	14.59%	14.51%	12.91%
13	50	10	40	10.01%	7.98%	6.71%	5.85%	8.39%	6.84%	5.85%	5.17%	7.10%
14	50	10	200	20.38%	20.59%	20.28%	19.97%	20.34%	19.72%	19.75%	19.44%	20.06%
15	50	10	400	21.05%	20.56%	20.45%	20.74%	20.61%	20.61%	20.43%	19.87%	20.54%
16	50	15	40	8.92%	7.33%	6.25%	5.48%	7.67%	6.37%	5.48%	4.84%	6.54%
17	50	15	200	55.46%	54.57%	54.09%	53.57%	53.65%	51.04%	52.70%	52.30%	53.42%
18	50	15	400	40.89%	42.26%	41.79%	41.20%	48.74%	47.73%	47.35%	46.85%	44.60%
19	100	5	40	16.56%	11.07%	7.93%	6.06%	12.22%	8.24%	6.06%	4.70%	9.11%
20	100	5	200	11.54%	11.31%	10.93%	5.81%	1.00%	1.77%	2.01%	2.50%	5.86%
21	100	5	400	3.70%	9.43%	10.87%	10.74%	15.56%	15.27%	14.89%	14.66%	11.89%
22	100	10	40	10.86%	8.43%	6.92%	5.88%	8.41%	6.59%	5.42%	3.93%	7.05%
23	100	10	200	2.57%	21.93%	21.82%	21.68%	21.37%	21.22%	21.45%	19.48%	18.94%
24	100	10	400	17.81%	21.15%	21.67%	21.81%	21.41%	21.46%	21.32%	21.11%	20.97%
25	100	15	40	8.60%	6.94%	5.82%	5.01%	7.38%	6.04%	5.13%	4.46%	6.17%
26	100	15	200	56.95%	56.57%	56.28%	56.01%	56.00%	55.41%	55.13%	54.69%	55.88%
27	100	15	400	57.09%	56.75%	56.47%	56.25%	56.01%	55.57%	55.28%	54.83%	56.03%
			Average	18.68%	19.32%	19.44%	17.53%	21.51%	18.90%	17.45%	15.69%	18.57%

Table 3.5: Comparison Best Lower Bound with Best Upper Bound

Also, we compare Lagrangian Relaxation (LR) as lower bound and Tabu Search (TS) for upper bound. All tables studies exactly the same instances and same configuration between δ and δ' but through TS method. Gap column values are the average percentage difference between LR and TS and calculated as gap=$\frac{TS-LR}{TS}$100%, respectively for each instance in the table. All the results can be seen in eight tables in Appendix B (B.54 - B.61). Table B.62 shows gap value can be low for a suitable combination of instance parameters, i.e. the second instance configuration ($|I|$=10, $|J|$=5 and $|K|$=200) provides the least value of gap between TS and LR. On the other hand, instance 25 ($|I|$=100, $|J|$=15 and $|K|$=40), 26 ($|I|$=100, $|J|$=15 and $|K|$=200) and 27 ($|I|$=100, $|J|$=15 and $|K|$=400) might be one of the worst configuration of $|I|$, $|J|$ and $|K|$, which provide a large value of gap between TS and LR.

Another approach observing the results on gap column shows that regardless of different combination between δ and δ', for $|J|$=15 we are likely to end up having a big gap percentage. As a simple explanation, having 5 or 10 potential proxies is a much better choice. We may also learn, in an instance when $|J|$ and $|K|$ values are bigger, then the gap value would also be high. Instances 9 ($|I|$=10, $|J|$=15 and $|K|$=400), 18 ($|I|$=50, $|J|$=15 and $|K|$=400) and 27 ($|I|$=100, $|J|$=15 and $|K|$=400) which all have $|J|$=15 and $|K|$=400 have resulted in high gap values. It is clear that the pattern of high value of gaps also repeats for all different values combination of δ and δ'. We have also learn that $|I|$ value (number of clients) has far less influence to increase the gap value, comparing to $|J|$ (number of potential proxies) and $|K|$ (number of objects). By looking at average value of each gap column, the combination of δ=0.8 and δ'=0.9 conclude the lowest average of 25%. This table shows instances such as $|I|$=10, $|J|$=5 and $|K|$=200 that provides least gap value irrespective of the combination of δ and δ'. On the other hand $|I|$=100, $|J|$=15 and $|K|$=400 provides highest gap value, which again is not related to the δ and δ' combination.

For another comparison, we compare Lagrangian Relaxation (LR) as lower bounds and Iterated Variable Neighbourhood Descent (IVND) considering for upper bounds. All tables consider exactly the same instances and the same configuration between δ and δ', but focus on the IVND method. Gap column values are the average percentage difference between LR and IVND and calculated as gap=$\frac{IVND-LR}{IVND}100\%$, respectively for each instance in the table. All the tables are in Appendix B (B.64 - B.71). With no change, in Table B.72 instance number 2 ($|I|$=10, $|J|$=5 and $|K|$=200) is the best configuration so far for the lowest gap value and instance 27 ($|I|$=100, $|J|$=15 and $|K|$=400) yet creates a gap value that is highest. It is simply indicates that the TS algorithm would be providing good quality solution comparing with IVND algorithm.

Table B.62 provides gaps between Lagrangian relaxation and TS, and Table B.72 shows the gaps between Lagrangian relaxation and IVND. By looking at these two tables (B.62 and B.72), it is clear that the average gap value for each gap column produces a higher value in Table B.72. Therefore, the proposed IVND algorithm compared to the TS algorithm will always provides a higher average gap value.

The last comparison is TS with LPR1 and LPR2. This time we compare only instances where CPLEX gives the solution for that instance. All the results are in Appendix B (B.73 - B.80). In the tables, LPR2 has the lower gap compare with LPR1 when capacity of the proxy is 0.8. LPR1 and LPR2 have the same average when capacity of the proxy is 0.9. As a general overview to these tables, what we learn from gap values is that LPR2 is a slightly better method of calculation by providing less gap. Note that all tables studies exactly the same instances and same configuration between δ and δ'.

3.6 Discussion

In this chapter, we propose a new non-linear integer programming formulation which overcome disadvantages of previous models by introducing delay in the formulation. The aim is to minimize overall latency of searching Web server due to availability of system resources and traffic pattern. Lagrangian relaxation and decomposition approaches are used to address this problem. Two complicated constraints, which are related to the amount of traffic in the link and the assignment, are used to relax the problem with two different set of multipliers. The relaxed problem decomposes into three sub-problems with respect to the component involving variables x, f and z. To upgrade the multipliers, step lengths for each multiplier are computed independently of each other. Also, two different sub-gradients are considered for each multiplier. We set 100 sub-gradient optimization iterations for each instance to get good performance from Lagrangian relaxation. Two different linearization schemes are also introduced. The first linearization (Scheme I) finds a lower bound through perspective cuts when tangents of the function are used as valid inequalities to provide a lower bound for auxiliary variables. The second linearization (Scheme II) uses a previous linear segment between pairs of consecutive integer points. The approach which is provided in scheme II has stronger bounds compare to scheme I.

Iterated Variable Neighbourhood Descent (IVND) is one of the metaheuristics method that we use to find upper bounds. Also, Tabu Search (TS) is another metaheuristic providing very good quality solutions for solving our optimization problem. Two local search operators are used: swap and remove-insert to improve. Constructive heuristic method is introduced to find an initial solution. It randomly assigns objects to proxies and calculates the capacity of each proxy by adding new object every time, until all objects find space allocated in a proxy and no proxy is left out.

In this chapter, for the computational experiments, two different designs are introduced as small and large for instances. The first category for small instances is generated with the following specifications: the number of clients ($|I|$) is 2, the number of potential proxy servers ($|J|$) is 3 and the number of objects ($|K|$) is 5. For small instances, we consider NEOS to get optimality. Mixed Integer Programming (MIP) and Linear Programming Relaxation (LPR) are two methods that run in CPLEX. We are looking at the minimum value of gap between Lagrangian relaxation and Linear Programming Relaxation (LPR). When s_j (capacity of proxy) equals to 15 and D is in range of [700,1000] minimum gap is reached between Lagrangian relaxation and LPR with scheme I. Also, the minimum gap between LR and LPR with scheme II is achieved, when s_j is in range of [20-25], but the value of D could float between all ranges. As the conclusion, for smaller gap, highest possible value for s_j should be considered. The computation time for each instance configuration is set to two hours (7200 seconds). Majority of instances, just passed two hours computational time with no optimality, so the results would be achievement of upper bound for both LPR from scheme I and II, but no optimality.

The second category of large instances has following specifications: number of clients ($|I|$) are chosen as 10, 50 or 100, the number of potential proxies ($|J|$) as 5, 25 or 50 and the number of objects ($|K|$) as 40, 200 or 400. There are twenty seven different combinations of these parameters. For large instances, Lagrangian relaxation, Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are considered to find lower bounds and upper bounds. According to the capacity restriction, different variations between δ and δ' which are coefficient of s_j and Q_{ij} (capacity of the link) are considered. Each percentage gap is calculated as: gap=$\frac{upperbound - lowerbound}{upperbound}$100%. The best number of shaking is considered 7 for all experiments (B.31 - B.33). Also, to get the best solution, five minutes is suggested for computational time (B.34). Mixed Integer Programming (MIP) and Linear Programming Relaxation (LPR) are compared together. Also, IVND and TS are compared together and concluded that TS is better method for upper bounds. From all lower bounds and upper bounds techniques that we use in this chapter, the best lower bounds and upper bounds are compared together. For more challenge, we compare Lagrangian relaxation with IVND and TS. In linearizations with two schemes, LPR from scheme I and II compared with Lagrangian relaxation, and finally, TS is compared with LPR1 and LPR2.

In next chapter, we are looking at one of the models of cache allocation problem which has local cluster for proxies by considering origin server. Also, all the proxies are identical and there is no difference to put object in which proxy. Two different linearization methods are introduced to find lower bounds. In metaheuristic section, two different methods are introduced to find initial solution: constructive heuristic and greedy algorithm. variable Neighbourhood (VND), Iterated Variable Neighbourhood Descent (IVND) and Tabu

Search (TS) with six neighbourhoods are suggested in next chapter.

Chapter 4

Lower Bounding Techniques and Metaheuristics for Cache Allocation Problem

Caching is the process of replicating Web contents in different locations where clients can access objects easily. Caching can be performed at various levels: at the proxy server, at the browser and Web server (Mookerjee and Tan (2002) and Hosanagar et al. (2005)). Dynamic replacement is focused on minimizing the cost for the cache under certain conditions, whereas cooperative caching refers to the sharing of objects between networks of proxy caches. In one of the articles about Web caches, Wong (2002) suggest savings of about 40% in download time regardless of the link speed by using cache proxies. This analysis shows that saving from a Web cache proxy might be insignificant, if download or network charges are considered, but the main reason to have cache proxies is to save time when multiple users are utilizing the limited available bandwidth. This value may be small for an individual or instance, but multiplied over a huge number of users and time, considerable productivity can be achieved.

Tawarmalani et al. (2009) explain the benefit of collaboration of objects which refers to the sharing objects between network of proxy caches (i.e., proxy servers serving as caches). We use this strategy, if an object is demanded from a cache that does not have a copy of the required object, then it can be found in neighbour caches. This strategy results in cost reduction, because instead of going to the origin server to get the object, it can be retrieved from neighbouring caches. The basic formulation that we study in this chapter regarding the cache allocation problem is based on an idea from Tawarmalani et al. (2009). To find lower bounds for this formulation, two different linearizations are proposed to solve this proposed mixed integer problem to obtain a global optimal solution. We claim that these two linearization methods give equivalent formulations of the original problem with regards to integer solutions.

Also, Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are two metaheuristics methods which are developed to find upper bounds. Six different local search operators are suggested as follows: insert from origin server to a proxy, insert from one proxy to another proxy, swap between origin server and a proxy, swap between two proxies, a cyclic exchange and remove-insert. Also, a constructive heuristic and a greedy algorithm are two methods which are introduced to find an initial solution.

In Section 4.1, the cache allocation problem and the way that we simplify the formulation is explained. In Section 4.2, two linearizations are suggested based on different studies. Section 4.3 presents metaheuristic algorithms for proposed problem, by using two different approaches to find initial solutions. In Section 4.4, our computational results are presented and finally, a discussion of the findings in this chapter are given in Section 4.5.

4.1 Problem Definition and Formulation

Tawarmalani et al. (2009) analyze the allocation of objects in a network of caches that collaborate to service requests from clients. Let $k \in K$ denotes a set of objects with different sizes and $j \in J$ represents a set of caches. The caches $j \in J$ have capacities that are denoted by s_j. According to the description of model (4.1.1), suppose the cost of serving an object locally from the cache, where it is requested, is c_l, from the other caches in the network is c_n, and from the origin server is c_o. Even if, c_l, c_n and c_o are dependent on the proxy, we treat them as constants. The expected total cost for cache allocation problem from the Tawarmalani et al. (2009) model is as follows:

$$\min \sum_{j\in J}\left\{(c_l - c_n)\sum_{k\in K} e_{jk}z_{jk} + c_n \sum_{k\in K} e_{jk} + (c_o - c_n)\sum_{k\in K} e_{jk}\prod_{j'\in J}(1 - z_{j'k})\right\} \tag{4.1.1}$$

$$\text{s.t.} \quad \sum_{k\in K} b_k z_{jk} \leq s_j \qquad \forall j \in J, \tag{4.1.2}$$

$$z_{jk} \in \{0, 1\} \qquad \forall j \in J, k \in K. \tag{4.1.3}$$

where z_{jk} is 1 if object k is held in cache j and 0 otherwise. The first term in (4.1.1) accounts for the objects served locally, and the second and third terms account for the ones retrieved from neighbouring caches and origin server, respectively. Obviously, an object is obtained from a neighbour only when it is not held locally and at least one of the neighbours holds it. The object is obtained from the origin server when it is not accessible locally or from the neighbouring caches.

Tawarmalani et al. (2009) analyse centralizing scenarios, linearize the objective function by using the techniques of Glover and Woolsey (1974). Next, they discuss the generalization of their results to objects with arbitrary sizes. In terms of complexity, the problem becomes much harder. The objects are not allowed to be split and stored in individual parts in different proxies. This is due to the fact that, if the object can be divided into packets for transmission, to access to the object from one source or different sources would require assembly to recreate the object, whether the whole object is brought from one source or different sources. Thus, the cost of combining different parts of an object may be too high to be ignored.

The proposed Formulation is non-linear due to the last term of objective function as a result of multiplication of z_{jk} for different value of j'. The Multi-dimensional Non-linear Knapsack Problem (MNKP) is a bounded non-linear integer programming problem that maximize a separable non-decreasing function subject to separable non-decreasing constraints. Non-linear knapsack problems have various applications in different fields, for instance, marketing, networking, production planing and so on. In our case, because we cannot separate the objective function, there is not a straightforward way to use MNKP to solve our problem. So, we work on finding an efficient linearization technique by referring to different papers. In the original model, Tawarmalani et al. (2009) consider a demand e_{jk} for each proxy j, but our model has d_k as a demand that we can assign to any proxy that holds the object of interest. Demand for object k is measured by $b_k\theta_k$ where b_k is the size of the popular object k, and θ_k measures how popular object k is going to be.

From the formulation, it is clear that each object can be allocated at most once, that means we cannot put all the objects in the proxy because solution will be dominated. We consider set of proxies $|J|$, origin server ($J_0 = J \bigcup \{0\}$) and objects ($k \in K$) with different sizes. We are looking at simpler model of cache allocation problem which have local cluster of proxies and trying to put an object in each proxy to avoid having to use slower origin server. Essentially all proxies are identical, that might be in different sizes in general. However, for computational testing, we consider all proxies to have the same size. In objective function there is no difference to put the object in which proxy; the cost will be the same. Basically we consider $c_l = c_n$ in the model of Tawarmalani et al. (2009), which means whenever demand occurs in the network, if some proxy j contains object k, it will be considered as local cost; alternative, k is obtained from the origin server. Another explanation is that when object k is not in a proxy, the cost is $c_o d_k$ and when object k is in a proxy the cost is $c_l d_k$. By considering above explanation, the propose model for the cache allocation problem can be written as follows:

$$\min \sum_{k\in K} \left\{ c_l d_k + (c_o - c_l) \left(d_k \prod_{j\in J} (1 - z_{jk}) \right) \right\} \tag{4.1.4}$$

$$\text{s.t.} \quad \sum_{k\in K} b_k z_{jk} \le s_j \qquad \forall j \in J, \tag{4.1.5}$$

$$z_{jk} \in \{0,1\} \qquad \forall j \in J, k \in K. \tag{4.1.6}$$

The objective function consists of two parts, the first part is $\sum_{k\in K} c_l d_k$ where c_l and d_k are positive constants. The second part is a mixed 0-1 polynomial term , $(c_o - c_l) d_k \prod_{j\in J}(1 - z_{jk})$, where z_{0k}, z_{1k}, z_{2k}, $\ldots$, $z_{|J|k}$ are 0-1 integer variables, and c_o and c_l are positive constants, where c_n=c_l . If $\prod_{j\in J}(1 - z_{jk})$=1, then all of z_{jk} for $j \in J$ are zero, and if $\prod_{j\in J}(1 - z_{jk})$=0, means z_{jk}=1 for at least one value $j \in J$.

4.2 Linearization Techniques

There are many problems in network design which are formulated as mixed 0-1 problems. To solve non-linear models, a transformation should be applied by using auxiliary constraints and additional 0-1 variables. The reason is that there are many powerful software tools for linear problems but the tools are not very good for non-linear problems. Also, solving non-linear problems is costly compared with linear problems. Some improvements are provided by Watters (1967) who discuss the method that reduces a 0-1 polynomial formulation to a 0-1 linear formulation. Also, Chen et al. (2006) use this method for the Multi-dimensional Nonlinear Knapsack Problem (MNKP) with a single constraint by considering the fact that the problem is separable, and Li et al. (2009) develop a convergent Lagrangian and cut method for solving the MNKP.

Equation (4.1.4) is a 0-1 polynomial. We solve the problem to optimality by two different models for linearization, to convert the original mixed 0-1 polynomial formulation to 0-1 linear program. The aim is to find out which one of the linearization models is more efficient and better respect to computational time. Therefore, the feasible region will be the same for different linearization models. The validity of each linearization is proved to show the equivalency between the original objective function and the linearized objective function.

4.2.1 Linearization I

We linearize (4.1.4)- (4.1.6) by using the method of Tawarmalani et al. (2009) for arbitrary object sizes without splitting the object, by introducing an auxiliary variable m_k to represent $\prod_{j \in J}(1 - z_{jk})$, and adding linear constraints to impose the non-linear relationship between z_{jk} variables. In our model z_{jk}=1 if object $k \in K$ is in proxy $j \in J$, and z_{jk}=0; otherwise. The model defined by (4.1.4), (4.1.5) and (4.1.6) with new auxiliary variables and constraints are given below:

$$\min c_l \sum_{k \in K} d_k + (c_o - c_l) \sum_{k \in K} d_k m_k \tag{4.2.1}$$

$$\begin{aligned}
\text{s.t.} \quad & m_k \geq 1 - \sum_{j \in J} z_{jk} && \forall k \in K, && (4.2.2)\\
& \sum_{k \in K} b_k z_{jk} \leq s_j && \forall j \in J, && (4.2.3)\\
& z_{jk} \in \{0, 1\} && \forall j \in J, k \in K, && (4.2.4)\\
& m_k \geq 0 && \forall k \in K. && (4.2.5)
\end{aligned}$$

Proposition 4.1: Linearization I is a valid formulation for cache allocation problem.

Proof. Note that Constraint sets (4.2.3) and (4.2.4) are identical to (4.1.5) and (4.1.6), hence any solution that is feasible for Linearization I is also feasible for cache allocation problem. We just need to show that the objective function values of the two solutions are the same. Variable m_k accounts for the term $\prod_{j \in J}(1 - z_{jk})$ in (4.1.4).

Case 1: $\sum_{j \in J} z_{jk} \geq 1$,

In this case, Constraint (4.2.5) dominates constraint (4.2.2) and results in $m_k \geq 0$. Since $c_o - c_l \geq 0$ the optimal solution is $m_k^* = 0$, which is equal to $\prod_{j \in J}(1 - z_{jk}) = 0$.

Case 2: $\sum_{j \in J} z_{jk} = 0$,

In this case, Constraint (4.2.2) results in $m_k \geq 1$. Similar to the previous case, the optimal solution is $m_k^* = 1$, which is equal to $\prod_{j \in J}(1 - z_{jk}) = 1$. □

4.2.2 Linearization II

Ghezavati and Saidi-Mehrabad (2011) develop a new linearization method by creating a set of auxiliary linear constraints that can be solved by exact algorithms such as branch and bound. They decrease the number of additional constraints and this considerably affects the computational time. Let σ_k be an auxiliary variable to represent $\prod_{j\in J}(1 - z_{jk})$. The model define by (4.1.4), (4.1.5) and (4.1.6) with new auxiliary variables and constraints are given below.

$$\min c_l \sum_{k\in K} d_k + (c_o - c_l)\sum_{k\in K} d_k\sigma_k \tag{4.2.6}$$

$$\text{s.t.}\quad (1 - z_{1k}) + \ldots + (1 - z_{|J|k}) \geq |J|\sigma_k, \qquad \forall k \in K, \tag{4.2.7}$$

$$(1 - z_{1k}) + \ldots + (1 - z_{|J|k}) - |J| + 1 \leq |J|\sigma_k, \qquad \forall k \in K, \tag{4.2.8}$$

$$\sum_{k\in K} b_k z_{jk} \leq s_j \qquad \forall j \in J, \tag{4.2.9}$$

$$\sigma_k \in \{0,1\} \qquad \forall k \in K, \tag{4.2.10}$$

$$z_{jk} \in \{0,1\} \qquad \forall j \in J, k \in K. \tag{4.2.11}$$

Proposition 4.2: Linearization II is a valid formulation for the cache allocation problem.

Proof. Note that Constraints (4.2.9) and (4.2.11) are identical to (4.1.5) and (4.1.6), so any solution that is feasible for linearization II is also feasible for cache allocation problem. We just need to show that the objective function values for the two solutions are the same.

Case 1:$\sum_{j\in J} z_{jk} \geq 1$,

In this case, Constraint (4.2.7) will imply $\sigma_k < 1$ and Constraint (4.2.10) will force $\sigma_k^* = 0$, which is equal to $\prod_{j\in J}(1 - z_{jk}) = 0$.

Case 2: $\sum_{j\in J} z_{jk} = 0$,

In this case, Constraint (4.2.8) will imply $\sigma_k > 0$, and Constraint (4.2.10) will force $\sigma_k^* = 1$, which is equal to $\prod_{j\in J}(1 - z_{jk}) = 1$. □

4.3 Metaheuristics

Metaheuritics are affective methods by providing a sufficiency good solution to an optimization problem. Two powerful metaheuristics are used to solve the propose cache allocation problem in this chapter. Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are considered to find upper bounds for our new model. We describe six different neighbourhoods for new model. In the next section, all neighbourhoods with related algorithm are explained. An initial solution is achieved by two different algorithms as follows: a constructive heuristic and a greedy algorithms.

4.3.1 Neighbourhoods

Metaheuristics may try to avoid being trapped in a local optimum by changing neighbourhood in the search (Mladenović and Hansen, 1997). Therefore, introducing a variety of different neighbourhoods for the cache allocation problem is relevant. We can assign each object to at most once to a proxy, so many objects are not assigned to any proxy and will be unassigned and stay in origin server. In each algorithm, F_{best} represents the best solution we currently found in that particular neighbourhood. Before we start searching the neighbourhood, F_{best} is infinity, because we do not have a neighbour solution, but when search the neighbourhood F_{best} is gradually decreased.

Six different local search operators (neighbourhoods) are suggested as: insert from origin server to a proxy (Algorithms 16), insert from one proxy to another proxy (Algorithms 17), swap between origin server and a proxy (Algorithms 18), swap between two proxies (Algorithms 19), cyclic exchange (Algorithms 20) and remove-insert (Algorithms 21). For all neighbourhoods, the objective function value $F(O)$ is evaluated for set of objects in each proxy, where $F(O)$ is evaluated from (4.1.4).

4.3.1.1 Insert from Origin Server to a Proxy

Algorithm 16 Insert from Origin Server to a Proxy

```
$F_{best} = \infty$
$F(O) \leftarrow$ objective function value (4.1.4) for sets $O_j$ $(j \in J)$ and $O_0$
Initialization:
Set $O_0 \leftarrow$ set of objects in origin server
Set $O_j \leftarrow$ set of objects in proxy $j$    $\forall j \in J$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k$    $\forall j \in J$
for $j \in J$ do
    for $k \in O_0$ do
        if $B_j + b_k \leq s_j$ then
            Set $\bar{O}_{j'} \leftarrow O_{j'}$    $\forall j' \in J \backslash \{j\}$
            Set $\bar{O}_j \leftarrow O_j \bigcup \{k\}$
            Set $\bar{O}_0 \leftarrow O_0 \backslash \{k\}$
            if $F(\bar{O}) < F_{best}$ then
                Set $O^*_j \leftarrow \bar{O}_j$    $\forall j \in J_0$
                $F_{best} \leftarrow F(\bar{O})$
            end
        end
    end
end
```

In Algorithm 16 object k is chosen from origin server and inserted to a proxy j after checking the capacity of the proxy (s_j).

4.3.1.2 Insert From One Proxy to Another Proxy

In this neighbourhood, an object k from one of the proxies j inserts into another proxy j' after checking the capacity of proxy $(s_{j'})$. Algorithm 17 explains for this neighbourhood.

Algorithm 17 Insert From One Proxy to Another Proxy

```
$F_{best} = \infty$
$F(O) \leftarrow$ objective function value (4.1.4) for sets $O_j$ $(j \in J)$
Initialization:
Set $O_j \leftarrow$ set of objects in proxy $j$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k \qquad \forall j \in J$
for $j \in J$ do
    for $j' \in J$ do
        for $k \in O_j$ do
            if $B_{j'} + b_k \leq s_{j'}$ then
                Set $\bar{O}_{j''} \leftarrow O_{j''} \qquad \forall j'' \in J_0 \backslash \{j, j'\}$
                Set $\bar{O}_{j'} \leftarrow O_{j'} \bigcup \{k\}$
                Set $\bar{O}_j \leftarrow O_j \backslash \{k\}$
                if $F(\bar{O}) < F_{best}$ then
                    Set $O^*_{j''} \leftarrow \bar{O}_{j''} \qquad \forall j'' \in J_0$
                    $F_{best} \leftarrow F(\bar{O})$
                end
            end
        end
    end
end
```

4.3.1.3 Swap Between Origin Server and a Proxy

Algorithm 18 Swap Between Origin Server and a Proxy

```
$F_{best} = \infty$
$F(O) \leftarrow$ objective function value (4.1.4) for sets $O_j$ $(j \in J)$
Initialization:
Set $O_0 \leftarrow$ set of objects in origin server
Set $O_j \leftarrow$ set of objects in proxy $j$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k \qquad \forall j \in J$
for $j \in J$ do
    for $k \in O_j$ do
        for $k' \in O_0$ do
            if $B_j + b_{k'} - b_k \leq s_j$ then
                Set $\bar{O}_{j'} \leftarrow O_j \qquad \forall j' \in J \backslash \{j\}$
                Set $\bar{O}_j \leftarrow O_j \backslash \{k\} \bigcup \{k'\}$
                Set $\bar{O}_0 \leftarrow O_0 \backslash \{k'\} \bigcup \{k\}$
                if $F(\bar{O}) < F_{best}$ then
                    Set $O^*_{j'} \leftarrow \bar{O}_{j'} \qquad \forall j' \in J_0$
                    $F_{best} \leftarrow F(\bar{O})$
                end
            end
        end
    end
end
```

In Algorithm 18 an object k in proxy j swaps with another object k' from the origin server.

4.3.1.4 Swap Between Two Proxies

In this neighbourhood two proxies with two different objects are considered. First we select two objects from two different proxies then check the capacity of each proxy, then objects can be interchanged. Note that if the capacity constraints are satisfied the swap can be applied.

Algorithm 19 Swap Between Two Proxies

```
$F_{best} = \infty$
$F(O) \leftarrow$ objective function value (4.1.4) for sets $O_j$ $(j \in J)$
Initialization:
Set $O_j \leftarrow$ set of objects in proxy $j$    $\forall j \in J$
Set $\bar{O}_j \leftarrow$ temporary set of objects in proxy $j$
Set $O_j^* \leftarrow$ best set of objects in proxy $j$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k$    $\forall j \in J$
for $j, j' \in J$    $j < j'$ do
    for $k \in O_j$ do
        for $k' \in O_{j'}$ do
            if $B_j + b_{k'} \leq s_j$ and $B_{j'} + b_k \leq s_{j'}$ then
                Set $\bar{O}_{j''} \leftarrow O_{j''}$    $\forall j'' \in J_0 \backslash \{j, j'\}$
                Set $\bar{O}_j \leftarrow O_j \backslash \{k\} \bigcup \{k'\}$
                Set $\bar{O}_{j'} \leftarrow O_{j'} \backslash \{k'\} \bigcup \{k\}$
                if $F(\bar{O}) < F_{best}$ then
                    Set $O_{j''}^* \leftarrow \bar{O}_{j''}$    $\forall j'' \in J$
                    Set $F_{best} \leftarrow F(\bar{O})$
                end
            end
        end
    end
end
```

4.3.1.5 Cyclic Exchange

Algorithm 20 explains how we can manage a triple swap between two assigned objects which are already allocated in two different proxies with an unassigned object from the origin server. All the objects which are in proxies, are assigned as a set. The idea is that such a triple swap allows a large search of the neighbourhood solutions.

Algorithm 20 Cyclic Exchange

```
$F_{best} = \infty$
$F(O) \leftarrow$ objective function value (4.1.4) for sets $O_j$ $(j \in J)$
Initialization:
Set $O_0 \leftarrow$ set of objects in origin server
Set $O_j \leftarrow$ set of assigned objects in proxy $j \qquad \forall j \in J$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k \qquad \forall j \in J$
for $j, j' \in J \qquad j \neq j'$ do
    for $k \in O_0$ do
        for $k' \in O_j$ do
            for $k'' \in O_{j'}$ do
                if $B_j + b_k - b_{k'} \leq s_j$ and $B_{j'} + b_{k'} - b_{k''} \leq s_{j'}$ then
                    Set $\bar{O}_{j''} \leftarrow O_{j''} \qquad \forall j'' \in J_0 \backslash \{j, j'\}$
                    Set $\bar{O}_j \leftarrow (O_j \backslash \{k'\}) \bigcup \{k\}$
                    Set $\bar{O}_{j'} \leftarrow (O_{j'} \backslash \{k''\}) \bigcup \{k'\}$
                    Set $\bar{O}_0 \leftarrow (O_0 \backslash \{k\}) \bigcup \{k''\}$
                    if $F(\bar{O}) < F_{best}$ then
                        Set $O_j^* \leftarrow \bar{O}_{j''} \qquad \forall j'' \in J_0$
                        Set $F_{best} \leftarrow F(\bar{O})$
                    end
                end
            end
        end
    end
end
```

4.3.1.6 Remove-Insert

Algorithm 21 explains the steps of neighbourhood remove-insert. We take one object out of proxy and try to put in several objects from the origin server. We are trying to put the objects in proxy with the highest demand, also we want small size objects. Therefore sequence π is a ratio of $\frac{d_k}{b_k}$ where d_k is demand, that is generating the popularity of that particular object, when b_k is the size of each object. All objects in origin server are sorted in descending order from largest to smallest. The aim is to put more objects in proxy if the capacity allows. If objective function value is better than previous one, the set is considered as the best set; otherwise, the process starts again.

Algorithm 21 Remove-Insert

```
$F_{best} = \infty$
$F(O) \leftarrow$ objective function value (4.1.4) for sets $O_j$ $(j \in J)$
Initialization:
Set $O_0 \leftarrow$ set of assigned objects in origin server
Set $O_j \leftarrow$ set of assigned objects in proxy $j$    $\forall j \in J$
Start:
Set $B_j \leftarrow \sum_{k \in O_j} b_k$    $\forall j \in J$
for $j \in J$ do
    for $k \in O_j$ do
        Set $\bar{B}_j \leftarrow B_j - b_k$
        Set $\bar{O}_{j'} \leftarrow O_{j'}$    $\forall j' \in J_0 \backslash \{j\}$
        Set $\bar{O}_j \leftarrow (O_j \backslash \{k\})$
        Set $\bar{O}_0 \leftarrow (O_0 \bigcup \{k\})$
        Set $\hat{k} \leftarrow |O_0|$ and compute
        Sequence $\pi$ of objects in $O_0$ such that $\frac{d_{\pi(1)}}{b_{\pi(1)}} \geq \frac{d_{\pi(2)}}{b_{\pi(2)}} \geq ... \geq \frac{d_{\pi(\hat{k})}}{b_{\pi(\hat{k})}}$
        for $k' = 1, ..., \hat{k}$ do
            if $\bar{B}_j + b_{\pi(k')} \leq s_j$ then
                Set $\bar{B}_j \leftarrow \bar{B}_j + b_{\pi(k')}$ ,
                Set $\bar{O}_j \leftarrow \bar{O}_j \bigcup \{\pi(k')\}$
                Set $\bar{O}_0 \leftarrow \bar{O}_0 \backslash \{\pi(k')\}$
            end
            if $F(\bar{O}_{\hat{j}}) < F(O^*_{\hat{j}})$ then
                Set $O^*_{\hat{j}} \leftarrow \bar{O}_{\hat{j}}$    $\forall \hat{j} \in J_0$
            end
        end
    end
end
```

4.3.2 Heuristics Algorithms to Find Initial Solution

Initial solution is used as a starting point for Variable Neighbourhood Descent (VND) and Tabu Search (TS). Two different methods are used to find an initial solution. The first method is a constructive heuristic that generates a solution from scratch by random assignment and the second one is a greedy algorithm. In both algorithms, each object can be used at most once, so some of the objects remains unassigned and therefore are placed in the origin server. In the constructive heuristic algorithm, each proxy j is considered in turn. An random sequence of objects in the origin server is created, and these are considered in turn as candidates to be inserted in the proxy. If the remaining capacity of the proxy allows, then the object will be inserted in the proxy. Algorithm 22 is a constructive heuristic method that describes this heuristic algorithm.

Algorithm 22 Constructive Heuristic Method to Find Initial Solution

```
$F(O) \leftarrow$ Objective function value of problem (4.1.4) for set $O_j$ $(j \in J)$
Initialization:
Set $O_j \leftarrow \emptyset$    $\forall j \in J$
Set $B_j \leftarrow 0$    $\forall j \in J$
for $k \in K$ do
    Set $\bar{J} \leftarrow J$
    while $k \notin O_1 \bigcup ... \bigcup O_J$ do
        Randomly pick a proxy $j \in \bar{J}$
        Set $\bar{J} \leftarrow \bar{J} \backslash \{j\}$
        if $b_k \leq s_j - B_j$ then
            Set $O_j \leftarrow O_j \bigcup \{k\}$
            Set $B_j \leftarrow B_j + b_k$
            Update $B_j$
        end
    end
end
Calculate $F(O)$
```

In the greedy algorithm, the capacity of each proxy has a different size and the proxies are sorted descending of their sizes. First, we consider the biggest proxy. Each proxy is empty to start. Before putting each object in a proxy, the capacity should be checked; if there is insufficient space another object is considered. This process is implemented proxy by proxy until all proxies are considered. Algorithm 23 explains the steps of greedy procedure, where space depends on the number of available proxies. So, at the end, maybe some of the objects remains unassigned.

