DISCRETE LOCATION MODELS FOR CONTENT DISTRIBUTION

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Tolga Bektaş
September, 2005

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assoc. Prof. Dr. Osman Oğuz (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. İmdat Kara

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Erhan Erkut

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assoc. Prof. Dr. Oya Ekin Karaşan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Asst. Prof. Dr. Osman Alp

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

DISCRETE LOCATION MODELS FOR CONTENT DISTRIBUTION

Tolga Bektaş
Ph.D. in Industrial Engineering
Supervisor: Assoc. Prof. Dr. Osman Oğuz
September, 2005

The advances in information and computer technology has tremendously eased the way to reach electronic information. This, however, also brought forth many problems regarding the distribution of electronic content. This is especially true in the Internet, where there is a phenomenal growth of demand for any kind of electronic information, placing a high burden on the underlying infrastructure. In this dissertation, we study problems arising in distribution of electronic content.

The first problem studied here is related to Content Distribution Networks (CDNs), which have emerged as a new technology to overcome the problems arising on the Internet due to the fast growth of the web-related traffic, such as slow response times and heavy server loads. They aim at increasing the effectiveness of the network by locating identical or partial copies of the origin server(s) throughout the network, which are referred to as proxy servers. In order for such structures to run efficiently, the CDN must be designed such that system resource are properly managed. To this purpose, we develop integer programming models for the problem of designing CDNs and investigate exact and heuristic algorithms for their solution.

The second problem considered in this dissertation is Video Placement and Routing, which is related to the so-called Video-on-Demand (VoD) services. Such services are used to deliver programs to the users on request and find many applications in education, entertainment and business. Although bearing similarities with the CDN phenomena, VoD services have special characteristics with respect to the structure of the network and the type of content distributed. We study the problem of Video Placement and Routing for such networks and offer an optimization based solution algorithm for the associated integer programming model.

The third problem studied here is the problem of allocating databases in distributed computing systems. In this context, we specifically focus on the well-known multidimensional Knapsack Problem (mKP). The mKP arises as a subproblem in solving the database location problem. We concentrate on the well known cover inequalities that are known to be important for the solution of the mKP. We then propose a novel separation procedure to identify violated cover inequalities and utilize this procedure in a branch-and-cut framework devised for the solution of the mKP.

Keywords: Content Distribution Networks, Video on Demand, Multidimensional Knapsack Problem, Integer Programming, Optimization.

ÖZET

İÇERİK DAĞITIMI İÇİN AYRIK YERSEÇİMİ MODELLERİ

Tolga Bektaş
Endüstri Mühendisliği, Doktora
Tez Yöneticisi: Doç. Dr. Osman Oğuz
Eylül, 2005

Bilgi ve bilgisayar teknolojisindeki ilerlemeler, elektronik bilgiye erişimi oldukça kolaylaştırmıştır. Ancak bu gelişmeler, elektronik bilginin dağıtımı ile ilgili bir çok problemi de beraberinde getirmiştir. Bu durum, özellikle her türlü elektronik bilgiye karşı olağanüstü artan bir talebin bulunduğu İnternet ortamı için geçerli olup, mevcut altyapı üzerine oldukça ağır bir yük getirmektedir. Bu tezde, elektronik içerik dağıtımında ortaya çıkan problemler incelenmiştir.

Dikkate alınan ilk problem, ağ baglantılı trafiğin hızlı bir şekilde artması sonucunda İnternet'te ortaya çıkan uzun yanıt süreleri ve sunucular üzerindeki ağır yükler gibi problemlerin üstesinden gelebilmek için geliştirilen yeni bir teknoloji olan İçerik Dağıtım Ağları (İDA)'nın tasarlanması ile ilişkilidir. Bu ağlar, *proxy* sunucuları olarak adlandırılan ve ana sunucuların tam ya da kısmi kopyaları olan ek sunucuları ağ üzerine yerleştirerek, ağın etkinliğini artırmayı hedeflemektedirler. Bu tür yapıların etkin bir şekilde çalısması için, İDA'nin mevcut sistem kaynaklarını doğru kullanacak şekilde tasarlanması gerekmektedir. Bu amaçla, çalışmada İDA tasarımı problemine yönelik tamsayılı programlama modelleri geliştirilmiş ve problemin çözümü için kesin ve yaklaşık çözüm yöntemleri geliştirilmiştir.

Tezde incelenen ikinci problem, İsteğe Bağlı Video (İBV) servislerinde ortaya çıkan Video Yerleştirme ve Yönlendirme Problemi (VYYP)'dir. İBV servisi, kullanıcılarının istekleri doğrultusunda onlara istenilen programları ulaştırma amacıyla geliştirilen ve eğitim ve iş dünyasında bir çok uygulamaya sahip olan bir uygulamadır. İDA ile benzer yönleri olmasına rağmen, İBV servisleri, üzerine kurulduklari ağ ve dağıtılan içerik açısından farklılık göstermektedirler. Bu çalışmada VYYP incelenerek, probleme ilişkin bir tamsayılı doğrusal karar modelinin çözümü için eniyilemeye dayalı bir yöntem önerilmiştir.

İncelenen üçüncü problem ise bilgisayar ağlarında veri tabanlarinin yerleştirilmesi problemidir. Bu kapsamda, sözkönusu problemin çözümünde bir altproblem olarak ortaya çıkan Çok Boyutlu Sırtçantası Problemi (cSP) üzerinde durulmuştur. Çalışmada, cSP'nin çözümünde önemli bir yeri olan örtü eşitsizlikleri incelenerek, ihlal edilen örtü eşitsizliklerinin bulunması için yeni bir yöntem önerilmiş, sözkonusu yöntem cSP'nin çözümü için geliştirilen bir dal ve kes algoritması çatısı altında kullanılmıştır.

Anahtar sözcükler: İçerik Dağıtım Ağları, İsteğe Bağlı Video, Çok Boyutlu Sırtçantası Problemi, Tamsayılı Programlama, Eniyileme.

Acknowledgement

I would like to express my sincere gratitude to a number of people, for which this doctoral dissertation would not have been possible without their support and encouragement.

This dissertation was supervised by Dr. Osman Oğuz. I would like to sincerely thank him for his unrestricted support and for allowing me the appropriate amount of freedom in following my own research ideas during my PhD studies. I am also profoundly grateful to him for generously sharing his experience and own research ideas with me, from which both myself and this dissertation indeed profitted a lot.

My sincere thanks also goes to Dr. İmdat Kara of Başkent University, who has been constantly supporting and guiding me since my MSc studies. I greatly appreciate his advices on professional issues during the time I've been at Başkent University. I would also like to thank him for accepting to be a member of the doctoral committee.

Parts of this work owe much to Dr. Iradj Ouveysi of University of Melbourne. I would like to thank him for graciously suggesting the two topics that are considered in this dissertation. Despite the very long distance, his support and encouragement, as well as our useful discussions on the technical issues, helped greatly in completing the dissertation.

I am deeply indebted to Dr. Erhan Erkut of Bilkent University for devoting his time to read an early draft of this dissertation and offering his valuable comments and suggestions. I would also like to thank him for accepting to be a member of the doctoral committee. I am grateful to Kürşad Asdemir of University of Alberta, and Paul Boustead of University of Wollongong, for their detailed constructive feedbacks on a part of this dissertation that has led to many improvements.

Further, I wish to thank Dr. Oya Karaşan and Dr. Osman Alp, both of

Bilkent University, for accepting to be a member of the doctoral committee and also providing constructive comments.

I would like to take this opportunity to express my gratitude to Dr. Berna Dengiz of Başkent University, for her encouragement and support in starting my academic career. She has helped me in taking the first footsteps.

My colleagues at Başkent University deserve many thanks. In specific, I would like to thank Dr. Ergün Eraslan for being such a nice officemate and understanding, M. Oya Çınar for her constant support and encouragement, Onur Özkök for his effort in answering all my endless questions, which in fact resulted in very useful discussions, and F. Buğra Çamlıca for his support in difficult times. I am also grateful to Halit Ergezer, Hasan Oğul and Güven Köse, for kindly providing support in many computer programming issues. Additional thanks are due to folks at Bilkent University, and in specific to Banu Yüksel, Aysegül Altın, Sibel Alumur.

I would like to express my sincere gratitude to my mother, my father and my brother, Burçin. I would not have been where I am now if it were not for their endless support, great care and everlasting love.

This Dissertation is Dedicated To

My Family

and

to the Memory of N.D., for She Will Always Be My Guarding Angel...

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Content Distribution

Efficient storage and distribution of any type of goods nowadays is of critical importance to both organizations and consumers, mainly due to a highly competitive market environment and ever-increasing number and variety of products. Moreover, it is not only the cost of the storage and distribution that matters, but distributing goods so as to ensure a predefined service quality level is also of crucial importance. Logistics, having a broad scope and composed of many such interrelated activities, has therefore become a very complex issue to deal with for many companies.

The concept of distribution is usually related with the transportation of physical goods from the production plant to several physical demand points. Consequently, a vast amount of research is devoted to what is called the *Supply Chain Management* and *Logistics Planning*, which deal with all activities that the goods are subjected to, ranging from the initial production to the consumer delivery. However, there is one other major field in which distribution is very important and deserves at least the same attention: *computer communication networks*.

The main characteristic of the information age we live in today is the need for the availability of any kind of information, anywhere and anytime. This is

primarily the case when we are talking about electronic information (or content) that is distributed via communication networks and computers. Parallel to the problems arising in the domain of logistics, companies are already facing difficulties in storing and distributing electronic information so as to make it available to the consumers of electronic content. This is especially true for the World Wide Web, which has experienced an explosive growth in the past decade. An obvious consequence of this high rate of usage is the enormous share of Internet traffic, which gives rise to problems such as web access delay, increasing loads on the server(s) and network congestion. Similar problems also arise in smaller networks, such as corporate intranets or Video-on-Demand systems. It is often the case that in such electronic distribution systems the delays experienced in the delivery time grow with the increasing amount of traffic in the network.

As the size of the content delivered via Internet and the number of users have increased tremendously in recent years, the clients have started to experience unacceptable response times, changing the Internet from the "World Wide Web" to a "World Wide Wait" [31]. In fact, as Saroiu et al. [90] demonstrate in a recent study, the average size of the delivered content has changed from about 2KB to 4MB, which is an increase in the magnitude of thousands. Consequently, the huge amount of traffic generated by the distribution of the content has made the Internet unable to efficiently support this growth, giving rise to increased response times. The delays experienced by the end users have consequences, particularly economic, from the perspective of content providers. As Zona Research reports [7], "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". A widely appreciated standard is that a typical client will abandon a Web site which fails to download in less than eight seconds. According to Zona Research, about $4.35 billion may have been lost in online sales in 1999 due to unacceptably slow response times. Moreover, the potential losses in 2001 were estimated to be over $25 billion [8]. Hence, distributing electronic content effectively has become a major problem of today.

Motivated by the discussion presented above, the objective of this dissertation is to investigate problems arising in distribution of content. The overall focus

will be on electronic content distribution, but we will nevertheless provide an extension of the interesting ideas that are to be discussed in this context to similar problems arising in logistics planning.

The aim of this chapter is to provide an overview of the research objectives of this dissertation and to provide a general outline. Before doing this, we will first give some insight on Internet and its infrastructure, as the main problems considered in this dissertation are based on these fundamental concepts.

1.1 The Internet Infrastructure

Internet is basically a network of computer networks, providing content to the users. The term *content*, with respect to Internet, refers to any kind of information that is available on the World Wide Web to public such as Web pages, multimedia files and text documents. We will also use the term *object* to refer to a specific item of the content. The term *content provider* refers to a unit, which holds the content for the access of others on its origin server(s). We will denote by the term *client* or *user*, the individuals (either a person or a company) who issue requests for electronic content.

Internet has a hierarchial structure. Briefly put forward, clients are connected to local *Internet Service Providers* (*ISP*s) which provide retail-level Internet access. These local ISPs are connected to *National Backbone Providers* (*NBP*s) via regional ISPs. NBPs are long-haul data networks (such as AT&T in USA) providing wholesale-level Internet access to regional ISPs. Finally, the NBPs are interconnected via either *Network Access Points* (NAPs) or peering points, forming the Internet backbone. For more details on the Internet infrastructure, the reader is referred to Datta et al. [32].

In a typical network application, the client issues a request and sends it to a site, after which the site responds to the client. The request for a content is made by using the *Universal Resource Locator* located in the browser. The browser then issues a query to a *Domain Name System* (*DNS*) to obtain the

Internet Protocol address of the server holding the requested content (named as *DNS lookup*). Based on this IP address, a connection is set up between the client and the corresponding server. The client then issues a *Hyper Text Transport Protocol* (*HTTP*) request, to which the server responds and the requested content is delivered to the client. For details, see Datta et al. [32], [31].

A typical metric used to measure the performance of a network is the *user response time* (or *latency*), which can be defined as the amount of time elapsed between the time of the request issued by the client and the time when the response is received. The best-case of an application is obtained when there is no other traffic in the network, which is also a lower bound on the latency experienced by the client. Hence, the primary goal of managing performance on the network is to design the network such that the client latency is within acceptable limits, which is directly related with the amount of traffic flowing over the World Wide Web.

Unless additional technological schemes are employed, these problems will continue to negatively affect the success of Web sites and their potential sales. One immediate solution seems to be adding new infrastructure, although Datta et al. [31] argue that new infrastructure is expensive and this alternative only shifts bottlenecks to other part of the network rather than eliminating them. Thus, the approach should concentrate on efficient usage of the existing infrastructure. This gives way to a widely used technique, known as caching.

1.1.1 Caching Strategies

A widely adopted technique to overcome the high rate of latency due to the intense Internet-based traffic is *caching*, which is aimed at improving the response time of web servers. Caching can be described as keeping an accessed content in storage centers called *cache*s, to where future accesses to this specific content are made. As Hosanagar et al. [46] point out, “Caches are storage centers - the digital equivalent of warehouses. In this context, the Internet infrastructure makes up the digital supply chain for information goods”. Hence, caching is a

viable strategy for such networking applications to alleviate the Internet traffic.

In general, there are three kinds of caching: Client-Based Caching, Server-Based Caching and Proxy-Based Caching. The first type implements caching at the client side (either in the browser located on the client's computer or at a gateway) and serves only the requests made from this specific location. Since these caches are usually of limited sizes, only a restricted amount of content can be stored. Thus, when a new content needs to be stored, it must replace an existing object in the proxy. This brings the need for a replacement policy to determine which objects should be replaced by the new ones. Client-Based Caching is an approach of limited use, 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 [54].

The second approach, Server-Based Caching, is performed by installing additional caches at various places within the server location. Although this type of caching helps to share the load on the server by distributing it to these side caches, it has a small effect on reducing the incoming traffic to this server.

The third type is usually performed using a web proxy located somewhere between the client site (such as a company or a university proxy) and the origin server. When the 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 further sent to the server and the content will be accessed from here. In the latter case, since this will be the first time the specific content will be accessed, a copy will be stored at the proxy to be used to serve further requests. These proxies are located at different points on the network, so they can serve a large number of clients and are very effective in reducing the network traffic. *Content Distribution Networks*, a new Internet technology, are based upon this approach and explained further in the next section.

1.2 Content Distribution Networks

The main idea of the new emerging technology referred to as *Content Distribution* (or *Delivery*) *Networks* (CDNs) is to replicate the content from the origin server(s) to geographically distributed surrogate sites, from which the client receives the requested content. The aim is to speed up the delivery of Internet content, reduce the load on the original servers, and improve service quality. The place where the replicated content is held is referred to as the *proxy server* (also named as *surrogate server* or *replica*). If the proxy is an exact copy of the origin server, then it is called a *mirror*.

A CDN can significantly improve the performance of a network, since the clients no longer have to be served by the origin server but instead they receive the content from a proxy server located nearby. Another important contribution of the CDN technology is an improvement in application reliability, i.e. the ability of the network to serve a client even when the origin server is down. The basic idea behind the operation of a CDN is depicted in Figure 1.1. As the figure demonstrates, a client retrieves the requested content from one of the proxies that are deployed at the edges of the network, as opposed to retrieving it from the origin server, without the interference of the heavy Internet traffic.

To this date, a number of companies have started to offer commercial hosting services for content distribution such as Akamai [1], Digital Island [2], Mirror Image [4] and Speedera [6]. As Vakali and Pallis [93] report, about 2500 companies are reported to be using CDNs as of December 2003. According to the same study, Akamai [1], for instance, has over 12,000 servers in 62 countries and hosts popular customers such as Apple, CNN, MSNBC, Reuters and Yahoo. Another large scale CDN, Digital Island [2], has about 2,500 surrogate servers spanning 35 countries and hosts popular pages such as AOL, Microsoft and Hewlett Packard. Medium sized CDNs, on the other hand, have smaller number of surrogate servers. Mirror Image [4], for instance, is a CDN with about 22 surrogate servers spanning North America, Europe and Asia. Being another commercial CDN, Inktomi [3] has 10 surrogate servers deployed throughout China.

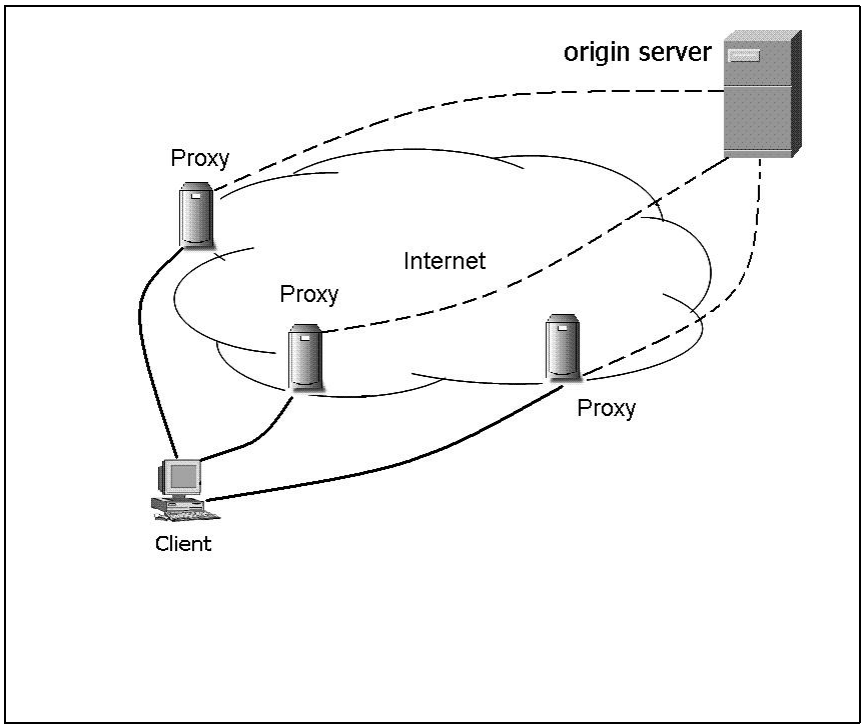

Figure 1.1: A typical CDN architecture

A CDN is composed of the following main components:

1. Multiple Surrogate Sites (where each may consist of single or multiple machines)
2. Client Routing Scheme (used to route the client to one of the surrogate sites instead of the origin site) and the Routing Table (which indicates the assignment of clients to surrogate servers)
3. Cache Management (to distribute the content to the surrogate servers, and to manage cache coherency and consistency among all the sites)
4. Networking Infrastructure (between the surrogate server sites and the origin servers)

For technical details on these components, the reader is referred to the book by Verma [95].