Algorithm 23 Greedy Algorithm to Find Initial Solution

```
Initialization
Index the proxies so that $s_1 \geq ... \geq s_{|J|}$
Set $\hat{k} = |K|$
Set $O_j = \emptyset$    $\forall j \in J$
for $j$=1 to $|J|$ do
    Set $s \leftarrow s_j$
    Compute sequence $\pi$ of unassigned objects such that $\frac{d_{\pi(1)}}{b_{\pi(1)}} \geq \frac{d_{\pi(2)}}{b_{\pi(2)}} \geq ... \geq \frac{d_{\pi(\hat{k})}}{b_{\pi(\hat{k})}}$
    Set $\bar{k}$=0
    for $k$=1 to $\hat{k}$ do
        if $b_{\pi(k)} \leq s$ then
            $O_j \leftarrow O_j \bigcup \pi(k)$
            Set $\bar{k} \leftarrow \bar{k} + 1$
            Set $s \leftarrow s - b_{\pi(k)}$
        end
        Set $\hat{k} \leftarrow \hat{k} - \bar{k}$
    end
end
```

We are trying to put small objects with highest demand in proxies to avoid the extra cost $((c_o - c_l)d_k)$. For this reason, we consider sequence of unassigned objects (in origin server) and sorted respect to their demand to size ratio $(\frac{d_k}{b_k})$ in descending order from largest to smallest.

4.3.3 Variable Neighbourhood Descent (VND) for Cache Allocation Problem

VND is a local improvement strategy which is a sub-ordinate in Variable Neighbourhood Search (VNS) and most heuristics methods. The idea behind VND is to switch between different neighbourhoods structures. Starting with first neighbourhood, VND implements local search until no more improvement is obtained. Algorithm 24 describes the steps of VND for cache allocation problem as below.

Algorithm 24 Variable Neighbourhood Descent (VND) for Cache Allocation Problem

```
$F(O) \leftarrow$ objective function value (4.1.4) for sets $O_j$ from Algorithms 22 or 23
Initialization:
Given an initial solution $O$ and a neighbourhood structures $N_k = 1, 2, ..., k_{\max}$
$k_{\max}$=6
Set $k \leftarrow 1$
while $k \leq k_{\max}$ do
    Find the best neighbour $\bar{O} \in N_k(O)$
    if $F(\bar{O}) < F(O)$ then
        Set $O \leftarrow \bar{O}$ and $k \leftarrow 1$
    else
        Set $k \leftarrow k + 1$
    end
end
```

From this local optimum, VND continues local search with next neighbourhood, If an improvement solution is found, VND remains in the same neighbourhood; otherwise, it continues with next neighbourhood and so forth. If last neighbourhood has been explored and no improvement has occurred, then the solution represents a local optimum with respect to all neighbourhoods and VND terminates (Hu and Raidl, 2006). For all algorithms, six different neighbourhoods are defined as follows: N_1= Insert from origin server to a proxy (Algorithms 16), N_2= Insert from one proxy to another proxy (Algorithms 17), N_3= Swap between origin server and a proxy (Algorithms 18), N_4= Swap between two proxies (Algorithms 19), N_5= Cyclic exchange (Algorithms 20) and N_6= Remove-insert (Algorithms 21). For all neighbourhoods, the objective function value $F(O)$ is evaluated for set of objects in each proxy, where $F(O)$ is evaluated from (4.1.4).

4.3.4 Iterated Variable Neighbourhood Descent (IVND) for Cache Allocation Problem

Iterated Variable Neighbourhood Descent (IVND) is another metaheuristic method that is applied to our formulation to find upper bounds. IVND integrates the strengths of ILS and VND (Lourenço et al., 2003) and is flexible and easy to implement compare with other heuristic methods. Algorithm 25 explains IVND and considers N_5= Cyclic exchange for shaking neighbourhood.

Algorithm 25 Iterated Variable Neighbourhood Descent (IVND) for Cache Allocation Problem

```
Initialization:
Given an initial solution $O$ and a shaking neighbourhood structure $N = N_{cyclic-exchange}$
Apply Algorithm 24 to find the best solution $O''$
Set $O \leftarrow O'$
while computation time < computation time limit do
    Shaking: Generate a point $O'$ by applying $n$ random moves from neighbourhood $N$ in succession to $O''$
    Generate a solution $O''$ by applying Algorithm 24
    if $F(O'') < F(O')$ then
        Set $O' \leftarrow O''$
    end
    Determine computation time
end
```

An initial solution O is found in two different ways: a constructive heuristic method and a greedy algorithm that can be found in Algorithms 22 and 23. Variable Neighbourhood Descent (VND) is applied to an initial solution until it finds a local optimum O'. Once a local optimum is found, IVND performs shaking. The computational experiment regarding to find the best usage of the shake is explained later. Then by knowing this size n, we generate a point O'', by knowing the shaking neighbourhood. If the value is better than local optimum ($F(O'') < F(O')$), the search is re-centered there ($O' \leftarrow O''$). Otherwise, should be back to initial solution and repeat the process again, until a computational time limit is reached. Stopping criterion is usually the maximum computing time since the last improvement, or the maximum number of iterations. In Algorithm 25, computational time is considered as the stopping condition. We use the limit of 6 minutes, the reason for which is explained later.

4.3.5 Tabu Search (TS) for Cache Allocation Problem

TS is a higher level heuristic procedure for optimization problems to obtain near optimal solutions. Tabu search continues the search whenever a local optimum is encountered by allowing non-improving moves. Also, cycling back to previous visited solution is avoided because a tabu list records recent attributes of moves. Therefore, moves from one potential solution to search for an improved solution with the same neighbourhood continue until some stopping criteria is met. Algorithm 26 describes TS for cache allocation problem.

Algorithm 26 Tabu Search for Cache Allocation Problem

```
Initialization:
Set computation time limit
Given an initial solution $O$ and a set of neighbourhood structures $N_k$ for $k$= 1,2,...,$k_{\max}$
$k_{\max}$=6
Set $\hat{O} \leftarrow O$
Set Tabu list $\leftarrow \emptyset$
while computation time < computation time limit do
    Set $X \leftarrow \emptyset$
    Set $O_0 \leftarrow O$
    while $O_0$=$O$ do
        Apply VND method and find the best neighbour $\bar{O} \in N_1(O) \bigcup ... \bigcup N_{k_{\max}}(O) \setminus X$
        if $\bar{O}$ is not tabu, or $\bar{O}$ is tabu but satisfies the aspiration criterion then
            Set $O \leftarrow \bar{O}$
            if $F(O) < F(\hat{O})$ then
                $\hat{O} \leftarrow O$
            end
        else
            $X = X \bigcup \{\bar{O}\}$
        end
    end
    Update Tabu list and aspiration criterion
    Determine computation time
end
```

In TS, instead of recording full solutions, attribute memory structures are used. Usually attribute memory are based on recording information about solution properties (attribute) that change in moving from one solution to another. If a potential solution has been previously visited within a certain period or if it has violated a rule, it is marked as "tabu" (forbidden), so the algorithm does not consider that solution again. Tabu lists are too powerful and sometimes prohibit attractive moves, even when there is no danger of cycling. So it is important to use an algorithmic device (aspiration criterion) that allows cancelation of tabu status. In aspiration criteria, when a solution value is better than the current best-known solution, the move is allowed even if it is tabu.

TS uses neighbourhood search procedure to iteratively move from one potential solution to a new improved solution in the neighbourhood until a stopping criterion has been satisfied. TS explores neighbourhood of each solution as the search progress. The solutions admitted to the new neighbourhood, are determined by the tabu list. The tabu list is a short-term set of solutions that have changed by moving from one solution to another. There are three main approaches to define tabu list size: fixed to a predetermined value, randomly chosen from a specific range, or dynamically change by adjusting the value (Salhi, 2002). We use 20 as the size of tabu list. Reasons are explained later in this chapter.

4.4 Computational Results

In this section, we present the results of computational experiments comparing lower bounds and upper bounds. Linearizations obtained using two different methods employed in computing the lower bounds. Also, Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS) are the methods for upper bounds.

All algorithms were coded in Visual C++ 2012 and C and run on a PC with Intel(R) Core(TM), 3.10GHz and 4GB RAM. IBM ILOG CPLEX Optimization Studio (often informally referred to simply as CPLEX) is an optimization software package, that is used for solving all linearizations to find lower bounds, that are implemented in the C programming language.

4.4.1 Design of Instances

Tawarmalani et al. (2009) present computational results when the number of proxies are seven. According to that, for all the experiments in this chapter we consider number of proxy servers ($|J|$) is 10. Number of objects ($|K|$) is generated with following specification: ten integer numbers with steps of 50 in the interval [50,500], five integer numbers with step of a hundred in the interval [600,1000] and nine integer numbers with steps of a thousand in the interval [2000,10000]. Therefore, there are twenty four different combination according to the different size of objects.

We generate size of each object (b_k) to be an integer random variable in the interval [1,100]. The capacity for each proxy is calculated as $s_j = \delta \sum_{k \in K} b_k$, when b_k is the size of each object k and δ is a coefficient which is calculated as $\frac{\gamma}{|J|}$. Test instances are calculated to find the best number for γ. When γ is one (10% of total capacity) nearly all the objects can go to proxies easily, so the problem is not particularly interesting. The number of γ should be small enough to limit the capacity, so many objects cannot fit into the proxies easily. Because of this reason, we choose 0.75 (7.5% of total size of all objects) and 0.5 (5% of total size of all objects) to have challenging instances. More details can be found in Appendix C (C.1 - C.2).

Probability for each object $k \in K$ is θ_k as a random number in interval [0,1]. Also c_o is a network cost to deliver an object from origin server and c_l is network cost to deliver an object from network locally. The ratio between c_l and c_o are important because it shows how having caches create a saving in Internet cost. Wong (2002) believes that web cache proxies saves 40% Internet cost, so if the aim is to save Internet cost, we should try the ratio near 40%. Therefore, we consider c_o=5 and c_l=2 for our instances.

4.4.2 Computational Results for Linearizations

We provide the results of computational experiments to demonstrate the efficiency of two proposed solution procedure for linearization. Optimization studio (CPLEX) is considered to solve the models, with the same data for both linearizations. The time limit is two CPU hours (7200 seconds) for each instance. For those instances that prove optimality before the set time, we consider the results as optimal solution and for the rest of instances the results count as an upper bound. Table 4.1 below is the list of acronyms for Tables 4.2 and 4.3.

Instance	Ins.
Number of objects	|K|
Integer Programming	IP
Linear Programming	LP
Number of objects in proxies	N_{object}
Computational time of the IP	T_{IP}

Table 4.1: List of Acronyms for Tables 4.2 and 4.3

Tables 4.2 and 4.3 describe computational results for linearization I and linearization II when capacity of each proxy is 5% and 7.5% of total size of all objects. In both Tables 4.2 and 4.3 we solve the integer program and the linear programming relaxation when time is set to two hours. It is clear that, Tables 4.2 for 5% capacity, linearization II has fastest computational time compared with linearization I. Also, computation time is getting bigger for |K| $\geq$2000. For one specific instance when |K|=900, linearization II takes 1178.41 seconds while linearization I takes 159.04 seconds, but when |K| is smaller than 2000, both linearizations have fairly small computational time.

Table 4.3 compares linearization I and linearization II when capacity of each proxy is 7.5% of total size of all objects. Linearization I has fastest computational time compare with linearization II. Computational time is getting bigger when |K| increases, especially for $K \geq$=5000. Just for one specific instance (|K|=300), both linearizations take nearly two hours to solve the problem. As a conclusion, For 7.5% capacity, on average, linearization I has fastest computational time compared with linearization II.

		Linearization I			Linearization II		
Ins.	| K|	IP	LP	T_{IP}	IP	LP	T_{IP}
1	50	38,772	38,096	340.55	38,772	38,096	3.26
2	100	158,062	157,089	1.62	158,062	158,055	1.03
3	150	344,013	340,253	3.26	344,021	340,733	2.62
4	200	556,423	556,423	2.01	556,423	556,423	1.55
5	250	905,661	905,593	4.03	905,661	905,593	2.72
6	300	1,242,402	1,236,318	3.93	1,242,458	1,238,290	3.12
7	350	1,683,366	1,683,293	1.43	1,683,360	1,683,293	2.9
8	400	2,281,415	2,273,341	7.19	2,281,431	2,273,341	4.23
9	450	2,907,556	2,890,297	4.98	2,907,382	2,891,418	4.98
10	500	3,471,393	3,457,982	6.79	3,471,297	3,457,982	4.9
11	600	5,198,983	5,177,227	9.63	5,198,999	5,177,858	7.38
12	700	6,791,495	6,772,326	15.43	6,791,495	6,773,535	8.53
13	800	9,049,558	9,031,618	15.43	9,049,563	9,031,618	12.45
14	900	11,986,783	11,982,900	159.04	11,986,795	11,982,900	1178.41
15	1000	14,512,502	14,489,600	30.39	14,512,482	14,489,600	22.82
16	2000	56,988,235	56,938,365	7206.64	56,988,261	56,941,200	4350.62
17	3000	130,228,446	130,208,000	7198.43	130,229,002	130,208,000	7237.37
18	4000	225,593,807	225,402,581	7204.05	225,593,503	225,417,000	5718.09
19	5000	359,745,118	359,690,820	7200.40	359,744,891	359,707,000	7339.71
20	6000	522,840,568	522,791,265	1571.07	522,840,566	522,806,000	2836.96
21	7000	718,498,348	718,351,737	7208.21	718,498,302	718,359,000	7235.98
22	8000	931,897,309	931,664,698	7207.40	931,896,538	931,705,000	7138.76
23	9000	1,178,351,539	1,178,306,631	4301.20	1,178,350,822	1,178,310,000	417.08
24	10000	1,439,815,907	1,439,672,376	1541.45	1,439,815,867	1,439,720,000	394.34
			Average	2135.19 sec		Average	1830.41 sec

Table 4.2: Linearization I & II Capacity of Each Proxy is 5%

		Linearization I			Linearization II		
Ins.	| K|	IP	LP	T_{IP}	IP	LP	T_{IP}
1	50	18043	17500	0.39	18043	18000	0.48
2	100	75677	71609	1852.67	75677	71788	1.76
3	150	157154	154509	2.08	157152	154663	3.00
4	200	254188	253639	3.29	254188	254029	2.54
5	250	410471	407046	3.06	410509	407821	2.76
6	300	584002	569745	7202.38	584092	571331	6989.89
7	350	758392	757222	5.80	758392	757950	4.32
8	400	1043436	1041283	3.52	1043340	1042998	5.43
9	450	1321291	1311676	7.38	1321201	1311728	8.53
10	500	1557948	1557813	8.13	1557948	1557948	6.88
11	600	2390527	2379367	8.75	2390594	2381839	9.61
12	700	3091745	3082626	14.32	3091623	3084146	13.46
13	800	4152082	4120777	18.67	4152188	4121541	14.63
14	900	5517771	5492808	20.31	5518008	5495972	14.26
15	1000	6634749	6630400	270.85	6634673	6632839	341.56
16	2000	25817505	25767054	27.99	25817459	25776200	187.81
17	3000	59023393	58958130	150.95	59023366	58963000	205.19
18	4000	103308823	103117562	190.63	103308664	103135000	73.76
19	5000	165010986	164884378	7209.15	165011691	164904000	5139.08
20	6000	238398170	238314949	628.19	238397969	238316000	6495.54
21	7000	328166133	328064783	1477.10	328166152	328094000	7193.00
22	8000	425416785	425086808	7263.24	425416920	425089000	7285.22
23	9000	538442611	537934360	7191.65	537996800	537960000	1205.98
24	10000	662239395	661726319	7450.75	662240360	661782000	7178.17
			Average	1708.80		Average	1765.95

Table 4.3: Linearization I & II Capacity of Each Proxy is 7.5%

In Tables 4.2 and 4.3 most instances solved easily, that means integer programming is more successful to solve our problem. The reason why integer programming is successful is linear programming relaxation gives good lower bounds. As we can see in tables, integer programming and linear programming relaxation's results are close together.

4.4.3 Computational Results for Metaheuristics

In previous section, two different linearizations for proposed model are introduced to find lower bounds. In this section, to obtain optimality for our formula (4.1.1), two known different methods are considered to find upper bounds: Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS). The results from applying a heuristic are upper bounds that can be compared with the lower bounds that are the results from linearizations. Through that we can assess the quality of the heuristics that we use. An initial solution is found in two different ways: constructive heuristic method and greedy algorithm that can be seen in Algorithms 22 and 23. As explained above two different capacities also are considered for computational results for metaheuristics, which are 5% and 7.5% of total size of all objects. Bear in mind that each object can go to proxy at most once, so when we decide the location of each object, then we can determine where clients can get the objects.

4.4.3.1 Parameter Setting

Before starting, we need to set a computational time limit, the best size for shaking and the best size for tabu list. For these reasons, we used four different instances out of twenty four, with two different capacities for our test instances. Based on initial experiments, we found out that reasonable results could be obtained with modest computation time of up to 10 minutes . In order to find the best computational time for our algorithms, we run each test instance on a five different time limits of 2, 4 , 6, 8 and 10 minutes. From the results, there is no significant difference on the results of computations if we run for 6, 8 and 10 minutes. Thus, we conclude that 6 minutes is sufficient enough to find the results we need. Objective function values are shown in Table 4.4.

Capacity is 5% of total size of all objects						
Ins.	|K|	2 min	4 min	6 min	8 min	10 min
1	500	3,785,885	3,787,802	3,781,446	3,780,480	3,780,480
2	1000	14,928,056	14,926,895	14,926,525	14,925,716	14,925,716
3	5000	388,978,631	378,617,960	361,073,306	361,073,306	361,073,306
4	10000	1,528,880,555	1,550,024,163	1,530,070,928	1,447,424,886	1,447,424,886
Capacity is 7.5% of total size of all objects						
Ins.	K	2 min	4 min	6 min	8 min	10 min
1	500	1,739,522	1,744,134	1,649,974	1,649,974	1,649,974
2	1000	7,614,109	7,181,605	7,089,519	7,089,519	7,089,519
3	5000	176,582,332	176,316,226	175,354,922	175,354,922	175,354,922
4	10000	727,597,321	720,647,718	672,427,516	671,973,632	671,973,632

Table 4.4: Finding the Best Computational Time Limit

Now we try to determine the best size for shaking. It is essential to note that n represents the size of shake. The solution of each n is considered as an objective function value. We start with n=2,3,...,10 and run IVND against different values of n. The aim is to find the minimum objective function value and the relevant value of n for each test instance. As we can observe in Table 4.5, minimum objective function values are more frequent when n=4. Therefore, we use n=4 for the size of shaking for our experiments.

Capacity is 5% of total size of all objects										
Ins.	|K|	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
1	500	5,466,998	4,245,321	3,801,256	5,115,708	4,999,465	5,791,343	5,683,221	5,575,100	5,521,039
2	1000	15,865,151	16,402,730	15,776,935	15,837,058	14,881,255	16,129,298	16,389,882	15,933,502	16,214,013
3	5000	366,950,540	370,631,635	362,596,004	361,932,120	366,190,849	369,391,270	378,559,517	364,908,373	371,826,279
4	10000	1,449,334,669	1,459,693,579	1,442,065,979	1,455,129,118	1,459,819,197	1,455,702,144	1,448,676,414	1,450,228,145	1,453,845,361
Capacity is 7.5% of total size of all objects										
Ins.	|K|	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
1	500	2,223,819	2,191,362	2,021,924	2,110,222	2,158,906	2,126,450	2,093,993	2,028,187	2,088,547
2	1000	8,138,951	7,375,708	7,489,792	7,828,614	7,455,684	7,852,298	7,919,560	7,637,221	7,466,210
3	5000	188,645,259	175,354,922	179,414,817	181,240,146	182,552,912	180,365,249	179,614,023	177,375,279	178,490,320
4	10000	685,998,873	690,046,279	689,261,350	678,892,533	687,622,209	699,638,458	696,901,369	694,382,852	686,184,234

Table 4.5: Finding the Best Size for Shaking

Finally, we are trying to find an appropriate tabu list size. The purpose of this is to find the best tabu list size for all problem instances. In our test instances, we set the size of tabu equals to 10, 15, 20, 25 and 30, in steps of five. We use same stopping condition as used for all algorithms (6 minutes) to provide a basis for comparison. In Table 4.6, after size 20, it is observed that there is no more improvements for the rest of sizes. Therefore, we set 20 as the size of tabu list in our final experiments for all instances.

Capacity is 5% of total size of all objects						
Ins.	|K|	size 10	size 15	size 20	size 25	size 30
1	500	5,553,567	4,859,572	4,684,139	5,016,630	4,728,601
2	1000	18,335,041	17,918,205	17,702,919	18,709,585	18,437,632
3	5000	417,040,766	392,239,076	377,771,938	381,408,235	446,392,385
4	10000	1,823,525,464	1,675,836,536	1,581,840,062	1,853,012,039	1,711,124,426
Capacity is 7.5% of total size of all objects						
Ins.	|K|	size 10	size 15	size 20	size 25	size 30
1	500	1,773,361	1,760,217	1,748,727	2,099,004	2,096,462
2	1000	8,287,151	8,815,430	8,025,195	8,976,563	8,274,716
3	5000	192,965,315	190,566,233	186,867,741	187,605,532	191,716,162
4	10000	710,879,064	677,202,483	675,495,203	687,315,019	693,939,666

Table 4.6: Finding the Best Size for Tabu List

4.4.3.2 Upper Bounds Results

As explained above, the instances are the same that we use for linearizations. We provide the results of computational experiments to demonstrate the effectiveness of two meta-heuristics algorithms: IVND and TS. Note that in the table gap_c represents the gap when the initial solution for IVND and TS is achieved through the constructive heuristic and gap_g when the initial solution is from the greedy algorithm. Two different capacities are considered for each proxy as explained before.

The greedy algorithm provides the better results compared with the constructive heuristic algorithm, for both capacities. The gap in each column represents the relative percentage gap that is calculated as $\frac{IVND-TS}{IVND}100\%$. Also, from the table, it is clear that except two of the results the rest are positive that shows TS provides better upper bounds compared with IVND. We can observe this difference, especially when initial solution is calculated through constructive heuristic algorithm. Moreover, for both capacities (5% and 7.5%) the recommended method for initial solution will be greedy algorithm rather than constructive heuristic due to the big average difference. Table 4.7 shows two different gap values are calculated, depending on whether the initial solution is attained from the constructive heuristic or the greedy algorithms for both IVND and TS.

		capacity is 5% of total size of all objects		capacity is 7.5% of total size of all objects	
Ins.	|K|	gap_c	gap_g	gap_c	gap_g
1	50	11.69%	0.25%	7.12%	-1.20%
2	100	25.68%	1.83%	16.79%	12.97%
3	150	11.79%	0.13%	16.91%	14.36%
4	200	19.89%	0.58%	29.00%	21.67%
5	250	28.06%	0.17%	26.56%	14.72%
6	300	22.60%	0.22%	36.88%	27.51%
7	350	25.94%	0.07%	26.91%	15.75%
8	400	26.88%	0.09%	19.41%	12.04%
9	450	2.29%	0.08%	18.17%	10.07%
10	500	0.71%	0.52%	24.67%	18.40%
11	600	4.32%	0.02%	27.72%	20.54%
12	700	4.91%	0.27%	10.16%	28.76%
13	800	0.73%	6.73%	6.05%	7.18%
14	900	0.50%	0.01%	5.48%	6.40%
15	1000	2.54%	5.39%	1.23%	5.34%
16	2000	3.36%	3.35%	5.72%	5.05%
17	3000	2.17%	2.54%	10.62%	4.59%
18	4000	2.42%	1.00%	0.59%	-0.64%
19	5000	1.41%	0.42%	2.00%	2.26%
20	6000	1.31%	1.81%	4.45%	4.36%
21	7000	0.79%	1.04%	3.42%	2.68%
22	8000	1.30%	7.77%	1.29%	2.11%
23	9000	0.83%	0.30%	0.44%	1.23%
24	10000	0.71%	0.07%	1.61%	1.02%
	Average	8.45%	1.44%	12.63%	9.88%

Table 4.7: Gaps Between Iterated Variable Neighbourhood Descent and Tabu Search

4.4.3.3 Comparison Upper Bounds and Lower Bounds

For the last part of computational results, we decide to compare the best lower bound with the best upper bounds. Table 4.8 shows the percentage of deviation between upper bounds and lower bounds. When the capacity of proxy is 5% of total size of all objects, linearization II shows better results compare with linearization I for lower bounds, while linearization I is better than linearization II when the capacity is 7.5%. Also, for upper bounds TS is more powerful compare with IVND for both capacities. Therefore, when capacity is 5%, linearization II is compared with TS, and for capacity 7.5% linearization I is contrasted with TS. The percentage of deviation is calculated as: $\frac{TS-LP}{TS}100\%$ when TS is the best upper bound and LPR is relevant linear programming relaxation for each capacity that is explained earlier. The superscript TS-LP shows the comparison between TS and LP (Linear Programming Relaxation). Each gap has a subscribe c or g that represents the constructive heuristic or the greedy algorithm that are used to find initial solutions.

		capacity is 5% of total size of all objects		capacity is 7.5% of total size of all objects	
Ins.	|K|	gap_c^{TS-LP}	gap_g^{TS-LP}	gap_c^{TS-LP}	gap_g^{TS-LP}
1	50	12.71%	6.45%	-0.25%	3.58%
2	100	11.63%	3.45%	6.16%	5.31%
3	150	25.85%	7.56%	2.43%	1.86%
4	200	14.43%	1.44%	7.45%	2.88%
5	250	8.79%	4.86%	4.60%	3.23%
6	300	15.49%	2.12%	3.85%	2.01%
7	350	17.44%	6.35%	7.33%	4.53%
8	400	14.04%	4.95%	1.14%	0.35%
9	450	33.45%	0.04%	3.41%	1.21%
10	500	26.18%	8.55%	10.91%	5.58%
11	600	32.86%	0.92%	13.18%	2.92%
12	700	24.33%	1.97%	22.50%	6.00%
13	800	25.25%	0.26%	23.66%	8.20%
14	900	25.56%	0.87%	20.37%	4.52%
15	1000	18.15%	2.93%	17.35%	6.44%
16	2000	12.74%	7.00%	11.84%	8.79%
17	3000	12.08%	7.13%	9.32%	7.50%
18	4000	7.81%	5.09%	12.62%	6.15%
19	5000	4.78%	0.38%	11.75%	5.96%
20	6000	5.24%	3.86%	8.66%	4.59%
21	7000	8.56%	5.08%	5.28%	3.79%
22	8000	5.76%	1.44%	5.28%	2.76%
23	9000	6.43%	3.08%	3.27%	1.95%
24	10000	8.98%	0.09%	2.03%	1.52%
	Average	15.77%	3.58%	8.92%	4.23%

Table 4.8: Comparison Best Upper Bound and Best Lower Bound

It is obvious that greedy algorithm for initial solution is much more powerful compared with constructive heuristic algorithm for both capacities. Especially, when capacity is 5%, to find initial solution, the greedy algorithm is recommended.

4.5 Discussion

In this chapter, we propose a new version of cache allocation problem from Tawarmalani et al. (2009), where the formulation leads to an allocation of objects in a network of caches that collaborate in serving requests from clients. According to the original version of this model, the unit cost of serving an object locally from a cache where it is requested is c_l, from other caches in the network is c_n and from the origin server is c_o. Even if c_l, c_n and c_o are in practice dependent on the proxy j , we assume them to be constants. Also, each object is allocated at most once, and interesting instances of the problem have the property that all of the objects cannot be put in the proxies.

The demand d_k for an object k can be assigned to any proxy that holds object k. In new propose model, we consider a local cluster of proxies and aim to put every object in a proxy to avoid having to service requests from the slower origin server. Thus, c_l=c_n in our model. All proxies are identical except for their capacity, and there is no difference in cost according to the choice of proxy for any object since c_l=c_n . This means that whenever a demand for some object k occurs in the network, if some proxy j contains object k, then it will be considered as local cost. Therefore, two different costs are introduced: a local cost $c_l d_k$ when object k is in a proxy and an origin server cost $c_o d_k$ when object is not in any of the proxies.

The proposed model is solved to optimality by two different linearizations that are used to convert the original mixed 0-1 polynomial formulation into a 0-1 linear program. The computational results for linearization I and linearization II are compared with each other. The aim is to discover which of the two linearization models is more efficient and better with respect to computational time, when the feasible region will be the same for both linearization models. The validity of each linearization is proved, to show the equivalence between the original objective function and the linearized objective function.

With a view to designing metaheuristics, six different local search operators involving the insertion and swapping of objects are suggested as follows: insert from origin server to a proxy; insert from one proxy to another proxy; swap between origin server and a proxy; swap between two proxies; a cyclic exchange of three objects in which there is an insert from the origin server to a proxy j, an insert from proxy j to another proxy j' and an insert from proxy j' to the origin server; and remove-insert in which an object is inserted from a proxy j to the origin server and several objects are inserted from the original server to proxy j. Also, a randomized constructive heuristic and a greedy algorithm are two methods are introduced to find initial solutions. These initial solutions and the six neighbourhoods are used in Iterated Variable Neighbourhood Descent (IVND) and Tabu Search (TS), which are two metaheuristic methods that aim to find good-quality solutions of our optimization problem.

A range of different types of test instances have been generated. The instances all have 10 proxies, and the number of objects ranges from 50 to 10,000 with their sizes being generated from a uniform distribution. Each proxy has an identical capacity, and two different capacities are considered that correspond to scenarios involving fairly tight capacity constraints (a significant number of objects cannot be placed in the proxies) and looser capacity constraints (many of the objects can be placed in the proxies).

In this chapter, we provide the results of computational experiments to demonstrate the efficiency of two proposed solution procedure for linearization. Optimization studio (CPLEX) is considered to solve the models, with the same data for both linearizations. The time limit is two CPU hours for each instance. We prove that with 5% capacity, linearization II has fastest computational time compared with linearization I and with 7.5% capacity, linearization I is faster. Also, in both linearizations, integer programming is more successful because linear programming relaxation gives good lower bounds. In metaheuristics, we compare percentage gap value for IVND and TS with two initial solutions and two different capacities. Finally, we compare percentage gap value for the best upper bounds (TS) with the best lower bounds (LPR).

Next chapter is concluding remarks and future works. We provide an overall conclusion of the results presented in this thesis and also given an outline of some research which are worthwhile investigating in the future.

Chapter 5

Concluding Remarks and Future Work

This chapter provides an overall conclusion of the findings presented in this thesis and also given an outline of some research directions which are worthwhile investigating in the future.

5.1 Research Outcomes

This thesis presents background materials require to understand a Content Distribution Network (CDN). A CDN focuses on building its network infrastructure to provide different services such as: cache management, delivery of data, storage and management of content and distribution of content among edge servers. A content provider (customer) uses a CDN provider for service and having its content on the cache servers. Typically, customers of a CDN are media and Internet advertisement companies, Internet service providers (ISPs), mobile operators and other carrier companies. Each of these customers wants to deliver its contents to end-users on the Internet in a reliable timely and manner.

These days, the efficiency of distributing electronic content is affected by increasingly popularity of the World Wide Web and high demand for electronic information. Unacceptable delays for end-users often occur because the size of delivered content and number of users grow gradually. Because of these reasons, distribution of electronic content has become an important problem. CDN tries to improve the performance of a network by reducing the total cost related to distributing content. This is achieved through the client no longer being served by the origin server, but instead is served by a nearby proxy server. For this reason, a good design for a CDN is to locate content as close as possible to the clients.

Object retrieval and request routing is one of the optimization models studied in Chapter 3. A novel feature of the model is the introduction of a delay term into the objective function. The performance of a CDN degrades when it is operating close to capacity, and the new model captures this loss of performance by introducing a new term into the objective function. This model is related to data placement of cooperative caching, which determines a placement of replicated objects between nodes in the network with a given access pattern and aims to minimize the average access cost. The objective is to minimize overall latency of searching the target Web server, subject to the availability of system resources and traffic pattern.

Another contribution of this thesis is to introduce in Chapter 3 suggested methodologies to deal with non-linear terms that are introduced into the objective function in this chapter. Lagrangian relaxation and decomposition approaches are used to find a lower bound for this problem. Two complicating constraints, which are related to the amount of traffic in a link and the consistency between assignment variables, are relaxed to form the Lagrangian problem. The relaxed problem decomposes into three sub-problems. To update the multipliers, two different sub-gradients are considered for each category of multiplier. Also, two different linearization schemes are introduced for this problem. The first linearization (Scheme I) finds a lower bound through perspective cuts when tangents of the function are used as valid inequalities to provide a lower bound for auxiliary variables representing the non-linear terms. The second linearization (Scheme II) uses linear segments between two consecutive integer points. Computational results indicate that the approach with Scheme II provides stronger lower bound compared to Scheme I.

Also, we design a new set of instances for the object retrieval and request routing problem. Test instances are categorized as small with 2 clients, 3 proxies and 5 objects, or large with 10, 50 or 100 clients, 5, 10 or 15 proxies, and 40, 200 or 400 objects. The capacity of each proxy for the large instances is 0.8 or 0.9 of the total size of all objects, and the capacity of each link is 0.6, 0.7, 0.8 of 0.9 of the total size of all objects. Thus, each size of the large instances is categorized as one of 8 types depending on the different combinations of these capacities.

Another contribution of this thesis is new optimization model in Chapter 4, which involves caching in a network (cache allocation problem). Caching is the process of replicating Web content in different locations so that clients can gain access to objects easily. Caching can be performed at various levels: at the proxy server, at the browser and Web server. The main idea for a basic formulation is from Tawarmalani et al. (2009). All the objects cannot put in any proxy because its capacity is limited. We consider a set of proxies of varying size, an origin server, and different size for objects. Further, the cache allocation problem that is addressed has a local cluster of proxies and attempts to put each object in a proxy so that the use of a slower origin server is avoided. In the objective function, there is no difference in cost between putting an object in one proxy or another. The cache allocation problem is naturally a non-linear model. Sometimes non-linear models can be transformed into linear form by using auxiliary constraints because there are many powerful software tools for linear problems but the tools are less good for non-linear problems. Also, solving non-linear problems is costly comparing with linear problems. Two different linearizations are considered for this problem to convert the original mixed polynomial formulation to linear program. The aim is to find out which one of the linearization model is more efficient with respect to computation time.

The next contribution is designing new set of instances and test instances. The test instances that we use each have 10 proxies, and the number of objects ranges from 50 to 10,000 with their sizes being generated from a uniform distribution. Each proxy has an identical capacity, and two different capacities (5% and 7.5% of total size of all objects) are considered that reflect the tightness of the capacity constraints of the proxies and hence the need for requests to be served from the origin server. It is clear that linearization II has smaller computational time compare with linearization I. Linearization I has smaller computational time compare with linearization II when capacity of proxy is 7.5% of total size of all objects, but when the capacity of 5% of total size of objects, Linearization II is faster. Integer programming is more successful to solve our problem, because each instance easily solved. The reason why integer programming is successful is linear programming relaxation gives good lower bounds. Our results show that integer programming and linear programming relaxations produce solution values that are close to each other.

As a general conclusion, Iterated Variable Neighbourhood Descent (IVND) from literature is supposed to be a good method but we found the better algorithm with Tabu Search (TS).

5.2 Further Research

The current trend in metaheuristics to design hybrids that incorporate evolutionary strategies and local search, and focus on balancing the quality of solutions with diversity. For further research, these strategies can be applied to our proposed optimization models in this thesis.

Moreover, origin server redirection can be studied, which is the problem of connecting the client's request for an object directly to the origin server, when the object is not available in any other proxy servers. When the storage capacities of proxy servers are limited, there are many requests forwarded to the origin server, which creates issues regarding bandwidth. Also, other request routing techniques can also be considered to direct client requests for objects to a proxy that could best serve that content, rather than other proxies. In the cache allocation problem, an object can be divided for transmission and irrespective of whether the entire object is brought from one source or different sources, and assembly would be provided to regenerate the original object.

In cache allocation problem a new method can be created to highlight what percentage of objects are allocated in proxies and what percentage of full capacity of each proxy has been consumed. Also, generalization of different costs can be considered for this problem where costs depends on which proxy stores the object.

5.3 Final Remark

Overall, we believe that the work presented in this thesis contributes positively to scientific research in the design and operation of a Content Distribution Network (CDN). This study provides an insight into assignment and cache allocation problems. All the algorithm that are used in this research can be integrated within larger frameworks as an optimization tools, thereby saving more on Internet costs.

Appendix A

Computational Results for Small Instances

Ins.	s_j	D	LR	LPR1	LPR2	$gap_{LR-LPR1}$	$gap_{LR-LPR2}$	T_{LR}	T_{LPR1}	T_{LPR2}
1	15	250	4,326.12	4,629.64	4,629.64	-7.02%	-7.02%	1.78	0.06	0.05
2	15	500	8,633.97	9,129.64	9,129.64	-5.74%	-5.74%	1.81	0.06	0.06
3	15	700	11,960.85	11,726.32	12,729.64	2.0%	-6.40%	2.05	0.06	0.05
4	15	1000	17,071.43	16,694.64	18,129.64	2.20%	-6.20%	1.68	0.06	0.05
5	20	250	4,758.89	4,619.20	4,619.20	2.94%	2.94%	2.47	0.06	0.07
6	20	500	9,406.46	9,119.2	9,119.20	3.05%	3.05%	3.53	0.06	0.06
7	20	700	13,103.94	11,786.64	12,719.20	10.05%	2.94%	2.00	0.09	0.08
8	20	1000	18,690.18	16,786.64	18,119.20	10.18%	3.05%	2.56	0.08	0.08
9	25	250	4,758.89	4,619.20	4,619.20	2.94%	2.94%	2.47	0.05	0.04
10	25	500	9,406.46	9,119.2	9,119.20	3.05%	3.05%	3.53	0.05	0.05
11	25	700	13,103.90	12,719.20	12,719.20	2.94%	2.94%	2.00	0.03	0.03
12	25	1000	18,690.18	18,119.20	18,119.20	3.05%	3.05%	2.56	0.03	0.03
13	30	250	5,125.38	4,619.20	4,619.20	9.88%	9.88%	1.95	0.01	0.01
14	30	500	10,140.50	9,119.20	9,119.20	10.07%	10.07%	1.71	0.02	0.01
15	30	700	14,148.33	12,719.20	12,719.20	10.10%	10.10%	1.73	0.03	0.02
16	30	1000	20,151.26	18,119.20	18,119.20	10.08%	10.08%	1.67	0.03	0.03
					Average	4.36%	2.42%			

Table A.1: Computational Results for Small Instances (1)

Ins.	s_j	D	Neos	LR	LPR1	LPR2	gap_{N-LR}	gap_{N-LPR1}	gap_{N-LPR2}	T_{LR}	T_{LPR1}	T_{LPR2}
1	15	250	6,884.06	4,326.12	4,629.64	4,629.64	37.16%	32.75%	32.75%	1.78	0.06	0.05
2	15	500	13,634.06	8,633.97	9,129.64	9,129.64	36.67%	33.04%	33.04%	1.81	0.06	0.06
3	15	700	18,817.48	11,960.85	11,726.32	12,729.64	36.44%	37.68%	32.35%	2.05	0.06	0.05
4	15	1000	27,095.86	17,071.43	16,694.64	18,129.64	37.00%	38.39%	33.09%	1.68	0.06	0.05
5	20	250	5,967.85	4,758.89	4,619.20	4,619.20	20.26%	22.60%	22.60%	2.47	0.06	0.07
6	20	500	11,801.18	9,406.46	9,119.20	9,119.20	20.29%	22.73%	22.73%	3.53	0.06	0.06
7	20	700	16,467.85	13,103.94	11,786.64	12,719.20	20.43%	28.43%	22.76%	2.00	0.09	0.08
8	20	1000	23,467.85	18,690.18	16,786.64	18,119.20	20.36%	28.47%	22.79%	2.56	0.08	0.08
9	25	250	5,967.85	4,758.89	4,619.20	4,619.20	20.26%	22.60%	22.60%	2.47	0.05	0.04
10	25	500	11,801.18	9,406.46	9,119.20	9,119.20	20.29%	22.73%	22.73%	3.53	0.05	0.05
11	25	700	16,467.85	13,103.90	12,719.20	12,719.20	20.43%	22.76%	22.76%	2.00	0.03	0.03
12	25	1000	23,467.85	18,690.18	18,119.20	18,119.20	20.36%	22.79%	22.79%	2.56	0.03	0.03
13	30	250	5,967.85	5,125.38	4,619.20	4,619.20	14.12%	22.60%	22.60%	1.95	0.01	0.01
14	30	500	11,801.18	10,140.50	9,119.20	9,119.20	14.07%	22.73%	22.73%	1.71	0.02	0.01
15	30	700	16,467.85	14,148.33	12,719.20	12,719.20	14.09%	22.76%	22.76%	1.73	0.03	0.02
16	30	1000	23,467.85	20,151.26	18,119.20	18,119.20	14.13%	22.79%	22.79%	1.67	0.03	0.03
						Average	22.90%	26.61%	25.24%			

Table A.2: Computational Results for Small Instances (2)

Appendix B

Computational Results

B.1 Report of the Results on the Network Transfer Cost and Delay Cost Individually

Ins.	|I|	|J|	|K|	Network Transfer Cost	Delay Cost
1	10	5	40	371,105.46	1,278,372.39
2	10	5	200	1,927,993.41	7,772,358.92
3	10	5	400	3,943,996.16	16,930,451.96
4	10	10	40	379,196.53	1,258,931.50
5	10	10	200	1,887,121.32	7,730,474.53
6	10	10	400	3,973,599.00	18,364,950.97
7	10	15	40	351,910.91	1,575,147.85
8	10	15	200	1,849,996.86	8,008,598.72
9	10	15	400	4,223,647.02	18,155,606.59
10	50	5	40	1,801,436.90	43,421,562.78
11	50	5	200	9,519,853.62	266,236,610.03
12	50	5	400	19,305,714.56	681,954,106.98
13	50	10	40	2,204,211.92	58,370,305.55
14	50	10	200	9,583,798.10	330,062,970.20
15	50	10	400	19,180,366.94	778,614,747.38
16	50	15	40	1,790,170.61	72,254,717.58
17	50	15	200	9,267,876.60	389,792,107.40
18	50	15	400	19,577,354.86	895,162,693.15
19	100	5	40	3,563,283.10	175,837,824.75
20	100	5	200	18,720,121.52	111,542,725.44
21	100	5	400	36,759,229.03	2,830,357,715.06
22	100	10	40	3,489,486.34	250,714,405.69
23	100	10	200	19,128,528.96	1,436,316,890.60
24	100	10	400	37,647,216.91	3,347,091,897.44
25	100	15	40	3,648,877.09	325,223,475.65
26	100	15	200	19,121,116.97	1,783,398,778.56
27	100	15	400	38,143,072.88	3,994,147,011.56

Table B.1: Results for Network Transfer Cost and Delay Cost

B.2 Finding the Best Maximum Number for Iteration in Lagrangian Relaxation

Iteration	Global bound	Iteration	Global bound	Iteration	Global bound	Iteration	Global bound
0	1,007,729.54	26	6,265,142.33	52	6,901,911.52	78	6,925,558.19
1	1,007,729.54	27	6,344,894.92	53	6,905,360.98	79	6,925,667.85
2	1,007,729.54	28	6,415,513.37	54	6,908,738.45	80	6,925,724.38
3	1,007,729.54	29	6,481,852.74	55	6,910,651.30	81	6,925,777.03
4	1,007,729.54	30	6,522,224.28	56	6,912,330.01	82	6,925,832.26
5	1,007,729.54	31	6,565,263.62	57	6,914,130.28	83	6,925,884.49
6	1,007,729.54	32	6,605,095.32	58	6,915,774.94	84	6,925,938.68
7	1,007,729.54	33	6,642,256.25	59	6,917,532.72	85	6,925,967.39
8	1,007,729.54	34	6,681,449.00	60	6,918,446.49	86	6,925,993.43
9	1,081,811.02	35	6,705,991.93	61	6,919,271.58	87	6,926,021.25
10	1,439,458.94	36	6,730,832.39	62	6,920,176.43	88	6,926,047.11
11	1,777,085.59	37	6,752,498.36	63	6,920,993.51	89	6,926,074.44
12	2,175,920.75	38	6,772,950.70	64	6,921,877.61	90	6,926,088.77
13	2,460,286.78	39	6,792,601.62	65	6,922,329.37	91	6,926,102.30
14	4,206,335.24	40	6,806,950.84	66	6,922,748.33	92	6,926,115.81
15	4,317,412.60	41	6,819,340.21	67	6,923,191.53	93	6,926,129.10
16	4,589,779.53	42	6,832,574.02	68	6,923,607.97	94	6,926,142.38
17	4,904,905.59	43	6,844,503.89	69	6,924,040.40	95	6,926,149.69
18	5,125,119.34	44	6,856,276.30	70	6,924,270.90	96	6,926,156.54
19	5,407,857.24	45	6,863,601.34	71	6,924,477.22	97	6,926,163.24
20	5,575,503.72	46	6,870,712.08	72	6,924,701.76	98	6,926,169.97
21	5,724,965.87	47	6,877,587.69	73	6,924,907.67	99	6,926,169.97
22	5,853,914.04	48	6,884,377.55	74	6,925,126.26		
23	5,968,789.05	49	6,891,103.85	75	6,925,239.99		
24	6,079,109.45	50	6,894,954.05	76	6,925,343.50		
25	6,183,368.88	51	6,898,527.06	77	6,925,455.31		

Table B.2: |I|=10, |J|=15, |K|=200, δ=0.8, δ'=0.9

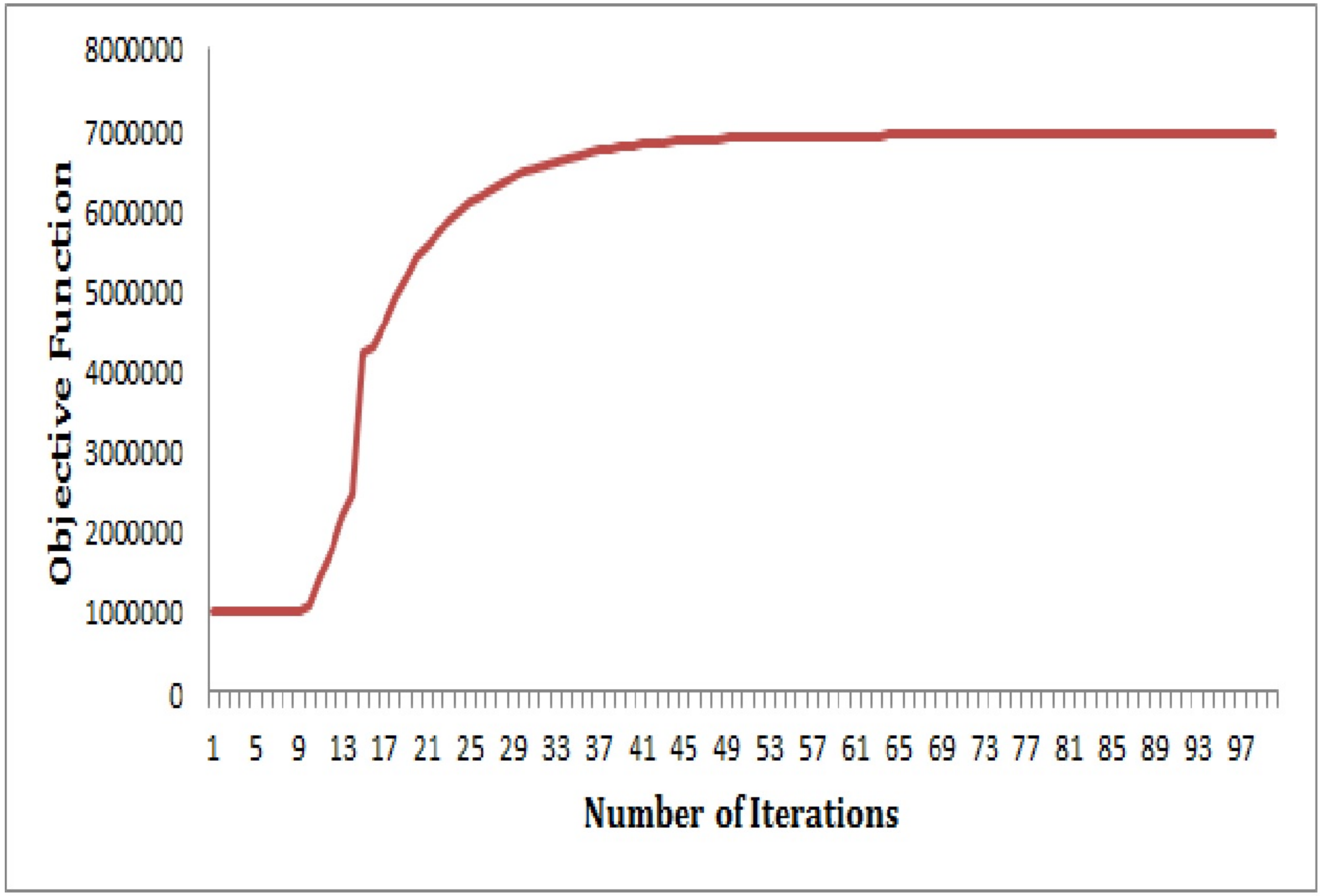

Figure B.1: Global Bound Respect to 100 Iterations 1

Iteration	Global bound	Iteration	Global bound	Iteration	Global bound	Iteration	Global bound
0	2,109,497.05	26	13,963,290.10	52	15,392,414.05	78	15,449,529.86
1	2,109,497.05	27	14,128,529.70	53	15,400,592.46	79	15,449,794.29
2	2,109,497.05	28	14,285,593.28	54	15,409,007.09	80	15,449,928.03
3	2,109,497.05	29	14,423,963.12	55	15,413,416.51	81	15,450,057.02
4	2,109,497.05	30	14,546,911.25	56	15,417,495.57	82	15,450,189.26
5	2,109,497.05	31	14,641,631.27	57	15,421,828.97	83	15,450,317.68
6	2,109,497.05	32	14,734,507.45	58	15,425,875.05	84	15,450,448.09
7	2,109,497.05	33	14,814,458.25	59	15,430,119.38	85	15,450,517.33
8	2,109,497.05	34	14,883,166.07	60	15,432,274.32	86	15,450,580.49
9	2,109,497.05	35	14,947,223.67	61	15,434,320.80	87	15,450,648.27
10	2,677,616.25	36	15,000,362.62	62	15,436,464.37	88	15,450,710.77
11	3,370,565.38	37	15,051,455.53	63	15,438,498.89	89	15,450,777.82
12	4,256,120.44	38	15,097,042.28	64	15,440,612.02	90	15,450,810.36
13	4,696,305.15	39	15,139,664.15	65	15,441,698.04	91	15,450,843.43
14	9,106,665.61	40	15,171,392.42	66	15,442,711.73	92	15,450,875.68
15	9,453,071.19	41	15,202,383.01	67	15,443,790.05	93	15,450,908.59
16	10,026,266.64	42	15,229,030.35	68	15,444,800.40	94	15,450,940.42
17	10,838,945.75	43	15,256,561.35	69	15,445,864.33	95	15,450,957.99
18	11,459,041.00	44	15,284,880.21	70	15,446,399.12	96	15,450,973.92
19	11,916,189.61	45	15,301,219.04	71	15,446,911.37	97	15,450,990.73
20	12,353,274.62	46	15,317,756.93	72	15,447,444.73	98	15,451,006.47
21	12,741,656.07	47	15,334,406.70	73	15,447,953.75	99	15,451,006.47
22	13,064,760.28	48	15,350,602.61	74	15,448,480.28		
23	13,346,301.36	49	15,366,499.94	75	15,448,752.79		
24	13,556,401.60	50	15,375,431.67	76	15,449,008.61		
25	13,805,933.28	51	15,383,766.60	77	15,449,276.01		

Table B.3: |I|=10, |J|=15, |K|=400, δ=0.8, δ'=0.9

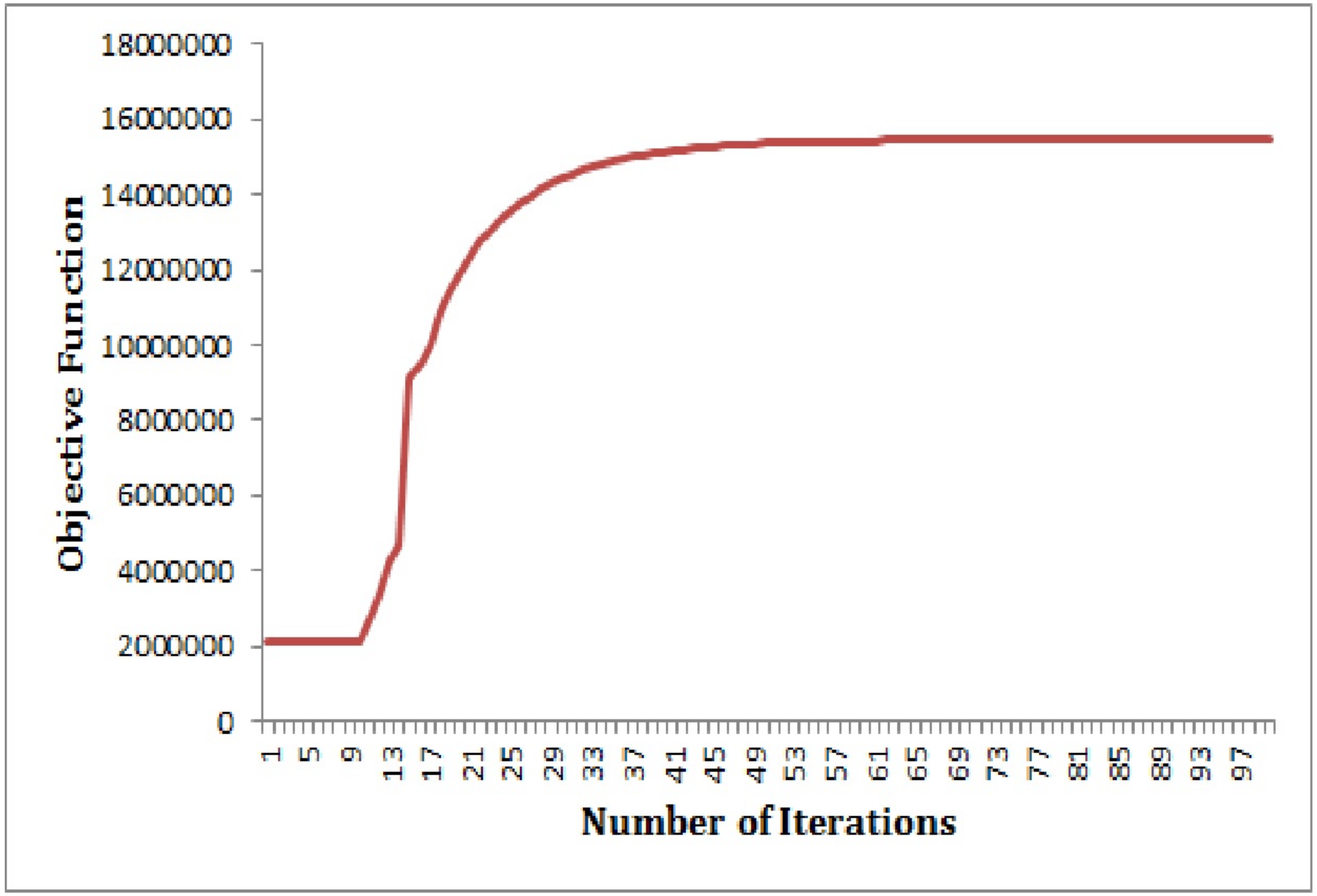

Figure B.2: Global Bound Respect to 100 Iterations 2

Iteration	Global bound	Iteration	Global bound	Iteration	Global bound	Iteration	Global bound
0	1,187,135.81	26	35,974,297.29	52	36,552,128.38	78	36,570,119.05
1	1,187,135.81	27	36,059,191.16	53	36,555,902.08	79	36,570,151.22
2	1,187,135.81	28	36,120,381.64	54	36,555,902.08	80	36,570,243.15
3	1,187,135.81	29	36,181,391.21	55	36,557,893.40	81	36,570,327.08
4	1,187,135.81	30	36,253,005.74	56	36,560,733.95	82	36,570,331.87
5	9,015,113.81	31	36,282,342.69	57	36,560,733.95	83	36,570,416.49
6	11,814,699.62	32	36,319,876.71	58	36,562,871.54	84	36,570,420.09
7	14,434,257.32	33	36,342,932.72	59	36,562,871.54	85	36,570,476.32
8	20,966,659.81	34	36,378,416.74	60	36,563,581.75	86	36,570,512.10
9	25,556,962.07	35	36,417,716.37	61	36,565,208.94	87	36,570,523.51
10	27,057,973.06	36	36,417,716.37	62	36,565,265.01	88	36,570,557.19
11	27,886,899.28	37	36,448,050.77	63	36,566,625.51	89	36,570,569.74
12	30,805,124.02	38	36,448,050.77	64	36,566,625.51	90	36,570,600.51
13	31,532,180.23	39	36,476,935.42	65	36,566,968.29	91	36,570,602.05
14	31,900,011.40	40	36,494,705.10	66	36,567,831.01	92	36,570,624.58
15	33,008,230.75	41	36,494,705.10	67	36,567,886.39	93	36,570,624.58
16	33,665,812.13	42	36,509,824.76	68	36,568,536.74	94	36,570,645.02
17	34,158,584.44	43	36,509,824.76	69	36,568,536.74	95	36,570,649.78
18	34,559,095.26	44	36,522,123.57	70	36,568,924.02	96	36,570,659.66
19	34,832,887.38	45	36,530,632.60	71	36,569,226.76	97	36,570,660.93
20	35,128,774.46	46	36,530,632.60	72	36,569,257.32	98	36,570,671.32
21	35,387,748.21	47	36,539,347.59	73	36,569,560.65	99	36,570,671.32
22	35,493,158.11	48	36,539,347.59	74	36,569,586.24		
23	35,636,119.82	49	36,546,221.69	75	36,569,798.30		
24	35,756,437.52	50	36,548,948.96	76	36,569,947.03		
25	35,923,821.21	51	36,551,400.52	77	36,569,983.43		

Table B.4: |I|=50, |J|=5, |K|=40, δ=0.9, δ'=0.9

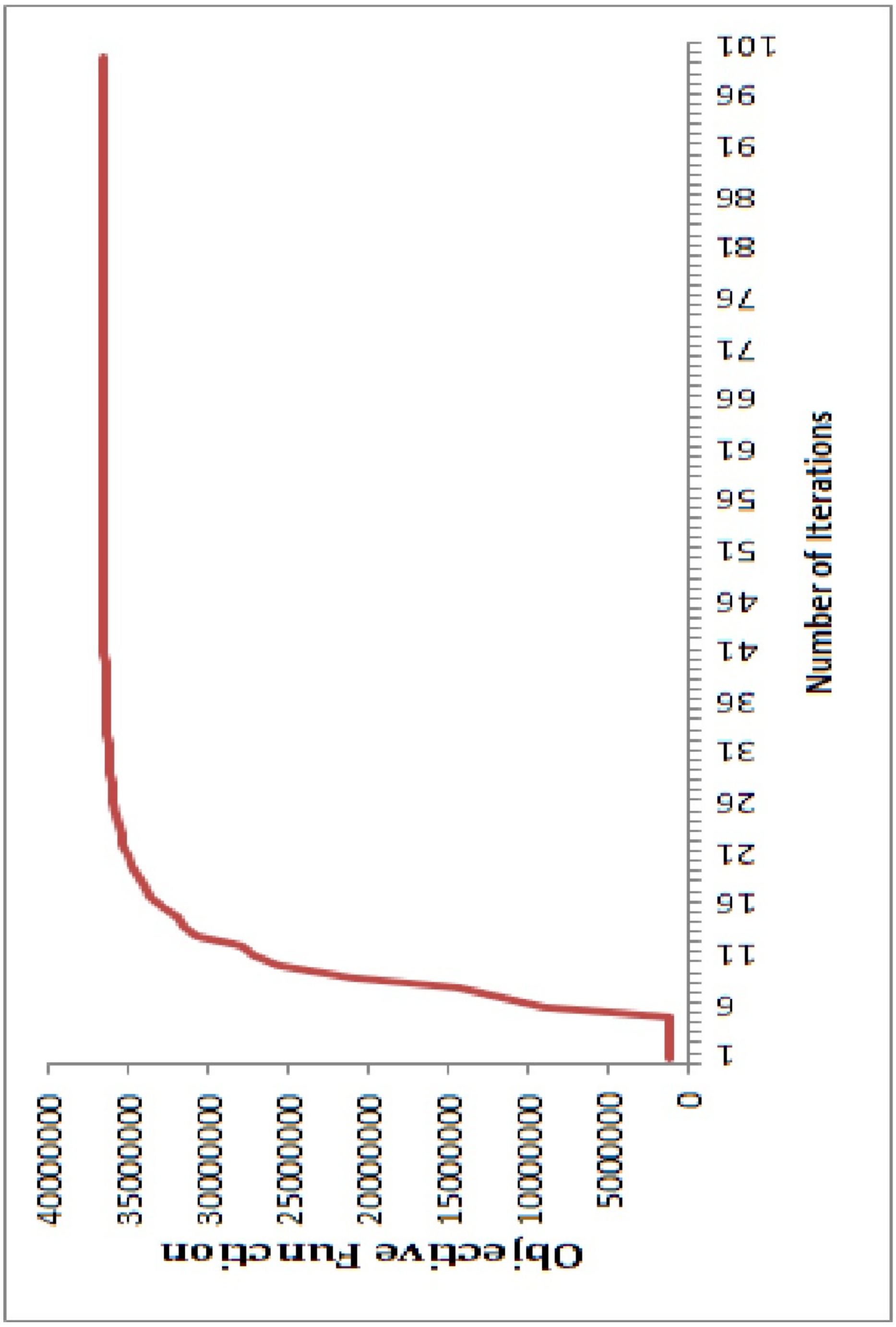

Figure B.3: Global Bound Respect to 100 Iterations 3

Iteration	Global bound	Iteration	Global bound	Iteration	Global bound	Iteration	Global bound
0	2,243,415.13	26	208,864,913.02	52	211,317,268.44	78	211,395,702.18
1	2,243,415.13	27	209,099,917.08	53	211,317,268.44	79	211,396,555.78
2	2,243,415.13	28	209,402,870.02	54	211,336,498.91	80	211,396,845.12
3	2,243,415.13	29	209,667,153.03	55	211,342,416.29	81	211,397,195.28
4	11,550,013.30	30	209,962,063.82	56	211,347,228.71	82	211,397,432.90
5	95,153,918.97	31	210,128,880.17	57	211,352,357.21	83	211,397,794.26
6	116,002,217.04	32	210,280,584.55	58	211,356,848.45	84	211,398,009.75
7	121,976,415.63	33	210,393,569.21	59	211,361,765.66	85	211,398,272.64
8	136,764,114.08	34	210,502,541.52	60	211,368,985.86	86	211,398,442.78
9	157,542,665.24	35	210,688,752.92	61	211,368,985.86	87	211,398,602.76
10	178,664,866.47	36	210,735,679.78	62	211,374,401.09	88	211,398,778.22
11	181,570,079.75	37	210,840,468.85	63	211,374,401.09	89	211,398,930.20
12	185,966,175.35	38	210,872,052.89	64	211,380,469.39	90	211,399,048.56
13	188,704,376.56	39	210,947,320.31	65	211,381,808.76	91	211,399,133.27
14	196,081,021.50	40	211,039,490.38	66	211,383,762.79	92	211,399,228.39
15	197,502,921.37	41	211,054,246.09	67	211,384,808.69	93	211,399,301.10
16	200,735,885.59	42	211,106,072.36	68	211,387,124.23	94	211,399,396.11
17	201,354,948.48	43	211,128,922.47	69	211,387,546.92	95	211,399,451.42
18	202,586,421.07	44	211,163,198.81	70	211,389,691.93	96	211,399,498.69
19	203,536,255.95	45	211,211,028.39	71	211,390,704.23	97	211,399,541.21
20	205,103,044.52	46	211,214,064.90	72	211,391,571.43	98	211,399,583.29
21	206,052,460.44	47	211,246,949.28	73	211,392,518.00	99	211,399,583.29
22	206,726,261.93	48	211,246,949.28	74	211,393,339.06		
23	207,274,353.09	49	211,280,984.89	75	211,394,292.66		
24	207,877,632.16	50	211,297,167.70	76	211,394,635.18		
25	208,478,159.58	51	211,300,756.06	77	211,395,442.31		

Table B.5: |I|=100, |J|=5, |K|=40, δ=0.8, δ'=0.8

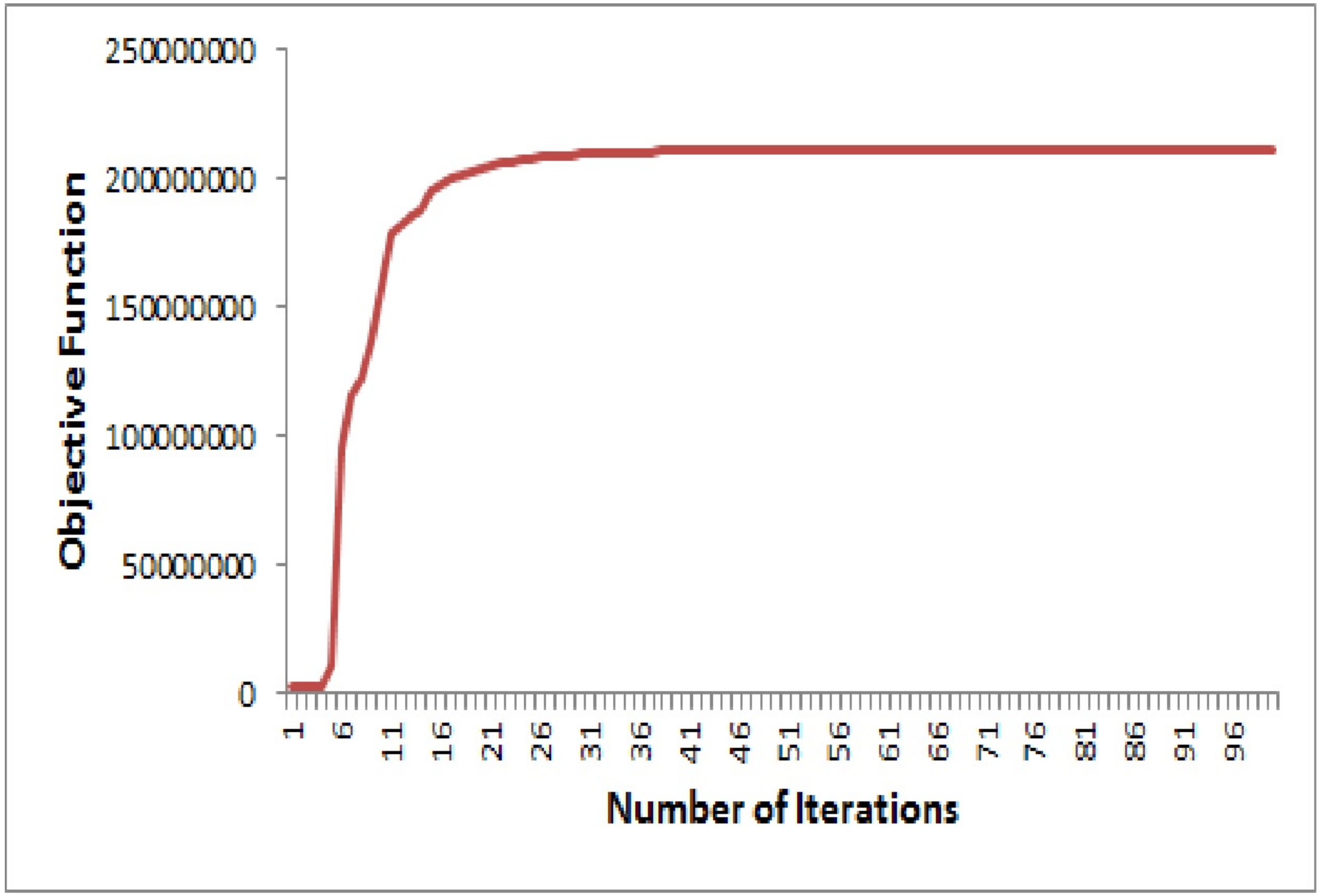

Figure B.4: Global Bound Respect to 100 Iterations 4

B.3 Mixed Integer Programming (MIP) and Linear Programming Relaxation (LPR)

δ=0.8 δ'=0.6									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	2,903,769.19	2,903,233.88	7199.99	2,903,797.50	2,903,235.57	7199.99
2	10	10	40	2,879,222.96	2,878,154.64	7204.66	2,879,136.02	2,878,167.77	7200.04
3	10	15	40	2,953,841.97	2,950,734.96	7208.82	2,953,759.94	2,950,738.56	7214.26
4	50	5	40	88,390,143.98	88,237,648.79	7209.15	88,362,286.87	88,237,996.95	6896.18
5	50	10	40	104,558,189.96	104,279,448.74	7204.98	104,521,873.84	104,279,427.29	7209.65
6	50	15	40	124,437,116.56	123,771,089.38	7314.53	124,283,403.64	123,771,576.99	7202.19
7	100	5	40	356,926,405.49	355,955,820.86	7360.63	356,686,789.48	355,956,347.43	7186.50
8	100	10	40	443,125,039.76	441,246,767.27	7182.43	443,209,467.02	441,247,492.38	7429.35
9	100	15	40	557,571,180.92	553,228,938.29	7247.07	557,616,082.40	553,197,822.97	7415.71

Table B.6: Mixed Integer Programming and Linear Programming Relaxation(1)

δ=0.9 δ'=0.6									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	2,447,686.01	2,447,358.32	7199.98	2,447,721.35	2,447,361.92	7200.06
2	10	10	40	2,529,236.68	2,528,327.39	7199.99	2,529,146.54	2,528,328.07	7200.18
3	10	15	40	2,616,267.15	2,614,470.79	7207.48	2,616,619.89	2,614,475.72	7208.64
4	50	5	40	72,971,666.14	72,946,713.02	7214.33	72,977,496.78	72,947,049.29	7200.34
5	50	10	40	90,515,177.71	90,296,777.83	7381.89	90,456,148.81	90,297,294.59	7210.30
6	50	15	40	108,631,439.55	108,272,566.82	7227.79	108,609,609.56	108,272,736.00	7741.80
7	100	5	40	294,107,821.33	293,669,592.32	7201.80	293,973,947.66	293,669,499.75	7210.82
8	100	10	40	382,482,025.69	381,483,522.77	7227.56	382,481,834.17	381,485,088.68	7203.22
9	100	15	40	486,735,021.13	483,449,460.00	7233.91	486,648,928.86	483,451,014.29	7226.95

Table B.7: Mixed Integer Programming and Linear Programming Relaxation (2)

δ=0.8 δ'=0.7									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	2,329,493.51	2,329,158.43	7199.98	2,329,490.49	2,329,168.92	7199.96
2	10	10	40	2,432,739.58	2,431,876.09	7205.58	2,432,695.75	2,431,866.54	7200.01
3	10	15	40	2,522,568.54	2,520,408.34	7210.16	2,522,049.42	2,520,415.95	7207.85
4	50	5	40	69,058,699.25	68,985,162.32	7214.72	69,075,382.05	68,985,281.48	7204.05
5	50	10	40	86,603,864.19	86,447,169.89	7192.97	86,625,722.06	86,447,310.81	7198.39
6	50	15	40	104,344,753.55	103,944,079.25	7233.36	104,396,166.12	103,944,492.39	7225.48
7	100	5	40	277,919,016.46	277,528,281.17	7189.70	278,047,867.52	277,527,971.25	7187.84
8	100	10	40	366,427,789.81	365,028,703.75	7413.99	366,424,817.99	365,029,144.93	7273.41
9	100	15	40	467,208,397.88	463,970,877.01	7214.98	467,105,602.94	463,972,293.71	7312.97