1.2.1 Caching in CDNs

CDNs mainly aim at reducing the load on the origin server and the backbone traffic by deploying a number of surrogate servers across the network, which serve on the origin server's behalf. This is possible through the replication of the content on the surrogate servers. Two types of replications are possible for content distribution. The first type, *full replication*, can be employed when the storage space of storage servers is sufficiently large and the whole content consists of small-sized objects (e.g. web pages, text files). The surrogate servers are then said to serve as *mirror*s of the origin server(s). However, a major problem in such an approach is that a large amount of storage space may be wasted by a fraction of the replicated content for which the requests are very small. This approach may also generate much more traffic on the Internet than it ought to be.

A second type of replication is where only a very selective set of content, based on the request rates, are replicated in the proxies. This situation generally occurs

when the size of the content is large (such as multimedia audio or video files) and the proxy has a limited storage space. Although this approach helps to balance the traffic on the network and improve the storage utilization, it brings up an additional decision problem regarding the objects to be replicated in each proxy.

The effectiveness of a CDN depends heavily on the probability that the proxy server is able to satisfy the client's request. This probability is usually referred to as the *hit ratio*. A high hit ratio results in a higher performance in a network whereas a low hit ratio indicates that a CDN is not likely to be of use [95]. If the proxy server does not have the requested object, then the request is further forwarded to the origin server with the expense of additional latency perceived by the customer. In some cases, this additional latency is much more than the case when the client has requested the object directly from the origin server. Hence, the assignment of each client to a suitable proxy server and the set of objects that are to be held in each proxy are issues that should be properly decided on.

1.2.2 Application Contexts

There are some applications where a CDN approach may be readily applied and the network is expected to gain a high improvement in performance. One application is where the network has a large amount of static data, i.e. kind of data which remains the same over a long period of time. These may consist of images, static HTML files or large multimedia files (such as music or video files). Also, traditional applications such as file transfer (FTP) servers or mail servers are suitable for a CDN application.

On the other hand, some applications may be highly unsuitable for a CDN application. These include applications that require frequent updates to data (i.e. dynamic content), simultaneous access from multiple locations or strong security needs (such as credit card/banking applications). Dynamic data may not always be suitable for a CDN application, since in this case frequent updates would be needed and it may become harder to manage the network. However, some dynamic data may be handled by a CDN using special approaches.

There are also several drawbacks of a CDN application. Since a CDN consists of many geographically dispersed servers, it becomes crucial to effectively manage the overall network. This is referred to as *manageability of the CDN*. Another drawback is that some existing applications may need to be rewritten to be suitable for the CDN approach. This is called *redesigning of applications*. For more detail on these issues, the reader is referred to Verma [95].

We would like to note that The World Wide Web is not the only application domain of CDNs. They can also be implemented for corporate intranets or extranets, which are obviously much more smaller networks than the Internet itself.

1.2.3 Problems in Content Distribution Networks

The objective in a content distribution problem may vary depending on the viewpoint of the decision maker, that is, the CDN operator. The CDN operator, providing service, charges some amount to their customers. However, the CDN also pays to the backbone network on which it operates to disseminate the content over this network to its clients and the amount of payment is a function of the traffic flowing on the network. Thus, it is important for a CDN to have as low traffic as possible in order to reduce its expenses. Besides, the more the traffic is reduced, the less the CDN charges to its customers and the less the delay the clients experience as a result of using the CDN. Hence, such an aim from the perspective of the CDN helps to improve the *Quality of Service* (QoS) offered to its customers and to better compete with other commercial CDNs. If this is the case, then the CDN is likely to have an objective function of a minimization type, where the function to be minimized is the total cost of the network (proportional to the total traffic flowing on the network). Another scenario may be of concern when the CDN wants to maximize its total revenue. In this case, the CDN needs to determine the optimal *pricing policy*, i.e. finding the price to charge to clients for serving an object. An objective function for this situation is given by Datta et al. [31], which is based on the (known) demand for each object.

Operating a CDN in an efficient manner requires a proper management of system resources. This issue, in general, excludes considerations such as installation of new links and allocation of additional link capacities, since a network is already in place and it is not in general feasible to expand the existing infrastructure of the Internet. Thus, efforts should concentrate on the efficient usage of the existing configuration rather than expanding it.

In operating a CDN, a service provider is usually faced with three important problems in resource managment. The first of these problems is concerned with the optimal placement of proxies and called the *replica server or proxy server placement problem* (also referred to as the *mirror placement* or *cache location problem*). More specifically, given an existing infrastructure, this problem consists of optimally placing a given number of proxy servers to a given number of potential sites, such that a cost function (overall flow of traffic, average delay the clients experience, total delivery cost, etc.) is minimized. The second problem is related to the placement of objects in proxy servers and called the *object replication or data/replica placement problem*. In other words, given the whole content to be distributed, the service provider must decide on the specific items of the content to be held in each proxy server. Finally, the third problem, referred to as the *request routing problem*, consists of guiding the clients' requests for a specific item of the content to suitable proxies that are able to address the corresponding requests so as the minimize the cost of serving.

1.3 Video-on-Demand Services

The advances in high-speed networking and multimedia technologies have made it possible to develop many applications, including the popular Video-on-Demand (VoD) service. A VoD service is a special type of electronic content distribution service in that it deals specifically with distribution of videos (e.g. movies) to a number of geographically distributed users. In other words, a VoD service can be described as a virtual video rental store in which a user has the option to choose

and watch any program on request, at the convenience of their time. Interactive VoD services offer the user a fine-grained control, enabling them to pause, resume, fast rewind and fast-forward the video. Applications of such services are not limited to home entertainment and can be extended to banking applications, education and home shopping. Quoting from Ghose and Kim [42], "the combination of Internet and VoD may very well be the basis for entertainment, business, and education of the future." An in-depth treatment of the subject is given by Little and Venkatesh [70].

A VoD can be regarded as a special CDN where significantly large amounts of data (multimedia files) are to be distributed and hence bandwidth and server capacities pose tight constraints. However, the characteristics of the Internet are not appropriate for these services. These services require special networks that are capable of supporting such high-bandwidth applications (such as cable networks).

1.3.1 System Architecture and Components

A complete VoD system consists of three fundamental components which may be stated as the storage servers, network on which the system is built and the user interface. The network architecture in general has a hierarchical structure (see e.g. [80], [19]). An example VoD architecture is depicted in Figure 1.2.

In such a network, there exists a central server, which can be considered as the main storage unit holding all programs. Connected to the central server, there are groups consisting of local video servers. Each group is a fully meshed network, i.e., units in the group are all connected to each other. Each user is connected to a local server, although users can watch programs transparently from other local servers in the group. However, this incurs an additional cost.

In a VoD system, users typically interact with the system using an interface, such as a remote control or a keyboard. The requests made by the user through the interface is forwarded to the network. Once the user request for a program is

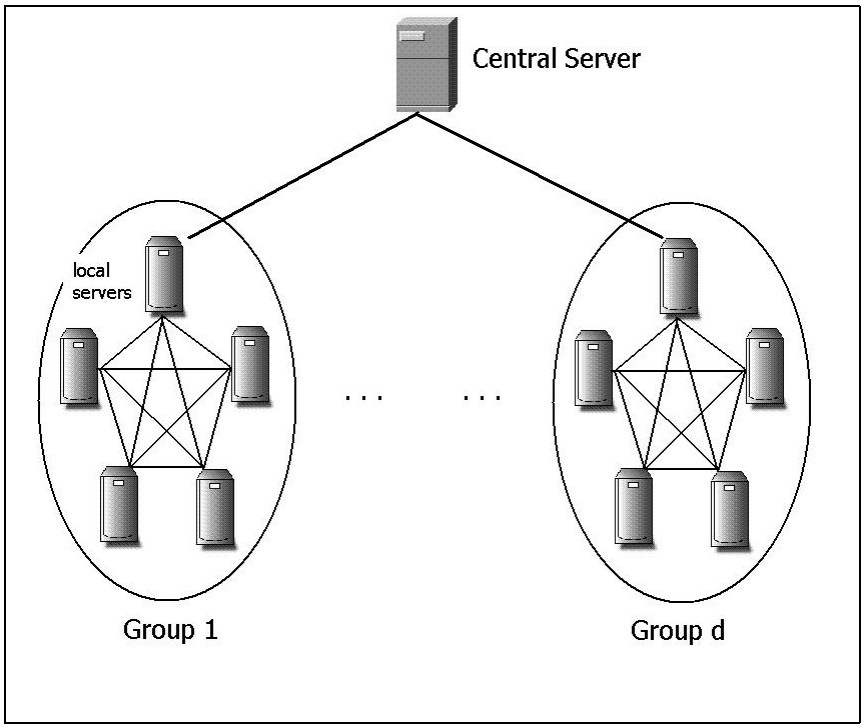

Figure 1.2: A typical VoD architecture

fetched from an available resource, it is served to the user.

A VoD provider may either choose to offer its services as *data-centered* or *user-centered* [42]. The former is in general called *broadcasting*, where the provider broadcasts the programs through a single channel in specific time periods and the user has to tune into the channel to receive the program. In this case, the user is a passive participant to the system and has no control over the program. In contrast, the user-centered approach specifically dedicates a channel and bandwidth to the user through which the program is immediately transmitted on request. Moreover, the user has complete control over the session. While the former approach requires less system resources and is less expensive, the latter has a higher quality. There are also hybrid approaches, such as *batching*, where the provider collects similar user requests in specific time intervals, which are then served using a single video stream. In this case, the user has to wait after issuing the request and does not have a control over the program. For more details on this topic, we refer the reader to the survey by Ghose and Kim [42].

1.3.2 Problems in Video-on-Demand Services

Movies in a VoD system are held in a repositories of huge sizes, called video servers, which are located throughout the network. A VoD system will typically be transmitting an enormous amount of data through its network every day. This would have significant consequences to the service provider in terms of the total cost. This is especially true for the groups of local servers, since most of the transmissions are expected to take place within these groups. Therefore, similar to the CDN case, it is very important for the service provider that the system resources are properly managed in order to minimize the total cost of providing the service and maintain an efficient distribution structure. According to Little and Venkatesh [70], this issue is directly related to what they refer to as *load balancing*. Proper load balancing in such a system can be performed through effectively allocating programs to the available servers and establishing the suitable connections between each potential user and program.

1.4 Research Objectives

As also mentioned in the beginning of this chapter, the main objective of this dissertation is to investigate problems arising in electronic content distribution. In particular, we focus on defining problems and proposing solution approaches in Content Distribution Networks and Video on Demand Systems. We state below the research objectives:

- **To identify and investigate problems in electronic content distribution**: Content distribution in telecommunications networks, being very complex structures as they are, pose many optimization problems to be solved in order to operate efficiently. These problems are generally interdependent, i.e., they are in general required to be solved simultaneously. In this dissertation, we attempt on identifying and defining problems in this avenue of research, with an emphasis on treating several subproblems jointly. This deviates from existing research, where it is generally the case that each study tends to consider one problem at a time, assuming that the others are already solved. In specific, we focus on the CDN environment and define a problem which we will hereafter refer to as the *Content Distribution Network Design Problem*. We then turn to investigate a special case of content distribution, namely the *Video Placement and Routing Problem* and investigate load balancing issues. Finally, we look into the problem of *Allocating Databases Distributed Computing Systems*.

- **To develop novel algorithmic approaches for the solution of problems in electronic content distribution**: As already indicated above, there exist many optimization problems in electronic content distribution, where their solution requires a number of subproblems to be solved simultaneously. This, in turn, renders such problems complex and hard to solve. It is generally the case that such problems are tackled using heuristic solution approaches in the literature. In this dissertation, in contrast to the existing literature, we focus on developing exact solution techniques utilizing combinatorial optimization and mathematical programming methods.

The benefits of such techniques are twofold. The first is that they will be useful in their own right, i.e., solving to optimality problems of electronic distribution. The second benefit lies in the use of these procedures to assist in evaluating the solution quality of many heuristic approaches proposed in the literature.

1.5 Outline of the Dissertation

The outline of this dissertation is provided below. We would like to state that the main contributions of this research are reported in Chapters 3, 4, and 5. Chapter 6 is composed of main results and conclusions.

- **Chapter 2** attempts to provide a literature review on the problems related to those studied in this dissertation. The main body is subdivided into two sections, consisting of a review on CDNs and VoD systems, respectively. Several formulations are provided where necessary. This chapter also includes a section where we attempt to provide the reader some insight on how the commercial CDNs operate.

- In **Chapter 3**, we formally define the problem of designing a CDN. Our design proposal consists of jointly deciding on (i) the number and placement of proxy servers on a given set of potential nodes, (ii) replicating content on the proxy servers, and (iii) routing the requests for the content to a suitable proxy server such that the total cost of distribution is minimized. We first provide two nonlinear integer programming formulations for the problem, for single and multiple server situations, respectively. We then offer a linearization for the first model. Based on the linearization, we develop an exact solution procedure based on Benders' decomposition and also utilize a variant of this procedure to accelerate the algorithm. In addition, we provide a fast and efficient heuristic that can be used to obtain near-optimal solutions to the problem. The chapter concludes with computational results

showing the performance of the decomposition procedure and the heuristic algorithm on randomly generated Internet topologies. In this chapter, to investigate whether the approach proposed here is beneficial or not, we compare the proposed joint approach for a CDN design to a two-stage approach that is inspired from practice. Through computational experiments, we investigate the potential benefits of using a joint approach.

- **Chapter 4** is related to the problem of video placement and routing in VoD systems. More specifically, we look into the problem of load balancing, which can be achieved through proper resource allocation and connection establishment. Although many heuristics are available for similar problems, not many exact solution procedures exist, mainly due to the complexity of the problem. In addition, such heuristics are incapable of indicating the quality of the solutions found. We devise a solution algorithm that is based on Lagrangean relaxation and decomposition algorithm for the problem of load balancing. Since a VoD system is partly dynamic in nature, it may call for a repeated solving of the problem in periodic and short time units. Taking such a situation into account, we propose a variant of the algorithm that is capable of producing good quality solutions in relatively short solution times. The chapter concludes with computational results demonstrating the efficiency of the proposed algorithm.

- **Chapter 5** deals with the problem of allocating databases in distributed computing systems. In this context, we specifically focus on the well-known multidimensional Knapsack Problem (mKP), which arises as a subproblem in the solution process of the former problem. We investigate a class of valid inequalities, namely cover inequalities, that are known to be very popular and important in the exactly solving the mKP. We propose a new separation procedure for these inequalities and implement the procedure in both a cutting plane and a branch-and-cut framework to demonstrate its efficiency. Computational results on both randomly generated and well-known literature problems are reported in the chapter.

- Summary of research findings, main results and issues for further research are stated in **Chapter 6**.

Chapter 2

Literature Review

Although being a relatively new topic for research, there is already a significant amount of research on CDNs. VoD services, on the other hand, is a topic that has been studied for more than 10 years. Therefore, there is a vast amount of literature on electronic content distribution. However, research on these topics are generally by the computer science community. The OR/MS community has only recently started expressing interest in this fruitful area of research.

This chapter, which aims at providing a review of the existing literature on the topic, will focus only on research relevant to the OR community. In specific, we will include in this review a subset of the existing research that utilize OR/MS approaches in solving the problems. As will shortly be shown, these approaches range from integer programming to nonlinear programming and from game theory to queuing models.

This chapter is composed of two main sections. The first section provides a review of the previous literature on Content Distribution Networks with respect to varying problems, and also includes an additional subsection that provides some insight on how the commercial CDNs operate. The second section is a literature review on the research relating to VoD Services. We also provide discussions on the existing research on these problems.

2.1 Content Distribution Networks

This section provides a literature review on the existing research for CDNs. As previously mentioned, three main issues that should be dealt with in a CDN design are the optimal placement of proxies (referred to as the *replica server or proxy server placement problem*, *mirror placement* or *cache location problem*), the placement of objects in proxy servers (referred to as the *object replication or data/replica placement problem*) and routing the requests of clients to a proxy (*request routing problem*). Additional issues such as *pricing* in CDNs also exist. The existing literature, in general, tends to investigate these problems independently and these are described in the relevant subsections. There also exist a number of publications that consider some of these problems jointly. The following review will be subdivided accordingly.

2.1.1 Proxy Server Placement

Given an existing infrastructure, the proxy server placement problem consists of optimally placing a given number of proxies to a given number of sites, such that a cost function (overall flow of traffic, average delay the clients experience, total delivery cost, etc.) is minimized.

The first study that we are aware of on the placement problem of proxy servers is due to Li et al. [68], who propose a dynamic programming approach to solve the problem with an assumption that the underlying network is a tree. However, as noted in recent studies (see for example [87]), the Internet topology is rarely a tree. In addition, the high computational complexity of the algorithm ($\mathcal{O}(n^3m^2)$ for choosing m proxy servers among n potential nodes) makes the approach highly inefficient since practical Internet topologies that have nodes in order of thousands.

Later, Woeginger [97] suggested an algorithm to optimally place m web proxies on a linear network topology with n nodes. By observing that the underlying cost function of such a problem has a Monge structure, his algorithm runs faster

than that proposed by Li et al. [68], with a reduced complexity of $\mathcal{O}(nm)$. A matrix $C = (c_{ij})$ is said to have a Monge structure if $c_{ij} + c_{rs} \leq c_{is} + c_{rj}$ for all $1 \leq i < r \leq m$ and $1 \leq j < s \leq n$.

Qiu et al. [87] investigated the proxy server placement problem with a single origin server, with an objective to minimize the total traffic load generated by the clients (i.e. bandwidth consumption) and the clients are directed to only a single replica. Their study seems to be the first to formally relate two mathematical models to the problem, which are the well known uncapacitated *p-median* and *facility location* problems (will henceforth be denoted by UPMP and UFLP, respectively). We will briefly review these two problems and the associated models below.

The UPMP is one of the first problems studied in location analysis. The reader is referred to Mirchandani [72] for an introduction to the problem and its generalizations. Among the vast amount of existing research on the UPMP, we also refer the reader the one by Beasley [17] and a recent work by Senne [91]. This problem consists of optimally locating at most (or exactly) p centers to a number of predefined sites and to assign each non-center site to a center so as to minimize the total assignment cost. In the CDN vocabulary, centers correspond to proxy servers and non-center sites correspond to clients.

The following decision variables are used to model the problem:

$$y_j = \begin{cases} 1, & \text{if node } j \in J \text{ is selected as a proxy server} \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if client } i \in I \text{ is assigned to proxy server } j \in J \\ 0, & \text{otherwise} \end{cases}$$

Although this is a well-studied problem, we provide the following p-median model for the sake of completeness:

$$\text{minimize} \sum_{i \in I} \sum_{j \in J} d_i c_{ij} x_{ij} \tag{2.1}$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \qquad \forall i \in I \tag{2.2}$$

$$x_{ij} \leq y_j, \quad \forall i \in I, j \in J \tag{2.3}$$

$$\sum_{j \in J} y_j \leq p \tag{2.4}$$

$$y_j \in \{0, 1\}, \qquad \forall j \in J \tag{2.5}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, j \in J \tag{2.6}$$

In this formulation, the objective (2.1) is to minimize the total cost of serving each client. Constraints (2.2) ensure that each client is assigned to a single proxy whereas constraints (2.3) ensure that a proxy must be installed at point j in order to be able to serve a client. Constraint (2.4) is only an upper bound on the number of proxies that can be opened. Note here that this formulation assumes the number of proxies that are to be installed is fixed a priori. If this is not the case and the number of installed proxies are to be minimized as well, then the objective function may be augmented by the expression $\sum_{j \in J} f_j y_j$, where f_j is the installation cost of a proxy at point j. This case is exactly the *uncapacitated facility location problem* (*UFLP*) (see [28]).