Table B.8: Mixed Integer Programming and Linear Programming Relaxation (3)

δ=0.9 δ'=0.7									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	2,001,760.45	2,001,457.31	7199.99	2,001,763.22	2,001,467.06	7200.04
2	10	10	40	2,151,690.87	2,151,040.77	7202.05	2,151,654.07	2,151,039.95	7200.12
3	10	15	40	2,244,863.58	2,243,228.30	7207.45	2,244,736.08	2,243,232.68	7210.54
4	50	5	40	58,060,318.91	58,013,287.41	7204.45	58,034,817.37	58,013,261.48	7212.49
5	50	10	40	75,356,619.21	75,259,039.69	7455.68	75,346,871.14	75,259,050.89	7212.76
6	50	15	40	91,506,197.73	91,214,412.63	7573.32	91,494,996.08	91,214,661.45	7628.14
7	100	5	40	233,063,738.29	232,836,226.90	7423.01	233,202,887.67	232,836,474.56	7468.31
8	100	10	40	318,049,230.79	317,211,632.29	7254.92	318,041,336.40	317,211,749.82	7248.57
9	100	15	40	409,125,247.01	406,686,628.77	7252.41	409,081,586.02	406,687,682.45	7353.95

Table B.9: Mixed Integer Programming and Linear Programming Relaxation (4)

δ=0.8 δ'=0.8									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	1,963,523.46	1,963,197.05	7200.06	1,963,500.18	1,963,190.13	7200.01
2	10	10	40	2,117,368.89	2,116,803.63	7204.78	2,117,426.24	2,116,801.80	7200.24
3	10	15	40	2,210,728.43	2,209,081.07	7214.97	2,210,518.35	2,209,083.02	7201.16
4	50	5	40	56,795,576.52	56,733,170.98	7630.46	56,772,459.36	56,733,195.94	7192.77
5	50	10	40	74,042,517.83	73,897,865.98	7187.06	74,056,044.09	73,897,956.80	7237.54
6	50	15	40	89,919,249.11	89,649,628.34	7203.52	89,938,791.56	89,649,946.00	7356.18
7	100	5	40	227,913,289.51	227,618,577.83	7181.47	227,945,718.58	227,619,090.47	7193.33
8	100	10	40	312,448,372.10	311,393,041.70	7305.20	312,480,487.95	311,394,015.64	7216.17
9	100	15	40	402,018,835.21	399,645,101.79	7278.38	399,646,005.10	399,646,005.10	7226.31

Table B.10: Mixed Integer Programming and Linear Programming Relaxation(5)

δ=0.9 δ'=0.8									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	1,709,431.61	1,709,110.28	7200.08	1,709,433.11	1,709,106.28	7200.01
2	10	10	40	1,882,551.90	1,882,100.15	7202.82	1,882,563.14	1,882,108.54	7202.11
3	10	15	40	1,974,122.68	1,972,974.21	7211.10	1,974,235.41	1,972,983.86	7214.09
4	50	5	40	48,269,492.48	48,248,579.64	7215.26	48,259,948.97	48,248,566.79	7203.08
5	50	10	40	64,667,481.63	64,586,078.98	7222.05	64,665,364.89	64,586,134.81	7255.29
6	50	15	40	79,056,611.93	78,854,991.69	7272.38	79,038,592.26	78,855,266.66	7449.05
7	100	5	40	193,245,750.12	193,060,366.81	7200.13	193,279,373.80	193,060,787.66	7338.75
8	100	10	40	272,266,359.84	271,598,280.27	7164.80	272,232,572.43	271,598,007.34	7444.80
9	100	15	40	352,765,917.28	351,072,232.33	7403.90	352,685,982.60	351,072,368.73	7899.98

Table B.11: Mixed Integer Programming and Linear Programming Relaxation (6)

δ=0.8 δ'=0.9									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	1,709,514.08	1,709,163.85	7200.07	1,709,515.49	1,709,160.56	7200.04
2	10	10	40	1,882,600.08	1,882,100.15	7200.29	1,882,564.81	1,882,108.54	7200.38
3	10	15	40	1,974,396.21	1,972,974.21	7211.08	1,974,329.37	1,972,983.86	7212.97
4	50	5	40	48,282,058.42	48,250,671.65	7264.53	48,316,086.04	48,250,656.76	7234.19
5	50	10	40	64,687,398.23	64,586,406.51	7197.25	64,705,118.09	64,586,461.72	7197.26
6	50	15	40	79,052,544.41	78,855,030.76	7219.51	79,103,985.79	78,855,305.75	7389.49
7	100	5	40	193,278,380.15	193,065,795.84	7216.98	193,319,306.60	193,066,214.51	7222.52
8	100	10	40	272,374,506.39	271,599,069.84	7241.80	272,405,230.76	271,598,797.92	7231.05
9	100	15	40	356,711,247.14	351,072,450.79	7234.14	352,790,813.41	351,072,587.18	7559.48

Table B.12: Mixed Integer Programming and Linear Programming Relaxation (7)

δ=0.9 δ'=0.9									
Ins.	|I|	|J|	|K|	MIP1	LPR1	T_{MIP1}	MIP2	LPR2	T_{MIP2}
1	10	5	40	1,502,543.14	1,502,201.15	7200.01	1,502,539.46	1,502,199.84	7200.04
2	10	10	40	1,680,750.70	1,680,286.16	72001.24	1,680,748.67	1,680,293.33	7200.38
3	10	15	40	1,768,009.12	1,767,017.50	7200.76	1,768,139.72	1,767,025.51	7198.87
4	50	5	40	41,381,344.48	41,364,775.31	7207.67	41,382,407.96	41,364,767.90	7200.71
5	50	10	40	56,691,710.12	56,618,027.95	7327.34	56,694,492.32	56,618,213.84	7248.84
6	50	15	40	69,648,060.25	69,487,950.96	7707.82	69,662,823.16	69,487,786.19	7233.77
7	100	5	40	165,138,499.60	165,021,092.37	7196.08	165,150,140.22	165,021,019.60	7186.37
8	100	10	40	238,160,768.30	237,548,011.38	7268.40	238,108,562.52	237,548,073.09	7344.85
9	100	15	40	310,271,193.94	308,924,039.79	7274.95	310,207,458.30	308,924,462.04	7442.48

Table B.13: Mixed Integer Programming and Linear Programming Relaxation (8)

Ins.	|I|	|J|	|K|	δ	δ'	gap_1	T_{MIP1}	gap_2	T_{MIP2}
1	10	5	40	0.9	0.8	0.02%	7200.07	0.02%	7200.04
2	10	10	40	0.9	0.8	0.03%	7200.29	0.02%	7200.38
3	10	15	40	0.9	0.8	0.07%	7211.08	0.07%	7212.97
4	50	5	40	0.9	0.8	0.07%	7264.53	0.14%	7234.19
5	50	10	40	0.9	0.8	0.16%	7197.25	0.18%	7197.26
6	50	15	40	0.9	0.8	0.25%	7219.51	0.31%	7389.49
7	100	5	40	0.9	0.8	0.11%	7216.98	0.13%	7222.52
8	100	10	40	0.9	0.8	0.28%	7241.80	0.30%	7231.05
9	100	15	40	0.9	0.8	1.58%	7234.14	0.49%	7559.48
10	10	5	40	0.9	0.9	0.02%	7200.01	0.02%	7200.04
11	10	10	40	0.9	0.9	0.03%	72001.24	0.03%	7200.38
12	10	15	40	0.9	0.9	0.06%	7200.76	0.06%	7198.87
13	50	5	40	0.9	0.9	0.04%	7207.67	0.04%	7200.71
14	50	10	40	0.9	0.9	0.13%	7327.34	0.13%	7248.84
15	50	15	40	0.9	0.9	0.23%	7707.82	0.25%	7233.77
16	100	5	40	0.9	0.9	0.07%	7196.08	0.08%	7186.37
17	100	10	40	0.9	0.9	0.26%	7268.40	0.24%	7344.85
18	100	15	40	0.9	0.9	0.43%	7274.95	0.41%	7442.48
					Average	0.21%		0.16%	

Table B.14: Final Gaps Between Mixed Integer Programming and Linear Programming Relaxation

B.4 Lagrangian Relaxation and Linear Programming Relaxation

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.6 LR	δ=0.8 δ'=0.6 LPR	gap
1	10	5	40	2,776,396.20	2,903,233.88	4.37%
2	10	5	200	13,087,774.98	18,127,243.24	27.80%
3	10	5	400	41,312,013.26	42,824,000.00	3.53%
4	10	10	40	2,639,277.72	2,878,154.64	8.30%
5	10	10	200	15,320,976.95	15,640,349.74	2.04%
6	10	10	400	32,469,458.97	—	—
7	10	15	40	1,883,242.80	2,950,734.96	36.18%
8	10	15	200	9,682,782.03	14,537,400.00	33.39%
9	10	15	400	20,847,716.31	—	—
10	50	5	40	79,434,163.20	88,237,648.79	9.98%
11	50	5	200	365,473,946.83	—	—
12	50	5	400	1,325,025,142.33	—	—
13	50	10	40	88,321,056.54	104,279,448.74	15.30%
14	50	10	200	471,156,745.34	—	—
15	50	10	400	1,153,073,595.99	—	—
16	50	15	40	60,090,980.97	123,771,089.38	51.45%
17	50	15	200	314,544,047.83	—	—
18	50	15	400	74,065,008.26	—	—
19	100	5	40	339,051,850.55	355,955,820.86	4.75%
20	100	5	200	1,509,304,162.96	1,283,593,391.80	-17.58%
21	100	5	400	5,450,113,895.27	5,943,877,314.00	8.31%
22	100	10	40	375,240,696.54	441,246,767.27	14.96%
23	100	10	200	2,114,801,496.82	—	—
24	100	10	400	4,851,144,734.32	5,124,851,741.20	5.34%
25	100	15	40	260,800,493.12	553,228,938.29	52.86%
26	100	15	200	1,379,556,565.81	—	—
27	100	15	400	3,069,414,984.16	—	—

Average 16.31%

Table B.15: Lagrangian Relaxation and Linear Programming Relaxation I (1)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.7 LR	δ=0.8 δ'=0.7 LPR	gap
1	10	5	40	2,241,299.98	2329158.431	3.77%
2	10	5	200	14,964,732.98	11,152,724.09	-34.18%
3	10	5	400	33,037,437.14	27,062,428.53	-22.08%
4	10	10	40	2,237,181.98	2,431,876.09	8.01%
5	10	10	200	12,877,915.93	14,131,382.80	8.87%
6	10	10	400	27,711,262.91	—	—
7	10	15	40	1,661,307.49	2520408.34	34.09%
8	10	15	200	8,532,722.37	—	—
9	10	15	400	18,492,319.13	—	—
10	50	5	40	66,125,091.61	68985162.32	4.15%
11	50	5	200	293,590,242.00	—	—
12	50	5	400	1,035,377,033.13	—	—
13	50	10	40	74,317,622.36	86447169.89	14.03%
14	50	10	200	408,629,142.63	—	—
15	50	10	400	737,484,300.01	—	—
16	50	15	40	50,402,625.59	103944079.3	51.51%
17	50	15	200	268,705,396.37	—	—
18	50	15	400	610,276,667.28	—	—
19	100	5	40	265,069,236.82	277528281.2	4.49%
20	100	5	200	1,207,511,287.33	—	—
21	100	5	400	4,515,369,064.95	—	—
22	100	10	40	310,430,599.75	365028703.8	14.96%
23	100	10	200	1,732,014,211.83	—	—
24	100	10	400	4,058,070,901.17	—	—
25	100	15	40	216,530,843.20	463970877	53.33%
26	100	15	200	1,163,787,011.00	—	—
27	100	15	400	2,591,004,189.62	—	—

Average 11.74%

Table B.16: Lagrangian Relaxation and Linear Programming Relaxation I (2)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.8 LR	δ=0.8 δ'=0.8 LPR	gap
1	10	5	40	1,895,114.59	1963197.047	3.47%
2	10	5	200	12,565,967.13	—	—
3	10	5	400	27,740,275.99	—	—
4	10	10	40	1,956,598.56	2,116,803.63	7.57%
5	10	10	200	11,175,126.63	—	—
6	10	10	400	24,264,641.08	—	—
7	10	15	40	1,479,421.88	2209081.074	33.03%
8	10	15	200	7,602,213.37	—	—
9	10	15	400	16,664,793.76	—	—
10	50	5	40	53,041,421.43	56733170.98	6.51%
11	50	5	200	246,200,491.59	—	—
12	50	5	400	850,472,591.19	—	—
13	50	10	40	62,352,813.61	73897865.98	15.62%
14	50	10	200	349,745,739.15	—	—
15	50	10	400	819,472,471.50	—	—
16	50	15	40	43,698,013.65	89649628.34	51.26%
17	50	15	200	233,764,241.05	—	—
18	50	15	400	532,465,561.23	—	—
19	100	5	40	211,399,583.29	227618577.8	7.13%
20	100	5	200	1,010,023,661.45	—	—
21	100	5	400	3,483,840,003.50	—	—
22	100	10	40	261,154,104.89	311393041.7	16.13%
23	100	10	200	1,475,495,317.38	—	—
24	100	10	400	3,433,560,340.47	—	—
25	100	15	40	185,922,722.82	399645101.8	53.48%
26	100	15	200	1,007,337,039.98	—	—
27	100	15	400	2,243,495,135.00	—	—
					Average	21.58%

Table B.17: Lagrangian Relaxation and Linear Programming Relaxation I (3)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.9 LR	δ=0.8 δ'=0.9 LPR	gap
1	10	5	40	1,653,667.42	1,709,163.85	3.25%
2	10	5	200	10,909,364.04	—	—
3	10	5	400	24,069,215.94	—	—
4	10	10	40	1,419,500.29	1,882,100.15	24.58%
5	10	10	200	9,913,248.62	—	—
6	10	10	400	21,685,264.30	—	—
7	10	15	40	1,359,631.86	1,972,974.21	31.09%
8	10	15	200	6,926,169.97	10,160,597.84	31.83%
9	10	15	400	15,451,006.47	12,273,267.65	-25.89%
10	50	5	40	44,888,209.02	48,250,671.65	6.97%
11	50	5	200	212,476,288.30	—	—
12	50	5	400	721,839,974.21	—	—
13	50	10	40	54,438,344.68	64,586,406.51	15.71%
14	50	10	200	306,265,517.54	—	—
15	50	10	400	712,431,296.89	—	—
16	50	15	40	38,431,865.76	78,855,030.76	51.26%
17	50	15	200	207,687,551.32	—	—
18	50	15	400	474,425,692.45	—	—
19	100	5	40	177,441,851.37	193,065,795.84	8.09%
20	100	5	200	916,162,057.39	—	—
21	100	5	400	2,951,717,087.83	—	—
22	100	10	40	225,937,476.82	271,599,069.84	16.81%
23	100	10	200	1,286,858,630.98	—	—
24	100	10	400	2,986,362,562.50	—	—
25	100	15	40	162,633,630.85	351,072,450.79	53.68%
26	100	15	200	888,981,996.25	—	—
27	100	15	400	1,979,482,180.41	—	—
					Average	19.76%

Table B.18: Lagrangian Relaxation and Linear Programming Relaxation I (4)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.6 LR	δ=0.9 δ'=0.6 LPR	gap
1	10	5	40	2,323,210.65	2,447,358.32	5.07%
2	10	5	200	15,749,534.16	—	—
3	10	5	400	34,926,487.98	—	—
4	10	10	40	2,309,862.90	2,528,327.39	8.64%
5	10	10	200	13,405,208.08	—	—
6	10	10	400	28,542,107.34	—	—
7	10	15	40	1,704,499.49	2,614,470.79	34.81%
8	10	15	200	8,755,092.85	—	—
9	10	15	400	19,234,461.08	—	—
10	50	5	40	67,105,119.11	72,946,713.02	8.01%
11	50	5	200	408,013,927.59	—	—
12	50	5	400	1,041,281,377.88	—	—
13	50	10	40	77,725,986.47	90,296,777.83	13.92%
14	50	10	200	428,483,645.00	—	—
15	50	10	400	1,001,939,273.07	—	—
16	50	15	40	53,752,392.63	108,272,566.82	50.35%
17	50	15	200	285,718,499.15	—	—
18	50	15	400	642,241,402.51	—	—
19	100	5	40	266,837,378.43	293,669,592.32	9.14%
20	100	5	200	1,688,994,657.01	—	—
21	100	5	400	4,279,052,131.57	—	—
22	100	10	40	321,763,337.87	381,483,522.77	15.65%
23	100	10	200	1,820,297,803.19	—	—
24	100	10	400	4,228,687,835.26	—	—
25	100	15	40	233,314,035.11	483,449,460.00	51.74%
26	100	15	200	1,230,405,232.28	—	—
27	100	15	400	2,747,609,838.06	—	—
					Average	21.93%

Table B.19: Lagrangian Relaxation and Linear Programming Relaxation I (5)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.7 LR	δ=0.9 δ'=0.7 LPR	gap
1	10	5	40	1,879,607.22	2,001,457.31	6.09%
2	10	5	200	12,816,958.26	—	—
3	10	5	400	28,433,801.08	—	—
4	10	10	40	1,969,729.20	2,151,040.77	8.43%
5	10	10	200	11,291,996.84	—	—
6	10	10	400	24,540,691.02	—	—
7	10	15	40	1,496,671.34	2,243,228.30	33.28%
8	10	15	200	7,775,096.17	8,446,407.52	7.95%
9	10	15	400	17,141,300.34	—	—
10	50	5	40	52,163,591.31	58,013,287.41	10.08%
11	50	5	200	324,236,960.45	—	—
12	50	5	400	826,928,616.15	—	—
13	50	10	40	64,513,151.08	75,259,039.69	14.28%
14	50	10	200	358,807,278.00	—	—
15	50	10	400	833,090,730.96	—	—
16	50	15	40	45,189,666.97	91,214,412.63	50.46%
17	50	15	200	242,277,261.55	251,341,792.00	3.61%
18	50	15	400	552,512,055.70	—	—
19	100	5	40	206,649,753.49	232,836,226.90	11.25%
20	100	5	200	1,337,262,069.27	—	—
21	100	5	400	3,389,070,285.45	—	—
22	100	10	40	266,978,620.99	317,211,632.29	15.84%
23	100	10	200	1,511,582,421.83	—	—
24	100	10	400	3,507,709,996.04	—	—
25	100	15	40	195,907,176.89	406,686,628.77	51.83%
26	100	15	200	1,046,330,098.57	—	—
27	100	15	400	2,330,389,273.81	—	—
					Average	19.37%

Table B.20: Lagrangian Relaxation and Linear Programming Relaxation I (6)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.8 LR	δ=0.9 δ'=0.8 LPR	gap
1	10	5	40	1,598,200.62	1,709,110.28	6.49%
2	10	5	200	10,909,105.06	—	—
3	10	5	400	24,192,749.98	—	—
4	10	10	40	1,743,757.77	1,882,100.15	7.35%
5	10	10	200	9,846,166.30	—	—
6	10	10	400	21,549,473.01	—	—
7	10	15	40	1,349,189.09	1,972,974.21	31.62%
8	10	15	200	6,995,423.42	10,160,597.84	31.15%
9	10	15	400	15,567,178.80	—	—
10	50	5	40	42,910,376.23	48,248,579.64	11.06%
11	50	5	200	270,004,462.95	—	—
12	50	5	400	689,386,984.04	—	—
13	50	10	40	55,163,436.97	64,586,078.98	14.59%
14	50	10	200	307,111,542.01	—	—
15	50	10	400	715,253,281.15	—	—
16	50	15	40	38,912,655.24	78,854,991.69	50.65%
17	50	15	200	211,568,195.84	—	—
18	50	15	400	481,648,162.76	—	—
19	100	5	40	169,401,587.30	193,060,366.81	12.25%
20	100	5	200	1,111,164,384.04	—	—
21	100	5	400	2,814,327,505.75	—	—
22	100	10	40	227,084,060.59	271,598,280.27	16.39%
23	100	10	200	1,287,708,484.35	—	—
24	100	10	400	3,004,714,685.69	—	—
25	100	15	40	168,471,801.86	351,072,232.33	52.01%
26	100	15	200	907,407,024.61	—	—
27	100	15	400	2,023,016,444.49	—	—
					Average	23.36%

Table B.21: Lagrangian Relaxation and Linear Programming Relaxation I (7)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.9 LR	δ=0.9 δ'=0.9 LPR	gap
1	10	5	40	1,410,971.08	1,502,201.15	6.07%
2	10	5	200	9,565,312.05	—	—
3	10	5	400	20,989,581.39	—	—
4	10	10	40	1,561,187.48	1,680,286.16	7.09%
5	10	10	200	8,791,303.96	—	—
6	10	10	400	19,323,807.46	—	—
7	10	15	40	1,227,527.96	1,767,017.50	30.53%
8	10	15	200	6,431,931.85	9,456,647.47	31.99%
9	10	15	400	7,727,388.32	20,923,752.24	63.07%
10	50	5	40	36,570,671.32	41,364,775.31	11.59%
11	50	5	200	231,259,704.04	—	—
12	50	5	400	589,888,224.30	—	—
13	50	10	40	48,135,891.93	56,618,027.95	14.98%
14	50	10	200	269,842,114.96	—	—
15	50	10	400	630,404,578.48	—	—
16	50	15	40	34,326,185.32	69,487,950.96	50.60%
17	50	15	200	187,836,237.90	—	—
18	50	15	400	428,862,527.00	—	—
19	100	5	40	143,906,325.95	165,021,092.37	12.80%
20	100	5	200	948,303,595.64	—	—
21	100	5	400	2,406,956,222.32	—	—
22	100	10	40	197,429,765.25	237,548,011.38	16.89%
23	100	10	200	1,030,126,236.98	—	—
24	100	10	400	2,632,393,839.02	—	—
25	100	15	40	147,501,204.32	308,924,039.79	52.25%
26	100	15	200	805,478,263.48	—	—
27	100	15	400	1,796,698,231.23	—	—
					Average	27.08%

Table B.22: Lagrangian Relaxation and Linear Programming Relaxation I (8)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.6 LR	δ=0.8 δ'=0.6 LPR	gap
1	10	5	40	2,776,396.20	2,903,235.57	4.37%
2	10	5	200	13,087,774.98	16,971,500.00	22.88%
3	10	5	400	41,312,013.26	45,982,115.34	10.16%
4	10	10	40	2,639,277.72	2,878,167.77	8.30%
5	10	10	200	15,320,976.95	16,094,534.97	4.81%
6	10	10	400	32,469,458.97	—	—
7	10	15	40	1,883,242.80	2,950,738.56	36.18%
8	10	15	200	9,682,782.03	14,541,900.00	33.41%
9	10	15	400	20,847,716.31	—	—
10	50	5	40	79,434,163.20	88,237,996.95	9.98%
11	50	5	200	365,473,946.83	—	—
12	50	5	400	1,325,025,142.33	—	—
13	50	10	40	88,321,056.54	104,279,427.29	15.30%
14	50	10	200	471,156,745.34	—	—
15	50	10	400	1,153,073,595.99	—	—
16	50	15	40	60,090,980.97	123,771,576.99	51.45%
17	50	15	200	314,544,047.83	—	—
18	50	15	400	74,065,008.26	—	—
19	100	5	40	339,051,850.55	355,956,347.43	4.75%
20	100	5	200	1,509,304,162.96	—	—
21	100	5	400	5,450,113,895.27	—	—
22	100	10	40	375,240,696.54	441,247,492.38	14.96%
23	100	10	200	2,114,801,496.82	2,623,414,420.40	19.39%
24	100	10	400	4,851,144,734.32	—	—
25	100	15	40	260,800,493.12	553,197,822.97	52.86%
26	100	15	200	1,379,556,565.81	—	—
27	100	15	400	3,069,414,984.16	—	—
					Average	20.63%

Table B.23: Lagrangian Relaxation and Linear Programming Relaxation II (1)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.7 LR	δ=0.8 δ'=0.7 LPR	gap
1	10	5	40	2,241,299.98	2,329,168.92	3.77%
2	10	5	200	14,964,732.98	11,156,605.54	-34.13%
3	10	5	400	33,037,437.14	36,532,558.88	9.57%
4	10	10	40	2,237,181.98	2,431,866.54	8.01%
5	10	10	200	12,877,915.93	15,793,114.44	18.46%
6	10	10	400	27,711,262.91	—	—
7	10	15	40	1,661,307.49	2,520,415.95	34.09%
8	10	15	200	8,532,722.37	—	—
9	10	15	400	18,492,319.13	—	—
10	50	5	40	66,125,091.61	68,985,281.48	4.15%
11	50	5	200	293,590,242.00	—	—
12	50	5	400	1,035,377,033.13	—	—
13	50	10	40	74,317,622.36	86,447,310.81	14.03%
14	50	10	200	408,629,142.63	—	—
15	50	10	400	737,484,300.01	—	—
16	50	15	40	50,402,625.59	103,944,492.39	51.51%
17	50	15	200	268,705,396.37	—	—
18	50	15	400	610,276,667.28	—	—
19	100	5	40	265,069,236.82	277,527,971.25	4.49%
20	100	5	200	1,207,511,287.33	—	—
21	100	5	400	4,515,369,064.95	—	—
22	100	10	40	310,430,599.75	365,029,144.93	14.96%
23	100	10	200	1,732,014,211.83	—	—
24	100	10	400	4,058,070,901.17	—	—
25	100	15	40	216,530,843.20	463,972,293.71	53.33%
26	100	15	200	1,163,787,011.00	—	—
27	100	15	400	2,591,004,189.62	—	—

Average	15.19%

Table B.24: Lagrangian Relaxation and Linear Programming Relaxation II (2)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.8 LR	δ=0.8 δ'=0.8 LPR	gap
1	10	5	40	1,895,114.59	1,963,190.13	3.47%
2	10	5	200	12,565,967.13	—	—
3	10	5	400	27,740,275.99	—	—
4	10	10	40	1,956,598.56	2,116,801.80	7.57%
5	10	10	200	11,175,126.63	—	—
6	10	10	400	24,264,641.08	—	—
7	10	15	40	1,479,421.88	2,209,083.02	33.03%
8	10	15	200	7,602,213.37	5,764,809.19	-31.87%
9	10	15	400	16,664,793.76	—	—
10	50	5	40	53,041,421.43	56,733,195.94	6.51%
11	50	5	200	246,200,491.59	—	—
12	50	5	400	850,472,591.19	—	—
13	50	10	40	62,352,813.61	73,897,956.80	15.62%
14	50	10	200	349,745,739.15	—	—
15	50	10	400	819,472,471.50	—	—
16	50	15	40	43,698,013.65	89,649,946.00	51.26%
17	50	15	200	233,764,241.05	—	—
18	50	15	400	532,465,561.23	—	—
19	100	5	40	211,399,583.29	227,619,090.47	7.13%
20	100	5	200	1,010,023,661.45	—	—
21	100	5	400	3,483,840,003.50	—	—
22	100	10	40	261,154,104.89	311,394,015.64	16.13%
23	100	10	200	1,475,495,317.38	—	—
24	100	10	400	3,433,560,340.47	—	—
25	100	15	40	185,922,722.82	399,646,005.10	53.48%
26	100	15	200	1,007,337,039.98	—	—
27	100	15	400	2,243,495,135.00	—	—

Average	16.23%

Table B.25: Lagrangian Relaxation and Linear Programming Relaxation II (3)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.9 LR	δ=0.8 δ'=0.9 LPR	gap
1	10	5	40	1,653,667.42	1,709,160.56	3.25%
2	10	5	200	10,909,364.04	—	—
3	10	5	400	24,069,215.94	—	—
4	10	10	40	1,419,500.29	1,882,108.54	24.58%
5	10	10	200	9,913,248.62	—	—
6	10	10	400	21,685,264.30	—	—
7	10	15	40	1,359,631.86	1,972,983.86	31.09%
8	10	15	200	6,926,169.97	10,156,276.64	31.80%
9	10	15	400	15,451,006.47	12,273,267.65	-25.89%
10	50	5	40	44,888,209.02	48,250,656.76	6.97%
11	50	5	200	212,476,288.30	—	—
12	50	5	400	721,839,974.21	—	—
13	50	10	40	54,438,344.68	64,586,461.72	15.71%
14	50	10	200	306,265,517.54	—	—
15	50	10	400	712,431,296.89	—	—
16	50	15	40	38,431,865.76	78,855,305.75	51.26%
17	50	15	200	207,687,551.32	—	—
18	50	15	400	474,425,692.45	—	—
19	100	5	40	177,441,851.37	193,066,214.51	8.09%
20	100	5	200	916,162,057.39	—	—
21	100	5	400	2,951,717,087.83	—	—
22	100	10	40	225,937,476.82	271,598,797.92	16.81%
23	100	10	200	1,286,858,630.98	—	—
24	100	10	400	2,986,362,562.50	—	—
25	100	15	40	162,633,630.85	351,072,587.18	53.68%
26	100	15	200	888,981,996.25	—	—
27	100	15	400	1,979,482,180.41	—	—
					Average	19.76%

Table B.26: Lagrangian Relaxation and Linear Programming Relaxation II (4)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.6 LR	δ=0.9 δ'=0.6 LPR	gap
1	10	5	40	2,323,210.65	2,447,361.92	5.07%
2	10	5	200	15,749,534.16	—	—
3	10	5	400	34,926,487.98	—	—
4	10	10	40	2,309,862.90	2,528,328.07	8.64%
5	10	10	200	13,405,208.08	—	—
6	10	10	400	28,542,107.34	—	—
7	10	15	40	1,704,499.49	2,614,475.72	34.81%
8	10	15	200	8,755,092.85	—	—
9	10	15	400	19,234,461.08	—	—
10	50	5	40	67,105,119.11	72,947,049.29	8.01%
11	50	5	200	408,013,927.59	—	—
12	50	5	400	1,041,281,377.88	—	—
13	50	10	40	77,725,986.47	90,297,294.59	13.92%
14	50	10	200	428,483,645.00	—	—
15	50	10	400	1,001,939,273.07	—	—
16	50	15	40	53,752,392.63	108,272,736.00	50.35%
17	50	15	200	285,718,499.15	—	—
18	50	15	400	642,241,402.51	—	—
19	100	5	40	266,837,378.43	293,669,499.75	9.14%
20	100	5	200	1,688,994,657.01	—	—
21	100	5	400	4,279,052,131.57	—	—
22	100	10	40	321,763,337.87	381,485,088.68	15.66%
23	100	10	200	1,820,297,803.19	—	—
24	100	10	400	4,228,687,835.26	—	—
25	100	15	40	233,314,035.11	483,451,014.29	51.74%
26	100	15	200	1,230,405,232.28	—	—
27	100	15	400	2,747,609,838.06	—	—
					Average	21.93%

Table B.27: Lagrangian Relaxation and Linear Programming Relaxation II (5)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.7 LR	δ=0.9 δ'=0.7 LPR	gap
1	10	5	40	1,879,607.22	2,001,467.06	6.09%
2	10	5	200	12,816,958.26	—	—
3	10	5	400	28,433,801.08	—	—
4	10	10	40	1,969,729.20	2,151,039.95	8.43%
5	10	10	200	11,291,996.84	—	—
6	10	10	400	24,540,691.02	—	—
7	10	15	40	1,496,671.34	2,243,232.68	33.28%
8	10	15	200	7,775,096.17	8,595,420.63	9.54%
9	10	15	400	17,141,300.34	—	—
10	50	5	40	52,163,591.31	58,013,261.48	10.08%
11	50	5	200	324,236,960.45	—	—
12	50	5	400	826,928,616.15	—	—
13	50	10	40	64,513,151.08	75,259,050.89	14.28%
14	50	10	200	358,807,278.00	—	—
15	50	10	400	833,090,730.96	—	—
16	50	15	40	45,189,666.97	91,214,661.45	50.46%
17	50	15	200	242,277,261.55	253,679,724.00	4.49%
18	50	15	400	552,512,055.70	—	—
19	100	5	40	206,649,753.49	232,836,474.56	11.25%
20	100	5	200	1,337,262,069.27	—	—
21	100	5	400	3,389,070,285.45	—	—
22	100	10	40	266,978,620.99	317,211,749.82	15.84%
23	100	10	200	1,511,582,421.83	—	—
24	100	10	400	3,507,709,996.04	—	—
25	100	15	40	195,907,176.89	406,687,682.45	51.83%
26	100	15	200	1,046,330,098.57	—	—
27	100	15	400	2,330,389,273.81	—	—

Average	19.60%

Table B.28: Lagrangian Relaxation and Linear Programming Relaxation II (6)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.8 LR	δ=0.9 δ'=0.8 LPR	gap
1	10	5	40	1,598,200.62	1,709,106.28	6.49%
2	10	5	200	10,909,105.06	—	—
3	10	5	400	24,192,749.98	—	—
4	10	10	40	1,743,757.77	1,882,108.54	7.35%
5	10	10	200	9,846,166.30	—	—
6	10	10	400	21,549,473.01	—	—
7	10	15	40	1,349,189.09	1,972,983.86	31.62%
8	10	15	200	6,995,423.42	10,156,276.64	31.12%
9	10	15	400	15,567,178.80	—	—
10	50	5	40	42,910,376.23	48,248,566.79	11.06%
11	50	5	200	270,004,462.95	—	—
12	50	5	400	689,386,984.04	—	—
13	50	10	40	55,163,436.97	64,586,134.81	14.59%
14	50	10	200	307,111,542.01	—	—
15	50	10	400	715,253,281.15	—	—
16	50	15	40	38,912,655.24	78,855,266.66	50.65%
17	50	15	200	211,568,195.84	—	—
18	50	15	400	481,648,162.76	—	—
19	100	5	40	169,401,587.30	193,060,787.66	12.25%
20	100	5	200	1,111,164,384.04	—	—
21	100	5	400	2,814,327,505.75	—	—
22	100	10	40	227,084,060.59	271,598,007.34	16.39%
23	100	10	200	1,287,708,484.35	—	—
24	100	10	400	3,004,714,685.69	—	—
25	100	15	40	168,471,801.86	351,072,368.73	52.01%
26	100	15	200	907,407,024.61	—	—
27	100	15	400	2,023,016,444.49	—	—
					Average	23.35%

Table B.29: Lagrangian Relaxation and Linear Programming Relaxation II (7)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.9 LR	δ=0.9 δ'=0.9 LPR	gap
1	10	5	40	1,410,971.08	1,502,199.84	6.07%
2	10	5	200	9,565,312.05	——	——
3	10	5	400	20,989,581.39	——	——
4	10	10	40	1,561,187.48	1,680,293.33	7.09%
5	10	10	200	8,791,303.96	——	——
6	10	10	400	19,323,807.46	——	——
7	10	15	40	1,227,527.96	1,767,025.51	30.53%
8	10	15	200	6,431,931.85	9,457,201.00	31.99%
9	10	15	400	7,727,388.32	20,920,142.27	63.06%
10	50	5	40	36,570,671.32	41,364,767.90	11.59%
11	50	5	200	231,259,704.04	——	——
12	50	5	400	589,888,224.30	——	——
13	50	10	40	48,135,891.93	56,618,213.84	14.98%
14	50	10	200	269,842,114.96	——	——
15	50	10	400	630,404,578.48	——	——
16	50	15	40	34,326,185.32	69,487,786.19	50.60%
17	50	15	200	187,836,237.90	——	——
18	50	15	400	428,862,527.00	——	——
19	100	5	40	143,906,325.95	165,021,019.60	12.80%
20	100	5	200	948,303,595.64	——	——
21	100	5	400	2,406,956,222.32	——	——
22	100	10	40	197,429,765.25	237,548,073.09	16.89%
23	100	10	200	1,030,126,236.98	——	——
24	100	10	400	2,632,393,839.02	——	——
25	100	15	40	147,501,204.32	308,924,462.04	52.25%
26	100	15	200	805,478,263.48	——	——
27	100	15	400	1,796,698,231.23	——	——

Average 27.08%

Table B.30: Lagrangian Relaxation and Linear Programming Relaxation II (8)

B.5 Finding the Best Number for the Size of Shaking

|I|=10 |J|=5 |K|=40
δ=0.9 δ'=0.9

IVND	n
1,715,626.41	2
1,624,507.07	3
1,712,106.94	4
1,632,484.20	5
1,678,820.57	6
1,625,073.94	7
1,643,547.84	8
1,593,297.92	9
1,611,787.03	10

|I|=10 |J|=5 |K|=200
δ=0.9 δ'=0.9

IVND	n
9,643,565.01	2
9,681,268.77	3
9,676,163.97	4
9,657,690.70	5
9,609,977.88	6
9,576,542.44	7
9,655,092.79	8
9,657,570.74	9
9,627,117.03	10

|I|=10 |J|=5 |K|=400
δ=0.9 δ'=0.9

IVND	n
23,377,489.25	2
23,274,956.71	3
23,252,740.25	4
23,213,634.00	5
23,263,056.85	6
23,210,568.48	7
23,343,023.54	8
23,290,664.26	9
23,272,813.78	10

|I|=10 |J|=10 |K|=40
δ=0.9 δ'=0.9

IVND	n
1,877,903.20	2
1,899,250.48	3
1,852,425.35	4
1,867,456.61	5
1,811,484.32	6
1,826,525.81	7
1,889,870.80	8
1,847,497.43	9
1,914,612.62	10

|I|=10 |J|=10 |K|=200
δ=0.9 δ'=0.9

IVND	n
10,965,340.53	2
10,948,904.95	3
10,860,330.59	4
10,918,624.13	5
10,882,842.91	6
10,953,596.58	7
10,906,173.15	8
10,919,681.54	9
10,884,918.37	10

|I|=10 |J|=10 |K|=400
δ=0.9 δ'=0.9

IVND	n
22,145,463.10	2
22,132,080.98	3
22,282,047.34	4
22,116,861.02	5
22,250,553.58	6
22,098,498.10	7
22,183,425.30	8
22,184,575.51	9
22,100,109.35	10

|I|=10 |J|=15 |K|=40
δ=0.9 δ'=0.9

IVND	n
1,928,031.69	2
1,900,867.61	3
1,988,451.10	4
1,946,361.38	5
1,938,463.07	6
1,968,593.76	7
1,948,149.72	8
1,918,248.42	9
1,999,940.65	10

|I|=10 |J|=15 |K|=200
δ=0.9 δ'=0.9

IVND	n
9818611.34	2
9781642.68	3
9731904.34	4
9796287.40	5
9812807.29	6
9835455.60	7
9758187.14	8
9796971.57	9
9765407.42	10

|I|=10 |J|=15 |K|=400
δ=0.9 δ'=0.9

IVND	n
22,170,024.60	2
22,127,880.58	3
22,155,443.67	4
22,153,113.80	5
22,202,427.75	6
22,117,443.22	7
22,206,226.06	8
22,133,134.64	9
22,141,695.78	10

Table B.31: Calculating the size of shaking (1)

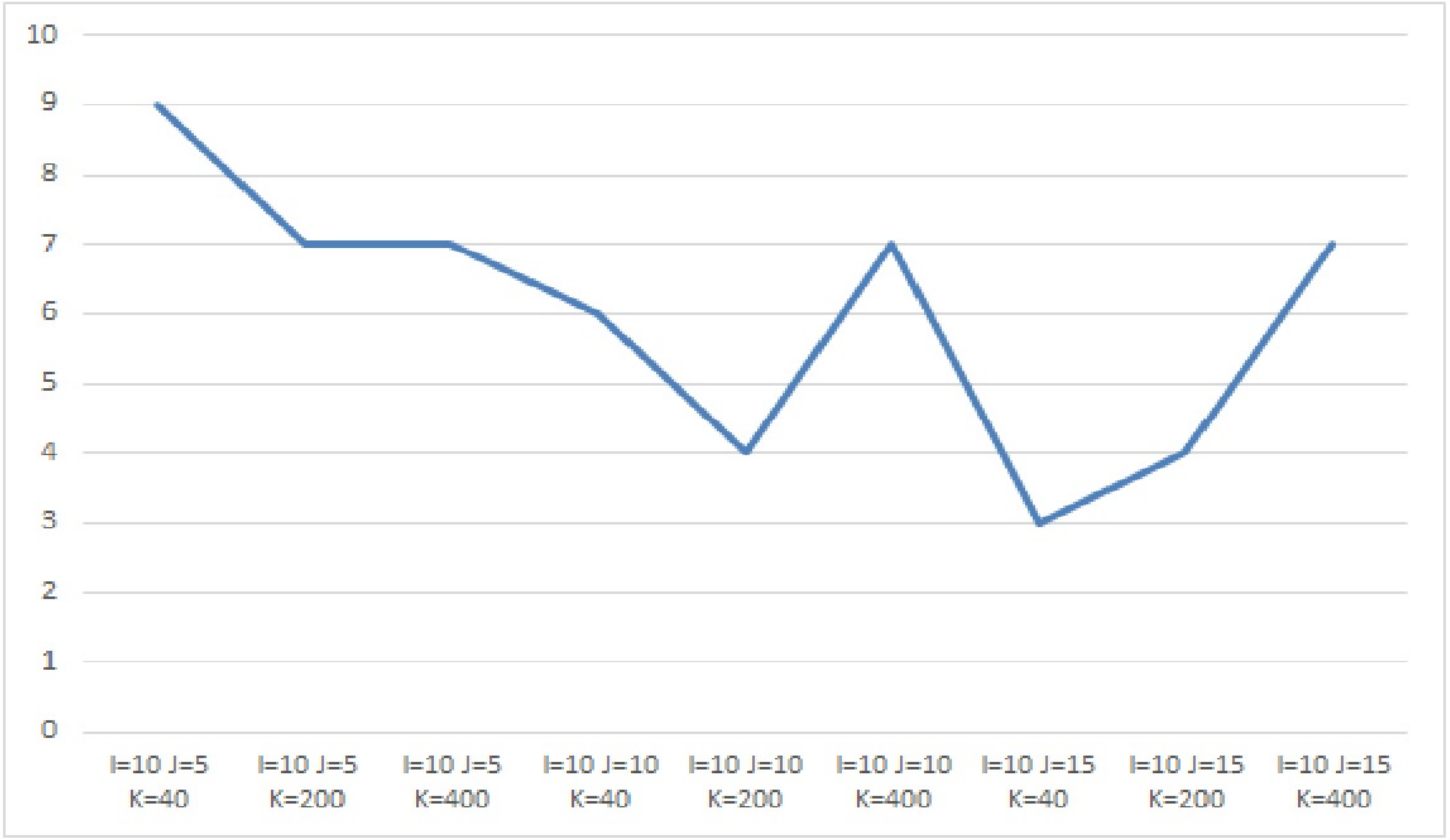

Figure B.5: Calculating the size of shaking (1)

|I|=50 |J|=5 |K|=40
δ=0.9 δ'=0.9

IVND	n
44,688,181.56	2
47,789,142.44	3
45,520,909.52	4
45,609,838.90	5
45,182,509.73	6
46,630,519.92	7
45,111,287.82	8
47,720,970.19	9
44,258,812.39	10

|I|=50 |J|=5 |K|=200
δ=0.9 δ'=0.9

IVND	n
395,259,855.66	2
395,291,927.60	3
396,748,257.59	4
394,929,454.50	5
396,817,837.10	6
394,614,732.54	7
394,582,160.80	8
395,566,398.07	9
394,456,296.14	10

|I|=50 |J|=5 |K|=400
δ=0.9 δ'=0.9

IVND	n
691,799,417.59	2
694,012,200.34	3
693,028,332.01	4
691,444,154.48	5
691,287,513.22	6
693,629,934.73	7
693,653,061.46	8
691,539,972.62	9
693,584,313.61	10

|I|=50 |J|=10 |K|=40
δ=0.9 δ'=0.9

IVND	n
61,047,620.29	2
61,689,128.56	3
60,404,163.23	4
61,904,949.92	5
60,908,057.24	6
61,021,354.58	7
62,090,309.51	8
60,335,118.22	9
59,814,210.87	10

|I|=50 |J|=10 |K|=200
δ=0.9 δ'=0.9

IVND	n
335,161,271.28	2
335,499,575.81	3
336,790,862.19	4
337,006,059.92	5
336,674,056.84	6
338,542,824.49	7
337,714,434.01	8
334,657,290.39	9
335,044,021.20	10

|I|=50 |J|=10 |K|=400
δ=0.9 δ'=0.9

IVND	n
786,957,146.43	2
787,547,749.03	3
789,759,973.93	4
789,327,338.87	5
790,184,077.44	6
789,462,154.58	7
789,044,973.37	8
789,774,686.01	9
787,449,107.25	10

|I|=50 |J|=15 |K|=40
δ=0.9 δ'=0.9

IVND	n
73,780,239.80	2
75,231,526.50	3
74,168,958.52	4
75,786,594.16	5
75,390,308.62	6
75,387,194.47	7
75,790,374.81	8
74,175,566.47	9
78,348,956.31	10

|I|=50 |J|=15 |K|=200
δ=0.9 δ'=0.9

IVND	n
394,780,698.24	2
395,377,247.61	3
397,178,461.12	4
393,270,905.49	5
395,373,381.16	6
393,956,813.74	7
397,637,134.35	8
395,346,385.17	9
396,531,230.22	10

|I|=50 |J|=15 |K|=400
δ=0.9 δ'=0.9

IVND	n
904,404,086.41	2
904,950,580.90	3
907,420,945.52	4
904,563,209.03	5
902,906,560.01	6
906,648,817.49	7
906,818,809.43	8
902,387,612.85	9
902,666,399.71	10

Table B.32: Calculating the size of shaking (2)

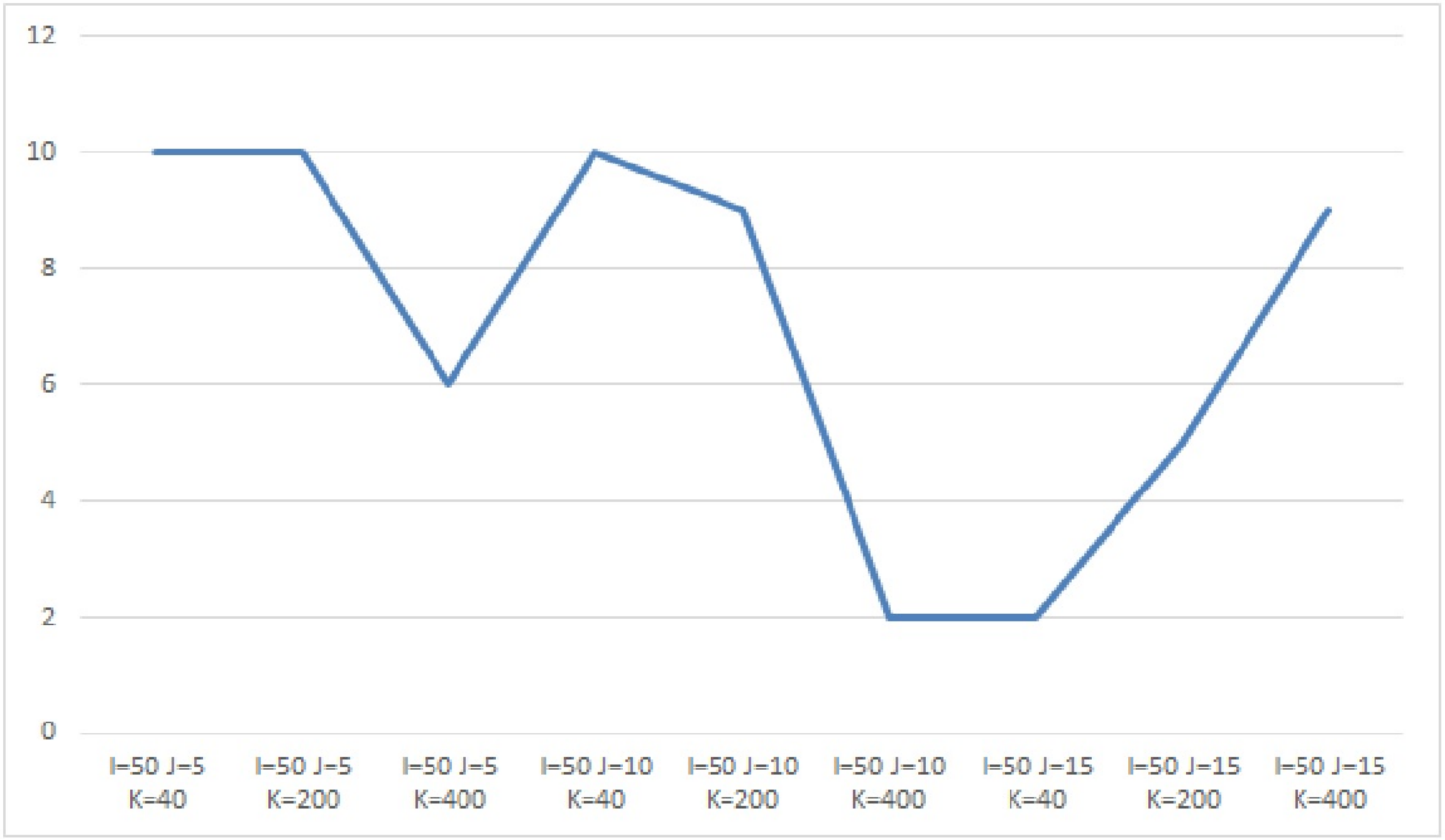

Figure B.6: Calculating the size of shaking (2)

|I|=100 |J|=5 |K|=40
δ=0.9 δ'=0.9

IVND	n
184,314,510.48	2
178,476,350.90	3
185,140,938.01	4
184,315,955.92	5
180,806,538.22	6
186,614,429.88	7
179,998,894.71	8
186,264,210.61	9
180,285,328.52	10

|I|=100 |J|=5 |K|=200
δ=0.9 δ'=0.9

IVND	n
1,123,018,534.64	2
1,122,749,359.37	3
1,127,946,571.88	4
1,117,404,665.56	5
1,116,110,574.62	6
1,125,996,088.30	7
1,118,540,053.55	8
1,113,688,001.29	9
1,119,575,727.14	10

|I|=100 |J|=5 |K|=400
δ=0.9 δ'=0.9

IVND	n
2,835,110,418.80	2
2,822,783,893.13	3
2,822,116,451.11	4
2,834,650,837.41	5
2,817,784,147.46	6
2,821,058,288.88	7
2,835,094,731.35	8
2,830,527,299.51	9
2,818,695,848.52	10

|I|=100 |J|=10 |K|=40
δ=0.9 δ'=0.9

IVND	n
258,946,900.36	2
259,436,278.50	3
277,958,094.88	4
277,565,600.74	5
259,558,796.21	6
255,697,618.81	7
259,900,737.11	8
261,702,670.67	9
257,886,203.02	10

|I|=100 |J|=10 |K|=200
δ=0.9 δ'=0.9

IVND	n
1,448,532,711.59	2
1,449,312,831.61	3
1,443,353,489.47	4
1,436,389,150.56	5
1,446,744,513.86	6
1,436,239,490.60	7
1,441,362,829.87	8
1,445,200,642.36	9
1,438,227,190.69	10

|I|=100 |J|=10 |K|=400
δ=0.9 δ'=0.9

IVND	n
184,367,038.98	2
181,960,252.68	3
182,340,661.43	4
178,246,050.14	5
180,970,311.16	6
176,208,252.94	7
188,335,940.85	8
173,949,642.98	9
183,807,543.83	10

|I|=100 |J|=15 |K|=40
δ=0.9 δ'=0.9

IVND	n
340,539,576.04	2
327,417,788.86	3
324,803,737.07	4
336,368,084.37	5
327,748,963.79	6
335,035,149.70	7
343,698,610.18	8
324,436,736.61	9
327,025,470.81	10

|I|=100 |J|=15 |K|=200
δ=0.9 δ'=0.9

IVND	n
1,792,072,957.98	2
1,795,809,950.35	3
1,789,290,981.49	4
1,789,461,097.34	5
1,777,289,838.96	6
1,783,203,093.86	7
1,788,694,696.41	8
1,781,298,465.67	9
1,795,228,033.06	10

|I|=100 |J|=15 |K|=400
δ=0.9 δ'=0.9

IVND	n
3,994,845,502.18	2
3,983,372,961.86	3
3,981,712,938.35	4
3,979,295,010.54	5
3,998,607,989.07	6
3,997,216,843.13	7
3,981,214,643.14	8
3,998,259,615.08	9
3,997,960,055.55	10

Table B.33: Calculating the size of shaking (3)

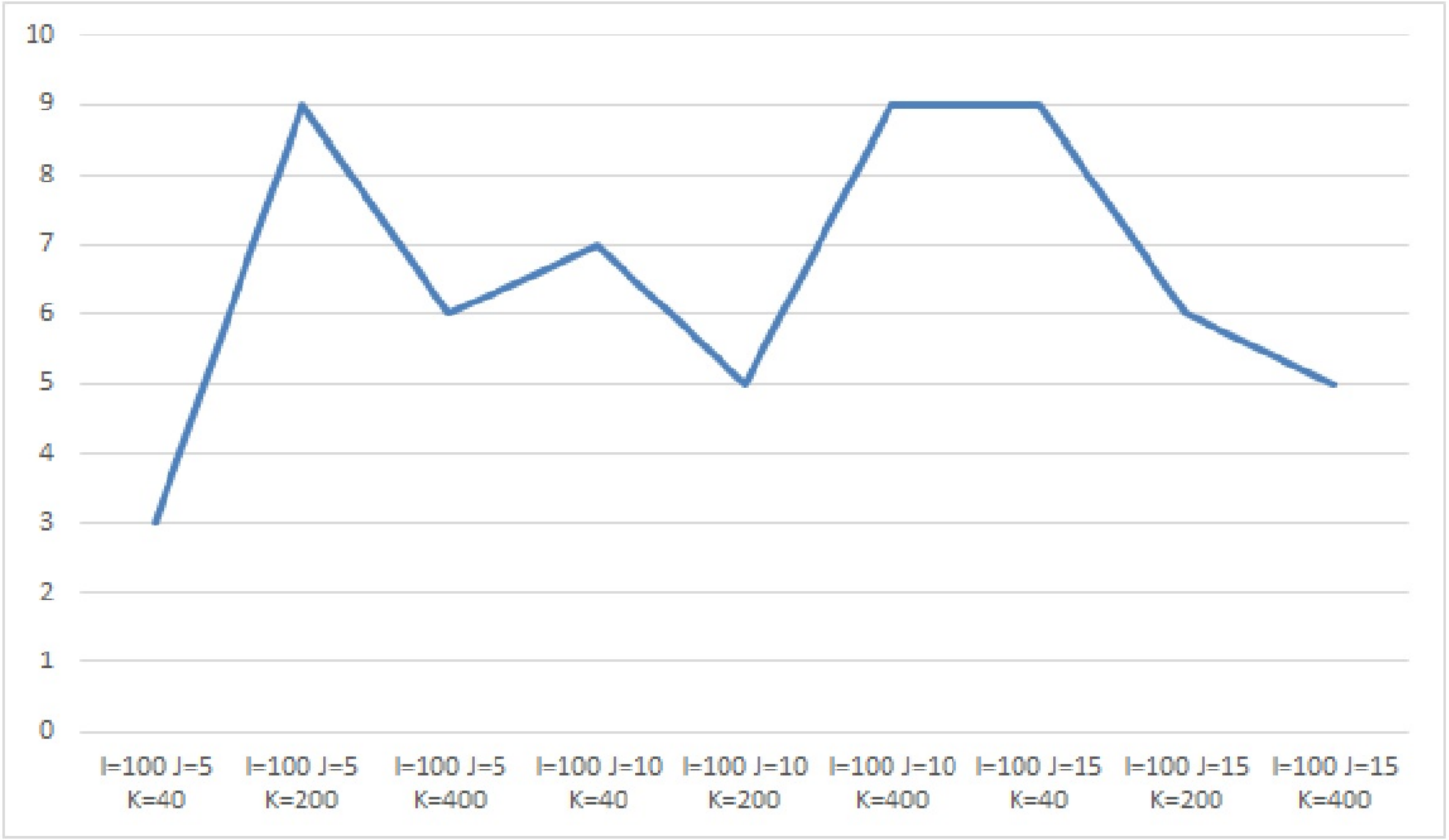

Figure B.7: Calculating the size of shaking (3)

B.6 Finding the Best Computational Time for Instances

|I|	|J|	|K|	5 min	10 min	15 min	20 min
100	5	200	1,134,021,984.46	1,134,021,984.46	1,133,926,967.90	1,133,926,967.90
50	15	400	902,635,654.56	902,635,654.56	902,635,654.56	902,635,654.56
50	5	200	271,321,341.26	271,321,341.26	271,224,355.54	271,224,355.54
10	10	40	1,802,876.47	1,802,876.47	1,802,876.47	1,802,876.47

Table B.34: Finding the Best Computational Time for Instances

B.7 Finding the Best Size for Tabu List

			δ=0.8 δ'=0.6				
|I|	|J|	|K|	size 10	size 15	size 20	size 25	size 30
100	5	200	1706833949	1706273464	1706131427	1708740207	1707258440
50	15	400	1253051906	1255626616	1252909254	1254308241	1255569550
50	5	200	413611502.6	412101329.1	411915587.1	412286959.8	413207988.9
10	10	40	2477414.38	2457741.42	2454756.83	2485208.36	2464128.77
			δ=0.8 δ'=0.7				
|I|	|J|	|K|	size 10	size 15	size 20	size 25	size 30
100	5	200	1363024788	1361526254	1361421246	1363372174	1365551748
50	15	400	1058867756	1057064252	1056945993	1057361660	1059981441
50	5	200	331175876.7	330041599.9	329903547.1	330177140.4	330508189
10	10	40	2103425.79	2205764.15	2084385.17	2086899.55	2091774.62
			δ=0.8 δ'=0.8				
|I|	|J|	|K|	size 10	size 15	size 20	size 25	size 30
100	5	200	1135159155	1134104937	1134021984	1135575489	1134674558
50	15	400	914841076.4	919475645.4	914740048	915606157.9	918249265.1
50	5	200	276760040	275886367.9	275744112.6	276237659.5	276087235.4
10	10	40	1844236.4	1843843.01	1827342.27	1841240.62	1829464.8
			δ=0.8 δ'=0.9				
|I|	|J|	|K|	size 10	size 15	size 20	size 25	size 30
100	5	200	972708954	972774865.3	972640532.5	982674514	973580014.6
50	15	400	806921122.3	808515982.5	806832905	807321674.9	807977924.4
50	5	200	238126539.4	237372075.3	237281361.5	237565138.9	238237189.6
10	10	40	1738580.08	1655861.27	1638128.03	1641916.75	1640002.11

Table B.35: Finding the Best Tabu List Size δ=0.8

			δ=0.9 δ'=0.6				
$\lvert I\rvert$	$\lvert J\rvert$	$\lvert K\rvert$	size 10	size 15	size 20	size 25	size 30
100	5	200	1706549665	1708464839	1706121405	1706131427	1710643299
50	15	400	1253125383	1254195901	1252909254	1255052915	1255374261
50	5	200	411955519.5	411915587.1	411915587.1	411983788.7	411940628.5
10	10	40	2457731.48	2457731.48	2454756.83	2457731.48	2457731.48
			δ=0.9 δ'=0.7				
$\lvert I\rvert$	$\lvert J\rvert$	$\lvert K\rvert$	size 10	size 15	size 20	size 25	size 30
100	5	200	1366717355	1361624927	1361399349	1363260654	1364083993
50	15	400	1057612933	1059503671	1056945993	1057439607	1364083993
50	5	200	331002619	331077555.9	329903547.1	330117953.3	330301586.1
10	10	40	2090504.28	2090504.28	2084385.17	2092219.46	2103889.32
			δ=0.9 δ'=0.8				
$\lvert I\rvert$	$\lvert J\rvert$	$\lvert K\rvert$	size 10	size 15	size 20	size 25	size 30
100	5	200	1138178163	1134760329	1133999446	1135394078	1135696566
50	15	400	915309573.1	916842373.9	914740048	918820626.9	917902896.3
50	5	200	276609571.3	275996596.5	275744112.6	276025214.1	276511224.4
10	10	40	1875378.18	1837260.51	1827342.27	1863206.27	1844831.4
			δ=0.9 δ'=0.9				
$\lvert I\rvert$	$\lvert J\rvert$	$\lvert K\rvert$	size 10	size 15	size 20	size 25	size 30
100	5	200	982000691.8	976931176.5	972619503.5	975178880.3	979592931.6
50	15	400	807330100.6	810560188.5	806832905	807272460.2	807585103.8
50	5	200	237993497.1	237281361.5	237281361.5	237383865.8	237422214.1
10	10	40	1694012.22	1693324.98	1638128.03	1726710.52	1657061.05

Table B.36: Finding the Best Tabu List Size δ=0.9

B.8 Computational Results for Iterated Variable Neighbourhood Descent and Tabu Search

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.6 IVND	δ=0.8 δ'=0.6 TS	gap
1	10	5	40	3,488,700.29	3,487,950.79	0.02%
2	10	5	200	18,812,281.82	18,803,555.02	0.05%
3	10	5	400	46,363,728.72	46,363,728.72	0.00%
4	10	10	40	3,200,035.57	3,200,035.57	0.00%
5	10	10	200	21,562,280.13	21,562,280.13	0.00%
6	10	10	400	38,284,345.04	37,992,533.93	0.76%
7	10	15	40	3,311,951.19	3,311,951.19	0.00%
8	10	15	200	16,341,603.87	16,174,974.71	1.02%
9	10	15	400	36,373,750.25	36,573,750.25	-0.55%
10	50	5	40	108,241,184.21	91,892,727.35	15.10%
11	50	5	200	591,621,401.61	411,915,587.11	30.38%
12	50	5	400	1,513,558,350.01	1,496,874,768.20	1.10%
13	50	10	40	122,907,731.27	115,880,267.28	5.72%
14	50	10	200	630,787,238.53	591,774,458.29	6.18%
15	50	10	400	1,460,589,116.77	1,460,589,116.77	0.00%
16	50	15	40	135,886,688.62	135,886,688.62	0.00%
17	50	15	200	706,232,388.69	706,232,388.69	0.00%
18	50	15	400	1,628,501,886.23	125,290,925.05	92.31%
19	100	5	40	449,043,662.47	426,609,675.07	5.00%
20	100	5	200	2,501,822,226.12	1,706,131,426.65	31.80%
21	100	5	400	6,194,760,935.03	6,172,465,833.53	0.36%
22	100	10	40	546,304,428.59	495,008,709.66	9.39%
23	100	10	200	2,718,049,303.38	2,692,716,281.33	0.93%
24	100	10	400	6,239,802,196.65	6,235,335,323.66	0.07%
25	100	15	40	661,526,208.58	605,309,158.41	8.50%
26	100	15	200	3,222,494,513.45	3,204,484,739.48	0.56%
27	100	15	400	7,164,172,713.18	7,153,766,416.78	0.15%

Average	7.74%

Table B.37: Iterated Variable Neighbourhood Descent and Tabu Search (1)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.7 IVND	δ=0.8 δ'=0.7 TS	gap
1	10	5	40	2,848,004.99	2,642,938.13	7.20%
2	10	5	200	14,976,705.91	14,965,332.07	0.08%
3	10	5	400	38,768,246.42	36,808,753.24	5.05%
4	10	10	40	2,653,641.80	2,653,641.80	0.00%
5	10	10	200	17,137,631.65	17,137,631.65	0.00%
6	10	10	400	32,013,192.56	32,013,192.56	0.00%
7	10	15	40	2,775,732.10	2,775,732.10	0.00%
8	10	15	200	13,814,155.09	13,814,155.09	0.00%
9	10	15	400	31,282,793.29	31,282,793.29	0.00%
10	50	5	40	81,552,094.63	78,717,439.44	3.48%
11	50	5	200	467,609,810.54	329,903,547.12	29.45%
12	50	5	400	1,169,139,075.50	1,162,617,226.61	0.56%
13	50	10	40	95,181,967.60	93,942,504.68	1.30%
14	50	10	200	514,550,913.12	514,550,913.12	0.00%
15	50	10	400	930,458,166.41	928,389,313.49	0.22%
16	50	15	40	112,166,187.45	112,166,187.45	0.00%
17	50	15	200	591,457,057.80	591,457,075.80	0.00%
18	50	15	400	1,361,301,659.98	1,056,945,992.84	22.36%
19	100	5	40	358,846,720.78	312,091,673.37	13.03%
20	100	5	200	1,900,295,161.18	1,361,421,245.77	28.36%
21	100	5	400	4,783,194,674.55	4,985,458,084.61	-4.23%
22	100	10	40	433,569,791.64	398,634,567.26	8.06%
23	100	10	200	2,218,508,059.33	2,218,459,716.09	0.00%
24	100	10	400	5,147,032,805.52	5,146,698,941.43	0.01%
25	100	15	40	500,767,740.62	498,599,160.19	0.43%
26	100	15	200	2,717,282,151.41	2,679,903,711.04	1.38%
27	100	15	400	6,017,927,423.33	5,990,370,288.23	0.46%

Average	4.34%

Table B.38: Iterated Variable Neighbourhood Descent and Tabu Search (2)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.8 IVND	δ=0.8 δ'=0.8 TS	gap
1	10	5	40	2,171,183.88	2,170,681.31	0.02%
2	10	5	200	12,566,452.96	12,566,452.96	0.00%
3	10	5	400	30,788,051.85	30,774,811.94	0.04%
4	10	10	40	2,355,003.31	2,288,505.99	2.82%
5	10	10	200	14,483,509.45	14,354,260.98	0.89%
6	10	10	400	28,010,316.76	27,823,743.27	0.67%
7	10	15	40	2,668,109.41	2,405,788.55	9.83%
8	10	15	200	12,122,330.63	12,122,330.63	0.00%
9	10	15	400	27,645,897.75	27,480,370.12	0.60%
10	50	5	40	63,667,128.14	62,003,433.47	2.61%
11	50	5	200	374,545,176.61	275,744,112.60	26.38%
12	50	5	400	961,018,514.45	952,065,665.03	0.93%
13	50	10	40	85,361,131.25	79,209,642.42	7.21%
14	50	10	200	438,713,649.94	438,713,649.94	0.00%
15	50	10	400	1,038,924,032.23	1,030,150,676.72	0.84%
16	50	15	40	95,630,062.11	95,630,062.11	0.00%
17	50	15	200	510,970,312.23	509,180,111.29	0.35%
18	50	15	400	1,176,362,598.43	914,740,048.02	22.24%
19	100	5	40	255,085,220.42	247,227,343.27	3.08%
20	100	5	200	1,579,281,422.05	1,134,021,984.46	28.19%
21	100	5	400	3,908,570,277.61	3,908,570,277.61	0.00%
22	100	10	40	344,696,269.01	334,529,186.77	2.95%
23	100	10	200	1,891,520,328.22	1,887,417,597.66	0.22%
24	100	10	400	4,383,998,476.75	4,383,727,904.39	0.01%
25	100	15	40	465,790,862.79	424,351,624.71	8.90%
26	100	15	200	2,339,456,987.00	2,303,854,003.24	1.52%
27	100	15	400	5,161,337,562.70	5,154,155,172.58	0.14%
					Average	4.46%

Table B.39: Iterated Variable Neighbourhood Descent and Tabu Search (3)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.9 IVND	δ=0.8 δ'=0.9 TS	gap
1	10	5	40	1,865,122.73	1,864,681.62	0.02%
2	10	5	200	10,909,736.54	10,909,736.54	0.00%
3	10	5	400	26,615,806.73	26,615,806.73	0.00%
4	10	10	40	2,161,566.00	2,025,836.16	6.28%
5	10	10	200	12,536,711.89	12,440,101.11	0.77%
6	10	10	400	24,902,737.96	24,724,464.60	0.72%
7	10	15	40	2,134,600.49	2,134,600.49	0.00%
8	10	15	200	10,986,793.54	10,850,153.72	1.24%
9	10	15	400	24,615,461.40	24,615,461.40	0.00%
10	50	5	40	52,749,373.54	52,163,338.32	1.11%
11	50	5	200	317,977,666.87	237,281,361.53	25.38%
12	50	5	400	814,361,990.20	807,143,301.90	0.89%
13	50	10	40	71,292,283.13	68,596,635.03	3.78%
14	50	10	200	387,247,321.51	382,707,853.20	1.17%
15	50	10	400	898,876,940.50	898,876,940.50	0.00%
16	50	15	40	86,993,149.54	83,427,178.19	4.10%
17	50	15	200	453,349,616.04	447,298,620.62	1.33%
18	50	15	400	1,033,355,516.07	806,832,905.01	21.92%
19	100	5	40	209,298,068.14	205,514,341.97	1.81%
20	100	5	200	1,311,908,320.82	972,640,532.48	25.86%
21	100	5	400	3,306,745,183.01	3,306,745,183.01	0.00%
22	100	10	40	314,283,951.71	288,564,398.57	8.18%
23	100	10	200	1,654,098,280.39	1,643,135,394.90	0.66%
24	100	10	400	3,841,066,926.28	3,819,193,540.98	0.57%
25	100	15	40	373,061,921.79	369,606,614.62	0.93%
26	100	15	200	2,031,832,986.28	2,021,023,068.94	0.53%
27	100	15	400	4,530,273,946.38	4,524,056,016.77	0.14%

Average	3.98%

Table B.40: Iterated Variable Neighbourhood Descent and Tabu Search (4)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.6 IVND	δ=0.9 δ'=0.6 TS	gap
1	10	5	40	2,805,735.38	2,803,391.18	0.08%
2	10	5	200	15,761,292.47	15,750,456.68	0.07%
3	10	5	400	38,778,802.31	38,768,246.42	0.03%
4	10	10	40	3,006,313.90	2,768,607.10	7.91%
5	10	10	200	18,043,239.79	18,043,239.79	0.00%
6	10	10	400	33,301,532.59	33,301,532.59	0.00%
7	10	15	40	3,190,525.27	2,890,385.33	9.41%
8	10	15	200	14,328,114.19	14,328,114.19	0.00%
9	10	15	400	32,436,176.37	32,436,176.37	0.00%
10	50	5	40	88,147,345.74	84,571,608.69	4.06%
11	50	5	200	492,226,984.81	411,915,587.11	16.32%
12	50	5	400	1,239,740,910.21	1,231,076,102.52	0.70%
13	50	10	40	102,201,656.07	98,572,309.54	3.55%
14	50	10	200	537,903,998.54	537,903,998.54	0.00%
15	50	10	400	1,272,837,004.85	1,262,077,359.69	0.85%
16	50	15	40	117,263,637.12	117,263,637.12	0.00%
17	50	15	200	616,447,813.49	616,447,813.49	0.00%
18	50	15	400	1,410,607,923.05	1,252,909,254.05	11.18%
19	100	5	40	355,568,998.37	334,554,085.53	5.91%
20	100	5	200	2,028,417,552.31	1,706,121,405.25	15.89%
21	100	5	400	5,080,757,003.24	5,067,645,767.65	0.26%
22	100	10	40	438,825,950.61	416,518,158.16	5.08%
23	100	10	200	2,348,583,464.96	2,314,882,490.01	1.43%
24	100	10	400	5,381,648,137.39	5,380,972,886.84	0.01%
25	100	15	40	561,827,133.59	521,990,191.75	7.09%
26	100	15	200	2,845,821,980.47	2,796,235,536.43	1.74%
27	100	15	400	6,244,174,979.88	6,246,364,753.44	-0.04%