The UFLP consists of choosing a subset of facilities in a given network among a potential set, such as plants or warehouses, to minimize the total cost of satisfying all the demands of the customers for a specific commodity. Based on the previously given definitions in the previous section, we provide the integer programming formulation of UFLP as follows:

$$\text{minimize} \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} d_i c_{ij} x_{ij} \tag{2.7}$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \qquad \forall i \in I \tag{2.8}$$

$$x_{ij} \leq y_j, \quad \forall i \in I, j \in J \tag{2.9}$$

$$y_j \in \{0, 1\}, \qquad \forall j \in J \tag{2.10}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, j \in J \tag{2.11}$$

The constraints of this formulation are similar to that of UPMP presented previously. The primary difference between the UPMP and the UFLP is that the latter has a fixed cost associated with opening a facility, whereas the former only imposes an upper bound on the number of facilities opened. Similar the the UPMP, the UFLP is an extensively studied problem in the context of locational analysis. For an excellent introduction and review, the reader is referred to Cornuejols et al. [28].

Qiu et al. [87] consider the p-median to model the proxy server placement problem. It should be noted that both the p-median and the facility location models assume that the entire content is stored in each installed proxy and thus only take into account the traffic flowing between the server and the clients. The authors propose several approaches for the solution, consisting of a tree-based algorithm, greedy algorithm, random algorithm, hot spot algorithm and an exact algorithm. The first four algorithms are heuristics, whereas the last algorithm is based on relaxing constraints (2.2) in a Lagrangean fashion and solve the resulting model using subgradient optimization. The basic idea behind the greedy algorithm is very simple, it evaluates a single potential location at every iteration and the location that yields the lowest cost in conjunction with the proxies already installed in previous iterations is selected as a proxy. The cost calculation is based under the assumption that each client receives content from the closest proxy. The algorithm stops when p proxies are chosen. Based on

randomly generated and real Internet topologies, the authors' results show that the greedy algorithm's performance is better than that of the others.

Jamin et al. [50] investigated the problem of web proxy placement where the goal is to minimize the maximum distance between any client and a proxy. This is similar to the well-known p-center problem. They make use of algorithms devised for the k-hierarchically well-separated trees (k-HST) and p-center problems to determine the number and the placement of network instrumentation. Later on, Cronin et al. [29] investigated specifically the mirror placement on the Internet with a small number of mirrors, where the placement is restricted to a given number of potential locations. They show that increasing the number of mirrors only for a small range of values is effective in reducing client latency and server load, regardless of the placement algorithm used.

Radoslavov et al. [88] consider the problem of placing a given number of proxy servers on a given network topology so as to minimize the average client latency and the overall network overhead. They argue that the greedy algorithm described above requires one to have a detailed knowledge of the network, including the client locations and all pairwise inter-node distances, which may not always be possible. In constrast, they investigate several heuristics for the problem which do not assume that a detailed network information is at hand, and are only based on node degrees. They assume that each client is assigned to the closest proxy.

The proxy server placement problem, which takes into account a hierarchial structure of the Internet as well as the routing policy constraints was considered by Bassali et al. [16]. The authors assume that all proxies to be replaced are identical and have an infinite amount of storage space. Several heuristic procedures are proposed along with their evaluation with simulations over real Internet topologies.

Yang and Fei [102] consider the problem of proxy server placement in multimedia applications and argue that the simplifying assumptions (such as the infinite storage space) made in previous studies may not always be valid in the case of multimedia objects (such as video, audio files). They stress on the fact

that the storage capacity of each proxy server is limited. More importantly, the distribution of objects from the origin server(s) to the proxy servers are as important as the distribution of objects from the proxy servers to the clients and should not be ignored. The problem is solved via a modification of previously proposed heuristics, namely the greedy [87], hot spot [87] and max fanout [88] heuristics.

The problem of cache placement on transparent caches was studied by Krishnan et al. [63]. The objective function considered in this study is interesting in that it considers the case where the requested content is *not* found in a specific proxy. Thus, the cost of serving client i from a server s is given by the following:

$$cost(i, s) = f_{is}(h_{is}d_{ij} + (1 - h_{is})(d_{ij} + d_{js})) \tag{2.12}$$

where f_{is} is the flow from server s to client i, and d_{ij} (resp. d_{js}) is the unit cost of sending traffic from node i to node j (resp. from node j to the server node). Then, the optimization problem is of the form $\sum_{i,s} min_{j \in J \cup S} cost(i, s)$, i.e., choosing the minimum number of locations to install proxies such that the total cost function is minimized. The authors provide optimal algorithms for line and ring networks and a dynamic programming algorithm for the single server case.

A recent study by Jia et al. [51] investigates the placement of transparent en-route proxies and also considers read and update operations of the data on the web server. The cost function they consider is similar to (2.12). The authors consider two cases of the problem, where the first consists of optimally placing a given number of proxies and the second involves finding the optimal number and placement of an unconstrained number of proxies. The problems are solved through dynamic programming and optimal solutions are obtained for randomly generated Internet topologies in polynomial time (in $\mathcal{O}(n^3k^2)$ time for the former and $\mathcal{O}(n^3)$ time for the latter).

Choi and Shavitt [24] propose a well-known model for optimally locating transparent servers, namely the *set cover problem*. The aim is to determine the minimum number of servers such that at least one server should be located along each router from clients to servers, i.e. all the demand requests should be covered by the set of chosen servers. They propose four heuristics for the problem and investigate the performance of each through simulation.

2.1.1.1 Discussion

Most of the previously published studies on the placement of proxy servers or web proxies in the context of CDNs utilize some well-known discrete location problems. However, these approaches usually make use of some underlying assumptions which may not be valid for all kinds of content distributed on the Internet. For instance, the UPMP model is used for the replica placement problem in some studies, where centers correspond to proxy servers and demand points correspond to clients (see, for example, [87]). However, this formulation may not always be adequate to correctly model the problem. The main assumption in UPMP is that each center has enough capacity to serve all the demand points, whereas in CDNs, the proxy servers may have restricted capacities. This situation may arise in the case of multimedia content, which indeed are huge in size and require a fair amount of space. In addition, this formulation assumes that the client is served from the proxy server only and does not take into account the cost of distributing objects from the origin server(s) to the proxy servers. This may especially be important when there are multiple origin servers and the cost of distributing content from origin servers to the proxies is significant. Another limitation is that the formulation does not yield the optimum number of centers, instead, only imposes an upper bound. However, one can not in general know a priori the number of proxies that will lead to the optimal performance, hence the optimal number of proxies should be determined as well. Another drawback is that this formulation does not decide on which objects should be placed in a proxy server.

The UFLP/CFLP model can also be used for the proxy server location problem, as mentioned by some studies (see [87] and [102]). This formulation is capable of determining the optimal number of proxy servers. Still, this model ignores some important decision problems, such as deciding on the distribution of objects from origin server(s) to the proxy servers and the type of objects that should be located in each proxy server.

In short, the proxy server location problem arising in CDNs is similar to well-known discrete facility location problems such as the UPMP and CFLP. However, the corresponding models should be extended so as to be able to adequately incorporate all the issues that should be addressed for a CDN design. These extensions may be stated as follows:

1. The typical cost in classical facility location problems is a fixed parameter defined for each node pair in the network. For a CDN, this cost is typically a function of the size and the request rate of data transferred between two nodes.

2. A typical CDN includes many objects to be distributed throughout the network. Consequently, the problems arising here are multi-commodity extensions of the classical facility location problems, which typically consider a single commodity.

3. In classical facility location models, the amount of commodity stored at each facility is important in that there should be enough supply to serve the requested demand. However, in the case of web objects, it is enough to locate only one unit of the object in the proxy server so as to satisfy any demand for this object required from this server. The amount of demand of the client only effects the cost incurred for transferring the data. This special feature is due to the characteristics of the Internet environment, which is clearly not the case in the facility location problem.

2.1.2 Object Replication

Previously mentioned studies assume that all the content held in the origin server has been replicated to the proxy servers, i.e. the proxy servers hold the entire content of the origin server. This may not always be the case where the objects are significantly large (for example in the case of multimedia files) and only a partial replication can be performed, since the proxy servers have finite capacity. In this case, any proxy server can only hold a subset of the content and these should be properly identified. This problem is referred to as the *Object Replication or Data/Replica placement problem.*

The object replication problem is in some ways similar to the *File Allocation Problem* (FAP) arising in distributed computer systems. More specifically, given a network with a number of computers installed at known locations and a number of files to be distributed over the network, the FAP is concerned with determining the number of files to be distributed and the specific location of each file copy. The problem also involves deciding on the allocation of each user to a specific computer from which the user's request is served.

One of the first studies dealing with the FAP is perhaps due to Chu [26]. Another study is due to Fisher and Hochbaum [35]. The problem considered in this study is to place additional copies of a database throughout a computer network with regards to the trade-off between the cost of accessing the various copies of the database in the network and the cost of storing and updating the additional copies. The authors refer to the problem as the *database location problem* and present an associated mixed-integer model. Pirkul [85] considers the specific problem of database allocation in a distributed computer system, where the database is to be partitioned without duplication. This kind of a problem generally arises in systems such as banking applications. Thus, this problem only involves deciding on the assignment of a number of users to computers subject to capacity constraints. The author proposes a Lagrangian based algorithm along with a heuristic procedure for the solution of the problem. Two other studies that are somewhat more recent are due to Ghosh et al. [43] and Murthy and Ghosh [73], which consider an extension of the problem considered by Pirkul [85] in that

duplication of the files is allowed. The aim is to find an allocation plan such that the total cost of storage and communication (involving query and update costs) is minimized subject to link capacity, storage capacity and delay constraints. Both studies offer a Lagrangean based branch-and-bound algorithm along with some primal and dual heuristics.

Karlsson et al. [57] propose a framework for replica placement algorithms in CDNs, which includes in detail the possible cost parameters and constraints that may arise in such problems. More specifically, the cost function of a replica placement algorithm may include parameters such as read and writes of data, distance, storage, object size, access time and hit ratio. On the other hand, the problem may have additional constraints associated with the storage, load and bandwidth capacity of each node, as well as link capacities, an upper bound on the number of replicas and the response time for requests and availability of objects in the system. The paper also includes a characterization of heuristics that may be used for the solution of the problem. In a similar paper, Karlsson and Mahalingam [58] compare simple caching schemes with replica placement algorithms in CDNs and conclude with the result that the former outperforms the latter if no hard performance guarantees are required.

Cidon et al. [27] offer a distributed algorithm to allocate electronic content over a network with a tree structure so as to minimize the total storage and communication costs. Contrast to the previous work, they allow the servers to be placed at different levels of the tree and consider a generalized cost structure in that the costs of storing an object at two different servers may be different. The algorithm is based on dynamic programming. The proposed algorithm is also shown to solve the joint problem of content and server allocation, where the latter corresponds to the problem of finding the number and the locations of servers in the distribution network, given the unit server cost. A related study by Tamir [92] proposes an algorithm to place a given number of servers on an undirected tree to minimize the overall distance where each client is connected to the closest server.

Optimally locating objects in CDNs with storage constraints on general graphs

has been studied by Kangasharju et al. [55]. The authors consider a network model where there are already a number of proxies deployed throughout the network, each having a limited capacity. The aim is to decide on the set of objects that should be stored at each proxy server. The problem is formulated as an integer program so as to minimize the average travel time of the objects. The following decision variable is used in the model:

$$z_{jk} = \begin{cases} 1, & \text{if proxy server } j \in J \text{ holds object } k \in K \\ 0, & \text{otherwise} \end{cases}$$

The integer model is given as follows:

$$\text{minimize} \frac{1}{\sum_j \lambda_j} \sum_{j \in J} \sum_{k \in K} \lambda_j p_k c_{jk}(\mathtt{x}) \tag{2.13}$$

s.t.

$$\sum_{k \in K} b_k z_{jk} \leq s_j, \qquad \forall j \in J \tag{2.14}$$

$$z_{jk} \in \{0, 1\}, \quad \forall j \in J, k \in K \tag{2.15}$$

The objective function is the average number of hops that a request should traverse, where p_k is the probability that a client will request object k, and λ_j is the aggregate request rate of the clients assigned to proxy j. Given a placement $\mathtt{x}$, $c_{jk}(\mathtt{x})$ is the shortest distance from proxy j to the copy of object k. Note that the objective function is dependent on a given placement $\mathtt{x}$. The only constraint (2.14) imposes capacity restrictions on each proxy server. This is a multiple-knapsack type constraint. The authors prove that this problem is $\mathcal{NP}$-Hard and propose the following four heuristic procedures: random, popularity, greedy-single and greedy-global heuristics. These are described briefly below:

1. *Random Heuristic.* The idea of this very simple heuristic is to assign objects to the proxies based on a uniform probability, without exceeding the capacity constraint.

2. *Popularity Heuristic.* This heuristic is based on the popularity of each object, determined according to their request probabilities. Each node then stores the most popular objects among its clients subject to its capacity constraint.

3. *Greedy-Single Heuristic.* This heuristic is based on the contribution of an object k to a proxy server j, calculated as $C_{jk} = p_k c_{jk}(\mathtt{X})$, where $(\mathtt{X})$ denotes the placement of objects at the origin servers. The placement is then performed, for each proxy j, according to a decreasing order of C_{jk} without exceeding the capacity constraint.

4. *Greedy-Global Heuristic.* For this heuristic, the contributions need to be calculated for each proxy and object pair ($C_{jk} = \lambda_j p_k c_{jk}(\mathtt{X})$). The proxy-object pair with the largest C_{jk} is chosen and this object is placed to this specific proxy. The contributions are then re-calculated and the procedure repeats itself until all capacity restrictions are violated.

The authors find that Greedy-Global heuristic performs the best among all the four heuristics.

Li and Liu [69] studied the growth of the gain in performance of a CDN with the increase of server replicas, where the performance of the CDN can be measured by client request latency, total network bandwidth consumption or an overall cost function. These authors also investigated the effects of client's content demand and distribution patterns on the growth of performance gain in CDNs. The authors model the problem using the p-median formulation and solve it using a heuristic method. It is shown that carefully choosing candidate sites yields the same performance of the network as that of replica placement on all candidate sites. Another result is that there is a decreasing performance gain when the number of replicas replaced exceeds a certain threshold value.

Cahill and Sreenan [22] investigate the design of a Video Content Distribution Network and identify the differences with the classical CDNs. The authors describe the architecture details and present an associated cost function that can

be used to locate proxies that will yield the minimum cost of serving the customers. However, the approach proposed in the paper consists of only a single media object.

The replica placement problem with a dynamic structure is studied by Bartolini et al. [15], who formulate the problem as a Semi-Markov Decision Process by assuming that the requests follow a Markovian structure. As a result, they were able to identify an optimal policy for dynamic replica placement. A heuristic was also offered for the problem, the performance of which was shown to be very close to the optimal placement.

2.1.3 Request Routing

The main goal of *routing* in a computer network is to send data from one or more sources to one or more destinations so as to minimize the total traffic flowing on the network. For a detailed review on the problem as well as a survey of combinatorial optimization applications, we refer to the recent survey by Oliveira and Pardalos [79]. *Request routing*, on the other hand, is basically the process of guiding the clients' requests to specific proxies that are able to serve the corresponding requests. The goal is to select the best server for a client request in terms of response time. As classified by Peng [84], there are five main techniques used for request routing: Client multiplexing, HTTP redirection, DNS indirection, Anycasting and Peer-to-Peer routing. Currently, CDNs such as Akamai use DNS indirection, which selects the best server for a request based on the current situation of the network. Technical details for each technique can be found in Peng [84].

Datta et al. [31] formally define the problem as follows: Given a request for an object, the request routing problem consists of selecting a server for the request such that a cost function to respond to the request is minimized. The authors indicate that the problem is closely related to the problem of distributed load balancing.

2.1.4 Pricing

A commercial CDN, providing service to its customers, usually charges some amount based on the quantity of content delivered. In this case, the CDN would like to maximize its total revenue, which gives way to the problem of determining how much to charge for serving each object. More specifically, if p_k denotes the optimal price charged by the CDN to serve an object $k \in K$ and r_{jk} represents the demand for object k at proxy j, the revenue function proposed by Datta et al. [31] for this problem is given as follows:

$$\sum_{j \in J} \sum_{k \in K} r_{jk} z_{jk} p_j \tag{2.16}$$

where z_{jk} is as defined previously. As indicated by the authors, this formulation is very simplistic and needs to be extended to capture the situation when an object is not found in the proxy server and needs to be retrieved from the origin server. In addition, the costs here are assumed to be linear, which is usually not the case.

We are aware of one more study that deals with the optimal pricing problem in CDNs, due to Hosanagar et al. [47], in which the authors develop an analytical model to determine optimal pricing policies for CDN operators. The model is based on the known distribution of requests to a CDN and uses queuing theory to mimic the characteristics of a web server. The authors find as a result that CDN providers should provide volume discounts to providers. In addition, they indicate that customers with high volume of traffic and low security requirements are likely to subscribe to CDNs.

2.1.5 Joint Considerations

In this section, we provide an overview of the literature on CDNs in which several problems presented above are jointly considered.

In an arbitrary network of nodes with limited capacities and given a demand function of each node for each object, the problem of placing objects so as to minimize the average access cost is investigated by Baev and Rajaraman [13]. The authors refer to this problem as the static *data placement* problem. In such a network, there is no set of origin servers, but rather each node acts as a server in cooperation with other nodes. In other words, the nodes cooperate to serve a given request and in making storage decisions. This is known as *cooperative caching*, which has applications in networks where the nodes trust each other (web hosting service, corporate internet, etc.). The request at a node is assumed to be satisfied by the nearest copy of the requested object. The authors model the problem as a binary program, which is a three-index extension to the p-median model, including capacity restrictions. In addition to the z_{jk} variable defined earlier, the authors also have the following three-index variable, used to indicate the routing decision.

$$x_{ijk} = \begin{cases} 1, & \text{if client } i \in I \text{ is assigned to node } j \in J \text{ holding the requested object } k \in K \\ 0, & \text{otherwise} \end{cases}$$

Then, their formulation is as follows:

$$\text{minimize} \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} b_k d_{ik} c_{ij} x_{ijk} \tag{2.17}$$

s.t.

$$\sum_{j \in J} x_{ijk} = 1, \qquad \forall i \in I, k \in K \tag{2.18}$$

$$x_{ijk} \leq z_{jk}, \quad \forall i \in I, j \in J, k \in K \tag{2.19}$$

$$\sum_{k \in K} b_k z_{jk} \leq s_j, \qquad \forall i \in I \tag{2.20}$$

$$z_{jk} \in \{0, 1\}, \qquad \forall j \in J, k \in K \tag{2.21}$$

$$x_{ijk} \in \{0, 1\}, \quad \forall i \in I, j \in J, k \in K \tag{2.22}$$

In this formulation, the objective function (2.17) expresses the average cost of an access request, taken over all nodes and objects. Note here that $J = I$, i.e. each node acts both as a client and a potential proxy. The first constraint (2.18) expresses that each node should be assigned to a single other node for a specific object request. The second constraint (2.19) indicates that an assignment to a node can only be made if that specific node is holding the requested object. Finally, constraint (2.20) denotes the capacity restriction for each node. The authors prove that the problem with uniform-length objects is $\mathcal{NP}$-Hard and propose an approximation algorithm with a factor of 20.5. They also indicate that this result is asymptotically the best possible. This formulation can also be extended to include additional costs, such as the cost of placing an object at a node.