Average	3.39%

Table B.41: Iterated Variable Neighbourhood Descent and Tabu Search (5)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.7 IVND	δ=0.9 δ'=0.7 TS	gap
1	10	5	40	2,218,809.56	2,216,719.25	0.09%
2	10	5	200	12,997,721.74	12,817,697.99	1.39%
3	10	5	400	31,404,750.20	31,404,750.20	0.00%
4	10	10	40	2,327,468.91	2,327,468.91	0.00%
5	10	10	200	14,790,885.44	14,644,484.18	0.99%
6	10	10	400	28,323,244.26	28,277,569.73	0.16%
7	10	15	40	2,445,665.60	2,445,665.60	0.00%
8	10	15	200	12,307,132.49	12,307,132.49	0.00%
9	10	15	400	28,098,824.98	27,896,140.75	0.72%
10	50	5	40	69,020,156.53	64,298,848.55	6.84%
11	50	5	200	386,184,437.94	329,903,547.12	14.57%
12	50	5	400	979,892,719.07	974,030,012.09	0.60%
13	50	10	40	80,783,218.20	80,783,218.20	0.00%
14	50	10	200	449,063,911.29	446,921,382.09	0.48%
15	50	10	400	1,049,370,020.21	1,049,370,020.21	0.00%
16	50	15	40	104,117,814.42	97,418,810.58	6.43%
17	50	15	200	521,857,120.34	518,168,389.35	0.71%
18	50	15	400	1,186,758,158.90	1,056,945,992.84	10.94%
19	100	5	40	281,664,914.85	253,737,256.78	9.92%
20	100	5	200	1,585,702,719.57	1,361,399,348.85	14.15%
21	100	5	400	3,999,794,411.56	3,999,794,411.56	0.00%
22	100	10	40	354,715,723.75	339,600,987.79	4.26%
23	100	10	200	1,947,380,372.04	1,918,719,960.21	1.47%
24	100	10	400	4,466,626,071.42	4,466,131,499.31	0.01%
25	100	15	40	440,527,676.83	432,843,435.19	1.74%
26	100	15	200	2,347,572,799.36	2,346,657,273.65	0.04%
27	100	15	400	5,245,727,896.80	5,245,282,964.66	0.01%

Average	2.80%

Table B.42: Iterated Variable Neighbourhood Descent and Tabu Search (6)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.8 IVND	δ=0.9 δ'=0.8 TS	gap
1	10	5	40	1,865,122.73	1,863,248.65	0.10%
2	10	5	200	11,080,427.19	10,909,736.54	1.54%
3	10	5	400	26,615,806.73	26,615,806.73	0.00%
4	10	10	40	2,153,078.95	2,025,836.16	5.91%
5	10	10	200	12,579,687.93	12,440,101.11	1.11%
6	10	10	400	24,934,739.91	24,724,464.60	0.84%
7	10	15	40	2,134,600.49	2,134,600.49	0.00%
8	10	15	200	10,850,153.72	10,850,153.72	0.00%
9	10	15	400	24,677,161.73	24,615,461.40	0.25%
10	50	5	40	52,749,373.54	52,154,363.54	1.13%
11	50	5	200	317,792,630.06	275,744,112.60	13.23%
12	50	5	400	812,039,329.58	807,143,301.90	0.60%
13	50	10	40	69,077,315.05	68,596,635.03	0.70%
14	50	10	200	385,820,058.24	382,707,853.20	0.81%
15	50	10	400	902,448,553.04	898,876,940.50	0.40%
16	50	15	40	87,358,187.40	83,427,178.19	4.50%
17	50	15	200	448,078,598.90	447,298,620.62	0.17%
18	50	15	400	1,029,992,563.19	914,740,048.02	11.19%
19	100	5	40	224,244,948.66	205,514,341.97	8.35%
20	100	5	200	1,314,482,508.29	1,133,999,446.40	13.73%
21	100	5	400	3,306,745,183.01	3,306,745,183.01	0.00%
22	100	10	40	313,631,666.91	287,158,702.46	8.44%
23	100	10	200	1,643,166,701.97	1,639,351,726.43	0.23%
24	100	10	400	3,819,420,526.22	3,818,983,851.62	0.01%
25	100	15	40	393,976,485.47	370,038,929.95	6.08%
26	100	15	200	2,034,342,312.29	2,022,476,796.00	0.58%
27	100	15	400	4,563,625,562.52	4,523,784,327.37	0.87%

Average	2.99%

Table B.43: Iterated Variable Neighbourhood Descent and Tabu Search (7)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.9 IVND	δ=0.9 δ'=0.9 TS	gap
1	10	5	40	1,625,073.94	1,625,159.62	-0.01%
2	10	5	200	9,576,542.44	9,566,320.01	0.11%
3	10	5	400	23,210,568.48	23,249,714.58	-0.17%
4	10	10	40	1,826,525.81	1,805,876.47	1.13%
5	10	10	200	10,953,596.58	10,893,720.48	0.55%
6	10	10	400	22,112,392.38	22,078,367.96	0.15%
7	10	15	40	1,968,593.76	1,904,484.69	3.26%
8	10	15	200	9,835,455.60	9,749,816.59	0.87%
9	10	15	400	22,117,443.22	22,133,761.02	-0.07%
10	50	5	40	46,630,519.92	44,064,095.37	5.50%
11	50	5	200	274,052,427.01	237,281,361.53	13.42%
12	50	5	400	693,629,934.73	689,995,487.73	0.52%
13	50	10	40	61,021,354.58	59,707,015.52	2.15%
14	50	10	200	338,542,824.49	334,953,744.58	1.06%
15	50	10	400	789,462,154.58	786,771,044.12	0.34%
16	50	15	40	75,387,194.47	73,022,207.95	3.14%
17	50	15	200	393,956,813.74	393,767,601.68	0.05%
18	50	15	400	906,648,817.49	806,832,905.01	11.01%
19	100	5	40	186,614,429.88	173,164,494.87	7.21%
20	100	5	200	1,125,996,088.30	972,619,503.46	13.62%
21	100	5	400	2,821,058,288.88	2,820,350,975.80	0.03%
22	100	10	40	255,697,618.81	247,253,099.96	3.30%
23	100	10	200	1,295,700,510.12	1,279,302,398.36	1.27%
24	100	10	400	3,762,082,522.94	3,336,964,592.96	11.30%
25	100	15	40	335,035,149.70	323,340,281.23	3.49%
26	100	15	200	1,783,203,093.96	1,777,617,262.08	0.31%
27	100	15	400	3,997,216,843.13	3,977,961,663.89	0.48%

Average	3.11%

Table B.44: Iterated Variable Neighbourhood Descent and Tabu Search (8)

B.9 Computational Results for Best Upper Bound and Best Lower Bound

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.6 TS	Best Lower Bound	gap
1	10	5	40	3,487,950.79	2,903,235.57	16.76%
2	10	5	200	18,803,555.02	18,127,243.24	3.60%
3	10	5	400	46,363,728.72	45,982,115.34	0.82%
4	10	10	40	3,200,035.57	2,878,167.77	10.06%
5	10	10	200	21,562,280.13	16,094,534.97	25.36%
6	10	10	400	37,992,533.93	32,469,458.97	14.54%
7	10	15	40	3,311,951.19	2,950,738.56	10.91%
8	10	15	200	16,174,974.71	14,541,900.00	10.10%
9	10	15	400	36,573,750.25	20,847,716.31	43.00%
10	50	5	40	91,892,727.35	88,237,996.95	3.98%
11	50	5	200	411,915,587.11	365,473,946.83	11.27%
12	50	5	400	1,496,874,768.20	1,325,025,142.33	11.48%
13	50	10	40	115,880,267.28	104,279,448.74	10.01%
14	50	10	200	591,774,458.29	471,156,745.34	20.38%
15	50	10	400	1,460,589,116.77	1,153,073,595.99	21.05%
16	50	15	40	135,886,688.62	123,771,576.99	8.92%
17	50	15	200	706,232,388.69	314,544,047.83	55.46%
18	50	15	400	125,290,925.05	74,065,008.26	40.89%
19	100	5	40	426,609,675.07	355,956,347.43	16.56%
20	100	5	200	1,706,131,426.65	1,509,304,162.96	11.54%
21	100	5	400	6,172,465,833.53	5,943,877,314.00	3.70%
22	100	10	40	495,008,709.66	441,247,492.38	10.86%
23	100	10	200	2,692,716,281.33	2,623,414,420.40	2.57%
24	100	10	400	6,235,335,323.66	5,124,851,741.20	17.81%
25	100	15	40	605,309,158.41	553,228,938.29	8.60%
26	100	15	200	3,204,484,739.48	1,379,556,565.81	56.95%
27	100	15	400	7,153,766,416.78	3,069,414,984.16	57.09%
					Average	18.68%

Table B.45: Best Upper Bound and Best Lower Bound (1)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.7 TS	Best Lower Bound	gap
1	10	5	40	2,642,938.13	2,329,168.92	11.87%
2	10	5	200	14,965,332.07	14,964,732.98	0.00%
3	10	5	400	36,808,753.24	36,532,558.88	0.75%
4	10	10	40	2,653,641.80	2,431,876.09	8.36%
5	10	10	200	17,137,631.65	15,793,114.44	7.85%
6	10	10	400	32,013,192.56	27,711,262.91	13.44%
7	10	15	40	2,775,732.10	2,520,415.95	9.20%
8	10	15	200	13,814,155.09	8,532,722.37	38.23%
9	10	15	400	31,282,793.29	18,492,319.13	40.89%
10	50	5	40	78,717,439.44	68,985,281.48	12.36%
11	50	5	200	329,903,547.12	293,590,242.00	11.01%
12	50	5	400	1,162,617,226.61	1,035,377,033.13	10.94%
13	50	10	40	93,942,504.68	86,447,310.81	7.98%
14	50	10	200	514,550,913.12	408,629,142.63	20.59%
15	50	10	400	928,389,313.49	737,484,300.01	20.56%
16	50	15	40	112,166,187.45	103,944,492.39	7.33%
17	50	15	200	591,457,075.80	268,705,396.37	54.57%
18	50	15	400	1,056,945,992.84	610,276,667.28	42.26%
19	100	5	40	312,091,673.37	277528281.2	11.07%
20	100	5	200	1,361,421,245.77	1,207,511,287.33	11.31%
21	100	5	400	4,985,458,084.61	4,515,369,064.95	9.43%
22	100	10	40	398,634,567.26	365,029,144.93	8.43%
23	100	10	200	2,218,459,716.09	1,732,014,211.83	21.93%
24	100	10	400	5,146,698,941.43	4,058,070,901.17	21.15%
25	100	15	40	498,599,160.19	463,972,293.71	6.94%
26	100	15	200	2,679,903,711.04	1,163,787,011.00	56.57%
27	100	15	400	5,990,370,288.23	2,591,004,189.62	56.75%
					Average	19.32%

Table B.46: Best Upper Bound and Best Lower Bound (2)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.8 TS	Best Lower Bound	gap
1	10	5	40	2,170,681.31	1,963,197.05	9.56%
2	10	5	200	12,566,452.96	12,565,967.13	0.00%
3	10	5	400	30,774,811.94	27,740,275.99	9.86%
4	10	10	40	2,288,505.99	2,116,803.63	7.50%
5	10	10	200	14,354,260.98	11,175,126.63	22.15%
6	10	10	400	27,823,743.27	24,264,641.08	12.79%
7	10	15	40	2,405,788.55	2,209,083.02	8.18%
8	10	15	200	12,122,330.63	7,602,213.37	37.29%
9	10	15	400	27,480,370.12	16,664,793.76	39.36%
10	50	5	40	62,003,433.47	56,733,195.94	8.50%
11	50	5	200	275,744,112.60	246,200,491.59	10.71%
12	50	5	400	952,065,665.03	850,472,591.19	10.67%
13	50	10	40	79,209,642.42	73,897,956.80	6.71%
14	50	10	200	438,713,649.94	349,745,739.15	20.28%
15	50	10	400	1,030,150,676.72	819,472,471.50	20.45%
16	50	15	40	95,630,062.11	89,649,946.00	6.25%
17	50	15	200	509,180,111.29	233,764,241.05	54.09%
18	50	15	400	914,740,048.02	532,465,561.23	41.79%
19	100	5	40	247,227,343.27	227,619,090.47	7.93%
20	100	5	200	1,134,021,984.46	1,010,023,661.45	10.93%
21	100	5	400	3,908,570,277.61	3,483,840,003.50	10.87%
22	100	10	40	334,529,186.77	311,394,015.64	6.92%
23	100	10	200	1,887,417,597.66	1,475,495,317.38	21.82%
24	100	10	400	4,383,727,904.39	3,433,560,340.47	21.67%
25	100	15	40	424,351,624.71	399,646,005.10	5.82%
26	100	15	200	2,303,854,003.24	1,007,337,039.98	56.28%
27	100	15	400	5,154,155,172.58	2,243,495,135.00	56.47%
					Average	19.44%

Table B.47: Best Upper Bound and Best Lower Bound (3)

Ins.	|I|	|J|	|K|	δ=0.8 δ'=0.9 TS	Best Lower Bound	gap
1	10	5	40	1,864,681.62	1,709,163.85	8.34%
2	10	5	200	10,909,736.54	10,909,364.04	0.00%
3	10	5	400	26,615,806.73	24,069,215.94	9.57%
4	10	10	40	2,025,836.16	1,882,108.54	7.09%
5	10	10	200	12,440,101.11	9,913,248.62	20.31%
6	10	10	400	24,724,464.60	21,685,264.30	12.29%
7	10	15	40	2,134,600.49	1,972,983.86	7.57%
8	10	15	200	10,850,153.72	10,160,597.84	6.36%
9	10	15	400	24,615,461.40	15,451,006.47	37.23%
10	50	5	40	52,163,338.32	48,250,671.65	7.50%
11	50	5	200	237,281,361.53	212,476,288.30	10.45%
12	50	5	400	807,143,301.90	721,839,974.21	10.57%
13	50	10	40	68,596,635.03	64,586,461.72	5.85%
14	50	10	200	382,707,853.20	306,265,517.54	19.97%
15	50	10	400	898,876,940.50	712,431,296.89	20.74%
16	50	15	40	83,427,178.19	78,855,305.75	5.48%
17	50	15	200	447,298,620.62	207,687,551.32	53.57%
18	50	15	400	806,832,905.01	474,425,692.45	41.20%
19	100	5	40	205,514,341.97	193,066,214.51	6.06%
20	100	5	200	972,640,532.48	916,162,057.39	5.81%
21	100	5	400	3,306,745,183.01	2,951,717,087.83	10.74%
22	100	10	40	288,564,398.57	271,599,069.84	5.88%
23	100	10	200	1,643,135,394.90	1,286,858,630.98	21.68%
24	100	10	400	3,819,193,540.98	2,986,362,562.50	21.81%
25	100	15	40	369,606,614.62	351,072,587.18	5.01%
26	100	15	200	2,021,023,068.94	888,981,996.25	56.01%
27	100	15	400	4,524,056,016.77	1,979,482,180.41	56.25%
					Average	17.53%

Table B.48: Best Upper Bound and Best Lower Bound (4)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.6 TS	Best Lower Bound	gap
1	10	5	40	2,803,391.18	2,447,361.92	12.70%
2	10	5	200	15,750,456.68	15,749,534.16	0.01%
3	10	5	400	38,768,246.42	34,926,487.98	9.91%
4	10	10	40	2,768,607.10	2,528,328.07	8.68%
5	10	10	200	18,043,239.79	13,405,208.08	25.71%
6	10	10	400	33,301,532.59	28,542,107.34	14.29%
7	10	15	40	2,890,385.33	1,704,499.49	41.03%
8	10	15	200	14,328,114.19	8,755,092.85	38.90%
9	10	15	400	32,436,176.37	19,234,461.08	40.70%
10	50	5	40	84,571,608.69	72,947,049.29	13.75%
11	50	5	200	411,915,587.11	408,013,927.59	0.95%
12	50	5	400	1,231,076,102.52	1,041,281,377.88	15.42%
13	50	10	40	98,572,309.54	90,297,294.59	8.39%
14	50	10	200	537,903,998.54	428,483,645.00	20.34%
15	50	10	400	1,262,077,359.69	1,001,939,273.07	20.61%
16	50	15	40	117,263,637.12	108,272,736.00	7.67%
17	50	15	200	616,447,813.49	285,718,499.15	53.65%
18	50	15	400	1,252,909,254.05	642,241,402.51	48.74%
19	100	5	40	334,554,085.53	293,669,592.32	12.22%
20	100	5	200	1,706,121,405.25	1,688,994,657.01	1.00%
21	100	5	400	5,067,645,767.65	4,279,052,131.57	15.56%
22	100	10	40	416,518,158.16	381,483,522.77	8.41%
23	100	10	200	2,314,882,490.01	1,820,297,803.19	21.37%
24	100	10	400	5,380,972,886.84	4,228,687,835.26	21.41%
25	100	15	40	521,990,191.75	483,451,014.29	7.38%
26	100	15	200	2,796,235,536.43	1,230,405,232.28	56.00%
27	100	15	400	6,246,364,753.44	2,747,609,838.06	56.01%
					Average	21.51%

Table B.49: Best Upper Bound and Best Lower Bound (5)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.7 TS	Best Lower Bound	gap
1	10	5	40	2,216,719.25	2,001,467.06	9.71%
2	10	5	200	12,817,697.99	12,816,958.26	0.01%
3	10	5	400	31,404,750.20	28,433,801.08	9.46%
4	10	10	40	2,327,468.91	2,151,040.77	7.58%
5	10	10	200	14,644,484.18	11,291,996.84	22.89%
6	10	10	400	28,277,569.73	24,540,691.02	13.21%
7	10	15	40	2,445,665.60	2,243,232.68	8.28%
8	10	15	200	12,307,132.49	8,595,420.63	30.16%
9	10	15	400	27,896,140.75	17,141,300.34	38.55%
10	50	5	40	64,298,848.55	58,013,287.41	9.78%
11	50	5	200	329,903,547.12	324,236,960.45	1.72%
12	50	5	400	974,030,012.09	826,928,616.15	15.10%
13	50	10	40	80,783,218.20	75,259,050.89	6.84%
14	50	10	200	446,921,382.09	358,807,278.00	19.72%
15	50	10	400	1,049,370,020.21	833,090,730.96	20.61%
16	50	15	40	97,418,810.58	91,214,661.45	6.37%
17	50	15	200	518,168,389.35	253,679,724.00	51.04%
18	50	15	400	1,056,945,992.84	552,512,055.70	47.73%
19	100	5	40	253,737,256.78	232,836,474.56	8.24%
20	100	5	200	1,361,399,348.85	1,337,262,069.27	1.77%
21	100	5	400	3,999,794,411.56	3,389,070,285.45	15.27%
22	100	10	40	339,600,987.79	317,211,749.82	6.59%
23	100	10	200	1,918,719,960.21	1,511,582,421.83	21.22%
24	100	10	400	4,466,131,499.31	3,507,709,996.04	21.46%
25	100	15	40	432,843,435.19	406,687,682.45	6.04%
26	100	15	200	2,346,657,273.65	1,046,330,098.57	55.41%
27	100	15	400	5,245,282,964.66	2,330,389,273.81	55.57%
					Average	18.90%

Table B.50: Best Upper Bound and Best Lower Bound (6)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.8 TS	Best Lower Bound	gap
1	10	5	40	1,863,248.65	1,709,110.28	8.27%
2	10	5	200	10,909,736.54	10,909,105.06	0.01%
3	10	5	400	26,615,806.73	24,192,749.98	9.10%
4	10	10	40	2,025,836.16	1,882,108.54	7.09%
5	10	10	200	12,440,101.11	9,846,166.30	20.85%
6	10	10	400	24,724,464.60	21,549,473.01	12.84%
7	10	15	40	2,134,600.49	1,972,983.86	7.57%
8	10	15	200	10,850,153.72	10,160,597.84	6.36%
9	10	15	400	24,615,461.40	15,567,178.80	36.76%
10	50	5	40	52,154,363.54	48,248,579.64	7.49%
11	50	5	200	275,744,112.60	270,004,462.95	2.08%
12	50	5	400	807,143,301.90	689,386,984.04	14.59%
13	50	10	40	68,596,635.03	64,586,134.81	5.85%
14	50	10	200	382,707,853.20	307,111,542.01	19.75%
15	50	10	400	898,876,940.50	715,253,281.15	20.43%
16	50	15	40	83,427,178.19	78,855,266.66	5.48%
17	50	15	200	447,298,620.62	211,568,195.84	52.70%
18	50	15	400	914,740,048.02	481,648,162.76	47.35%
19	100	5	40	205,514,341.97	193,060,787.66	6.06%
20	100	5	200	1,133,999,446.40	1,111,164,384.04	2.01%
21	100	5	400	3,306,745,183.01	2,814,327,505.75	14.89%
22	100	10	40	287,158,702.46	271,598,280.27	5.42%
23	100	10	200	1,639,351,726.43	1,287,708,484.35	21.45%
24	100	10	400	3,818,983,851.62	3,004,714,685.69	21.32%
25	100	15	40	370,038,929.95	351,072,368.73	5.13%
26	100	15	200	2,022,476,796.00	907,407,024.61	55.13%
27	100	15	400	4,523,784,327.37	2,023,016,444.49	55.28%
					Average	17.45%

Table B.51: Best Upper Bound and Best Lower Bound (7)

Ins.	|I|	|J|	|K|	δ=0.9 δ'=0.9 TS	Best Lower Bound	gap
1	10	5	40	1,625,159.62	1,502,201.15	7.57%
2	10	5	200	9,566,320.01	9,565,312.05	0.01%
3	10	5	400	23,249,714.58	20,989,581.39	9.72%
4	10	10	40	1,805,876.47	1,680,293.33	6.95%
5	10	10	200	10,893,720.48	8,791,303.96	19.30%
6	10	10	400	22,078,367.96	19,323,807.46	12.48%
7	10	15	40	1,904,484.69	1,767,025.51	7.22%
8	10	15	200	9,749,816.59	9,457,201.00	3.00%
9	10	15	400	22,133,761.02	20,923,752.24	5.47%
10	50	5	40	44,064,095.37	41,364,775.31	6.13%
11	50	5	200	237,281,361.53	231,259,704.04	2.54%
12	50	5	400	689,995,487.73	589,888,224.30	14.51%
13	50	10	40	59,707,015.52	56,618,213.84	5.17%
14	50	10	200	334,953,744.58	269,842,114.96	19.44%
15	50	10	400	786,771,044.12	630,404,578.48	19.87%
16	50	15	40	73,022,207.95	69,487,950.96	4.84%
17	50	15	200	393,767,601.68	187,836,237.90	52.30%
18	50	15	400	806,832,905.01	428,862,527.00	46.85%
19	100	5	40	173,164,494.87	165,021,092.37	4.70%
20	100	5	200	972,619,503.46	948,303,595.64	2.50%
21	100	5	400	2,820,350,975.80	2,406,956,222.32	14.66%
22	100	10	40	247,253,099.96	237,548,073.09	3.93%
23	100	10	200	1,279,302,398.36	1,030,126,236.98	19.48%
24	100	10	400	3,336,964,592.96	2,632,393,839.02	21.11%
25	100	15	40	323,340,281.23	308,924,462.04	4.46%
26	100	15	200	1,777,617,262.08	805,478,263.48	54.69%
27	100	15	400	3,977,961,663.89	1,796,698,231.23	54.83%
					Average	15.69%

Table B.52: Best Upper Bound and Best Lower Bound (8)

B.10 Computational Results for Lagrangian Relaxation and Tabu Search

All the acronyms for computational results for lagrangian relaxation and tabu search from Table B.62, are as below:

Instance	Ins.
Number of clients	|I|
Number of potential proxies	|J|
Number of objects	|K|
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.8$ and $\delta' = 0.6$	gap_{1-1}
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.8$ and $\delta' = 0.7$	gap_{1-2}
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.8$ and $\delta' = 0.8$	gap_{1-3}
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.8$ and $\delta' = 0.9$	gap_{1-4}
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.9$ and $\delta' = 0.6$	gap_{2-1}
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.9$ and $\delta' = 0.7$	gap_{2-2}
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.9$ and $\delta' = 0.8$	gap_{2-3}
Gap between Lagrangian Relaxation and Tabu search when $\delta = 0.9$ and $\delta' = 0.9$	gap_{2-4}

Table B.53: List of Acronyms for Table B.62

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.6 LR	δ=0.8 and δ'=0.6 TS	gap
1	10	5	40	2,776,396.20	3,487,950.79	20%
2	10	5	200	13,087,774.98	18,803,555.02	30%
3	10	5	400	41,312,013.26	46,363,728.72	11%
4	10	10	40	2,639,227.72	3,200,035.57	18%
5	10	10	200	15,320,976.95	21,562,280.13	29%
6	10	10	400	32,469,458.97	37,992,533.93	15%
7	10	15	40	1,883,242.80	3,311,951.19	43%
8	10	15	200	9,682,782.03	16,174,974.71	40%
9	10	15	400	20,847,716.31	36,573,750.25	43%
10	50	5	40	79,434,163.20	91,892,727.35	14%
11	50	5	200	520,430,718.38	591,621,401.61	12%
12	50	5	400	1,325,025,142.33	1,496,874,768.20	11%
13	50	10	40	88,321,056.54	115,880,267.28	24%
14	50	10	200	471,156,745.34	591,774,458.29	20%
15	50	10	400	1,153,073,595.99	1,460,589,116.77	21%
16	50	15	40	60,090,980.97	135,886,688.62	56%
17	50	15	200	314,544,047.83	706,232,388.69	55%
18	50	15	400	722,166,553.52	1,614,608,566.65	55%
19	100	5	40	339,051,850.55	426,609,675.07	21%
20	100	5	200	2,159,884,314.42	2,462,633,256.46	12%
21	100	5	400	5,450,113,895.27	6,172,465,833.53	12%
22	100	10	40	375,240,696.54	495,008,709.66	24%
23	100	10	200	2,114,801,496.82	2,692,716,281.33	21%
24	100	10	400	4,851,144,734.32	6,235,335,323.66	22%
25	100	15	40	260,800,493.12	605,309,158.41	57%
26	100	15	200	1,379,556,565.81	3,204,484,739.48	57%
27	100	15	400	3,069,414,984.16	7,153,766,416.78	57%
					Average	30%

Table B.54: Lagrangian Relaxation and Tabu Search (1)

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.7 LR	δ=0.8 and δ'=0.7 TS	gap
1	10	5	40	2,241,299.98	2,642,938.13	15%
2	10	5	200	14,964,732.98	14,965,332.07	0%
3	10	5	400	33,037,437.14	36,808,753.24	10%
4	10	10	40	2,237,181.98	2,653,641.80	16%
5	10	10	200	12,877,915.93	17,137,631.65	25%
6	10	10	400	27,711,262.91	32,013,192.56	13%
7	10	15	40	1,661,307.49	2,775,732.10	40%
8	10	15	200	8,532,722.37	13,814,155.09	38%
9	10	15	400	18,492,319.13	31,282,793.29	41%
10	50	5	40	66,125,091.61	78,717,439.44	16%
11	50	5	200	405,360,381.22	457,984,277.70	11%
12	50	5	400	1,035,377,033.13	1,162,617,226.61	11%
13	50	10	40	74,317,622.36	93,942,504.68	21%
14	50	10	200	408,629,142.63	514,550,913.12	21%
15	50	10	400	737,484,300.01	928,389,313.49	21%
16	50	15	40	50,402,625.59	112,166,187.45	55%
17	50	15	200	268,705,396.37	591,457,075.80	55%
18	50	15	400	610,276,667.28	1,353,740,000.11	55%
19	100	5	40	265,069,236.82	312,091,673.37	15%
20	100	5	200	1,678,008,058.78	1,899,742,058.39	12%
21	100	5	400	4,515,369,064.95	4,985,458,084.61	9%
22	100	10	40	310,430,599.75	398,634,567.26	22%
23	100	10	200	1,732,014,211.83	2,218,459,716.09	22%
24	100	10	400	4,058,070,901.17	5,146,698,941.43	21%
25	100	15	40	216,530,843.20	498,599,160.19	57%
26	100	15	200	1,163,787,011.00	2,679,903,711.04	57%
27	100	15	400	2,591,004,189.62	5,990,370,288.23	57%

Average	27%

Table B.55: Lagrangian Relaxation and Tabu Search (2)

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.8 LR	δ=0.8 and δ'=0.8 TS	gap
1	10	5	40	1,895,114.59	2,170,681.31	13%
2	10	5	200	12,565,967.13	12,566,452.96	0%
3	10	5	400	27,740,275.99	30,774,811.94	10%
4	10	10	40	1,956,598.56	2,288,505.99	15%
5	10	10	200	11,175,126.63	14,354,260.98	22%
6	10	10	400	24,264,641.08	27,823,743.27	13%
7	10	15	40	1,479,421.88	2,405,788.55	39%
8	10	15	200	7,602,213.37	12,122,330.63	37%
9	10	15	400	16,664,793.76	27,480,370.12	39%
10	50	5	40	53,041,421.43	62,003,433.47	14%
11	50	5	200	332,713,190.23	374,545,176.61	11%
12	50	5	400	850,472,591.19	952,065,665.03	11%
13	50	10	40	62,352,813.61	79,209,642.42	21%
14	50	10	200	349,745,739.15	438,713,649.94	20%
15	50	10	400	819,472,471.50	1,030,150,676.72	20%
16	50	15	40	43,698,013.65	95,630,062.11	54%
17	50	15	200	233,764,241.05	509,180,111.29	54%
18	50	15	400	532,465,561.23	1,166,258,081.36	54%
19	100	5	40	211,399,583.29	247,227,343.27	14%
20	100	5	200	1,371,082,902.54	1,548,872,287.12	11%
21	100	5	400	3,483,840,003.50	3,908,570,277.61	11%
22	100	10	40	261,154,104.89	334,529,186.77	22%
23	100	10	200	1,475,495,317.38	1,887,417,597.66	22%
24	100	10	400	3,433,560,340.47	4,383,727,904.39	22%
25	100	15	40	185,922,722.82	424,351,624.71	56%
26	100	15	200	1,007,337,039.98	2,303,854,003.24	56%
27	100	15	400	2,243,495,135.00	5,154,155,172.58	56%

Average 27%

Table B.56: Lagrangian Relaxation and Tabu Search (3)

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.9 LR	δ=0.8 and δ'=0.9 TS	gap
1	10	5	40	1,653,667.42	1,864,681.62	11%
2	10	5	200	10,909,364.04	10,909,736.54	0%
3	10	5	400	24,069,215.94	26,615,806.73	10%
4	10	10	40	1,741,309.67	2,025,836.16	14%
5	10	10	200	9,913,248.62	12,440,101.11	20%
6	10	10	400	21,685,264.30	24,724,464.60	12%
7	10	15	40	1,359,631.86	2,134,600.49	36%
8	10	15	200	6,926,169.97	10,850,153.72	36%
9	10	15	400	15,451,006.47	24,615,461.40	37%
10	50	5	40	44,888,209.02	52,163,338.32	14%
11	50	5	200	282,621,557.75	317,390,174.59	11%
12	50	5	400	721,839,974.21	807,143,301.90	11%
13	50	10	40	54,438,344.68	68,596,635.03	21%
14	50	10	200	306,265,517.54	382,707,853.20	20%
15	50	10	400	712,431,296.89	898,876,940.50	21%
16	50	15	40	38,431,865.76	83,427,178.19	54%
17	50	15	200	207,687,551.32	447,298,620.62	54%
18	50	15	400	474,425,692.45	1,024,999,734.33	54%
19	100	5	40	177,441,851.37	205,514,341.97	14%
20	100	5	200	1,161,780,218.07	1,308,784,118.83	11%
21	100	5	400	2,951,717,087.83	3,306,745,183.01	11%
22	100	10	40	225,937,476.82	288,564,398.57	22%
23	100	10	200	1,286,858,630.98	1,643,135,394.90	22%
24	100	10	400	2,986,362,562.50	3,819,193,540.98	22%
25	100	15	40	162,633,630.85	369,606,614.62	56%
26	100	15	200	888,981,996.25	2,021,023,068.94	32%
27	100	15	400	1,979,482,180.41	4,524,056,016.77	56%

Average	25%

Table B.57: Lagrangian Relaxation and Tabu Search (4)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.6 LR	δ=0.9 and δ'=0.6 TS	gap
1	10	5	40	2,323,210.65	2,803,391.18	17%
2	10	5	200	15,749,534.16	15,750,456.68	0%
3	10	5	400	34,926,487.98	38,768,246.42	10%
4	10	10	40	2,309,862.90	2,768,607.10	17%
5	10	10	200	13,405,208.08	18,043,239.79	26%
6	10	10	400	28,542,107.34	33,301,532.59	14%
7	10	15	40	1,704,499.49	2,890,385.33	41%
8	10	15	200	8,755,092.85	14,328,114.19	39%
9	10	15	400	19,234,461.08	32,436,176.37	41%
10	50	5	40	67,105,119.11	84,571,608.69	21%
11	50	5	200	408,013,927.59	411,915,587.11	1%
12	50	5	400	1,041,281,377.88	1,231,076,102.52	15%
13	50	10	40	77,725,986.47	98,572,309.54	21%
14	50	10	200	428,483,645.00	537,903,998.54	20%
15	50	10	400	1,001,939,273.07	1,262,077,359.69	21%
16	50	15	40	53,752,392.63	117,263,637.12	54%
17	50	15	200	285,718,499.15	616,447,813.49	54%
18	50	15	400	642,241,402.51	1,252,909,254.05	49%
19	100	5	40	266,837,378.43	334,554,085.53	20%
20	100	5	200	1,688,994,657.01	1,706,121,405.25	1%
21	100	5	400	4,279,052,131.57	5,067,645,767.65	16%
22	100	10	40	321,763,337.87	416,518,158.16	23%
23	100	10	200	1,820,297,803.19	2,314,882,490.01	21%
24	100	10	400	4,228,687,835.26	5,380,972,886.84	21%
25	100	15	40	233,314,035.11	521,990,191.75	55%
26	100	15	200	1,230,405,232.28	2,796,235,536.43	56%
27	100	15	400	2,747,609,838.06	6,246,364,753.44	56%

Average	27%

Table B.58: Lagrangian Relaxation and Tabu Search (5)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.7 LR	δ=0.9 and δ'=0.7 TS	gap
1	10	5	40	1,879,607.22	2,216,719.25	15%
2	10	5	200	12,816,958.26	12,817,697.99	0%
3	10	5	400	28,433,801.08	31,404,750.20	9%
4	10	10	40	1,969,729.20	2,327,468.91	15%
5	10	10	200	11,291,996.84	14,644,484.18	23%
6	10	10	400	24,540,691.02	28,277,569.73	13%
7	10	15	40	1,496,671.34	2,445,665.60	39%
8	10	15	200	7,775,096.17	12,307,132.49	37%
9	10	15	400	17,141,300.34	27,896,140.75	39%
10	50	5	40	52,163,591.31	64,298,848.55	19%
11	50	5	200	324,236,960.45	329,903,547.12	2%
12	50	5	400	826,928,616.15	974,030,012.09	15%
13	50	10	40	64,513,151.08	80,783,218.20	20%
14	50	10	200	358,807,278.00	446,921,382.09	20%
15	50	10	400	833,090,730.96	1,049,370,020.21	21%
16	50	15	40	45,189,666.97	97,418,810.58	54%
17	50	15	200	242,277,261.55	518,168,389.35	53%
18	50	15	400	552,512,055.70	1,056,945,992.84	48%
19	100	5	40	206,649,753.49	253,737,256.78	19%
20	100	5	200	1,337,262,069.27	1,361,399,348.85	2%
21	100	5	400	3,389,070,285.45	3,999,794,411.56	15%
22	100	10	40	266,978,620.99	339,600,987.79	21%
23	100	10	200	1,511,582,421.83	1,918,719,960.21	21%
24	100	10	400	3,507,709,996.04	4,466,131,499.31	21%
25	100	15	40	195,907,176.89	432,843,435.19	55%
26	100	15	200	1,046,330,098.57	2,346,657,273.65	48%
27	100	15	400	2,330,389,273.81	5,245,282,964.66	48%
					Average	26%