Ryoo and Panwar [89] consider the problem of distributing multimedia files in networks in designing a network which involves determining the communication link capacities, sizing the multimedia servers and distributing different types of content to each server. They assume a tree topology and the file placement is done according the their popularity. Due to the nature of the content they consider, i.e. continuous stream of data such as video-on-demand, multiple copies of a file needs to be placed in a server for multiple accesses. This, in turn, limits the independent number of accesses to this file to the number of copies located. The client assignment is predetermined.

Xu et al. [99] study the problem of, given a maximum number of potential proxies, determining the optimal number and location of proxies along with placement of objects on the installed proxies. However, the topology they consider is restricted to a tree topology with the origin server being located at the root. They also restrict the problem to a single object. The problem is formulated as an optimization problem and a dynamic programming algorithm is offered for its solution.

Another work by Xuanping et al. [100] discusses the problem of jointly replacing the proxies in a CDN along with the decision of optimally replicating objects. They consider a cost of storage usage for replicating objects at each

proxy and a total available budget constraining the amount of replication that can be performed. The overall aim is to minimize the total access cost. The authors propose two heuristics and evaluate their performance using simulation. However, the client assignment is done by assigning a client to its closest proxy, which may not always be the best assignment.

A recent study by Avella et al. [11] considers web cache location and design issues in Virtual Private Networks (VPNs). VPNs are private networks which are implemented over an existing public network infrastructure. The authors investigate a combined cache location and network design problem on VPNs, where network design refers to dimensioning of the links. Two models and a two stage solution approach are proposed for the problem and the approach is tested on a small VPN (with 8 sites and 3 servers).

Erçetin and Tassiulas [33] consider the content delivery problem in the Internet as a non-cooperative game, where each agent (either the CDN or the content provider) behaves selfishly so as to maximize its net benefit. The authors subdivide the content delivery problem into two, namely content distribution and request routing. The former seeks for the best strategy to place the objects into proxy servers, where the latter searches for the best strategy to route the user requests. Although these two problems are mentioned to be interrelated, the authors develop, for practical purposes, an iterative algorithm where each problem is solved repeatedly, and the routing strategy is determined only after the distribution strategy has been decided on.

Deciding on the placement of the objects together with the capacities of the nodes have been investigated in a recent work by Laoutaris et al. [65]. The discussed problem in this study is named as the capacity allocation problem, and a two step algorithm, requiring the resolution of a p-median problem and a packing problem, is proposed for its solution. The algorithm is capable of optimally solving the problem for tree graphs and yields an approximate solution for general graphs. Another study by the same authors presents optimal and heuristic approaches to solve the storage capacity allocation problem for content distribution

networks, which takes into account decisions regarding the location of the proxies to be installed, the capacity that should be allocated to each proxy and the objects that should be placed in each proxy [66]. Still, the assignment of clients is not considered as a decision problem, in that they assume a given hierarchial topology where the assignment of clients is pre-determined. The authors provide an integer linear programming model for the problem, prove that the problem is $\mathcal{NP}$-Hard, and develop greedy type heuristics to obtain approximate solutions.

Nguyen et al. [77] consider the problem of provisioning CDNs on shared infrastructures. The authors argue that it would be too demanding for a new CDN to deploy a network due to financial, technical and administrative restrictions. They assert that the future of CDNs will either lie in the cooperation of regional CDNs to form a large, Internet-wide CDN or in the CDN hiring resources from an existing infrastructure. The second alternative would be less demanding in terms of investment and resource allocation. Based on such a paradigm, the authors propose a joint provisioning and object replication model so as to minimize the total cost of storage, request serving and startup. The problem is subject to the following constraints: capacities must be respected, all requests must be served, average-customer distance for each object should be less than a predefined threshold, a server should be installed for the objects to be replicated on the server and a server can only serve a request for an object if the object is available there. The authors prove that the problem is $\mathcal{NP}$-Hard and propose a solution procedure for the model based on Lagrangean Relaxation, decomposition and subgradient optimization. They report computational results on problems with up to 36 customers and 100 objects.

Recently, Almedia et al. [10] considered the problem of optimally solving the proxy placement and request routing problems jointly, for which they provided a flow-based optimization model.

2.1.6 Commercial CDNs: Some Insight

In this section, we will try to give some insight on how the commercial CDNs handle the problems presented above. We note that our aim here is not to give an explicit description of how each system works in detail, but rather, to give an insight of in what ways the content distribution is performed. We specifically investigate three commercial CDNs, namely Akamai, Speedera and Nortel Networks. These are presented below.

2.1.6.1 Akamai

Being one of the first commercial CDNs, Akamai [1] is now running a very large network of more than 14000 servers in about 65 countries. Although the way Akamai works is not known in detail, the basic operations as indicated in [1] can be described as follows. Akamai's servers are located at the same site or a very close site to the local ISP of the end user, geographically. A client, requesting information from a web site, only contacts the Akamai server located near by. It is the Akamai's web server that retrieves the required content from the origin web site. This way, the client does not have to contact the origin web site, thereby experiencing a reduced response time. Such an approach also results in a secure application, since all the transactions are done locally, between the Akamai web server and the client. On technical details on how Akamai works, the reader is referred to Peng [84] and Pan et al. [83].

2.1.6.2 Speedera

Speedera, another commercial CDN, works in a similar way as Akamai. It has a network of already deployed caching servers at the Points of Presence in the Americas, Europe and the Asia-Pacific region [6]. When the content provider publishes content to Speedera, this content is replicated on the caching servers using a classic 80/20 scenario that is assumed to be typical for the clients, stating that 20% of the whole content generates 80% of the actual network traffic. Thus,

only the top 20% of the content (the popular content) is placed on the installed servers while the remaining content is placed on or close to the origin server. When a client issues a request, Speedera's traffic management system finds the best caching server based on multiple criteria, such as the location of the client, latency, the availability and the load of the servers and the ability of the server to deliver valid content. If the requested content is found in the caching server, then it is server to the customer from here. Otherwise, the caching server retrieves the content from the origin server.

2.1.6.3 Nortel Networks

Another commercial CDN provider, Nortel Networks uses a content manager and a content director to automatically and intelligently distribute content to a number of proxies and to dynamically direct the client requests to the best proxy server holding the requested content, respectively [75]. Note, however, that the content director works on top of the content manager, i.e. the request direction is performed after the content has been distributed to the available proxy servers.

2.1.7 Discussion

As the overview also indicates, there is an active and a growing literature on CDNs. However, the existing literature generally tends to focus on a single problem while assuming the remaining problems are decided on a priori. For instance, studies on proxy server placement assume that a client's request routing is done according to the geographically closest available proxy, which may not always be the best assignment. Similarly, object replication studies are usually based on the assumption that there is already a set of proxies installed at given locations, which might not be the best installation configuration. The main argument here is that all these problems are interdependent as the decision for the one effects the other, and thus should be considered simultaneously.

Another important aspect of the existing literature is that the Internet is

usually modelled as a tree-network, while it rarely has such a structure. Furthermore, studies considering the object replication problem usually assume that there is a single object to be distributed, while it is generally the case that there are multiple objects to be distributed.

The current implementations in practice have several common aspects. First of all, it is clear that these commercial CDNs have already deployed a network with a known number of proxy servers. The content is replicated on the proxy servers based on the popularity of the content. The important aspect is that the request routing is generally handled dynamically, i.e. the request is directed to the best available proxy server available at the time of the request. Such an implementation suggests a two-stage approach for content distribution where the first stage consists of determining the best proxy locations, whereas the second stage includes the object replication and request routing issues. We will elaborate more on this in Chapter 3. This approach seems appropriate from the management point of view. More specifically, the proxy server location problem can be regarded as a long-term decision at the strategic level, while the others are either at the tactical or operational levels. However, it should be noted that this approach may result in a suboptimal solution in some situations since these three problems are interdependent and the decision of one depends on the decisions of the others. In fact, Johnson et al. [53], as a result of an investigation of two major commercial CDN providers, state that neither of these "can consistently pick the best server of those available".

2.2 Video on Demand

We provide in this section a review of the literature on VoD, especially related with design and distribution issues.

A very good introduction to the topic, considering all possible aspects of a VoD system, such as system components, technologies, communication services, resource management, load balancing is provided by Little and Venkatesh [70].

More recently, Ghose and Kim [42] surveyed the scheduling issues VoD systems, providing a comprehensive overview of the major policies used to schedule the video streams. They conclude that batching, in which a user placing a request for specific a program has to wait until the system collects a batch of requests for the same program, is a viable alternative in improving the performance of a VoD system. In addition, the authors mention the necessity to consider a distributed server architecture for such systems, due to the huge capacity requirements for storing the programs.

Bisdikian and Patel [19] consider designing multimedia distribution systems from a cost perspective and present two networking architectures, where each user is allocated to a specific server but can watch any program transparently from any other server. The authors discuss the trade-offs between traffic flow, link utilization, number of copies of programs and the storage and transmission costs in such systems. A relevant study to the issue of network architectures to VoD systems is due to Barnett and Anido [14], who compared distributed and centralized approaches in terms of costs incurred. These costs include cabling costs, network bandwidth costs, user set-up costs and video server costs. The authors show that a distributed approach, having no greater cost than a centralized approach, has also significant benefits in terms of reducing network bandwidth requirements, improving response time and reliability.

Kim et al. [60] consider designing a VoD system on a network with storage capacity constraints on each node and no capacity limitations on links between each pair of nodes, so as to decide on where to place the video servers and the amount of programs transmitted via each link. They offer an integer linear programming formulation of the problem along with an efficient tabu search algorithm for its solution. The authors present computational results for networks with up to 40 nodes and 200 programs.

Wang et al. [96] study the optimal video distribution problem in VoD systems with multiple multicast sessions. Multicasting is performed when a set of clients require the same program at approximately the same time. In this case, clients are grouped as a multicast tree and the server sends the program through this

tree. The authors present a branch and bound algorithm to find an optimal solution when the network is a directed acyclic graph and propose an approximation algorithm for general graphs.

Hwang and Chi [49] consider the problem of placing a number of programs on a number of servers such that the total installation cost that is composed of the network transmission cost and the video storage cost. The authors provide fast heuristic algorithms for optimal program placement with and without storage capacity constraints on video servers.

Leung and Wong [67] address a somewhat different aspect of the problem as what kind of a charging scheme should a service provider adopt in order to maximize the mean revenue.

Ouveysi et al. [80] proposed an integer programming formulation to determine the location of the video programs so as to minimize the total cost of storage and transmission, subject to storage and transmission capacity constraints. They refer to this problem as the Video Placement and Routing Problem, for the solution of which the authors propose heuristic approaches. Finally Huang and Fang [48] propose a dynamic load balancing among the servers in a multi-server VoD system. Through simulations, the authors demonstrate that their algorithms perform well on an example network.

2.2.1 Discussion

VoD systems, unlike CDNs, are being studied for more than 10 years. Although there is a considerable amount of research on these systems, they are mainly focused on building different network structures. As for the problems arising in such systems, they are usually tackled by heuristic approaches, approximation algorithms and simulations, where exact algorithms are limited. We believe that VoD providers may benefit from exact solution algorithms, especially for problems involving load balancing.

Chapter 3

Content Distribution Network Design

According to the type of the content to be distributed, the structure of the distribution network, and the objective of the design problem, various formulations can be developed to model the distribution of electronic content. It is observed that the formulations proposed in the literature are mainly based on and are extensions of some well-known facility location models. In this chapter, we study the problem of the *content distribution network design*.

This chapter is organized as follows: In Section 3.1, we formally define the general problem setting and develop two novel formulations based on location models. We then propose exact and heuristic solution procedures to solve one of the proposed formulations, and present the associated computational results in Section 3.2. In Section 3.3, we investigate the potential savings of the proposed approach through computational experiments. Conclusions are stated in Section 3.4.

3.1 Location Models for Content Distribution

The two models presented in this section differ with respect to the number of origin servers in the network. In specific, we consider single and multiple server architectures and present the associated models in the subsections that follow:

3.1.1 CDN with a Single Server

The problem we consider here is formally defined as follows. We consider a complete network $G = (V, E)$, where V is the set of nodes and $E = (\{i, j\} : i, j \in V)$ is the set of logical links. The node set V is further partitioned into three nonempty, mutually exclusive and exhaustive subsets as $V = I \cup J \cup S$, where I is the set of clients, J is the set of potential nodes on which proxy servers can be installed and S is the set of origin servers. We may either have single or multiple origin servers. We limit our study here to a single origin server (i.e. $|S| = 1$) but the model can easily be extended to the multiple server case ($|S| > 1$). Further we assume without loss of generality that no client can directly access the origin server (e.g. for security reasons).

In practice, we generally have a network that is not necessarily complete (the *physical layer*). However we assume that there are logical direct links (one-hop paths) that connect every pair of nodes in the *link layer*. A logical link may pass through one or more physical links. The correspondence between the logical and physical links on the network can easily be established via an indicator function defined below:

$$\delta_{ij}^{l} = \begin{cases} 1, & \text{if the logical link } \{i, j\} \in E \text{ uses link } l \text{ in the physical network} \\ 0, & \text{otherwise} \end{cases}$$

Having defined such a correspondence, we can concentrate to work with the logical links. These links constitute the backbone of the network and are used for the distribution of the contents.

In designing such a network, there are two important cost components that should be considered. The first is related to with the cost of placing a server at node j and this is denoted by f_j. In specific, f_j is equal to the capital recovery cost of server plus the annual running cost of it (hosting center, power supply, air condition, maintenance and etc.). The fixed cost depends on the type of the server and its backbone capacity. Normally in practice there are clusters of servers with a load balancing switching devices to divert the incoming request to particular servers. For instance, an amount of $100,000 does not provide a very big server and it might be assumed as a minimum requirement.

The second cost component is related with transferring the traffic. The cost of per unit traffic that is transferred over the link $\{i, j\} \in E$ is denoted by c_{ij} (similarly c_{jS} denotes the unit cost from proxy j to the origin server). This cost may represent unit bandwidth cost in terms of latency, number of hops, etc. and is usually a linear-traffic dependent function [40]. In reality the bandwidth would be purchased in incremental amounts. Therefore, it could be approximated as a linear relationship as being adapted in this chapter. There may be pricing mechanisms where a larger bandwidth-link is priced as the same as a small bandwidth-link which leads to a nonlinear pricing exercise. Transmission costs can be determined on an annual basis, taking into account the expected traffic.

It is assumed that the link capacities are sufficiently large to allow the transfer of the available content. This assumption is valid when we are talking about nation-wide networks or networks of small size such as intranets (Similar assumptions have also been made in several previous studies, e.g. see [27] and [12]).

We assume here that a suitable routing protocol is used to ensure a guaranteed end-to-end bandwidth reservation to achieve some Quality of Service (QoS) capability in the network. The QoS functionalities can then be implemented in the physical layer of the backbone network parallel to the optimization task of content distribution that is performed in the link layer. However, QoS related concepts are out of the scope of this study and we focus only on the content distribution optimization task here.

Table 3.1: Summary of Notation used for the CDN Model

Notation	Indication
f_j	instantiation cost of proxy server on node $j \in V$
s_j	capacity of the potential proxy server on node $j \in V$
b_k	size of object $k \in K$
d_{ik}	probability of requesting object $k \in K$ by client $i \in I$
c_{ij}	unit cost of transferring an object over link $\{i, j\} \in E$
c_{jS}	unit cost of transferring an object from the origin server to proxy server $j \in J$

Each client is assumed to be served by exactly (or at least) one proxy server. In any case, we assume that the client retrieves the requested object from only a single server. This consideration is based on a well-stated result given by Kangasharju et al. [56], who demonstrated through simulation that retrieving an object as a whole from a single proxy results in a better performance compared to the situation where the client receives different parts of the object from different proxies. If a content requested by a client is not found in the assigned proxy server, then the client is able to access it from the origin server via the path from the corresponding proxy server to the origin server, but at the expense of an additional transfer cost.

We assume that the capacity of the potential server at site j is s_j. We define K as the set of objects located in the origin server and assume that the size of each object $k \in K$ is b_k. Also we consider that the probability of client $i \in I$ requesting object $k \in K$ is denoted by d_{ik}.

The cost components and the demands should be determined based on a pre-specified time frame (e.g. on an annual basis). In this case, the client demands should be calculated as an annual estimate of previous demand patterns. Once this is done, the unit cost of transmission should also be determined accordingly.

We emphasize that we consider an existing network in our study and that we ignore the option of network expansion here. The notation used here is summarized in Table 3.1.

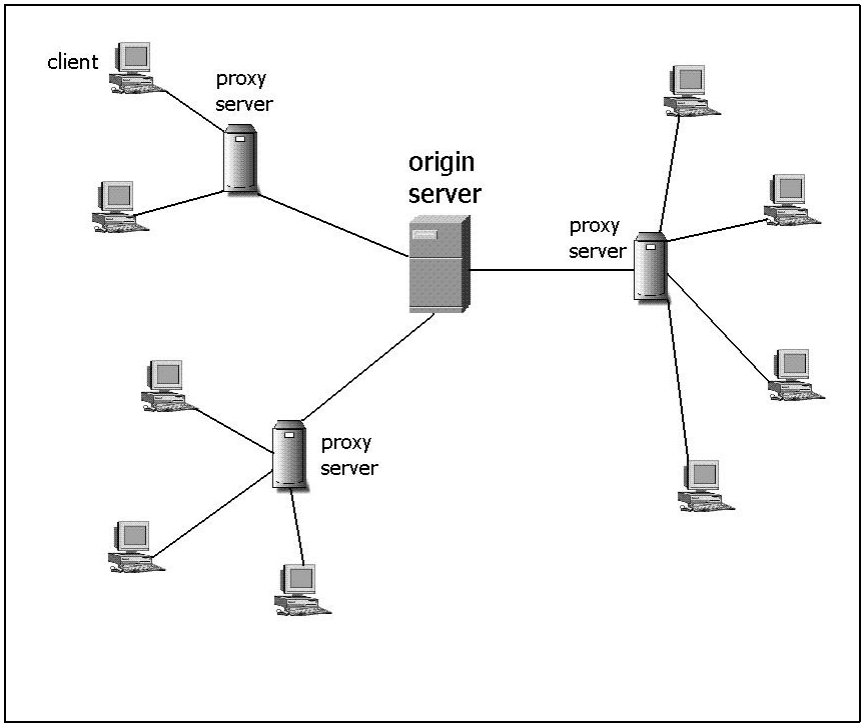

Figure 3.1: A typical single server CDN architecture with 3 proxy servers and 9 clients

The CDN architecture just described is depicted in Figure 3.1, with the origin server (in the box), 3 proxy servers and 9 clients, where each client is connected to a single proxy server.

The problem then consists of simultaneously deciding on the following issues:

- the assignment of each client to a single (or multiple) proxy servers
- the number and the location of the proxy servers to be used in the CDN among a given set of potential sites
- the objects to be located in each proxy server

The objective is to design a CDN such that the total cost is minimized. We refer to this problem as the *Single Server Content Distribution Network Problem* and denote it by *SCDNP*. It is important to note here that we consider this

problem from the perspective of the CDN provider, who would like to reduce its expenses by minimizing the total cost. There may be several other schemes that can be employed in a CDN, such as dynamically selecting the best proxy for a client that is offering the lowest response time. We ignore such a scheme as we do not consider real-time decisions and our design consists of making all the decisions a priori. Although this may seem like a very static approach for such an application, replication for content distribution based on steady state demand rates are shown to have significant benefits in [94].