Table B.59: Lagrangian Relaxation and Tabu Search (6)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.8 LR	δ=0.9 and δ'=0.8 TS	gap
1	10	5	40	1,598,200.62	1,863,248.65	14%
2	10	5	200	10,909,105.06	10,909,736.54	0%
3	10	5	400	24,192,749.98	26,615,806.73	9%
4	10	10	40	1,743,757.77	2,025,836.16	14%
5	10	10	200	9,846,166.30	12,440,101.11	21%
6	10	10	400	21,549,473.01	24,724,464.60	13%
7	10	15	40	1,349,189.09	2,134,600.49	37%
8	10	15	200	6,995,423.42	10,850,153.72	36%
9	10	15	400	15,567,178.80	24,615,461.40	37%
10	50	5	40	42,910,376.23	52,154,363.54	18%
11	50	5	200	270,004,462.95	275,744,112.60	2%
12	50	5	400	689,386,984.04	807,143,301.90	15%
13	50	10	40	55,163,436.97	68,596,635.03	20%
14	50	10	200	307,111,542.01	382,707,853.20	20%
15	50	10	400	715,253,281.15	898,876,940.50	20%
16	50	15	40	38,912,655.24	83,427,178.19	53%
17	50	15	200	211,568,195.84	447,298,620.62	53%
18	50	15	400	481,648,162.76	914,740,048.02	47%
19	100	5	40	169,401,587.30	205,514,341.97	18%
20	100	5	200	1,111,164,384.04	1,133,999,446.40	2%
21	100	5	400	2,814,327,505.75	3,306,745,183.01	15%
22	100	10	40	227,084,060.59	287,158,702.46	21%
23	100	10	200	1,287,708,484.35	1,639,351,726.43	21%
24	100	10	400	3,004,714,685.69	3,818,983,851.62	21%
25	100	15	40	168,471,801.86	370,038,929.95	54%
26	100	15	200	907,407,024.61	2,022,476,796.00	48%
27	100	15	400	2,023,016,444.49	4,523,784,327.37	55%
					Average	25%

Table B.60: Lagrangian Relaxation and Tabu Search (7)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.9 LR	δ=0.9 and δ'=0.9 TS	gap
1	10	5	40	1,410,971.08	1,625,159.62	13%
2	10	5	200	9,565,312.05	9,566,320.01	0%
3	10	5	400	20,989,581.39	23,249,714.58	10%
4	10	10	40	1,561,187.48	1,805,876.47	14%
5	10	10	200	8,791,303.96	10,893,720.48	19%
6	10	10	400	19,323,807.46	22,078,367.96	12%
7	10	15	40	1,227,527.96	1,904,484.69	36%
8	10	15	200	6,431,931.85	9,749,816.59	34%
9	10	15	400	7,727,388.32	22,133,761.02	65%
10	50	5	40	36,570,671.32	44,064,095.37	17%
11	50	5	200	231,259,704.04	237,281,361.53	3%
12	50	5	400	589,888,224.30	689,995,487.73	15%
13	50	10	40	48,135,891.93	59,707,015.52	19%
14	50	10	200	269,842,114.96	334,953,744.58	19%
15	50	10	400	630,404,578.48	786,771,044.12	20%
16	50	15	40	34,326,185.32	73,022,207.95	53%
17	50	15	200	187,836,237.90	393,767,601.68	52%
18	50	15	400	428,862,527.00	806,832,905.01	47%
19	100	5	40	143,906,325.95	173,164,494.87	17%
20	100	5	200	948,303,595.64	972,619,503.46	3%
21	100	5	400	2,406,956,222.32	2,820,350,975.80	15%
22	100	10	40	197,429,765.25	247,253,099.96	20%
23	100	10	200	1,030,126,236.98	1,279,302,398.36	19%
24	100	10	400	2,632,393,839.02	3,336,964,592.96	21%
25	100	15	40	147,501,204.32	323,340,281.23	54%
26	100	15	200	805,478,263.48	1,777,617,262.08	55%
27	100	15	400	1,796,698,231.23	3,977,961,663.89	55%
					Average	26%

Table B.61: Lagrangian Relaxation and Tabu Search (8)

No.	|I|	|J|	|K|	gap_{1-1}	gap_{1-2}	gap_{1-3}	gap_{1-4}	gap_{2-1}	gap_{2-2}	gap_{2-3}	gap_{2-4}
1	10	5	40	20%	15%	13%	11%	17%	15%	14%	13%
2	10	5	200	30%	0%	0%	0%	0%	0%	0%	0%
3	10	5	400	11%	10%	10%	10%	10%	9%	9%	10%
4	10	10	40	18%	16%	15%	14%	17%	15%	14%	14%
5	10	10	200	29%	25%	22%	20%	26%	23%	21%	19%
6	10	10	400	15%	13%	13%	12%	14%	13%	13%	12%
7	10	15	40	43%	40%	39%	36%	41%	39%	37%	36%
8	10	15	200	40%	38%	37%	36%	39%	37%	36%	34%
9	10	15	400	43%	41%	39%	37%	41%	39%	37%	65%
10	50	5	40	14%	16%	14%	14%	21%	19%	18%	17%
11	50	5	200	12%	11%	11%	11%	16%	15%	15%	15%
12	50	5	400	11%	11%	11%	11%	15%	15%	15%	15%
13	50	10	40	24%	21%	21%	21%	21%	20%	20%	19%
14	50	10	200	20%	21%	20%	20%	20%	20%	20%	19%
15	50	10	400	21%	21%	20%	21%	21%	21%	20%	20%
16	50	15	40	56%	55%	54%	54%	54%	54%	53%	53%
17	50	15	200	55%	55%	54%	54%	54%	53%	53%	52%
18	50	15	400	55%	55%	54%	54%	54%	53%	53%	52%
19	100	5	40	21%	15%	14%	14%	20%	19%	18%	17%
20	100	5	200	12%	12%	11%	11%	16%	16%	15%	15%
21	100	5	400	12%	9%	11%	11%	16%	15%	15%	15%
22	100	10	40	24%	22%	22%	22%	23%	21%	21%	20%
23	100	10	200	21%	22%	22%	22%	21%	21%	21%	21%
24	100	10	400	22%	21%	22%	22%	21%	21%	21%	21%
25	100	15	40	57%	57%	56%	56%	55%	55%	54%	54%
26	100	15	200	57%	57%	56%	56%	56%	55%	55%	55%
27	100	15	400	57%	57%	56%	56%	56%	56%	55%	55%
Average				29%	27%	27%	26%	28%	27%	27%	27%

Table B.62: Gaps Between Lagrangian Relaxation and Tabu Search

B.11 Computational Results for Lagrangian Relaxation and Iterated Variable Neighbourhood Descent

All the acronyms for computational results for lagrangian relaxation and iterated variable neighbourhood descent from Table B.64, are as below:

Instance	Ins.
Number of clients	|I|
Number of potential proxies	|J|
Number of objects	|K|
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.8$ and $\delta' = 0.6$	gap_{3-1}
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.8$ and $\delta' = 0.7$	gap_{3-2}
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.8$ and $\delta' = 0.8$	gap_{3-3}
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.8$ and $\delta' = 0.9$	gap_{3-4}
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.9$ and $\delta' = 0.6$	gap_{4-1}
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.9$ and $\delta' = 0.7$	gap_{4-2}
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.9$ and $\delta' = 0.8$	gap_{4-3}
Gap between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent when $\delta = 0.9$ and $\delta' = 0.9$	gap_{4-4}

Table B.63: List of Acronyms for Table B.64

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.6 LR	δ=0.8 and δ'=0.6 IVND	gap_{3-1}
1	10	5	40	2,776,396.20	3,488,700.29	20%
2	10	5	200	13,087,774.98	18,812,281.82	30%
3	10	5	400	41,312,013.26	46,363,728.72	11%
4	10	10	40	2,639,227.72	3,200,035.57	18%
5	10	10	200	15,320,976.95	21,562,280.13	29%
6	10	10	400	32,469,458.97	38,284,345.04	15%
7	10	15	40	1,883,242.80	3,311,951.19	43%
8	10	15	200	9,682,782.03	16,341,603.87	41%
9	10	15	400	20,847,716.31	36,373,750.25	43%
10	50	5	40	79,434,163.20	108,241,184.21	27%
11	50	5	200	520,430,718.38	591,621,401.61	12%
12	50	5	400	1,325,025,142.33	1,513,558,350.01	12%
13	50	10	40	88,321,056.54	122,907,731.27	28%
14	50	10	200	471,156,745.34	630,787,238.53	25%
15	50	10	400	1,153,073,595.99	1,460,589,116.77	21%
16	50	15	40	60,090,980.97	135,886,688.62	56%
17	50	15	200	314,544,047.83	706,232,388.69	55%
18	50	15	400	722,166,553.52	1,628,501,886.23	56%
19	100	5	40	339,051,850.55	449,043,662.47	24%
20	100	5	200	2,159,884,314.42	2,501,822,226.12	14%
21	100	5	400	5,450,113,895.27	6,194,760,935.03	12%
22	100	10	40	375,240,696.54	546,304,428.59	31%
23	100	10	200	2,114,801,496.82	2,718,049,303.38	22%
24	100	10	400	4,851,144,734.32	6,239,802,196.65	22%
25	100	15	40	260,800,493.12	661,526,208.58	61%
26	100	15	200	1,379,556,565.81	3,222,494,513.45	57%
27	100	15	400	3,069,414,984.16	7,164,172,713.18	57%

Average	31%

Table B.64: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (1)

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.7 LR	δ=0.8 and δ'=0.7 IVND	gap_{3-2}
1	10	5	40	2,241,299.98	2,848,004.99	21%
2	10	5	200	14,964,732.98	14,976,705.91	0%
3	10	5	400	33,037,437.14	38,768,246.42	15%
4	10	10	40	2,237,181.98	2,653,641.80	16%
5	10	10	200	12,877,915.93	17,137,631.65	25%
6	10	10	400	27,711,262.91	32,013,192.56	13%
7	10	15	40	1,661,307.49	2,775,732.10	40%
8	10	15	200	8,532,722.37	13,814,155.09	38%
9	10	15	400	18,492,319.13	31,282,793.29	41%
10	50	5	40	66,125,091.61	81,552,094.63	19%
11	50	5	200	405,360,381.22	467,609,810.54	13%
12	50	5	400	1,035,377,033.13	1,169,139,075.50	11%
13	50	10	40	74,317,622.36	95,181,967.60	22%
14	50	10	200	408,629,142.63	514,550,913.12	21%
15	50	10	400	737,484,300.01	930,458,166.41	21%
16	50	15	40	50,402,625.59	112,166,187.45	55%
17	50	15	200	268,705,396.37	591,457,057.80	55%
18	50	15	400	610,276,667.28	1,361,301,659.98	55%
19	100	5	40	265,069,236.82	358,846,720.78	26%
20	100	5	200	1,678,008,058.78	1,900,295,161.18	12%
21	100	5	400	4,515,369,064.95	4,783,194,674.55	6%
22	100	10	40	310,430,599.75	433,569,791.64	28%
23	100	10	200	1,732,014,211.83	2,218,508,059.33	22%
24	100	10	400	4,058,070,901.17	5,147,032,805.52	21%
25	100	15	40	216,530,843.20	500,767,740.62	57%
26	100	15	200	1,163,787,011.00	2,717,282,151.41	57%
27	100	15	400	2,591,004,189.62	6,017,927,423.33	57%

Average	28%

Table B.65: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (2)

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.8 LR	δ=0.8 and δ'=0.8 IVND	gap_{3-3}
1	10	5	40	1,895,114.59	2,171,183.88	13%
2	10	5	200	12,565,967.13	12,566,452.96	0%
3	10	5	400	27,740,275.99	30,788,051.85	10%
4	10	10	40	1,956,598.56	2,355,003.31	17%
5	10	10	200	11,175,126.63	14,483,509.45	23%
6	10	10	400	24,264,641.08	28,010,316.76	13%
7	10	15	40	1,479,421.88	2,668,109.41	45%
8	10	15	200	7,602,213.37	12,122,330.63	37%
9	10	15	400	16,664,793.76	27,645,897.75	40%
10	50	5	40	53,041,421.43	63,667,128.14	17%
11	50	5	200	332,713,190.23	374,545,176.61	11%
12	50	5	400	850,472,591.19	961,018,514.45	12%
13	50	10	40	62,352,813.61	85,361,131.25	27%
14	50	10	200	349,745,739.15	438,713,649.94	20%
15	50	10	400	819,472,471.50	1,038,924,032.23	21%
16	50	15	40	43,698,013.65	95,630,062.11	54%
17	50	15	200	233,764,241.05	510,970,312.23	54%
18	50	15	400	532,465,561.23	1,176,362,598.43	55%
19	100	5	40	211,399,583.29	255,085,220.42	17%
20	100	5	200	1,371,082,902.54	1,579,281,422.05	13%
21	100	5	400	3,483,840,003.50	3,908,570,277.61	11%
22	100	10	40	261,154,104.89	344,696,269.01	24%
23	100	10	200	1,475,495,317.38	1,891,520,328.22	22%
24	100	10	400	3,433,560,340.47	4,383,998,476.75	22%
25	100	15	40	185,922,722.82	465,790,862.79	60%
26	100	15	200	1,007,337,039.98	2,339,456,987.00	57%
27	100	15	400	2,243,495,135.00	5,161,337,562.70	57%
					Average	28%

Table B.66: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (3)

Ins.	|I|	|J|	|K|	δ=0.8 and δ'=0.9 LR	δ=0.8 and δ'=0.9 IVND	gap_{3-4}
1	10	5	40	1,653,667.42	1,865,122.73	11%
2	10	5	200	10,909,364.04	10,909,736.54	0%
3	10	5	400	24,069,215.94	26,615,806.73	10%
4	10	10	40	1,741,309.67	2,161,566.00	19%
5	10	10	200	9,913,248.62	12,536,711.89	21%
6	10	10	400	21,685,264.30	24,902,737.96	13%
7	10	15	40	1,359,631.86	2,134,600.49	36%
8	10	15	200	6,926,169.97	10,986,793.54	37%
9	10	15	400	15,451,006.47	24,615,461.40	37%
10	50	5	40	44,888,209.02	52,749,373.54	15%
11	50	5	200	282,621,557.75	317,977,666.87	11%
12	50	5	400	721,839,974.21	814,361,990.20	11%
13	50	10	40	54,438,344.68	71,292,283.13	24%
14	50	10	200	306,265,517.54	387,247,321.51	21%
15	50	10	400	712,431,296.89	898,876,940.50	21%
16	50	15	40	38,431,865.76	86,993,149.54	56%
17	50	15	200	207,687,551.32	453,349,616.04	54%
18	50	15	400	474,425,692.45	1,033,355,516.07	54%
19	100	5	40	177,441,851.37	209,298,068.14	15%
20	100	5	200	1,161,780,218.07	1,311,908,320.82	11%
21	100	5	400	2,951,717,087.83	3,306,745,183.01	11%
22	100	10	40	225,937,476.82	314,283,951.71	28%
23	100	10	200	1,286,858,630.98	1,654,098,280.39	22%
24	100	10	400	2,986,362,562.50	3,841,066,926.28	22%
25	100	15	40	162,633,630.85	373,061,921.79	56%
26	100	15	200	888,981,996.25	2,031,832,986.28	56%
27	100	15	400	1,979,482,180.41	4,530,273,946.38	56%
					Average	27%

Table B.67: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (4)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.6 LR	δ=0.9 and δ'=0.6 IVND	gap_{4-1}
1	10	5	40	2,323,210.65	2,805,735.38	17%
2	10	5	200	15,749,534.16	15,761,292.47	0%
3	10	5	400	34,926,487.98	38,778,802.31	10%
4	10	10	40	2,309,862.90	3,006,313.90	23%
5	10	10	200	13,405,208.08	18,043,239.79	26%
6	10	10	400	28,542,107.34	33,301,532.59	14%
7	10	15	40	1,704,499.49	3,190,525.27	47%
8	10	15	200	8,755,092.85	14,328,114.19	39%
9	10	15	400	19,234,461.08	32,436,176.37	41%
10	50	5	40	67,105,119.11	88,147,345.74	24%
11	50	5	200	408,013,927.59	492,226,984.81	17%
12	50	5	400	1,041,281,377.88	1,239,740,910.21	16%
13	50	10	40	77,725,986.47	102,201,656.07	24%
14	50	10	200	428,483,645.00	537,903,998.54	20%
15	50	10	400	1,001,939,273.07	1,272,837,004.85	21%
16	50	15	40	53,752,392.63	117,263,637.12	54%
17	50	15	200	285,718,499.15	616,447,813.49	54%
18	50	15	400	642,241,402.51	1,410,607,923.05	54%
19	100	5	40	266,837,378.43	355,568,998.37	25%
20	100	5	200	1,688,994,657.01	2,028,417,552.31	17%
21	100	5	400	4,279,052,131.57	5,080,757,003.24	16%
22	100	10	40	321,763,337.87	438,825,950.61	27%
23	100	10	200	1,820,297,803.19	2,348,583,464.96	22%
24	100	10	400	4,228,687,835.26	5,381,648,137.39	21%
25	100	15	40	233,314,035.11	561,827,133.59	58%
26	100	15	200	1,230,405,232.28	2,845,821,980.47	57%
27	100	15	400	2,747,609,838.06	6,244,174,979.88	56%

Average	30%

Table B.68: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (5)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.7 LR	δ=0.9 and δ'=0.7 IVND	gap_{4-2}
1	10	5	40	1,879,607.22	2,218,809.56	15%
2	10	5	200	12,816,958.26	12,997,721.74	1%
3	10	5	400	28,433,801.08	31,404,750.20	9%
4	10	10	40	1,969,729.20	2,327,468.91	15%
5	10	10	200	11,291,996.84	14,790,885.44	24%
6	10	10	400	24,540,691.02	28,323,244.26	13%
7	10	15	40	1,496,671.34	2,445,665.60	39%
8	10	15	200	7,775,096.17	12,307,132.49	37%
9	10	15	400	17,141,300.34	28,098,824.98	39%
10	50	5	40	52,163,591.31	69,020,156.53	24%
11	50	5	200	324,236,960.45	386,184,437.94	16%
12	50	5	400	826,928,616.15	979,892,719.07	16%
13	50	10	40	64,513,151.08	80,783,218.20	20%
14	50	10	200	358,807,278.00	449,063,911.29	20%
15	50	10	400	833,090,730.96	1,049,370,020.21	21%
16	50	15	40	45,189,666.97	104,117,814.42	57%
17	50	15	200	242,277,261.55	521,857,120.34	54%
18	50	15	400	552,512,055.70	1,186,758,158.90	53%
19	100	5	40	206,649,753.49	281,664,914.85	27%
20	100	5	200	1,337,262,069.27	1,585,702,719.57	16%
21	100	5	400	3,389,070,285.45	3,999,794,411.56	15%
22	100	10	40	266,978,620.99	354,715,723.75	25%
23	100	10	200	1,511,582,421.83	1,947,380,372.04	22%
24	100	10	400	3,507,709,996.04	4,466,626,071.42	21%
25	100	15	40	195,907,176.89	440,527,676.83	56%
26	100	15	200	1,046,330,098.57	2,347,572,799.36	55%
27	100	15	400	2,330,389,273.81	5,245,727,896.80	56%

Average 28%

Table B.69: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (6)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.8 LR	δ=0.9 and δ'=0.8 IVND	gap_{4-3}
1	10	5	40	1,598,200.62	1,865,122.73	14%
2	10	5	200	10,909,105.06	11,080,427.19	2%
3	10	5	400	24,192,749.98	26,615,806.73	9%
4	10	10	40	1,743,757.77	2,153,078.95	19%
5	10	10	200	9,846,166.30	12,579,687.93	22%
6	10	10	400	21,549,473.01	24,934,739.91	14%
7	10	15	40	1,349,189.09	2,134,600.49	37%
8	10	15	200	6,995,423.42	10,850,153.72	36%
9	10	15	400	15,567,178.80	24,677,161.73	37%
10	50	5	40	42,910,376.23	52,749,373.54	19%
11	50	5	200	270,004,462.95	317,792,630.06	15%
12	50	5	400	689,386,984.04	812,039,329.58	15%
13	50	10	40	55,163,436.97	69,077,315.05	20%
14	50	10	200	307,111,542.01	385,820,058.24	20%
15	50	10	400	715,253,281.15	902,448,553.04	21%
16	50	15	40	38,912,655.24	87,358,187.40	55%
17	50	15	200	211,568,195.84	448,078,598.90	53%
18	50	15	400	481,648,162.76	1,029,992,563.19	53%
19	100	5	40	169,401,587.30	224,244,948.66	24%
20	100	5	200	1,111,164,384.04	1,314,482,508.29	15%
21	100	5	400	2,814,327,505.75	3,306,745,183.01	15%
22	100	10	40	227,084,060.59	313,631,666.91	28%
23	100	10	200	1,287,708,484.35	1,643,166,701.97	22%
24	100	10	400	3,004,714,685.69	3,819,420,526.22	21%
25	100	15	40	168,471,801.86	393,976,485.47	57%
26	100	15	200	907,407,024.61	2,034,342,312.29	55%
27	100	15	400	2,023,016,444.49	4,563,625,562.52	56%

Average 28%

Table B.70: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (7)

Ins.	|I|	|J|	|K|	δ=0.9 and δ'=0.9 LR	δ=0.9 and δ'=0.9 IVND	gap_{4-4}
1	10	5	40	1,410,971.08	1,625,073.94	13%
2	10	5	200	9,565,312.05	9,576,542.44	0%
3	10	5	400	20,989,581.39	23,210,568.48	10%
4	10	10	40	1,561,187.48	1,826,525.81	15%
5	10	10	200	8,791,303.96	10,953,596.58	20%
6	10	10	400	19,323,807.46	22,112,392.38	13%
7	10	15	40	1,227,527.96	1,968,593.76	38%
8	10	15	200	6,431,931.85	9,835,455.60	35%
9	10	15	400	7,727,388.32	22,117,443.22	65%
10	50	5	40	36,570,671.32	46,630,519.92	22%
11	50	5	200	231,259,704.04	274,052,427.01	16%
12	50	5	400	589,888,224.30	693,629,934.73	15%
13	50	10	40	48,135,891.93	61,021,354.58	21%
14	50	10	200	269,842,114.96	338,542,824.49	20%
15	50	10	400	630,404,578.48	789,462,154.58	20%
16	50	15	40	34,326,185.32	75,387,194.47	54%
17	50	15	200	187,836,237.90	393,956,813.74	52%
18	50	15	400	428,862,527.00	906,648,817.49	53%
19	100	5	40	143,906,325.95	186,614,429.88	23%
20	100	5	200	948,303,595.64	1,125,996,088.30	16%
21	100	5	400	2,406,956,222.32	2,821,058,288.88	15%
22	100	10	40	197,429,765.25	255,697,618.81	23%
23	100	10	200	1,030,126,236.98	1,295,700,510.12	20%
24	100	10	400	2,632,393,839.02	3,762,082,522.94	30%
25	100	15	40	147,501,204.32	335,035,149.70	56%
26	100	15	200	805,478,263.48	1,783,203,093.96	55%
27	100	15	400	1,796,698,231.23	3,997,216,843.13	55%
					Average	29%

Table B.71: Lagrangian Relaxation and Iterated Variable Neighbourhood Descent (8)

No.	|I|	|J|	|K|	gap_{3-1}	gap_{3-2}	gap_{3-3}	gap_{3-4}	gap_{4-1}	gap_{4-2}	gap_{4-3}	gap_{4-4}
1	10	5	40	20%	21%	13%	11%	17%	15%	14%	13%
2	10	5	200	30%	0%	0%	0%	0%	1%	2%	0%
3	10	5	400	11%	15%	10%	10%	10%	9%	9%	10%
4	10	10	40	18%	16%	17%	19%	23%	15%	19%	15%
5	10	10	200	29%	25%	23%	21%	26%	24%	22%	20%
6	10	10	400	15%	13%	13%	13%	14%	13%	14%	13%
7	10	15	40	43%	40%	45%	36%	47%	39%	37%	38%
8	10	15	200	41%	38%	37%	37%	39%	37%	36%	35%
9	10	15	400	43%	41%	40%	37%	41%	39%	37%	65%
10	50	5	40	27%	19%	17%	15%	24%	24%	19%	22%
11	50	5	200	12%	13%	11%	11%	17%	16%	15%	16%
12	50	5	400	12%	11%	12%	11%	16%	16%	15%	15%
13	50	10	40	28%	22%	27%	24%	24%	20%	20%	21%
14	50	10	200	25%	21%	20%	21%	20%	20%	20%	20%
15	50	10	400	21%	21%	21%	21%	21%	21%	21%	20%
16	50	15	40	56%	55%	54%	56%	54%	57%	55%	54%
17	50	15	200	55%	55%	54%	54%	54%	54%	53%	52%
18	50	15	400	56%	55%	55%	54%	54%	53%	53%	53%
19	100	5	40	24%	26%	17%	15%	25%	27%	24%	23%
20	100	5	200	14%	12%	13%	11%	17%	16%	15%	16%
21	100	5	400	12%	6%	11%	11%	16%	15%	15%	15%
22	100	10	40	31%	28%	24%	28%	27%	25%	28%	23%
23	100	10	200	22%	22%	22%	22%	22%	22%	22%	20%
24	100	10	400	22%	21%	22%	22%	21%	21%	21%	30%
25	100	15	40	61%	57%	60%	56%	58%	56%	57%	56%
26	100	15	200	57%	57%	57%	56%	57%	55%	55%	55%
27	100	15	400	57%	57%	57%	56%	56%	56%	56%	55%
	Average			31%	28%	28%	27%	30%	28%	28%	29%

Table B.72: Gaps between Lagrangian Relaxation and Iterated Variable Neighbourhood Descent

B.12 Computational Results for Tabu Search and Linear Programming Relaxation

Ins.	|I|	|J|	|K|	δ'	TS	LPR1	gap
1	10	5	40	0.6	3,487,950.79	2,903,233.88	17%
2	10	5	200	0.6	18,803,555.02	18,127,243.24	4%
3	10	5	400	0.6	46,363,728.72	42,824,000.00	8%
4	10	10	40	0.6	3,200,035.57	2,878,154.64	10%
5	10	10	200	0.6	21,562,280.13	15,640,349.74	27%
6	10	15	40	0.6	3,311,951.19	2,950,734.96	11%
7	10	15	200	0.6	16,174,974.71	14,537,400.00	10%
8	50	5	40	0.6	91,892,727.35	88,237,648.79	4%
9	50	10	40	0.6	115,880,267.28	104,279,448.74	10%
10	50	15	40	0.6	135,886,688.62	123,771,089.38	9%
11	100	5	40	0.6	426,609,675.07	355,955,820.86	17%
12	100	5	200	0.6	1,706,131,426.65	1,283,593,391.80	25%
13	100	5	400	0.6	6,172,465,833.53	594,387,731.40	90%
14	100	10	40	0.6	495,008,709.66	441,246,767.27	11%
15	100	10	400	0.6	6,235,335,323.66	5,124,851,741.20	18%
16	100	15	40	0.6	605,309,158.41	553,228,938.29	9%
17	10	5	40	0.7	2,642,938.13	2,329,158.43	12%
18	10	5	200	0.7	14,965,332.07	11,152,724.09	25%
19	10	5	400	0.7	36,808,753.24	27,062,428.53	26%
20	10	10	40	0.7	2,653,641.80	2,431,876.09	8%
21	10	10	200	0.7	17,137,631.65	14,131,382.80	18%
22	10	15	40	0.7	2,775,732.10	2,520,408.34	9%
23	50	5	40	0.7	78,717,439.44	68,985,162.32	12%
24	50	10	40	0.7	93,942,504.68	86,447,169.89	8%

Average	17%

Table B.73: Gaps between Tabu Search and LPR1 (δ=0.8) (1)

Ins.	|I|	|J|	|K|	δ'	TS	LPR1	gap
25	50	15	40	0.7	112,166,187.45	103,944,079.25	7%
26	100	5	40	0.7	312,091,673.37	277,528,281.17	11%
27	100	10	40	0.7	398,634,567.26	365,028,703.75	8%
28	100	15	40	0.7	498,599,160.19	463,970,877.01	7%
29	10	5	40	0.8	2,170,681.31	1,963,197.05	10%
30	10	10	40	0.8	2,288,505.99	2,116,803.63	8%
31	10	15	40	0.8	2,405,788.55	2,209,081.07	8%
32	50	5	40	0.8	62,003,433.47	56,733,170.98	8%
33	50	10	40	0.8	79,209,642.42	73,897,865.98	7%
34	50	15	40	0.8	95,630,062.11	89,649,628.34	6%
35	100	5	40	0.8	247,227,343.27	227,618,577.83	8%
36	100	10	40	0.8	334,529,186.77	311,393,041.70	7%
37	100	15	40	0.8	424,351,624.71	399,645,101.79	6%
38	10	5	40	0.9	1,864,681.62	1,709,163.85	8%
39	10	10	40	0.9	2,025,836.16	1,882,100.15	7%
40	10	15	40	0.9	2,134,600.49	1,972,974.21	8%
41	10	15	200	0.9	10,850,153.72	10,160,597.84	6%
42	10	15	400	0.9	24,615,461.40	12,273,267.65	50%
43	50	5	40	0.9	52,163,338.32	48,250,671.65	8%
44	50	10	40	0.9	68,596,635.03	64,586,406.51	6%
45	50	15	40	0.9	83,427,178.19	78,855,030.76	5%
46	100	5	40	0.9	205,514,341.97	193,065,795.84	6%
47	100	10	40	0.9	288,564,398.57	271,599,069.84	6%
48	100	15	40	0.9	369,606,614.62	351,072,450.79	5%

Average	9%

Table B.74: Gaps between Tabu Search and LPR1 (δ=0.8) (2)

Ins.	|I|	|J|	|K|	δ'	TS	LPR2	gap
1	10	5	40	0.6	3,487,950.79	2,903,235.57	17%
2	10	5	200	0.6	18,803,555.02	16,971,500.00	10%
3	10	5	400	0.6	46,363,728.72	45,982,115.34	1%
4	10	10	40	0.6	3,200,035.57	2,878,167.77	10%
5	10	10	200	0.6	21,562,280.13	16,094,534.97	25%
6	10	15	40	0.6	3,311,951.19	2,950,738.56	11%
7	10	15	200	0.6	16,174,974.71	14,541,900.00	10%
8	50	5	40	0.6	91,892,727.35	88,237,996.95	4%
9	50	10	40	0.6	115,880,267.28	104,279,427.29	10%
10	50	15	40	0.6	135,886,688.62	123,771,576.99	9%
11	100	5	40	0.6	426,609,675.07	355,956,347.43	17%
12	100	10	40	0.6	495,008,709.66	441,247,492.38	11%
13	100	10	200	0.6	2,692,716,281.33	2,623,414,420.40	3%
14	100	15	40	0.6	605,309,158.41	553,197,822.97	9%
15	10	5	40	0.7	2,642,938.13	2,329,168.92	12%
16	10	5	200	0.7	14,965,332.07	11,156,605.54	25%
17	10	5	400	0.7	36,808,753.24	36,532,558.88	1%
18	10	10	40	0.7	2,653,641.80	2,431,866.54	8%
19	10	10	200	0.7	17,137,631.65	15,793,114.44	8%
20	10	15	40	0.7	2,775,732.10	2,520,415.95	9%
21	50	5	40	0.7	78,717,439.44	68,985,281.48	12%
22	50	10	40	0.7	93,942,504.68	86,447,310.81	8%
23	50	15	40	0.7	112,166,187.45	103,944,492.39	7%

Average 10%

Table B.75: Gaps between Tabu Search and LPR2 (δ=0.8) (1)

Ins.	|I|	|J|	|K|	δ'	TS	LPR2	gap
24	100	5	40	0.7	312,091,673.37	277,527,971.25	11%
25	100	10	40	0.7	398,634,567.26	365,029,144.93	8%
26	100	15	40	0.7	498,599,160.19	463,972,293.71	7%
27	10	5	40	0.8	2,170,681.31	1,963,190.13	10%
28	10	10	40	0.8	2,288,505.99	2,116,801.80	8%
29	10	15	40	0.8	2,405,788.55	2,209,083.02	8%
30	10	15	200	0.8	12,122,330.63	5,764,809.19	52%
31	50	5	40	0.8	62,003,433.47	56,733,195.94	8%
32	50	10	40	0.8	79,209,642.42	73,897,956.80	7%
33	50	15	40	0.8	95,630,062.11	89,649,946.00	6%
34	100	5	40	0.8	247,227,343.27	227,619,090.47	8%
35	100	10	40	0.8	334,529,186.77	311,394,015.64	7%
36	100	15	40	0.8	424,351,624.71	399,646,005.10	6%
37	10	5	40	0.9	1,864,681.62	1,709,160.56	8%
38	10	10	40	0.9	2,025,836.16	1,882,108.54	7%
39	10	15	40	0.9	2,134,600.49	1,972,983.86	8%
40	10	15	200	0.9	10,850,153.72	10,156,276.64	6%
41	10	15	400	0.9	24,615,461.40	12,273,267.65	50%
42	50	5	40	0.9	52,163,338.32	48,250,656.76	8%
43	50	10	40	0.9	68,596,635.03	64,586,461.72	6%
44	50	15	40	0.9	83,427,178.19	78,855,305.75	5%
45	100	5	40	0.9	205,514,341.97	193,066,214.51	6%
46	100	10	40	0.9	288,564,398.57	271,598,797.92	6%
47	100	15	40	0.9	369,606,614.62	351,072,587.18	5%
						Average	11%

Table B.76: Gaps between Tabu Search and LPR2 (δ=0.8) (2)

Ins.	|I|	|J|	|K|	δ'	TS	LPR1	gap
1	10	5	40	0.6	2,803,391.18	2,447,358.32	13%
2	10	10	40	0.6	2,768,607.10	2,528,327.39	9%
3	10	15	40	0.6	2,890,385.33	2,614,470.79	10%
4	50	5	40	0.6	84,571,608.69	72,946,713.02	14%
5	50	10	40	0.6	98,572,309.54	90,296,777.83	8%
6	50	15	40	0.6	117,263,637.12	108,272,566.82	8%
7	100	5	40	0.6	334,554,085.53	293,669,592.32	12%
8	100	10	40	0.6	416,518,158.16	381,483,522.77	8%
9	100	15	40	0.6	521,990,191.75	483,449,460.00	7%
10	10	5	40	0.7	2,216,719.25	2,001,457.31	10%
11	10	10	40	0.7	2,327,468.91	2,151,040.77	8%
12	10	15	40	0.7	2,445,665.60	2,243,228.30	8%
13	10	15	200	0.7	12,307,132.49	8,446,407.52	31%
14	50	5	40	0.7	64,298,848.55	58,013,287.41	10%
15	50	10	40	0.7	80,783,218.20	75,259,039.69	7%
16	50	15	40	0.7	97,418,810.58	91,214,412.63	6%
17	50	15	200	0.7	518,168,389.35	251,341,792.00	51%
18	100	5	40	0.7	253,737,256.78	232,836,226.90	8%
19	100	10	40	0.7	339,600,987.79	317,211,632.29	7%
20	100	15	40	0.7	432,843,435.19	406,686,628.77	6%

Average	12%

Table B.77: Gaps between Tabu Search and LPR1 (δ=0.9) (1)

Ins.	|I|	|J|	|K|	δ'	TS	LPR1	gap
21	10	5	40	0.8	1,863,248.65	1,709,110.28	8%
22	10	10	40	0.8	2,025,836.16	1,882,100.15	7%
23	10	15	40	0.8	2,134,600.49	1,972,974.21	8%
24	10	15	200	0.8	10,850,153.72	10,160,597.84	6%
25	50	5	40	0.8	52,154,363.54	48,248,579.64	7%
26	50	10	40	0.8	68,596,635.03	64,586,078.98	6%
27	50	15	40	0.8	83,427,178.19	78,854,991.69	5%
28	100	5	40	0.8	205,514,341.97	193,060,366.81	6%
29	100	10	40	0.8	287,158,702.46	271,598,280.27	5%
30	100	15	40	0.8	370,038,929.95	351,072,232.33	5%
31	10	5	40	0.9	1,625,159.62	1,502,201.15	8%
32	10	10	40	0.9	1,805,876.47	1,680,286.16	7%
33	10	15	40	0.9	1,904,484.69	1,767,017.50	7%
34	10	15	200	0.9	9,749,816.59	9,456,647.47	3%
35	10	15	400	0.9	22,133,761.02	20,923,752.24	5%
36	50	5	40	0.9	44,064,095.37	41,364,775.31	6%
37	50	10	40	0.9	59,707,015.52	56,618,027.95	5%
38	50	15	40	0.9	73,022,207.95	69,487,950.96	5%
39	100	5	40	0.9	173,164,494.87	165,021,092.37	5%
40	100	10	40	0.9	247,253,099.96	237,548,011.38	4%
41	100	15	40	0.9	323,340,281.23	308,924,039.79	4%

Average	6%

Table B.78: Gaps between Tabu Search and LPR1 (δ=0.9) (2)

Ins.	|I|	|J|	|K|	δ'	TS	LPR2	gap
1	10	5	40	0.6	2,803,391.18	2,447,361.92	13%
2	10	10	40	0.6	2,768,607.10	2,528,328.07	9%
3	10	15	40	0.6	2,890,385.33	2,614,475.72	10%
4	50	5	40	0.6	84,571,608.69	72,947,049.29	14%
5	50	10	40	0.6	98,572,309.54	90,297,294.59	8%
6	50	15	40	0.6	117,263,637.12	108,272,736.00	8%
7	100	5	40	0.6	334,554,085.53	293,669,499.75	12%
8	100	10	40	0.6	416,518,158.16	381,485,088.68	8%
9	100	15	40	0.6	521,990,191.75	483,451,014.29	7%
10	10	5	40	0.7	2,216,719.25	2,001,467.06	10%
11	10	10	40	0.7	2,327,468.91	2,151,039.95	8%
12	10	15	40	0.7	2,445,665.60	2,243,232.68	8%
13	10	15	200	0.7	12,307,132.49	8,595,420.63	30%
14	50	5	40	0.7	64,298,848.55	58,013,261.48	10%
15	50	10	40	0.7	80,783,218.20	75,259,050.89	7%
16	50	15	40	0.7	97,418,810.58	91,214,661.45	6%
17	50	15	200	0.7	518,168,389.35	253,679,724.00	51%
18	100	5	40	0.7	253,737,256.78	232,836,474.56	8%
19	100	10	40	0.7	339,600,987.79	317,211,749.82	7%
20	100	15	40	0.7	432,843,435.19	406,687,682.45	6%
						Average	12%

Table B.79: Gaps between Tabu Search and LPR2 (δ=0.9) (1)

Ins.	|I|	|J|	|K|	δ'	TS	LPR2	gap
21	10	5	40	0.8	1,863,248.65	1,709,106.28	8%
22	10	10	40	0.8	2,025,836.16	1,882,108.54	7%
23	10	15	40	0.8	2,134,600.49	1,972,983.86	8%
24	10	15	200	0.8	10,850,153.72	10,156,276.64	6%
25	50	5	40	0.8	52,154,363.54	48,248,566.79	7%
26	50	10	40	0.8	68,596,635.03	64,586,134.81	6%
27	50	15	40	0.8	83,427,178.19	78,855,266.66	5%
28	100	5	40	0.8	205,514,341.97	193,060,787.66	6%
29	100	10	40	0.8	287,158,702.46	271,598,007.34	5%
30	100	15	40	0.8	370,038,929.95	351,072,368.73	5%
31	10	5	40	0.9	1,625,159.62	1,502,199.84	8%
32	10	10	40	0.9	1,805,876.47	1,680,293.33	7%
33	10	15	40	0.9	1,904,484.69	1,767,025.51	7%
34	10	15	200	0.9	9,749,816.59	9,457,201.00	3%
35	10	15	400	0.9	22,133,761.02	20,920,142.27	5%
36	50	5	40	0.9	44,064,095.37	41,364,767.90	6%
37	50	10	40	0.9	59,707,015.52	56,618,213.84	5%
38	50	15	40	0.9	73,022,207.95	69,487,786.19	5%
39	100	5	40	0.9	173,164,494.87	165,021,019.60	5%
40	100	10	40	0.9	247,253,099.96	237,548,073.09	4%
41	100	15	40	0.9	323,340,281.23	308,924,462.04	4%

Average 6%

Table B.80: Gaps between Tabu Search and LPR2 (δ=0.9) (2)

Appendix C

Computational Results for Cache Allocation Problem

C.1 Finding the Best Percentage for Capacity of Proxy

Linearization I

			Capacity of each proxy 5%		
|I|	|J|	|K|	Integer Programming	Linear Programming	percentage
10	10	50	38,771.74	38,095.90	70.00%
10	10	450	2,907,556.31	2,890,297.17	72.89%
10	10	300	1,242,402.28	1,236,317.98	74.33%
10	10	150	344,012.99	340,252.50	76.00%

			Capacity of each proxy 7.5%		
|I|	|J|	|K|	Integer Programming	Linear Programming	percentage
10	10	50	18,043.32	17,499.75	86.00%
10	10	450	1,321,291.16	1,311,675.82	86.66%
10	10	300	584,002.12	569,745.18	83.66%
10	10	150	157,153.72	154,508.83	86.00%

			Capacity of each proxy 10%		
|I|	|J|	|K|	Integer Programming	Linear Programming	percentage
10	10	50	2,474.93	2,474.78	98.00%
10	10	450	21,911.43	21,910.30	99.77%
10	10	300	14,171.76	14,171.42	99.66%
10	10	150	7,382.20	7,381.87	99.33%

Table C.1: Test Instance Linearization I

Linearization II

			Capacity of each proxy 5%		
|I|	|J|	|K|	Integer Programming	Linear Programming	percentage
10	10	50	38,771.74	38,095.90	70.00%
10	10	450	2,907,381.71	2,890,297.17	72.89%
10	10	300	1,242,457.64	1,236,317.98	74.33%
10	10	150	344,021.38	340,252.50	76.00%

			Capacity of each proxy 7.5%		
|I|	|J|	|K|	Integer Programming	Linear Programming	percentage
10	10	50	18,043.32	17,997.54	86.00%
10	10	450	1,321,200.93	1,311,675.86	86.66%
10	10	300	584,092.28	569,745.18	83.66%
10	10	150	157,152.00	154,508.83	86.00%

			Capacity of each proxy 10%		
|I|	|J|	|K|	Integer Programming	Linear Programming	percentage
10	10	50	2,476.99	2,474.78	98.00%
10	10	450	21,911.55	21,910.30	99.77%
10	10	300	14,172.77	14,171.42	99.66%
10	10	150	7,387.25	7,381.87	99.33%

Table C.2: Test Instance Linearization II

C.2 Upper Bound Computational Results

Ins.	|K|	Linear Programming Relaxation	TS Constructive	gap_c^{TS-LP}
1	50	38,096	43,644	12.71%
2	100	158,055	178,846	11.63%
3	150	340,733	459,526	25.85%
4	200	556,423	650,238	14.43%
5	250	905,593	992,878	8.79%
6	300	1,238,290	1,465,196	15.49%
7	350	1,683,293	2,038,791	17.44%
8	400	2,273,341	2,644,714	14.04%
9	450	2,891,418	4,344,814	33.45%
10	500	3,457,982	4,684,139	26.18%
11	600	5,177,858	7,712,104	32.86%
12	700	6,773,535	8,951,189	24.33%
13	800	9,031,618	12,082,230	25.25%
14	900	11,982,900	16,097,491	25.56%
15	1000	14,489,600	17,702,919	18.15%
16	2000	56,941,200	65,254,359	12.74%
17	3000	130,208,000	148,095,004	12.08%
18	4000	225,417,000	244,511,788	7.81%
19	5000	359,707,000	377,771,938	4.78%
20	6000	522,806,000	551,694,100	5.24%
21	7000	718,359,000	785,627,325	8.56%
22	8000	931,705,000	988,624,134	5.76%
23	9000	1,178,310,000	1,259,245,368	6.43%
24	10000	1,439,720,000	1,581,840,062	8.98%
			Average	15.77%

Table C.3: TS with Constructive Heuristic Algorithm with 5% Capacity

Ins.	|K|	Linear Programming Relaxation	TS-greedy	gap_g^{TS-LP}
1	50	38,096	40,724	6.45%
2	100	158,055	163,694	3.45%
3	150	340,733	368,605	7.56%
4	200	556,423	564,541	1.44%
5	250	905,593	951,822	4.86%
6	300	1,238,290	1,265,130	2.12%
7	350	1,683,293	1,797,410	6.35%
8	400	2,273,341	2,391,741	4.95%
9	450	2,891,418	2,892,615	0.04%
10	500	3,457,982	3,781,446	8.55%
11	600	5,177,858	5,225,953	0.92%
12	700	6,773,535	6,909,715	1.97%
13	800	9,031,618	9,055,020	0.26%
14	900	11,982,900	12,087,946	0.87%
15	1000	14,489,600	14,926,525	2.93%
16	2000	56,941,200	61,226,843	7.00%
17	3000	130,208,000	140,199,245	7.13%
18	4000	225,417,000	237,510,927	5.09%
19	5000	359,707,000	361,073,306	0.38%
20	6000	522,806,000	543,783,929	3.86%
21	7000	718,359,000	756,778,071	5.08%
22	8000	931,705,000	945,343,023	1.44%
23	9000	1,178,310,000	1,215,779,646	3.08%
24	10000	1,439,720,000	1,441,070,928	0.09%
			Average	3.58%

Table C.4: TS with Greedy Heuristic Algorithm with 5% Capacity

Ins.	|K|	Linear Programming Relaxation	IVND Constructive	$gap_c^{IVND-LP}$
1	50	38,096	49,419	22.91%
2	100	158,055	240,649	34.32%
3	150	340,733	520,955	34.59%
4	200	556,423	811,716	31.45%
5	250	905,593	1,380,175	34.39%
6	300	1,238,290	1,893,057	34.59%
7	350	1,683,293	2,752,798	38.85%
8	400	2,273,341	3,617,063	37.15%
9	450	2,891,418	4,446,798	34.98%
10	500	3,457,982	4,717,867	26.70%
11	600	5,177,858	8,060,392	35.76%
12	700	6,773,535	9,413,297	28.04%
13	800	9,031,618	12,171,196	25.80%
14	900	11,982,900	16,177,814	25.93%
15	1000	14,489,600	18,165,041	20.23%
16	2000	56,941,200	67,525,114	15.67%
17	3000	130,208,000	151,382,975	13.99%
18	4000	225,417,000	250,579,440	10.04%
19	5000	359,707,000	383,192,970	6.13%
20	6000	522,806,000	559,004,873	6.48%
21	7000	718,359,000	791,895,991	9.29%
22	8000	931,705,000	1,001,647,860	6.98%
23	9000	1,178,310,000	1,269,730,041	7.20%
24	10000	1,439,720,000	1,593,187,654	9.63%

Average	22.96%

Table C.5: IVND with Constructive Heuristic Algorithm with 5% Capacity

Ins.	|K|	Linear Programming Relaxation	IVND-greedy	$gap_g^{IVND-LP}$
1	50	38,096	40,825	6.69%
2	100	158,055	166,750	5.21%
3	150	340,733	369,078	7.68%
4	200	556,423	567,834	2.01%
5	250	905,593	953,484	5.02%
6	300	1,238,290	1,267,972	2.34%
7	350	1,683,293	1,798,682	6.42%
8	400	2,273,341	2,394,013	5.04%
9	450	2,891,418	2,894,838	0.12%
10	500	3,457,982	3,801,256	9.03%
11	600	5,177,858	5,227,211	0.94%
12	700	6,773,535	6,928,272	2.23%
13	800	9,031,618	9,708,216	6.97%
14	900	11,982,900	12,088,859	0.88%
15	1000	14,489,600	15,776,935	8.16%
16	2000	56,941,200	63,350,259	10.12%
17	3000	130,208,000	143,860,052	9.49%
18	4000	225,417,000	239,916,953	6.04%
19	5000	359,707,000	362,596,004	0.80%
20	6000	522,806,000	553,803,954	5.60%
21	7000	718,359,000	764,748,683	6.07%
22	8000	931,705,000	1,024,988,416	9.10%
23	9000	1,178,310,000	1,219,457,484	3.37%
24	10000	1,439,720,000	1,442,065,979	0.16%

Average	4.98%

Table C.6: IVND with Greedy Algorithm with 5% Capacity

Ins.	|K|	TS Constructive	IVND Constructive	gap_c
1	50	43,644	49,419	11.69%
2	100	178,846	240,649	25.68%
3	150	459,526	520,955	11.79%
4	200	650,238	811,716	19.89%
5	250	992,878	1,380,175	28.06%
6	300	1,465,196	1,893,057	22.60%
7	350	2,038,791	2,752,798	25.94%
8	400	2,644,714	3,617,063	26.88%
9	450	4,344,814	4,446,798	2.29%
10	500	4,684,139	4,717,867	0.71%
11	600	7,712,104	8,060,392	4.32%
12	700	8,951,189	9,413,297	4.91%
13	800	12,082,230	12,171,196	0.73%
14	900	16,097,491	16,177,814	0.50%
15	1000	17,702,919	18,165,041	2.54%
16	2000	65,254,359	67,525,114	3.36%
17	3000	148,095,004	151,382,975	2.17%
18	4000	244,511,788	250,579,440	2.42%
19	5000	377,771,938	383,192,970	1.41%
20	6000	551,694,100	559,004,873	1.31%
21	7000	785,627,325	791,895,991	0.79%
22	8000	988,624,134	1,001,647,860	1.30%
23	9000	1,259,245,368	1,269,730,041	0.83%
24	10000	1,581,840,062	1,593,187,654	0.71%
			Average	8.45%

Table C.7: Compare TS and IVND with Constructive Heuristic Algorithm with 5% Capacity

Ins.	|K|	TS-greedy	IVND-greedy	gap_g
1	50	40,724	40,825	0.25%
2	100	163,694	166,750	1.83%
3	150	368,605	369,078	0.13%
4	200	564,541	567,834	0.58%
5	250	951,822	953,484	0.17%
6	300	1,265,130	1,267,972	0.22%
7	350	1,797,410	1,798,682	0.07%
8	400	2,391,741	2,394,013	0.09%
9	450	2,892,615	2,894,838	0.08%
10	500	3,781,446	3,801,256	0.52%
11	600	5,225,953	5,227,211	0.02%
12	700	6,909,715	6,928,272	0.27%
13	800	9,055,020	9,708,216	6.73%
14	900	12,087,946	12,088,859	0.01%
15	1000	14,926,525	15,776,935	5.39%
16	2000	61,226,843	63,350,259	3.35%
17	3000	140,199,245	143,860,052	2.54%
18	4000	237,510,927	239,916,953	1.00%
19	5000	361,073,306	362,596,004	0.42%
20	6000	543,783,929	553,803,954	1.81%
21	7000	756,778,071	764,748,683	1.04%
22	8000	945,343,023	1,024,988,416	7.77%
23	9000	1,215,779,646	1,219,457,484	0.30%
24	10000	1,441,070,928	1,442,065,979	0.07%
			Average	1.44%

Table C.8: Compare TS and IVND with Greedy Algorithm with 5% Capacity

Ins.	|K|	Linear Programming Relaxation	TS Constructive	gap_c^{TS-LP}
1	50	18,000	17,956	-0.25%
2	100	71,788	76,497	6.16%
3	150	154,663	158,515	2.43%
4	200	254,029	274,467	7.45%
5	250	407,821	427,473	4.60%
6	300	571,331	594,223	3.85%
7	350	757,950	817,902	7.33%
8	400	1,042,998	1,054,989	1.14%
9	450	1,311,728	1,358,097	3.41%
10	500	1,557,948	1,748,727	10.91%
11	600	2,381,839	2,743,350	13.18%
12	700	3,084,146	3,979,796	22.50%
13	800	4,121,541	5,398,990	23.66%
14	900	5,495,972	6,901,776	20.37%
15	1000	6,632,839	8,025,195	17.35%
16	2000	25,776,200	29,236,425	11.84%
17	3000	58,963,000	65,023,633	9.32%
18	4000	103,135,000	118,034,118	12.62%
19	5000	164,904,000	186,867,741	11.75%
20	6000	238,316,000	260,911,861	8.66%
21	7000	328,094,000	346,376,990	5.28%
22	8000	425,089,000	448,797,071	5.28%
23	9000	537,960,000	556,158,640	3.27%
24	10000	661,782,000	675,495,203	2.03%
			Average	8.92%

Table C.9: TS with Constructive Heuristic Algorithm with 7.5% Capacity

Ins.	|K|	Linear Programming Relaxation	TS-greedy	gap_g^{TS-LP}
1	50	18,000	18,669	3.58%
2	100	71,788	75,817	5.31%
3	150	154,663	157,596	1.86%
4	200	254,029	261,559	2.88%
5	250	407,821	421,433	3.23%
6	300	571,331	583,024	2.01%
7	350	757,950	793,924	4.53%
8	400	1,042,998	1,046,623	0.35%
9	450	1,311,728	1,327,826	1.21%
10	500	1,557,948	1,649,974	5.58%
11	600	2,381,839	2,453,517	2.92%
12	700	3,084,146	3,281,114	6.00%
13	800	4,121,541	4,489,571	8.20%
14	900	5,495,972	5,756,414	4.52%
15	1000	6,632,839	7,089,520	6.44%
16	2000	25,776,200	28,259,000	8.79%
17	3000	58,963,000	63,744,932	7.50%
18	4000	103,135,000	109,890,154	6.15%
19	5000	164,904,000	175,354,922	5.96%
20	6000	238,316,000	249,786,000	4.59%
21	7000	328,094,000	341,028,260	3.79%
22	8000	425,089,000	437,136,449	2.76%
23	9000	537,960,000	548,664,145	1.95%
24	10000	661,782,000	671,973,632	1.52%

Average	4.23%

Table C.10: TS with Greedy Algorithm with 7.5% Capacity

Ins.	|K|	Linear Programming Relaxation	IVND constructive	gap_c^{TS-LP}
1	50	17,500	19,333	9.48%
2	100	71,609	91,928	22.10%
3	150	154,509	190,783	19.01%
4	200	253,639	386,563	34.39%
5	250	407,046	582,077	30.07%
6	300	569,745	941,421	39.48%
7	350	757,222	1,119,051	32.33%
8	400	1,041,283	1,309,056	20.46%
9	450	1,311,676	1,659,574	20.96%
10	500	1,557,813	2,321,308	32.89%
11	600	2,379,367	3,795,248	37.31%
12	700	3,082,626	4,430,040	30.42%
13	800	4,120,777	5,746,803	28.29%
14	900	5,492,808	7,302,031	24.78%
15	1000	6,630,400	8,125,195	18.40%
16	2000	25,767,054	31,011,325	16.91%
17	3000	58,958,130	72,746,425	18.95%
18	4000	103,117,562	118,730,780	13.15%
19	5000	164,884,378	190,677,280	13.53%
20	6000	238,314,949	273,053,763	12.72%
21	7000	328,064,783	358,627,547	8.52%
22	8000	425,086,808	454,645,462	6.50%
23	9000	537,934,360	558,616,579	3.70%
24	10000	661,726,319	686,542,164	3.61%
			Average	20.75%

Table C.11: IVND with Constructive Heuristic Algorithm with 7.5% Capacity

Ins.	|K|	Linear Programming Relaxation	IVND-greedy	gap_g^{TS-LP}
1	50	17,500	18,448	5.14%
2	100	71,609	87,117	17.80%
3	150	154,509	184,031	16.04%
4	200	253,639	333,914	24.04%
5	250	407,046	494,197	17.63%
6	300	569,745	804,327	29.17%
7	350	757,222	942,349	19.65%
8	400	1,041,283	1,189,945	12.49%
9	450	1,311,676	1,476,589	11.17%
10	500	1,557,813	2,021,924	22.95%
11	600	2,379,367	3,087,908	22.95%
12	700	3,082,626	4,606,013	33.07%
13	800	4,120,777	4,836,920	14.81%
14	900	5,492,808	6,149,825	10.68%
15	1000	6,630,400	7,489,792	11.47%
16	2000	25,767,054	29,762,331	13.42%
17	3000	58,958,130	66,811,633	11.75%
18	4000	103,117,562	109,191,225	5.56%
19	5000	164,884,378	179,414,817	8.10%
20	6000	238,314,949	261,166,489	8.75%
21	7000	328,064,783	350,404,626	6.38%
22	8000	425,086,808	446,575,523	4.81%
23	9000	537,934,360	555,471,631	3.16%
24	10000	661,726,319	678,892,533	2.53%
			Average	13.90%

Table C.12: IVND with Greedy Algorithm with 7.5% Capacity

Ins.	|K|	TS Constructive	IVND constructive	gap_c
1	50	17,956	19,333	7.12%
2	100	76,497	91,928	16.79%
3	150	158,515	190,783	16.91%
4	200	274,467	386,563	29.00%
5	250	427,473	582,077	26.56%
6	300	594,223	941,421	36.88%
7	350	817,902	1,119,051	26.91%
8	400	1,054,989	1,309,056	19.41%
9	450	1,358,097	1,659,574	18.17%
10	500	1,748,727	2,321,308	24.67%
11	600	2,743,350	3,795,248	27.72%
12	700	3,979,796	4,430,040	10.16%
13	800	5,398,990	5,746,803	6.05%
14	900	6,901,776	7,302,031	5.48%
15	1000	8,025,195	8,125,195	1.23%
16	2000	29,236,425	31,011,325	5.72%
17	3000	65,023,633	72,746,425	10.62%
18	4000	118,034,118	118,730,780	0.59%
19	5000	186,867,741	190,677,280	2.00%
20	6000	260,911,861	273,053,763	4.45%
21	7000	346,376,990	358,627,547	3.42%
22	8000	448,797,071	454,645,462	1.29%
23	9000	556,158,640	558,616,579	0.44%
24	10000	675,495,203	686,542,164	1.61%
			Average	12.63%

Table C.13: Compare TS and IVND with Constructive Heuristic Algorithm with 7.5% Capacity

Ins.	|K|	TS-greedy	IVND-greedy	gap_g
1	50	18,669	18,448	-1.20%
2	100	75,817	87,117	12.97%
3	150	157,596	184,031	14.36%
4	200	261,559	333,914	21.67%
5	250	421,433	494,197	14.72%
6	300	583,024	804,327	27.51%
7	350	793,924	942,349	15.75%
8	400	1,046,623	1,189,945	12.04%
9	450	1,327,826	1,476,589	10.07%
10	500	1,649,974	2,021,924	18.40%
11	600	2,453,517	3,087,908	20.54%
12	700	3,281,114	4,606,013	28.76%
13	800	4,489,571	4,836,920	7.18%
14	900	5,756,414	6,149,825	6.40%
15	1000	7,089,520	7,489,792	5.34%
16	2000	28,259,000	29,762,331	5.05%
17	3000	63,744,932	66,811,633	4.59%
18	4000	109,890,154	109,191,225	-0.64%
19	5000	175,354,922	179,414,817	2.26%
20	6000	249,786,000	261,166,489	4.36%
21	7000	341,028,260	350,404,626	2.68%
22	8000	437,136,449	446,575,523	2.11%
23	9000	548,664,145	555,471,631	1.23%
24	10000	671,973,632	678,892,533	1.02%
			Average	9.88%

Table C.14: Compare TS and IVND with Greedy Algorithm with 7.5% Capacity

Bibliography

W. Adams and H. Sherali. Linearization strategies for a class of zero-one mixed integer programming problems. *Operations Research*, 38(2):217–226, 1990.

J.M. Almeida, D.L. Eager, M.K. Vernon, and S.J. Wright. Minimizing delivery cost in scalable streaming content distribution systems. *IEEE Transactions on Multimedia*, 6 (2):356–365, 2004.

I.D. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 661–670. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2001.

B. Bailey and E. Schaffer. Acceptable computer response times. *UI Design Update Newsletter*, 2001.

S. Bakiras and T. Loukopoulos. Combining replica placement and caching techniques in content distribution networks. *Computer Communications*, 28(9):1062–1073, 2005.

A. Barbir, B. Cain, R. Nair, and O. Spatscheck. Known content network (cn) request-routing mechanisms. *Internet Engineering Task Force RFC*, 3568, 2003.

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

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 and Operations Research*, 35(12):3860–3884, 2008.

T. Bektaş, M. Chouman, and T.G. Crainic. Lagrangean-based decomposition algorithms for multicommodity network design problems with penalized constraints. *Networks*, 55 (3):171–180, 2009.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

E.K. Burke and G. Kendall. *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2005.

R. Buyya, M. Pathan, and A. Vakali. *Content delivery networks*, volume 9. Springer, 2008.

CH. Chang. An efficient linearization approach for mixed-integer problems. *European Journal of Operational Research*, 123(3):652–659, 2000.

CH. Chang and CH. Chang. A linearization method for mixed 0–1 polynomial programs. *Computers & Operations Research*, 27(10):1005–1016, 2000.

J. Chen, X. Sun, and H. Guo. An efficient algorithm for multi-dimensional nonlinear knapsack problems. *Journal of Shanghai University (English Edition)*, 10(5):393–398, 2006.

I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. *Computer Networks*, 40(2):205–218, 2002.

Edward G. Coffman J., M.R. Garey, and D.S. Johnson. Approximation algorithms for bin-packingan updated survey. In *Algorithm design for computer system design*, pages 49–106. Springer, 1984.

S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

R.C. Coutinho, L. Drummond, and Y. Frota. A distributed transportation simplex applied to a content distribution network problem. *arXiv preprint arXiv:1210.6384*, 2012.

J.K. Deane, T.R. Rakes, and A. Agarwal. Designing content distribution networks for optimal cost and performance. *Information Technology and Management*, 13(1):1–15, 2012.

M. Drwal and J. Jozefczyk. Decomposition algorithms for data placement problem based on lagrangian relaxation and randomized rounding. *Annals of Operations Research*, pages 1–17, 2013.

O. Ercetin. Market-based resource allocation for content delivery in the internet. *IEEE Journal on Selected Areas in Communications*, 2002.

G. Erdoğan. The orienteering problem with variable profits. *Networks*, volume 61(1): 104–116, 2012.

G. Erdoğan, J.F. Cordeau, and G. Laporte. The attractive traveling salesman problem. *European Journal of Operational Research*, 203(1):59–69, 2010.

M.L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management science*, pages 1861–1871, 2004.

M.R. Garey and D.S. Johnson. *Computers and intractability*, volume 29. wh freeman, 2002a.

M.R. Garey and D.S. Johnson. *Computers and intractability*, volume 29. wh freeman, 2002b.

B. Gavish. Topological design of telecommunication networks-local access design methods. *Annals of Operations Research*, 33(1):17–71, 1991.

M. Gendreau and J.Y. Potvin. *Handbook of metaheuristics*, volume 2. Springer, 2010.

A. Geoffrion. *Lagrangean relaxation for integer programming.* Springer, 1974.

V.R. Ghezavati and M. Saidi-Mehrabad. An efficient linearization technique for mixed 0ā1 polynomial problem. *Journal of Computational and Applied Mathematics*, 235(6): 1730–1738, 2011.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

F. Glover and G.A. Kochenberger. *Handbook of metaheuristics*. Springer, 2003.

F. Glover and M. Laguna. *Tabu search*. Springer, 1999.

F. Glover and E. Woolsey. Further reduction of zero-one polynomial programming problems to zero-one linear programming problems. *Operations Research*, 21(1):156–161, 1973.

F. Glover and E. Woolsey. Technical noteconverting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22(1):180–182, 1974.

M. Held, Ph. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical programming*, 6(1):62–88, 1974.

A. Hertz and M. Mittaz. A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation science*, 35(4):425–434, 2001.

K. Hosanagar, R. Krishnan, J. Chuang, and V. Choudhary. Pricing and resource allocation in caching services with multiple levels of quality of service. *Management Science*, 51 (12):1844–1859, 2005.

B. Hu and G. Raidl. Variable neighborhood descent with self-adaptive neighborhood-ordering. In *Proceedings of the 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*. Citeseer, 2006.

C. Huang and T. Abdelzaher. Bounded-latency content distribution feasibility and evaluation. *Computers, IEEE Transactions on*, 54(11):1422–1437, 2005.

Al. Imtiaz, Md . Hossain, et al. Distributed cache management architecture: To reduce the internet traffic by integrating browser and proxy caches. In *Electrical Engineering and Information & Communication Technology (ICEEICT), 2014 International Conference on*, pages 1–4. IEEE, 2014.

H. Jabraili, S. Yousefi, B. Boukani, and M.B. Rad. Replication based on objects iteration frequency and load using a genetic algorithm under a content distribution network. In *Electrical Engineering (ICEE), 2013 21st Iranian Conference on*, pages 1–6. IEEE, 2013.

X. Jia, D. Li, X. Hu, and D.Z. Du. Optimal placement of web proxies for replicated web servers in the internet. *The Computer Journal*, 44(5):329–339, 2001.

J. Kangasharju, J. Roberts, and K.W. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25(4):376–383, 2002.

R.M. Karp. *Reducibility among combinatorial problems.* Springer, 1972.

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems.* Springer, 2004a.

H. Kellerer, U. Pferschy, and D. Pisinger. Basic algorithmic concepts. In *Knapsack Problems*, pages 15–42. Springer, 2004b.

O. Kettani and M. Oral. Equivalent formulations of nonlinear integer problems for efficient optimization. *Management Science*, 36(1):115–119, 1990.

S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671, 1983.

R.E. Korf. A new algorithm for optimal bin packing. In *AAAI/IAAI*, pages 731–736, 2002.

P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Transactions on Networking (TON)*, 8(5):568–582, 2000.

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.

N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis. Joint object placement and node dimensioning for internet content distribution. *Information Processing Letters*, 89(6): 273–279, 2004.

N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis. On the optimization of storage capacity allocation for content distribution. *Computer Networks*, 47(3):409–428, 2005.

E.L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

B. Li, M.J. Golin, G.F. Italiano, X. Deng, and K. Sohraby. On the optimal placement of web proxies in the internet. In *IEEE INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, volume 3, 1999.

D. Li, X.L. Sun, J. Wang, and Ken I.M. McKinnon. Convergent lagrangian and domain cut method for nonlinear knapsack problems. *Computational Optimization and Applications*, 42(1):67–104, 2009.

H.L. Li. Global optimization for mixed 0-1 programs with convex or separable continuous functions. *Journal of the Operational Research Society*, pages 1068–1076, 1994.

H.L. Li and Ch. Chang. An approximate approach of global optimization for polynomial programming problems. *European Journal of Operational Research*, 107(3):625–632, 1998.

H.R. Lourenço, O.C. Martin, and T. Stützle. *Iterated local search.* Springer, 2003.

H. Luss. Optimal content distribution in video-on-demand tree networks. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 40(1):68–75, 2010.

Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc., 1990.

N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

V. Mookerjee and Y. Tan. Analysis of a least recently used cache management policy for web browsers. *Operations Research*, 50(2):345–357, 2002.

F.F.H. Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

A. Nedić. Lagrangian optimization methods for nonlinear programming. *Wiley Encyclopedia of Operations Research and Management Science*, 2011.

T. Neves, L.S. Ochi, and Cé. Albuquerque. A new hybrid heuristic for replica placement and request distribution in content distribution networks. *Optimization Letters*, pages 1–16, 2014.

T.A. Neves, L. Drummond, L.S. Ochi, C. Albuquerque, and E. Uchoa. Solving replica placement and request distribution in content distribution networks. *Electronic Notes in Discrete Mathematics*, 36:89–96, 2010.

I.H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623, 1996.

Susan Hesse Owen and Mark S Daskin. Strategic facility location: A review. *European Journal of Operational Research*, 111(3):423–447, 1998.

M. Pathan, R. Buyya, and A. Vakali. Content delivery networks: State of the art, insights, and imperatives. *Content Delivery Networks*, pages 3–32, 2008.

G. Peng. Cdn: Content distribution network. *arXiv preprint cs/0411069*, 2004.

G. Polya. *How to solve it: A new aspect of mathematical method.* Princeton University Press, 2008.

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

R. Ruiz and B. Naderi. Variable neighborhood descent methods for the distributed permutation flowshop problem. In *Multidisciplinary International Conference on Scheduling: Theory and Applications*, 2009.

S. Salhi. Defining tabu list size and aspiration criterion within tabu search methods. *Computers & Operations Research*, 29(1):67–86, 2002.

E. Sandgren. Nonlinear integer and discrete programming in mechanical design optimization. *Journal of Mechanical Design*, 112:223, 1990.

P. Selvidge. How long is too long to wait for a website to load. *Usability news*, 1(2), 1999.

G. Sen, M. Krishnamoorthy, N. Rangaraj, and V. Narayanan. Exact approaches for static data segment allocation problem in an information network. *Computers & Operations Research*, 2014.

H.D. Sherali and C.H. Tuncbilek. A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2(1):101–112, 1992.

K. Sörensen. Metaheuristicsthe metaphor exposed. *International Transactions in Operational Research*, 2013.

J. Sun, S. Gao, W. Yang, and Z. Jiang. Heuristic replica placement algorithms in content distribution networks. *Journal of networks*, 6(3):416–423, 2011.

H.A. Taha. A balasian-based algorithm for zero-one polynomial programming. *Management Science*, 18(6):B–328, 1972.

M. Tawarmalani, K. Kannan, and P. De. Allocating Objects in a Network of Caches: Centralized and Decentralized Analyses. *Management Science*, 55(1):132–147, 2009.

H. Thomas and D. VanderMeer. World Wide Wait: a study of Internet scalability and cache-based approaches to alleviate it. *Management Science*, pages 1425–1444, 2003.

C.A. Tovey. Tutorial on computational complexity. *Interfaces*, pages 30–61, 2002.

L.J. Watters. Reduction of integer polynomial programming problems to zero-one linear programming problems. *Operations Research*, 15(6):1171–1174, 1967.

G.J. Woeginger. Monge strikes again: optimal placement of web proxies in the internet. *Operations Research Letters*, 27(3):93–96, 2000.

S. Wong. Web: how web cache proxies do and don't save on internet costs. *Sys Admin*, 11(2):34–38, 2002.

J. Xu, B. Li, and D.L. Lee. Placement problems for transparent data replication proxy services. *IEEE Journal on Selected Areas in Communications*, 20(7):1383–1398, 2002.

Z. Xuanping, W. Weidong, T. Xiaopeng, and Z. Yonghu. Data Replication at Web Proxies in Content Distribution Network, volume 2642 of Lecture Notes in Computer Science, 2003.

M. Yang and Z. Fei. A model for replica placement in content distribution networks for multimedia applications. In *Proceedings of the IEEE International Conference on Communications*, 2003.

Zona. Zona market bulletin, April 2001. URL `http://www.keynote.com/downloads/Zona_Need_For_Speed.pdf`.