We define the following binary decision variables:

$$y_j = \begin{cases} 1, & \text{if node } j \in J \text{ is selected as a proxy server} \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if client } i \in I \text{ is assigned to proxy server } j \in J \\ 0, & \text{otherwise} \end{cases}$$

$$z_{jk} = \begin{cases} 1, & \text{if proxy server } j \in J \text{ holds object } k \in K \\ 0, & \text{otherwise} \end{cases}$$

We now present a novel integer programming formulation for the SCDNP problem below. The novelty of the proposed formulation lies in simultaneously addressing the three interdependent problems mentioned above as well as explicitly representing the distribution structure of a CDN through the objective function.

$$\text{minimize} \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} (b_k d_{ik} c_{ij} z_{jk} x_{ij} + b_k d_{ik} (1 - z_{jk})(c_{jS} + c_{ij}) x_{ij}) \quad (3.1)$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \qquad \forall i \in I \quad (3.2)$$

$$x_{ij} \leq y_j, \quad \forall i \in I, j \in J \quad (3.3)$$

$$\sum_{k \in K} b_k z_{jk} \leq s_j y_j, \qquad \forall j \in J \quad (3.4)$$

$$y_j \in \{0, 1\}, \qquad \forall j \in J \quad (3.5)$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, j \in J \quad (3.6)$$

$$z_{jk} \in \{0, 1\}, \quad \forall j \in J, k \in K \quad (3.7)$$

In the objective function (3.1), the first summation represents the total cost of installing proxy servers. The second summation denotes the total cost of transferring the content. The first part of this summation is for the case when client i receives content k that is located in proxy j (reflected by the cost $b_k d_{ik} c_{ij} z_{jk} x_{ij}$ summed over all the proxies, clients and objects). In the case when the requested object is not located in a proxy server, an additional cost is incurred to further request the object from the origin server. This is reflected in the second part of the summation by $(b_k d_{ik}(1 - z_{jk})(c_{jS} + c_{ij})x_{ij}))$, over all proxies, clients and objects. The objective function (3.1) is nonlinear, which makes the problem harder to solve. Similar cost functions have also been suggested in [63], [102], and [31].

Constraint (3.2) is the assignment constraint indicating that each client must be assigned to exactly one proxy server. Note that the case where each client can receive objects from different proxies may easily be accommodated by using the following constraint instead of (3.2):

$$\sum_{j \in J} x_{ij} \geq 1, \forall i \in I \quad (3.8)$$

Constraint (3.3) implies that a client can be assigned to a node only if a proxy server is installed on that node. Constraint (3.4) implies that the total size of objects held in each proxy server is constrained by the available capacity. Finally, constraints (3.5) - (3.7) denote the integrality of the decision variables.

The model just presented for the SCDNP has $|J| + |I||J| + |J||K|$ binary variables and $|I| + |I||J| + |J|$ constraints. Next, we show the complexity status of the problem.

Proposition 1 *The SCDNP is $\mathcal{NP}$-Hard.*

Proof We prove the proposition by restriction (see [39]). Let us consider the following instance of the SCDNP: $K = \{1\}$ (i.e. there is only a single object), $b_1 = b$, $d_{i1} = d_i$ and $s_j \geq b$, $\forall j$ (i.e. all the proxies have a sufficiently large capacity). Since there are no capacity constraints in this case, (3.4) becomes redundant and $z_{j1} = 1$, $\forall j$. In this case, the problem becomes the Uncapacitated Facility Location Problem (FLP), which is known to be $\mathcal{NP}$-Hard (see e.g. [28]). □

In the following, we give some observations regarding the connections of SCDNP with other well-known problems:

1. If each proxy server has an infinite capacity, i.e. $s_j >> M, \forall j \in J$ then the capacity restriction on the storage amount becomes redundant. In this case, since the whole content is replicated in each proxy server, the clients receive all their demands from the proxy servers. This problem is the *Multicommodity Uncapacitated FLP* (see [38], [61]). When there is only a single object, the SCDNP becomes the well-known *Uncapacitated FLP*.

2. If there is no fixed cost of installing a proxy server and each proxy has an infinite storage capacity, then the problem becomes the well-known *p-Median* problem.

Unlike the work of Nguyen et al. [76], we do not assume that the total capacity of established proxies is greater than the total demand. In fact our model is a better representation of a real life operation in a CDN, where a requested object not available in the proxy server is cached from the origin server. This is exactly the reason for the nonlinearity of the objective function in the SCDNP model. In addition, Nguyen et al. [76] allow the client request to be fractionally served by proxies holding the requested content, where, we do not allow for such a situation (as explained previously).

We will provide exact and heuristic approaches to solve SCDNP in Section 3.2. Before doing that, we extend the SCDNP model to the case of multiple origin servers in the following subsection.

3.1.2 CDN with Multiple Servers

In this section, we consider the situation where there are multiple origin servers (denoted by the set S) in the CDN. The problem in this case, in addition to the requirements of SCDNP, is to decide on which proxy should be assigned to each origin server. We refer to this problem as the *Multiple Server CDN problem* (*MCDNP*). A feasible sample configuration for a MCDNP is given in Figure 3.2 with 3 origin servers, 3 proxy servers and 9 clients. In addition to the assumptions and definitions provided in the previous section, we further define the following decision variable, which is used to decide which specific proxy server will be connected to a specific origin server:

$$t_{js} = \begin{cases} 1, & \text{if proxy server } j \in J \text{ is assigned to the origin server } s \in S \\ 0, & \text{otherwise} \end{cases}$$

In MCDNP, we assume that each client is served by a single proxy server, which, in turn is connected to only one origin server. The integer formulation is given below:

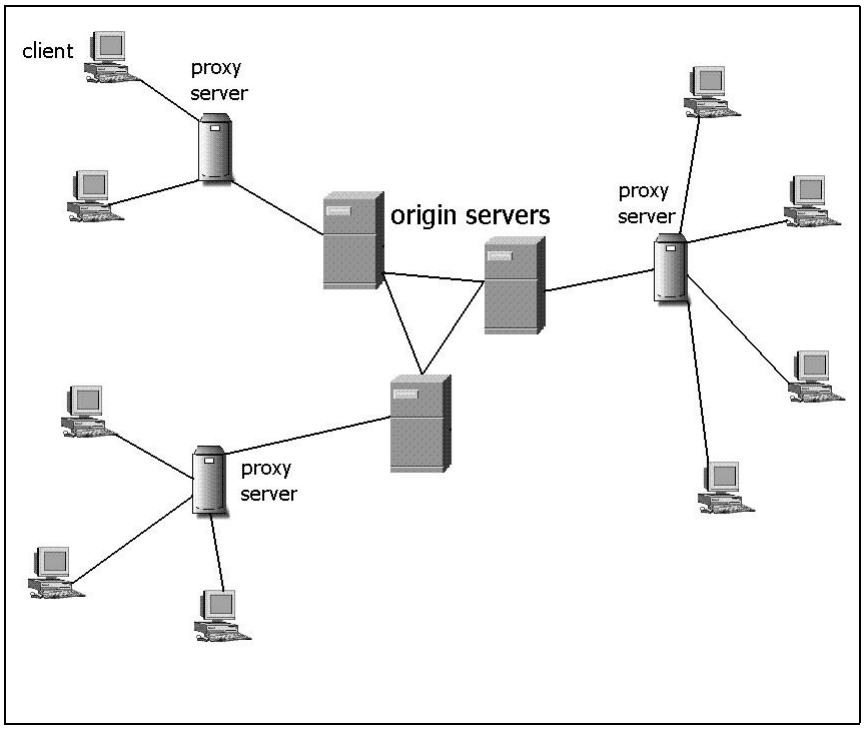

Figure 3.2: A typical multiple server CDN architecture with 3 origin servers, 3 proxy servers and 9 clients

$$\text{minimize} \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} \sum_{s \in S} \sum_{k \in K} (b_k d_{ik} c_{ij} z_{jk} x_{ij} + b_k d_{ik} (1 - z_{jk})(c_{ij} + c_{js} t_{js}) x_{ij}) \tag{3.9}$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \qquad \forall i \in I \tag{3.10}$$

$$x_{ij} \leq y_j, \quad \forall i \in I, j \in J \tag{3.11}$$

$$\sum_{k \in K} b_k z_{jk} \leq s_j y_j, \qquad \forall j \in J \tag{3.12}$$

$$\sum_{s \in S} t_{js} = 1, \qquad \forall j \in J \tag{3.13}$$

$$y_j \in \{0, 1\}, \qquad \forall j \in J \tag{3.14}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, j \in J \tag{3.15}$$

$$z_{jk} \in \{0, 1\}, \quad \forall j \in J, k \in K \tag{3.16}$$

$$t_{js} \in \{0, 1\}, \quad \forall j \in J, s \in S \tag{3.17}$$

In this formulation, constraints (3.10)-(3.12) are the same as that of SCDNP model. The only different constraint, given in (3.13), ensures that each proxy server is connected to a single origin server. We would like to note that this formulation is again quadratic in the objective function given in (3.9). A linearization of the MCDNP model is provided in Appendix B.

It is straightforward to show that when there is only a single origin server, i.e. $|S| = 1$, constraints (3.13) become redundant and the formulation reduces to that of SCDNP. Therefore, this problem is also $\mathcal{NP}$-Hard.

3.2 Exact and Heuristic Solutions for SCDNP

There are several ways to solve the nonlinear integer programming model presented for the SCDNP. One strategy would be to use techniques specifically devised for quadratic optimization problems. In this study, we consider a solution approach based on a special linearization of the model. Heuristic solution procedures are also developed. These are presented in the following.

3.2.1 Model Linearization

It is clearly seen that the objective function (3.1) contains a quadratic term due to the multiplication of the x_{ij} and z_{jk} variables. One way for linearization is to introduce a new binary variable into the formulation along with three sets of constraints (as shown in [82] and [86]). We hereby propose a more efficient linearization, using a continuous variable and only two sets of constraints. Although there are other linearization techniques available for such quadratic functions, as we will show later in this section, our linearization brings forth a special structure that will enable us to use the exact solution approach to our model. The following is the basis of our linearization.

Proposition 2 *The following constraints are sufficient to linearize the objective function* (3.1) *of the SCDNP model,*

$$\varphi_{ijk} \leq x_{ij}, \quad \forall i \in I, j \in J, k \in K \tag{3.18}$$

$$\varphi_{ijk} \leq z_{jk}, \quad \forall i \in I, j \in J, k \in K \tag{3.19}$$

where $\varphi_{ijk} = z_{jk}x_{ij}$ *and is a continuous variable in* $[0, 1]$.

Proof By simplifying the objective function as $\sum_{i\in I}\sum_{j\in J}\sum_{k\in K}(b_k d_{ik}(c_{ij} + c_{jS})x_{ij} - b_k d_{ik} c_{jS}\varphi_{ijk})$, where $\varphi_{ijk} = z_{jk}x_{ij}$, the proof relies on the observation

that the coefficient of φ_{ijk} in the objective function is $-b_k d_{ik} c_{jS}$, which is always negative. By definition, φ_{ijk} should be 1 if and only if $z_{jk} = 1$ and $x_{ij} = 1$, and 0 for all other cases. Now, assume that $z_{jk} = 1$ and $x_{ij} = 1$ for a specific (i, j, k) triplet. Then, according to constraints (3.18) and (3.19), φ_{ijk} is only constrained by the upper bound 1 and the minimizing objective function implies $\varphi_{ijk} = 1$. In all other cases (i.e. $x_{ij} = 1$, $z_{jk} = 0$; or $x_{ij} = 0$, $z_{jk} = 1$; or $x_{ij} = 0$, $z_{jk} = 0$) constraints (3.18) and (3.19) together imply $\varphi_{ijk} = 0$. □

Note that the linearizing variable φ_{ijk} is actually an indicator of whether client i is connected to the proxy server j *and* the proxy server holds the requested object k or not. Under the proposed linearization, the objective function (3.1) reduces to the following:

$$\sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} (b_k d_{ik} (c_{ij} x_{ij} + c_{jS} (x_{ij} - \varphi_{ijk}))) \tag{3.20}$$

We can now construct the integer linear programming formulation of the SCDNP, denoted by M(SCDNP), as Minimize (3.20): s.t. (3.2)-(3.4), (3.18), (3.19), (3.5)-(3.7), $\varphi_{ijk} \in [0, 1]$.

3.2.2 A Preliminary Analysis of the SCDNP Model

In order to assess the computational performance of the proposed model for the SCDNP, we have performed some preliminary computational experiments. These experiments are based on randomly generated Internet topologies, using an Internet topology generator, available at the web address http://topology.eecs.umich.edu/inet/, which mimics the characteristics of the real Internet topology. The instances used for comparison purposes have different number of potential proxy locations, clients and objects (denoted by $|J|$, $|I|$, and $|K|$ respectively). Based on the generated topology, we have created a hierarchical network as follows: The topology was modified such that the average

cost between the origin server and the clients is doubled with respect to the average cost between the origin server and the potential proxy locations, i.e., it is two times more expensive to send content to a client than sending content to a proxy server. The server is selected to be the first node of the generated network. The unit transfer cost on a link is generated in accordance with the hierarchical network. The size of each object is chosen from a uniform random variable between 0 and 1. The fixed cost of installing a proxy server is also a uniform random variable between 200 and 1000. The capacity of each proxy server is calculated as a random number between 1 and the 50% of the total size of all objects. The demand distribution for the objects have been modelled using a Zipf-like distribution. This distribution is known to obey the characteristics of web requests (see [21]). The Zipf-like distribution assumes that, the probability of a request for an object is inversely proportional to its popularity. More specifically, let a number of objects be ranked in order of their popularity where object i in this order is the i^{th} most popular object. Then, given an arrival for a request, the conditional probability that the request is for object i is given by the following:

$$P_K(i) = \frac{\Omega}{i^\alpha}$$

where

$$\Omega = (\sum_{i=1}^{K} \frac{1}{i^\alpha})^{-1}$$

is a normalization constant and α is an exponent. When $\alpha = 1$, we have the true Zipf-distribution. In [21], it is shown that α varies from 0 to 1 for different access patterns and is usually between 0.64 and 0.83 for web objects. This value was recorded to be $\alpha = 0.733$ for multimedia files in [102]. We have used this specific value in our implementation.

The metric used to assess each solution is a *normalized cost* metric, as used in other studies (e.g. [99]), which is defined as follows:

Table 3.2: Computational analysis of M(SCDNP)

$\|J\|$[1]	$\|I\|$[2]	$\|K\|$[3]	v_{IP}[4]	Time (sec)[5]	v_{LP}[6]	Gap (%)[7]
2	5	10	0.196743	0.00	0.176784	10.15
2	5	20	0.220278	2.20	0.160009	26.79
2	5	30	0.243757	3.40	0.203921	22.16
2	5	40	0.203196	14.40	0.164390	23.61
2	5	50	0.159261	120.20	0.117373	27.75
2	10	10	0.259229	1.20	0.175000	34.56
2	10	20	0.232459	33.00	0.173245	28.54
2	10	30	0.195147	83.20	0.143140	26.21
2	10	40	0.185774	562.60	0.129690	34.28
2	10	50	0.153416	3266.40	0.103495	32.28
3	5	10	0.118412	2.20	0.053974	59.11
3	5	20	0.106240	13.60	0.050549	52.48
3	5	30	0.087297	18.20	0.027413	72.85
3	5	40	0.079468	29.80	0.018525	77.12
3	5	50	0.097295	27.80	0.052646	49.92
3	10	10	0.136549	3.00	0.059750	56.60
3	10	20	0.109765	578.20	0.032691	72.67
3	10	30	0.114386	1007.44	0.042822	63.59
3	10	40	0.129712	1265.60	0.057250	59.16
3	10	50	0.081450	3668.60	0.026444	67.54

[1] number of potential proxy servers,
[2] number of clients, [3] number of objects,
[4] Optimal solution value of the integer program,
[5] Solution time of the integer program (in seconds),
[6] Optimal solution value of the integer programming relaxation,
[7] Percentage gap between v_{IP} and v_{LP}

$$normalized\ cost = \frac{cost\ of\ the\ network\ output\ by\ the\ procedure}{cost\ of\ the\ network\ without\ any\ replicated\ proxies}$$

Here, the cost of the network without any replicated proxies is the scheme where all the clients are assumed to retrieve the requested content from the origin server. Note that the smaller the normalized cost, the better the solution found by the procedure is.

Table 3.2 presents, the computational efficiency of the proposed model on randomly generated instances with varying configurations as shown in the first

three columns of the table. The values presented in each line are the averages of 5 randomly generated instances with the same configuration. Columns v_{IP}, Time and v_{LP} respectively present the optimal solution value of the instance found by CPLEX 9.0, the required solution time and the optimal LP relaxation value obtained by using the model. Finally, the last column presents the gap between the LP-bound and the integer optimal solution and calculated by $\frac{v_{IP}-v_{LP}}{v_{IP}} * 100$.

As the table indicates, it becomes harder to solve the instances to optimality as the size of the instance increases. Furthermore, as also shown in the last column of the table, the gaps can be quite high. This is due to the type of linearization used in the previous section that forces us not to use any approach based on the LP-relaxations (such as a cutting plane algorithm). Although there are alternative tighter linearizations (e.g. see Adams and Sherali [9]), we will continue to use the proposed linearization in the rest of the chapter, since it brings forth a special structure of the model that enables us to develop an exact solution algorithm based on decomposition. This is explained further in the next section.

3.2.3 An Exact Solution Algorithm: Benders' Decomposition

To solve M(SCDNP), we use the decomposition approach of Benders [18]. We additionally note that the solution approach presented here is a demonstration of an exact solution procedure for similar integer models with a quadratic objective function. The approach is simply based on fixing some of the variables to some predetermined values, yielding a *subproblem* (henceforth denoted by SP), from which a *master problem* (henceforth denoted by MP) is derived and the approach iterates back and forth between two problems. Consider, now, fixing variables z_{jk} and y_j to either 0 or 1 (while maintaining feasibility) and denote these fixed values with $\overline{z}_{jk}$ and $\overline{y}_j$, respectively. Consequently, we can omit the capacity constraint (3.4) and the integrality constraints (3.5) and (3.7). The resulting SP is as follows:

$$(SP) \qquad \text{minimize} \sum_{j \in J} f_j \overline{y}_j + \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} (b_k d_{ik} (c_{ij} x_{ij} + c_{jS} (x_{ij} - \varphi_{ijk})))$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \qquad \forall i \in I \tag{3.21}$$

$$-x_{ij} \geq -\overline{y}_j, \qquad \forall i \in I, j \in J \tag{3.22}$$

$$-\varphi_{ijk} + x_{ij} \geq 0, \quad \forall i \in I, j \in J, k \in K \tag{3.23}$$

$$-\varphi_{ijk} \geq -\overline{z}_{jk}, \quad \forall i \in I, j \in J, k \in K \tag{3.24}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in I, j \in J \tag{3.25}$$

$$\varphi_{ijk} \in [0, 1], \quad \forall i \in I, j \in J, k \in K \tag{3.26}$$

It is obvious that SP still includes $|I||J|$ binary variables. In what follows, we show that we can relax the binary variables in the interval $[0, 1]$ and still obtain an integral solution. But before doing that, we present a definition and some results from Wolsey [98]:

Definition 1 ([98], p.38) *A matrix A is Totally Unimodular (TU) if every square submatrix of A has determinant* $+1$, -1 *or* 0.

Proposition 3 ([98], p.39) *A matrix A is TU if and only if the matrix (A,I) is TU.*

Proposition 4 ([98], p.39) *(Sufficient Condition) A matrix A is TU if*

1. *Each entry of A is either* $+1$, -1 *or* 0.
2. *Each column contains at most two nonzero coefficients.*
3. *There exists a partition* (M_1, M_2) *of the set M of rows such that each column j containing two nonzero coefficients satisfies* $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$.

Proposition 5 ([98], p.40) *The linear program max$\{cx$: $Ax \leq b$, $x \in \Re^n_+\}$ has an integral optimal solution for all integer vectors b for which it has a finite optimal value if and only if A is totally unimodular.*

Proposition 6 *The constraint matrix A of SP with the constraints (3.21)-(3.24) is Totally Unimodular.*

Proof Rearranging the constraints of SP, we first observe that the constraint matrix is of the form (A, I), where I corresponds to the constraints (3.22) and (3.24), and A corresponds to the remaining constraints. Therefore, it suffices to show that A is TU by Proposition 5. It is easy to see that each entry of A is either $+1$, -1 or 0 and each column contains at most two nonzero coefficients. Now, consider a partition $M_1 = M$ and $M_2 = \emptyset$ of the set M of rows. In this partition, each column j containing two nonzero coefficients satisfies $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$. Thus, A is TU and by Proposition 5 the LP relaxation of SP provides an integral solution. □

Based on the result of the preceding proposition, we relax constraints (3.25) such that $0 \leq x_{ij} \leq 1$. We can then omit constraints (3.25) and (3.26), since these are already implied by the remaining constraints.

We can now proceed with the dual of SP (denoted by SP_D), as shown below:

$$(SP_D) \quad \text{maximize} \sum_{j\in J} f_j\overline{y}_j + \sum_{i\in I} u_i - \sum_{i\in I}\sum_{j\in J} \overline{y}_j w_{ij} - \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} \overline{z}_{jk} n_{ijk}$$

s.t.

$$\begin{aligned} u_i - w_{ij} + \sum_{k\in K} m_{ijk} &\le \sum_{k\in K} b_k d_{ik}(c_{ij} + c_{jS}), && \forall i \in I, j \in J \\ -m_{ijk} - n_{ijk} &\le -b_k d_{ik} c_{jS}, && \forall i \in I, j \in J, k \in K \\ u_i &: free, && \forall i \in I \\ w_{ij}, m_{ijk}, n_{ijk} &\ge 0, && \forall i \in I, j \in J, k \in K \end{aligned}$$

where u_i, w_{ij}, m_{ijk} and n_{ijk} are dual variables associated with constraints (3.21), (3.22), (3.23) and (3.24), respectively. In the objective function, $\sum_j f_j\overline{y}_j$ is a constant since the y_j's are fixed. We show in the following that the SP_D is bounded and has a feasible solution.

Proposition 7 *SP_D is always bounded and feasible for a given set of feasible y_j and z_{jk} variables.*

Proof Since SP_D is generated from SP, it suffices to show that the latter is always bounded and feasible. Notice that the set of $\overline{z}_{jk}$ and $\overline{y}_j$'s are always chosen to be feasible (satisfying constraints (3.4), (3.5) and (3.7)) to construct SP. Thus, it is always possible to find a feasible set of x_{ij}'s (e.g. assigning each client to the closest proxy server). Since each of the variables can take values in the interval [0,1], SP and SP_D are bounded and feasible. □

The objective function of SP_D provides a cut for MP, which is shown in the following:

$$
\begin{aligned}
(MP)\ \text{minimize } & z_0 \\
\text{s.t.} & \\
z_0 \geq \sum_{i\in I} u_i^* - \sum_{i\in I}\sum_{j\in J} w_{ij}^* y_j - \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} n_{ijk}^* z_{jk} + \sum_{j\in J} f_j y_j & \qquad (3.27) \\
\sum_{k\in K} b_k z_{jk} \leq s_j y_j, \qquad \forall j \in J & \\
z_0 \geq 0 & \\
y_j \in \{0,1\}, \qquad \forall j \in J & \\
z_{jk} \in \{0,1\}, \quad \forall j \in J, k \in K &
\end{aligned}
$$

In the MP, constraint (3.27) is the Benders' cut and u_i^*, w_{ij}^*, m_{ijk}^* and n_{ijk}^* are the optimal values of the respective variables in SP_D. The MP, in general, includes an exponential number of Benders' cuts (3.27), and thus is not computationally tractable. However, this problem may be overcome by relaxing these cuts and dynamically generating only a subset of them at each iteration. This is accomplished by using the values of the dual variables to compute an extreme point and generate the corresponding optimality cut. We provide below a formal definition of the algorithm:

Benders' Decomposition Algorithm:

1. Let LB=$-\infty$, UB=$+\infty$. Fix z_{jk} and y_j, $\forall j \in J, k \in K$ to some feasible configuration.

2. Repeat the following until (UB-LB)/UB $\leq \epsilon$, where ϵ is a convergence parameter indicating the maximum allowable gap between the upper and lower bounds.

   (a) Solve SP and set UB=min{z^*,UB}, where z^* is the optimal solution value of SP.

   (b) Add the corresponding Benders' cut to MP. Let the optimal solution value of MP be z_0^*. Set LB=z_0^*, update $\overline{z}_{jk}$, $\overline{y}_j$.

3. Stop and conclude with the optimal solution.

In any iteration of the Benders' algorithm, the optimal solution value of MP provides a lower bound on the optimal solution value of the main problem, which monotonically increases at every iteration. The solution value of each SP, on the other hand, is an upper bound, not necessarily decreasing at each iteration. This is why the upper bound is chosen as UB=min$\{z^*$,UB$\}$ in Step (2a) of the algorithm, where z^* is as defined therein.

Note that MP is a mixed integer linear program with $|J| + |J||K|$ binary variables and a single continuous variable. Although MP has a number of binary variables much smaller than the original problem, our preliminary computational experiments show that it is still difficult to solve directly. Since a MP needs to be solved at each iteration of the algorithm, this may turn out to be quite costly. To overcome this drawback, we make use of a modification that will help to accelerate the procedure.

3.2.3.1 Modified Algorithm

The modification, as proposed in [71], partitions the procedure into two stages. In the first stage, only the LP-relaxations of the MPs are solved, where the relaxations are obtained by substituting the integrality conditions on y_j and z_{jk} by $0 \leq y_j \leq 1$ and $0 \leq z_{jk} \leq 1$, respectively. This procedure is continued (a) until no further iterations are possible (which indicates that we have found the optimal solution value of the LP-relaxation of the original problem), (b) for a prespecified number of iterations, or (c) when the gap between the upper and lower bounds is less than a prespecified amount. One can then switch to the second stage, in which the mixed-integer MP is solved. Note that this modification results in the optimal solution of the model provided that the "integer" MP is used in the algorithm or otherwise, the modified Benders' algorithm only ends up with the optimal LP-relaxation value of the original model. We refer to this modification as algorithm MB1. It should also be noted that the complexity of algorithm MB1 is proportional to the complexity of the mixed integer MP (i.e. number of integer

variables) that is solved in each iteration of the second stage.

3.2.4 A Heuristic Algorithm

Considering the size of the instances that may arise in designing a CDN, we also present a fast and simple heuristic algorithm for the solution of the SCDNP. Given any ordering $\{1, 2, \ldots, |J|\}$ of the set of potential proxy locations, the heuristic begins with opening the first proxy in the first iteration and continues on opening the i^{th} proxy of the ordering in the i^{th} iteration, until all the potential proxies are opened. The ordering chosen here can be such that the proxies are sorted in the nondecreasing order of f_j's or the nonincreasing order of s_j's. At every iteration, the request routing and object replication problems are solved. The former is done through assigning each client to the proxy with minimum total cost. The latter is based on what we refer to here as the *saving* of an object $k \in K$, calculated as follows:

$$\pi_{jk} = b_k c_{jS} \sum_{N(j)} d_{ik} \tag{3.28}$$

where $N(j)$ denotes the set of clients connected to proxy j. A similar measure has also been used by Xuanping et al. [100]. Verbally, the saving of an object is the amount of cost reduced by placing object k on a proxy j. Then, the objects are sorted in the non-increasing order of their savings. Let $\{O_{j1}, O_{j2}, \ldots\}$ denote this order. The objects are placed in the available proxy server(s) using this order without violating the capacity constraints. We also use $b(L)$ to denote $\sum_{k \in L} b_k$. The outline of the procedure is given in the following:

procedure GH

Let the best solution be $\overline{c} = +\infty$ and $l = 1$.

Repeat the following while $l \leq |J|$.

Select the first l sites to be opened, denoted by $J \supseteq \overline{J} = \{1, \ldots, l\}$.

for each client $i \in I$

let $x_{ij^*} := 1$ such that $\sum_{k \in K} b_k d_{ik} c_{ij^*} = min_{j \in \overline{J}}(\sum_{k \in K} b_k d_{ik} c_{ij})$

for each server $j \in \overline{J}$

Sort the objects and let the ordering be $\{O_{j1}, O_{j2}, \ldots, O_{j|K|}\}$.

$L := \emptyset$.

$t := 1$

while $b(L) \leq s_j$

begin

$k := O_{jt}$

if $b(L \cup \{k\}) \leq s_j$

$z_{jk} := 1$

$L := L \cup \{k\}$

$t \leftarrow t + 1$

end

Calculate the cost of the current solution c^l.

If $c^l < \overline{c}$, set $\overline{c} = c^l$.

$l \longleftarrow l + 1$.

Here, set L given in the algorithm is used to record the set of objects located on a proxy, for each $j \in J$. We name this procedure as the *greedy heuristic* and denote it by *GH*.

3.2.5 Computational Results

In this section, we report our computational experience with the decomposition algorithm and the heuristic procedure presented in the previous sections. All the algorithms were implemented in C on a Sun UltraSPARC 12x400 MHz with 3 GB RAM. State-of-the-art optimization software CPLEX 9.0 was used to solve the linear and integer programs at each step of MB1. For this algorithm, we have set the convergence parameter to $\epsilon = 0.01\%$ (i.e. the algorithm stops as soon as the gap between the upper and lower bounds is less than 0.01%). The algorithm was specified to switch from the first stage to the second stage (i.e. start to solve the MP as a mixed-integer problem) when the gap between the upper and lower bounds falls under 5%. We have also imposed an upper limit on the number of integer MPs that can be solved in MB1, which is 200.

One alternative way to solve the proposed model is to use a commercial software. We have chosen to use the state-of-the-art optimization software CPLEX 9.0 as a benchmark to assess the performance of MB1. For comparison purposes, a time limit of 10800 seconds (3 hours) was imposed on CPLEX.

The performance of the algorithm was tested on random instances, which are generated using the parameters explained in Section 3.2.2. For the experiments, a batch of sixty problems were generated with the following specifications: The number of potential proxy locations ($|J|$) range between 2 and 3. That of clients ($|I|$) range between 50 and 200 and that of objects ($|K|$) range between 10 and 20. There are 12 different combinations of these dimension parameters as may be seen in the first three columns of Table 3.5. The numerical values in each row are calculated over 5 problems for each configuration.

The average final gap, the average number of iterations, the average number of integer MPs solved and the average time required by algorithm MB1 to solve the instances are given under the columns *Gap*, *Iter*, *Int*, and t_{MB1} respectively. The next column, t_C, corresponds to the average time required for CPLEX 9.0 to solve the instances. In the case that either algorithm MB1 or CPLEX exceed the predefined limit, we consider the value of the best feasible solution obtained

Table 3.3: Comparison results of MB1, CPLEX and GH

$\|J\|$[1]	$\|I\|$[2]	$\|K\|$[3]	Gap[4]	$Iter$[5]	Int[6]	t_{MB1}[7]	t_C[8]	$d_{C/MB1}$[9]	$d_{C/GH}$[10]	$d_{MB1/GH}$[11]
2	50	10	0.0000	88	60	31	80	0.00	0.55	0.55
2	50	10	0.0046	18	17	2	8	0.00	0.00	0.00
2	50	10	0.0000	86	58	21	161	0.00	2.50	2.50
2	50	10	0.0000	30	2	4	7	0.00	2.62	2.62
2	50	10	0.0000	32	31	5	10	0.00	0.23	0.23
Average			**0.0009**	**50.8**	**33.6**	**12.6**	**53.20**	**0.00**	**1.18**	**1.18**
2	100	10	0.0000	115	82	66	729	0.00	7.85	7.85
2	100	10	0.0000	23	6	6	15	0.00	3.19	3.19
2	100	10	0.0000	170	127	160	10802*	0.36	2.99	3.36
2	100	10	0.0000	42	41	13	64	0.00	2.95	2.95
2	100	10	0.0000	37	14	12	62	0.00	0.83	0.83
Average			**0.0000**	**77.4**	**54**	**51.4**	**2334.40**	**0.07**	**3.56**	**3.63**
2	150	10	0.0002	17	16	7	25	0.00	1.87	1.87
2	150	10	0.0003	10	8	3	2	0.00	0.32	0.32
2	150	10	0.0000	14	2	4	6	0.00	0.31	0.31
2	150	10	0.0000	49	48	19	296	0.00	0.41	0.41
2	150	10	0.0000	141	121	85	10813*	0.00	3.23	3.23
Average			**0.0001**	**46.2**	**39**	**23.6**	**2228.40**	**0.00**	**1.23**	**1.23**
2	200	10	0.0009	33	8	19	173	0.00	0.77	0.77
2	200	10	0.0000	32	2	19	73	0.00	3.62	3.62
2	200	10	0.3509	201	200	237	1556	0.00	3.86	3.86
2	200	10	6.4124	248	200	503	3996	0.14	4.61	4.46
2	200	10	5.9014	201	200	287	8925	0.00	4.69	4.69
Average			**2.5331**	**143**	**122**	**213**	**2944.60**	**0.03**	**3.51**	**3.48**

[1] number of potential proxy servers, [2] number of clients, [3] number of objects, [4] final optimality gap between upper and lower bounds, [5] number of total iterations, [6] number of integer master problems solved, [7] total computation time spent by MB1, [8] total computation time spent by CPLEX 9.0, [9] percentage deviation of the final solutions found by MB1 and CPLEX, [10] percentage deviation of the final solutions found by GH and CPLEX, [11] percentage deviation of the final solutions found by MB1 and GH, * Indicates that the solution time of CPLEX for the corresponding problem exceeded 3 hours (10800 seconds) without reaching the optimal solution

Table 3.4: Comparison results of MB1, CPLEX and GH

$\|J\|$	$\|I\|$	$\|K\|$	Gap	$Iter$	Int	t_{MB1}	t_C	$d_{C/MB1}$	$d_{C/GH}$	$d_{MB1/GH}$
2	50	20	0.0096	121	74	158	98	0.00	0.31	0.31
2	50	20	16.3910	201	200	894	5940	0.14	1.63	1.48
2	50	20	27.1888	256	200	2976	10804*	0.72	8.46	7.67
2	50	20	12.2098	201	200	447	2331	0.00	8.06	8.05
2	50	20	23.0422	307	200	9401	9861	0.29	5.46	5.16
Average			**15.7683**	**217.2**	**174.8**	**2775.2**	**5806.80**	**0.23**	**4.78**	**4.53**
2	100	20	0.0012	51	30	29	39	0.00	1.51	1.51
2	100	20	5.1648	204	200	312	864	0.00	0.00	0.00
2	100	20	0.0016	67	66	53	61	0.00	1.70	1.70
2	100	20	20.8517	249	200	5320	10805*	0.44	0.59	1.03
2	100	20	3.1150	259	200	693	3078	0.00	3.28	3.28
Average			**5.8269**	**166**	**139.2**	**1281.4**	**2969.40**	**0.09**	**1.42**	**1.50**
2	150	20	14.1654	201	200	722	10802*	0.00	1.31	1.31
2	150	20	27.2585	288	200	7328	10802*	1.29	0.42	1.71
2	150	20	34.5923	201	200	4098	10803*	0.38	5.44	5.84
2	150	20	3.2074	215	200	688	10806*	0.00	4.26	4.26
2	150	20	5.8157	211	200	418	10802*	0.05	0.15	0.20
Average			**17.0078**	**223.2**	**200**	**2650.8**	**10803.00**	**0.34**	**2.32**	**2.66**
2	200	20	15.1687	221	200	1667	10801*	0.10	1.77	1.87
2	200	20	8.2099	222	200	699	10801*	0.01	0.95	0.95
2	200	20	0.0001	17	16	13	62	0.00	0.22	0.22
2	200	20	0.0009	22	21	16	55	0.00	0.21	0.21
2	200	20	17.1616	151	100	1930	10802*	0.13	1.62	1.75
Average			**8.1083**	**126.6**	**107.4**	**865**	**6504.20**	**0.05**	**0.95**	**1.00**

*: Indicates that the solution time of CPLEX for the corresponding problem exceeded 3 hours (10800 seconds) without reaching the optimal solution

Table 3.5: Comparison results of MB1, CPLEX and GH

$\|J\|$	$\|I\|$	$\|K\|$	Gap	$Iter$	Int	t_{MB1}	t_C	$d_{C/MB1}$	$d_{C/GH}$	$d_{MB1/GH}$
3	50	10	23.5536	275	200	850	573	0.00	3.30	3.29
3	50	10	2.3218	245	200	270	926	0.00	5.28	5.28
3	50	10	23.2103	628	200	3489	171	0.01	5.49	5.48
3	50	10	0.0044	119	66	68	19	0.00	1.60	1.60
3	50	10	0.4282	254	200	187	78	0.00	3.21	3.21
Average			**9.9037**	**304.2**	**173.2**	**972.8**	**353.40**	0.00	3.77	3.77
3	100	10	18.0875	202	200	279	359	0.00	0.00	0.00
3	100	10	6.0378	292	200	763	1361	0.00	1.24	1.24
3	100	10	11.7135	272	200	801	5102	0.00	5.87	5.86
3	100	10	21.3673	215	200	373	3623	0.00	0.66	0.66
3	100	10	11.9358	246	200	482	275	0.00	0.11	0.11
Average			**13.8284**	**245.4**	**200**	**539.6**	**2144.00**	**0.00**	1.58	1.57
3	150	10	0.0023	253	175	398	400	0.00	2.60	2.60
3	150	10	12.0348	204	200	271	235	0.00	3.20	3.20
3	150	10	0.0012	69	15	47	202	0.00	1.89	1.89
3	150	10	0.0022	113	38	166	748	0.00	1.13	1.13
3	150	10	20.8257	266	200	681	10145	0.33	1.04	0.71
Average			**6.5732**	**181**	**125.6**	**312.6**	**2346.00**	**0.07**	1.97	1.90
3	200	10	0.0039	146	82	180	691	0.00	0.02	0.02
3	200	10	0.0006	71	70	63	18	0.00	0.75	0.75
3	200	10	35.4799	293	200	2140	10807*	1.43	0.08	1.34
3	200	10	26.0671	269	200	1362	10800*	0.10	1.45	1.35
3	200	10	1.135656	276	200	2475	4076	0.86	0.86	0.00
Average			**12.5374**	**211**	**150.4**	**1244**	**5278.40**	**0.33**	**0.88**	**0.55**

*: Indicates that the solution time of CPLEX for the corresponding problem exceeded 3 hours (10800 seconds) without reaching the optimal solution

so far as the output of the algorithm.

As far as the results given in Table 3.5 are concerned, we see that algorithm MB1 is able to solve to optimality problems with $|I| = 2$, $|K| = 10$ and $|J| \leq 150$ considerably faster than CPLEX. We observe that as the instances increase in size, the final gap of algorithm MB1 increases as well. For this purpose, we additionally provide column $d_{C/MB1}$, which shows the percent difference between the final solutions found by MB1 and CPLEX for each instance. This value is calculated by $\frac{v_C - v_{MB1}}{v_{MB1}} * 100$, where v_C and v_{MB1} are the values of the solutions found by CPLEX and MB1, respectively. The differences presented in this column indicate that the solutions found by both algorithms do not significantly differ. However, algorithm MB1 is able to obtain such solutions in substantially shorter computation times as compared to those of CPLEX. The reason for the increasing gap is due to the linearization used here, which is shown to output very large integer gaps, as previously presented.

Table 3.5 also includes the results obtained by the GH, given in the last two columns. The columns $d_{C/GH}$ and $d_{C/MB1}$ respectively present the percent deviations of the *GH* from that of CPLEX and MB1. These are calculated by $\frac{v_{GH} - v_C}{v_C} \cdot 100$ and $\frac{v_{GH} - v_{MB1}}{v_{MB1}} \cdot 100$, respectively, where v_{GH} is the value of the solutions found by *GH*. As indicated in these two columns, the heuristic is able to find solutions that are within at most 5% of the solutions found by the exact solution algorithms. Since this heuristic runs very fast (within a few seconds for all the instances given in the table), we have not recorded the solution times.

We have also tried to solve instances with $|J| \geq 4$ using the proposed algorithms. However, for such instances, computational results showed that MB1 was not able to produce satisfactory solutions, which is mainly due to high gaps output by the algorithm.

3.3 Justification of the Combined Approach

We have mentioned in Chapter 3 that designing CDNs essentially involves solving three main problems, which are proxy location, object replication and client assignment. In practice, the replication of objects is done according to the popularity of each object. The clients' content requests, on the other hand, are directed in real-time to the best proxy that is the fastest to respond to the request. Details for commercial CDNs were supplied in Chapter 2. The current implementations in practice suggest a simple two step approach where the proxies are installed according to some optimization criteria in the first stage and the content is replicated according to the locations of the proxies and the 80/20 rule afterwards.

Contradicting the existing studies, we believe the three important decision problems in a CDN should be considered simultaneously. These three problems are inter-related in that the object replication and the client assignment is done according to the installed proxies and the placement of the proxies is affected by the replicated objects on the proxy as well as the clients assigned. However, the efficiency of such a combined approach over the two-stage approach, taking into consideration all the decision problems simultaneously, should be investigated.

In what follows, we will examine this specific argument by comparing the approach used in practice with the proposed (combined) approach. The combined approach was presented and modelled in the Chapter 3 for single and multi-origin server networks. In this chapter, we only consider networks with a single origin server, i.e. the problem SCDNP.

3.3.1 A Two-Stage Approach

Based on the approach used by commercial CDN providers in practice, we propose a two-stage approach formally stated below. The definitions and notations follow that of the preceding chapter.

Procedure `TWOSTAGE`:

1. *Stage* 1: Solve an Uncapacitated Facility Location Problem (UFLP) on the network. The solution to this problem will output the locations of the proxies to be installed with minimum total cost, as well as the assignment of clients to each installed proxy.

2. *Stage* 2: Calculate the popularity of each object $k \in K$ as $pop(k) = \sum_{i \in I} d_{ik}$. Locate only top 20% of the objects to each of the installed proxies with respect to their popularity, without violating the capacities.

Simply put forward, the procedure `TWOSTAGE` involves solving an integer linear model of the UFLP for the proxy location and client assignment decisions. Then, object replication is performed in the second stage, based on the output of the first stage and the 80/20 scenario.

3.3.2 A Combined Approach

The combined approach is actually the solution of the SCDNP presented in Chapter 3. Note that we may use the proposed model to solve the problem. However, for comparison purposes, we use the greedy algorithm.

3.3.3 Computational Results

In this section, we computationally compare both approaches explained above on randomly generated instances. The computational setting and all the details are as explained in Chapter 3. The problems have varying parameters, as shown in the first three columns of Table 3.3.3. Columns v(`TWOSTAGE`) and v(`COMBINED`) present the normalized costs obtained by using the respective procedures. For each problem configuration, five instances have been generated and each row presents the values averaged over these five instances. The last column name *imp*

presents the improvement achieved by using COMBINED over the solution found by TWOSTAGE and calculated as:

$$imp = \frac{v(\texttt{TWOSTAGE}) - v(\texttt{COMBINED})}{v(\texttt{TWOSTAGE})} * 100$$

In this table, we do not report the computational solution times required by each approach, since neither requires a significant amount of solution time for all of the instances considered.

As the results given in Table 5.1 indicate, the suggested COMBINED approach is highly superior to TWOSTAGE approach, especially for larger instances with $|J| = 20, 50$ and $|I| \geq 100$. Noting that the COMBINED approach is heuristic in nature and only results in an upper bound for the problem, we conclude that the savings could be even higher once the optimal solution of the problem is calculated (if possible). For some of the smaller instances, i.e. with $n = 5$ and $m = 50$, the TWOSTAGE approach results in a better solutions as compared to that obtained by the COMBINED approach. This, however, is an expected situation since in this case the number of potential proxies is very small and no significant savings by the heuristic can be obtained by deploying additional proxies in the network.

3.3.4 Discussion

In this subsection, we have tried to justify the approach of considering all three important decisions in a CDN design simultaneously. We have demonstrated through computational experiments that the combined approach can result in substantially better solutions than a straightforward two-stage approach. The two-stage approach, as inspired from practice, is clearly bound to result in a sub-optimal solution, since the decision problems are solved in a subsequent fashion, making the solution of the second stage dependent on that of the first one. As a result of the experiments, we can conclude that, for a network with a large number of potential proxy sites and objects, the combined approach is worthwhile in terms of the total network cost.

Table 3.6: Comparison of TWOSTAGE and COMBINED approaches

$\|J\|$[1]	$\|I\|$[2]	$\|K\|$[3]	v(TWOSTAGE)[4]	v(COMBINED)[5]	imp[6]
2	10	50	0.307036	0.253901	17.31
2	10	60	0.298925	0.238017	20.38
2	10	70	0.276902	0.260607	5.88
2	10	80	0.238348	0.227061	4.74
2	10	90	0.301522	0.253954	15.78
2	10	100	0.302111	0.242322	19.79
5	50	50	0.080986	0.078015	3.67
5	50	60	0.078979	0.101588	-28.63
5	50	70	0.081200	0.086612	-6.66
5	50	80	0.081916	0.080837	1.32
5	50	90	0.111066	0.108187	2.59
5	50	100	0.091940	0.113033	-22.94
20	100	500	0.031825	0.023669	25.63
20	100	600	0.034662	0.021637	37.58
20	100	700	0.037372	0.020593	44.90
20	100	800	0.036329	0.022581	37.84
20	100	900	0.034300	0.020798	39.36
20	100	1000	0.030742	0.018878	38.59
50	500	500	0.015379	0.009962	35.23
50	500	600	0.015970	0.009941	37.75
50	500	700	0.016094	0.010996	31.68
50	500	800	0.015397	0.008003	48.02
50	500	900	0.015509	0.009297	40.06
50	500	1000	0.015506	0.008827	43.07

[1] number of potential proxy servers, [2] number of clients, [3] number of objects, [4] solution value found by the TWOSTAGE approach, [5] solution value found by the COMBINED approach, [6] percentage improvement of the COMBINED approach over the TWOSTAGE approach

3.4 Conclusions and Further Issues

Content Distribution Network is a new technology aimed at increasing the effectiveness of the distributing content. In this chapter, we have developed integer programming models to optimally locate proxy servers, allocate each object to a proxy server and assign clients to the proxies so as to minimize the total transfer cost of the content. Our approach differs from the related studies considering a similar problem with respect to the objective function, which allows one to consider the case when the requested object is not found in any proxy server.

The potential savings of such a joint approach is also investigated in this chapter. Computational results suggest that the approach presented herein is superior to a two-stage approach that seems to be used in practice.

We have devised exact and heuristic solution approaches for the single server CDN design problem. Since the proposed model includes a quadratic objective function, we have made use of a linearization in devising a decomposition-based exact algorithm, along with a variant to accelerate it. Computational experiments based on randomly generated Internet topologies suggest that the proposed algorithm is superior to CPLEX and is very efficient especially when the number of clients is huge and the number of objects is comparably small (as generally is the case in multimedia environments). The algorithm is also a demonstration of an exact solution technique for similar integer models with quadratic objective functions.

For larger networks, we have also proposed a greedy algorithm that runs very fast. Comparison results with the exact algorithm suggests that the heuristic algorithm is able to output near-optimal solutions to the problem. The greedy algorithm proposed here is simple and is based on the concept of savings. However, one may go further in this direction to investigate the potentials of other heuristic procedures for this problem. In specific, metaheuristics such as tabu search or genetic algorithms may prove to be useful in obtaining better solutions in considerably small running times.

It is clear that the problems considered here are of dynamic nature. In specific, the level and the distribution of traffic tend to vary in a telecommunications network, which is partly due to the stochastic nature of customer demand and partly due to the variability in usage pattern over the course of a day. The aim of a CDN provider is to configure the system efficiently given the demand of each client node. Therefore, the system should be reconfigured periodically, and this reconfiguration is especially needed when the request rates of the clients for the objects change significantly. The time intervals for reconfiguration of the system is also dependent on the nature of the content that is being distributed. There would be some applications that are very time sensitive where we may also have applications (like online newspapers, etc.) that are not very time sensitive.

To accomplish this task, the algorithm presented in this chapter needs to run as frequently as possible to achieve the highest availability and performance. This should be in the order of minutes and further optimization to minimize this time is desirable. This may be achieved by parallel running of the algorithm or further enhancing of the methodologies suggested in this work.

Chapter 4

Video on Demand

In this chapter, we develop a solution algorithm for the *Video Placement and Routing Problem* that arises in local groups of VoD systems. Our algorithm is based on Lagrangean relaxation and decomposition, coupled with some integer programming techniques to convert infeasible solutions to feasible solutions. This approach is in contrast to similar algorithms in the literature, where heuristics are used to obtain feasible solutions.

The outline of this chapter is as follows. In the next section, we define the problem and present an integer linear programming formulation. Section 4.2 provides the full details of the Lagrangean relaxation and decomposition algorithm, with the implementation results on randomly generated instances given in Section 4.3. Conclusions and further remarks are given in Section 4.4.

4.1 Problem Definition And Formulation

In defining the problem, we adhere to the notation of Ouveysi et al [80], given as follows. There exists a fully meshed network modelled by an undirected graph $G = (V, A)$ corresponding to a cluster of local servers in a VoD network. Here, $V = \{1, 2, ..., n\}$ is the set of nodes and A is the set of edges including the

$n(n-1)/2$ links of the network. The set of programs (videos) to be placed and routed is denoted by $P = \{1, 2, ..., m\}$, where each program $k \in P$ has a capacity requirement denoted by m_k and a bandwidth requirement for transmission denoted by μ_k. Each node $j \in V$ corresponds to a potential location for storing the programs with capacity denoted by $C_s(j)$. In addition, each node has a demand for each program. The cost of storing a program $k \in P$ at node $j \in V$ is shown by $s_k(j)$ and the transmission cost of the program on the link $\{i, j\} \in A$ is shown by $t_k(i, j)$. Finally, each link $\{i, j\} \in A$ has a transmission capacity that is denoted by $C_t(i, j)$. A fully meshed group of a VoD architecture with 5 local servers is given in Figure 4.1.

Given the demand forecast of the programs, the *Video Placement and Routing Problem* (*VRPR*) is to find a placement scheme for the programs such that the total cost of storage and transmission of the programs in the network is minimized and the demand of each node for each program is satisfied.

Ouveysi et al. [80] provide an integer linear programming formulation for this problem, using the following binary decision variables:

$$x_{ij}^k = \begin{cases} 1, & \text{if program } k \in P \text{ is transmitted to node } j \in V \text{ from node } i \in V \\ 0, & \text{otherwise} \end{cases}$$

$$y_j^k = \begin{cases} 1, & \text{if program } k \in P \text{ is stored at node } j \in V \\ 0, & \text{otherwise} \end{cases}$$

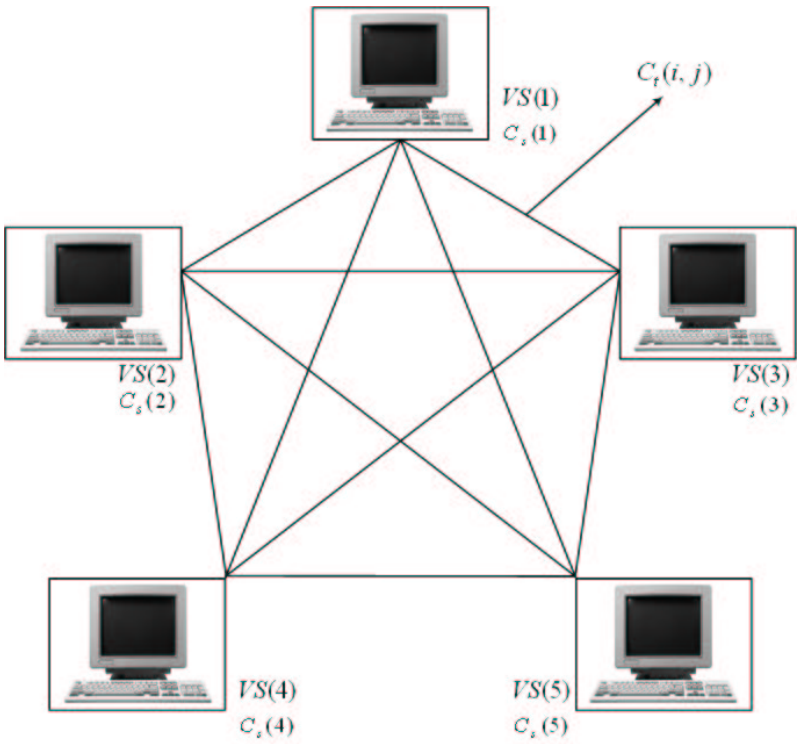

Figure 4.1: A fully meshed VoD architecture with 5 servers

The formulation is then given as follows (denoted by **F**):

$$\textbf{(F)} \qquad \text{minimize} \sum_{k\in P}\sum_{j\in V} s_k(j)y_j^k + \sum_{k\in P}\sum_{j\in V}\sum_{i\in V, i\neq j} t_k(i,j)x_{ij}^k \qquad (4.1)$$

s.t.

$$\sum_{k\in P} m_k y_j^k \leq C_s(j), \quad \forall j \in V \qquad (4.2)$$

$$\sum_{k\in P} \mu_k x_{ij}^k \leq C_t(i,j), \quad \forall i \neq j \in V \qquad (4.3)$$

$$\sum_{i\in V, i\neq j} x_{ij}^k + y_j^k = 1, \quad \forall j \in V, k \in P \qquad (4.4)$$

$$x_{ij}^k \leq y_i^k, \quad \forall i \neq j \in V, k \in P \qquad (4.5)$$

$$x_{ij}^k, y_j^k \in \{0,1\}, \quad \forall i \neq j \in V, k \in P \qquad (4.6)$$

In this model, constraints (4.2) and (4.3) correspond to the capacity constraints related with storage nodes and transmission links, respectively. Constraints (4.4) state that each node either stores a program or receives it from another node that stores it. Finally, constraints (4.5) imply that a program can be transmitted from a node only if the program is stored at that node. It is clear that this formulation only allows one-hop paths in transmitting a program. The reader is referred to Ouveysi et al. [81] for the generalization of this problem to two-hop paths. Constraints (4.6) impose integrality restrictions on the decision variables. Problem **F** has in general $(n^2m + nm)/2$ binary variables and $(n^2m + n^2 + n)/2$ constraints.

We now study the complexity of the VPRP. Ouveysi et al. [80] mention the possibility that the VPRP may be $\mathcal{NP}$-Hard. In the proposition stated below, we prove that it indeed is:

Proposition 8 *The video placement and routing problem (VPRP) is $\mathcal{NP}$-Hard.*

Proof The proof is based on the following restriction. Consider a single program (i.e., $P = \{1\}$) and let $\mu_1 \leq min_{\{i,j\}\in A} C_t(i,j)$ and $m_1 \leq min_{i\in V} C_s(i)$. Since

there is a single program, we can drop the index k in the formulation. In this case, constraints (4.2) and (4.3), pertaining to node and link capacities, become redundant. Now, partition the node set such that $V = I \bigcup J$ where $y_j \leq 0$ for all $j \in J$. Then, constraints (4.4) and (4.5) can be written as $\sum_{i \in I} x_{ij} = 1, \forall j \in J$ and $x_{ij} \leq y_i, \forall i \in I, j \in J$. But then **F** reduces to the well-known uncapacitated facility location problem (see [28]) with I as the set of potential facility locations and J as the set of customers. Since this problem is known to be $\mathcal{NP}$-Hard, the VPRP is also $\mathcal{NP}$-Hard. □

The complexity of the VPRP implies that the solution of **F** using standard off-the-shelf software will not be practical, especially with the increasing size of the problem. Ouveysi et al. [80] have proposed a heuristic to solve problem **F**. We propose a solution algorithm for problem **F** based on Lagrangean relaxation and decomposition. The details of the algorithm are presented in the next section.

4.2 A Lagrangean Relaxation and Decomposition Algorithm

Our algorithm is based on relaxing the capacity constraints (4.2) and (4.3) in a Lagrangean fashion, by respectively associating the Lagrange multipliers β_j and α_{ij} to each constraint. As a result, we obtain the following relaxed problem (denoted by $F(\beta, \alpha)$):

$$(F(\beta,\alpha)) \qquad \text{minimize} \sum_{k \in P} \sum_{j \in V} A_j^k y_j^k + \sum_{k \in P} \sum_{j \in V} \sum_{i \in V, i \neq j} B_{ij}^k x_{ij}^k - C_0$$

$$\text{s.t.}$$

$$(4.4), (4.5), (4.6)$$

where

$$\begin{aligned} A_j^k &= s_k(j) + \beta_j m_k, \\ B_{ij}^k &= t_k(i,j) + \alpha_{ij}\mu_k, \text{ and} \\ C_0 &= \sum_{j\in V} \beta_j C_s(j) + \sum_{i\in V}\sum_{j\in V} \alpha_{ij} C_t(i,j). \end{aligned}$$

Next, we observe that $F(\beta,\alpha)$ decomposes into $|P|$ subproblems, one for each program $k \in P$, each denoted by $F_k(\beta,\alpha)$ and shown as follows for a single k^*:

$$\begin{aligned} (F_{k^*}(\beta,\alpha)) \qquad & \text{minimize } \sum_{j\in V} A_j^{k^*} y_j^{k^*} + \sum_{j\in V}\sum_{i\in V, i\neq j} B_{ij}^{k^*} x_{ij}^{k^*} \\ & \text{s.t.} \\ & \sum_{i\in V, i\neq j} x_{ij}^{k^*} + y_j^{k^*} = 1, \quad \forall j \in V \\ & x_{ij}^{k^*} \leq y_i^{k^*}, \quad \forall i \neq j \in V \\ & x_{ij}^{k^*}, y_j^{k^*} \in \{0,1\}, \quad \forall i \neq j \in V \end{aligned}$$

Each subproblem has $(n^2+n)/2$ binary variables and $(n^2+n)/2$ constraints. Let $v(F)$ denote the optimal objective function value of problem **F**. Preliminary computational experimentation shows that the subproblems do not have any integrality properties, i.e., the solution to the LP-relaxation of the subproblems may output fractional solutions.

As a result of the decomposition procedure, the optimal objective function of $F(\beta,\alpha)$ can be calculated as $v(F(\beta,\alpha)) = \sum_{k\in P} v(F_k(\beta,\alpha)) - C_0$. We are now ready to provide a general outline of the algorithm for the solution to problem **F**.

The Algorithm:

- Start with an initial vector of multipliers β^1, α^1. Let the incumbent lower

bound be $lb = -\infty$, incumbent upper bound be $ub = \infty$ and $t = 1$.

- Perform the following until $gap = \frac{ub-lb}{ub} < 0.01$ or the maximum amount of iterations have been reached.
  - Solve $F(\beta^t, \alpha^t)$. Set $lb = v(F(\beta^t, \alpha^t))$ if $v(F(\beta^t, \alpha^t)) > lb$.
  - Modify the solution of $F(\beta^t, \alpha^t)$ into a feasible solution $\widehat{F}(\beta^t, \alpha^t)$ using the two-stage procedure that will be explained later on. If $v(\widehat{F}(\beta^t, \alpha^t)) < ub$, set $ub = v(\widehat{F}(\beta^t, \alpha^t))$.
  - Update the multipliers as follows:

$$\beta^{t+1} = max\{0, \beta^t + s_1^t \cdot g_1^t\}$$
$$\alpha^{t+1} = max\{0, \alpha^t + s_2^t \cdot g_2^t\}$$

    Here, g_1^t and g_2^t are the subgradient vectors. The j^{th} component of g_1^t is defined as:

$$(g_1^t)_j = \sum_{k \in P} m_k y_j^k - C_s(j)$$

    Similarly, the $(i,j)^{th}$ component of g_2^t is defined as:

$$(g_2^t)_{ij} = \sum_{k \in P} \mu_k x_{ij}^k - C_t(i,j)$$

    In updating the multipliers, the step sizes s_1^t and s_2^t are calculated as follows:

$$s_i^t = \lambda \frac{1.05 \cdot ub - v(V(\beta^t, \alpha^t))}{\|g_i^t\|^2}, \qquad i = 1, 2 \tag{4.7}$$

  - Increment t as $t + 1$.

- Output ub as the best feasible solution.

In calculating the steplengths, equation (4.7) is used where λ is a convergence parameter. This equation is the classical formula given in Held et al. [45]. In this

equation, the coefficient of the incumbent upper bound (ub) is chosen as 1.05, although it is generally considered 1 in the literature. The purpose for choosing such a coefficient is to increase the value of the step sizes at every iteration, with an expectation of obtaining a better solution at the next iteration. The value of this coefficient has been optimized by performing additional computational testing for values 1.05, 1.1, 1.15 and 1.2 as the coefficient and choosing the best one in terms of the solutions produced.

The *gap* calculated at each iteration of the algorithm shows how far the current feasible solution is from the optimal solution. Therefore, in the case that the algorithm is unable to find the optimal solution, it is capable of indicating the quality of the final solution.

At any step of the algorithm, the solution of $F(\beta^t, \alpha^t)$ provides an integral solution that is feasible with respect to constraints (4.4) and (4.5), but do not necessarily satisfy the capacity constraints (4.2) and (4.3). This (infeasible) solution needs to be converted into a feasible solution with respect to Problem **F** in order to be able to provide an upper bound. In general, in such circumstances, some fast heuristics are used to convert the infeasible relaxed solution to a feasible solution, at the expense of a possibly bad feasible solution. However, we consider a reverse approach and utilize integer programs (IPs) to obtain feasible solutions. Our motivation is to make use of the information provided by the current relaxed solution as much as possible. Such an approach, although at the expense of a higher computational effort, will be proven to be effective in providing feasible solutions of good quality. The details of our procedure is as follows:

4.2.1 Obtaining Feasible Solutions

Let $\widehat{y}_j^k$ and $\widehat{x}_{ij}^k$ be the optimal solution to the $F(\beta, \alpha)$. Using this solution, we attempt to achieve a feasible solution to **F** using a two stage procedure (named as $2SP$). In brief terms, the first stage of the $2SP$ attempts to obtain a feasible configuration of y variables using an IP named *FeasY*. Using the result of the first stage, we construct another IP named *FeasX* in the second stage, whose solution

provides a feasible configuration of x variables. Details are provided below:

4.2.1.1 Stage 1

The first stage of the $2SP$ consists of converting the $\widehat{y}_j^k$'s so as to satisfy constraint (4.2) with a minimal amount of modification. The modification is done for each node $j \in V$, through repositioning any program that violates the capacity constraint. To achieve this, we define the set $O(j,k) = \{j \in V, k \in P | \widehat{y}_j^k = 1\}$. Then, the feasibility is accomplished through the use of the following feasibility IP model (henceforth denoted by *FeasY*):

$$(FeasY) \qquad \text{minimize} \sum_{k\in P}\sum_{j\in V} s_k(j) y_j^k + \sum_{k\in P}\sum_{j\in V} R m_j^k \tag{4.8}$$

s.t.

$$\sum_{k\in P} m_k y_j^k \leq C_s(j), \quad \forall j \in V$$

$$y_j^k \geq 1 - m_j^k, \quad \forall j,k \in O(j,k) \tag{4.9}$$

$$\sum_{j\in V} y_j^k \geq 1, \quad \forall k \in P \tag{4.10}$$

$$y_j^k \in \{0,1\}, \quad \forall j \in V, k \in P$$

$$m_j^k \in \{0,1\}, \quad \forall j,k \in O(j,k) \tag{4.11}$$

In *FeasY*, the additional binary variable m_j^k is equal to one if program k on node j is repositioned to another node, with, however, a penalty for each repositioning. This is reflected in the second summation of the objective function of *FeasY* with a penalty coefficient R and ensures that the modification performed to the solution is minimal. The motivation for such an approach is to benefit as much as possible from the information provided by the relaxed Lagrangean solution. Constraints (4.9) stipulate that if a program k already located at node j is repositioned to another node, then $y_j^k = 0$. Constraints (4.10) are used to ensure that after the modification, each program is located on at least one node.

In short, the solution to *FeasY* yields a placement scheme for the programs such that no node constraint is violated. Our computational experience shows that *FeasY* is easily solved to optimality with standard optimization software.

We also note that in *FeasY*, it is possible to relax the binary variables m_j^k in the interval $[0, 1]$, since it is easy to see that in any optimal solution to *FeasY*, no m_j^k will attain a fractional value.

4.2.1.2 Stage 2

In the second stage of the $2SP$, we attempt to find a feasible configuration of x_{ij}^k variables, based on the optimal values of the variables $\overline{y}_j^k$ of *FeasY*. In other words, we would like to obtain a vector of x variables satisfying the following IP model (henceforth referred to as *FeasX*):

$$(FeasX) \qquad \text{minimize} \sum_{k\in P}\sum_{j\in V}\sum_{i\in V, i\neq j} t_k(i,j)x_{ij}^k \tag{4.12}$$

s.t.

$$\sum_{k\in P} m_k x_{ij}^k \leq C_t(i,j), \quad \forall i \neq j \in V : \overline{y}_i^k = 1, \overline{y}_j^k = 0$$

$$\sum_{i\in V, i\neq j, \overline{y}_i^k=1} x_{ij}^k = 1, \quad \forall j \in V, k \in P : \overline{y}_j^k = 0 \tag{4.13}$$

$$x_{ij}^k \in \{0,1\}, \quad \forall i \neq j \in V : \overline{y}_i^k = 1, \overline{y}_j^k = 0, k \in P \tag{4.14}$$

Note that in model $FeasX$, the binary variables x_{ij}^k are only defined if $\overline{y}_i^k = 1$ and $\overline{y}_j^k = 0$. Therefore, the size of the model is greatly reduced as compared to problem **F**. The optimal solution to the *FeasX* yields a feasible configuration of x_{ij}^k variables.

As a result of stages 1 and 2, we obtain the optimal objective values for the formulations *FeasY* and *FeasX*. The objective value of the corresponding (feasible) solution for problem **F** is then found through $v(FeasY) + v(FeasX) -$

$\sum_{k \in P} \sum_{j \in V} Rm_j^k$.

4.3 Computational Results

In this section, we describe our computational results with the proposed algorithm on randomly generated test problems. The Lagrangean relaxation and decomposition algorithm proposed for the solution of **F** has been implemented in C and all the tests are performed on a Sun UltraSPARC 12x400 MHz with 3 GB RAM, using CPLEX 9.0 as the optimization package to solve the IPs.

For the computational experiments, a batch of sixteen random problems have been generated with the number of nodes (n) ranging from 50 to 80, and the number of programs (m) ranging from 20 to 50. As for the parameters, $\mu_k, t_k(i,j), s_k(j)$ are randomly generated from a uniform distribution between 1 and 100. m_k is modelled as $m_k = \mu_k T_k$, where T_k is the total transmission time for program k. In the experiments, $T_k = 10$ min. for all k. $C_t(i,j)$ values have been chosen from the uniform distribution between $max_{i,j}\{t_k(i,j)\}$ and $\sum_{i,j}\{t_k(i,j)\}$). The capacity of each node ($C_s(j)$) is set to be 40% of the total size of all the programs and the penalty parameter R is set to $10max_{k \in P, j \in V}\{s_k(j)\}$.

Prior to solving the randomly generated problems, the algorithm's parameters have been fine-tuned and selected as follows: The convergence parameter λ is initially set to 2.00 and multiplied by 0.87 if there is not any improvement in the best known upper bound for 5 consecutive iterations. The algorithm is stopped if either $gap < 0.01$, $t > 100$, there is no improvement in the best known upper bound for 9 successive iterations, or a pre-specified time limit has been reached. All the subproblems are solved to optimality using CPLEX 9.0.

We present the computational results in Table 4.1. Each row of the table contains the average values of five randomly generated problems. The columns of the table are explained below:

- n: number of nodes

- m: number of programs
- n_L: number of iterations required by the algorithm
- t_{sub}: average time required to solve all the subproblems to optimality
- t_{FeasY}: average time required to solve $FeasY$ to optimality
- t_{FeasX}: average time required to solve $FeasX$ to optimality
- $igap$: initial gap obtained at the beginning of the algorithm
- gap: final gap obtained at the end of the algorithm
- d: comparison of the algorithm with CPLEX (more on this below)

It is previously stated that the algorithm proposed here is a solution procedure that is able to provide both upper and lower bounds at every iteration. This, in turn, outputs an integrality gap that is an indicator of the quality of the solution found. Therefore, we do not compare our algorithm with the heuristic procedure proposed by Ouveysi et al. [80]. However, we do compare it with CPLEX, a powerful commerical optimization package. To be fair in comparisons, we impose a common time limit of 300 seconds on both algorithms, considering the dynamic nature of the problem requiring repeated resolving to adopt to the changes in the demand pattern and available programs. The last column of Table 4.1, column d, shows the average percent difference between the best solution found by the proposed algorithm (denoted by v_{opt}) and that of CPLEX (denoted by v_C) within the given time limit, and calculated as $\frac{v_{opt}-v_C}{v_{opt}} \cdot 100$.

The results presented in Table 4.1 indicate that the algorithm presented here is able to produce good quality solutions, even in the first iteration, for most of the problems (typically around 2% of the optimal). In addition, the proposed algorithm is observed to be capable of providing better solutions than those found by CPLEX in the same amount of time, especially as the instances grow in size. As also indicated in Table 4.1, the time required to obtain feasible solution at every step of the algorithm is quite short. However, one drawback of the algorithm lies in obtaining lower bounds. The lower bound computation time, as shown

Table 4.1: Computational results for the Lagrangean relaxation and decomposition algorithm

n[1]	m[2]	n_L[3]	t_{sub}[4]	t_{FeasY}[5]	t_{FeasX}[6]	$igap$[7]	gap[8]	d[9]
50	20	13.2	13.78	0.12	0.53	3.23	2.03	0.82
60	20	12.4	23.91	0.15	0.78	4.04	2.20	0.91
70	20	7	36.45	0.17	1.08	3.37	1.82	0.12
80	20	6.4	58.20	0.20	1.39	2.92	2.45	-0.30
50	30	16.4	15.93	0.18	0.97	4.01	2.70	0.98
60	30	10	30.02	0.23	1.37	3.01	2.27	0.72
70	30	6.4	52.20	0.25	2.02	2.95	2.58	-2.74
80	30	4	76.55	0.31	2.47	2.65	1.99	-5.08
50	40	10.6	26.72	0.24	1.49	4.43	2.81	0.53
60	40	7.6	40.64	0.27	2.02	2.95	2.38	-0.56
70	40	5.4	61.94	0.34	2.77	2.56	2.26	-3.63
80	40	3.4	98.26	0.41	3.43	2.15	1.72	-4.80
50	50	9.4	33.75	0.30	2.01	3.28	2.67	-0.10
60	50	5.6	58.94	0.35	2.93	3.38	2.60	-2.61
70	50	4	90.00	0.40	3.83	2.92	2.56	-3.89
80	50	3	139.18	0.47	4.89	2.24	2.02	-4.07

[1] number of nodes,
[2] number of programs,
[3] number of iterations,
[4] average computation time (in seconds) to solve a subproblem to optimality,
[5] average computation time (in seconds) to solve *FeasY* to optimality,
[6] average computation time (in seconds) to solve *FeasX* to optimality,
[7] initial gap,
[8] final gap,
[9] percent difference between best solution found by the algorithm and that of CPLEX

under column t_{sub}, increases heavily with the number of nodes. This is due to the fact that, at every iteration, the algorithm needs to solve m integer subproblems to optimality. This, in turn, makes the algorithm computationally inefficient for larger sized instances, as the size and the number of subproblems will increase as well. To overcome this drawback, we propose a simple modification to the algorithm that appears to be quite efficient, as described in detail below.

4.3.1 A Modified Algorithm

Since solving m integer subproblems at every iteration of the algorithm is costly, we propose to solve the LP-relaxation of each integer subproblem. The lower bound obtained in this case will surely be below the lower bound obtained by solving integer subproblems, but will help in speeding up the algorithm. The only complication with this modification is that the solutions of the subproblems will in general be fractional, if not always. However, the fractional solutions can easily be converted to integer solutions that will be used to obtain feasible solutions to the original problem. This is done through rounding up (to 1) every fractional variable with value greater or equal to 0.50, and rounding down the rest. Using this solution, a feasible solution can easily be computed using formulations *FeasY* and *FeasX*, as discussed previously.

Our computational experience with this version of the algorithm shows that such a modification greatly helps in reducing the solution time of calculation of the lower bound at each iteration of the algorithm. To see this, we present Table 4.2, where we compare the original and the modified algorithm on some test problems. The first five columns are self explanatory. In the next three columns, we report the average computation time required to find lower bounds (denoted by $\widehat{t}_{sub}$), to solve problem *FeasY* (denoted by $\widehat{t}_{FeasY}$) and to solve *FeasX* (denoted by $\widehat{t}_{FeasX}$), respectively. Finally , the last column denoted by *imp*, shows the improvement in the solution time in finding lower bounds (calculated as $\frac{t_{sub}-\widehat{t}_{sub}}{t_{sub}} * 100$.)

One may expect that the performance of the modified algorithm in terms of the final gaps will deteriorate since one is solving linear programming problems

Table 4.2: Comparison of the original and modified algorithm in terms of computation time

		Original Algorithm			Modified Algorithm			
n[1]	m[2]	t_{sub}[3]	t_{FeasY}[4]	t_{FeasX}[5]	$\widehat{t}_{sub}$[6]	$\widehat{t}_{FeasY}$[7]	$\widehat{t}_{FeasX}$[8]	imp[9]
50	10	9.07	0.06	0.25	2.24	0.06	0.25	75.33
60	10	15.84	0.07	0.34	3.18	0.07	0.35	79.90
70	10	21.44	0.09	0.50	4.50	0.09	0.57	79.01
80	10	25.65	0.10	0.62	5.88	0.09	0.63	77.08
90	10	30.91	0.11	0.79	7.79	0.10	0.79	74.81
100	10	131.87	0.12	0.88	11.08	0.12	0.94	91.60
50	20	14.15	0.11	0.58	4.62	0.12	0.58	67.36
60	20	23.31	0.13	0.84	6.17	0.13	0.82	73.53
70	20	24.68	0.16	1.20	9.02	0.17	1.38	63.46
80	20	35.27	0.18	1.32	11.51	0.18	1.43	67.37
90	20	69.01	0.23	1.93	16.03	0.19	1.84	76.77
100	20	80.55	0.23	2.14	20.43	0.24	2.13	74.63
50	30	12.54	0.18	1.03	6.72	0.18	1.18	46.46
60	30	46.24	0.25	1.35	10.07	0.22	1.52	78.21
70	30	26.37	0.23	2.08	12.72	0.23	2.08	51.75
80	30	94.47	0.33	2.11	17.79	0.28	2.15	81.17
90	30	188.64	0.32	2.97	25.84	0.35	3.23	86.30
100	30	168.30	0.33	3.51	31.01	0.33	3.67	81.57
50	40	23.62	0.23	1.64	8.79	0.26	1.66	62.79
60	40	49.34	0.30	2.62	13.49	0.31	2.50	72.67
70	40	58.85	0.42	2.70	17.83	0.31	2.74	69.70
80	40	120.21	0.38	4.17	25.90	0.38	4.16	78.45
90	40	218.63	0.34	4.97	34.54	0.43	4.86	84.20
100	40	211.71	0.37	4.88	40.45	0.36	4.84	80.90

[1] number of nodes, [2] number of programs,
[3] average computation time (in seconds) to solve a subproblem to optimality by the original algorithm,
[5] average computation time (in seconds) to solve *FeasY* to optimality by the original algorithm,
[6] average computation time (in seconds) to solve *FeasX* to optimality by the original algorithm,
[7] average computation time (in seconds) to solve a subproblem to optimality by the modified algorithm,
[8] average computation time (in seconds) to solve *FeasY* to optimality by the modified algorithm,
[9] average computation time (in seconds) to solve *FeasX* to optimality by the modified algorithm

instead of integer programs. To see how much is lost in terms of the gaps produced by the modified algorithm with respect to the original algorithm, we provide additional results given in Table 4.3. In the table, the first two columns are self explanatory. Columns *igap* and *gap* refer to the initial and final gaps found by the original algorithm, respectively. Similarly, the last two columns $\widehat{igap}$ and $\widehat{gap}$ refer to the initial and final gaps found by the modified algorithm, respectively.

The numerical values given in Table 4.2 show that our modification proposal does not have any effect on reducing the computation times to obtain feasible solutions. However, it does have a tremendous effect in reducing the necessary computation times to find lower bounds. As indicated under column *imp*, these improvements can be up to 90%. Furthermore, results given in table Table 4.3 indicate that not much is lost with respect to the final gaps output by the algorithm. Based on these results, we proceed with solving larger instances using this modification and the corresponding results are presented in Table 4.4, where the number of nodes range from 50 to 100, and that of programs range from 50 to 90. In solving these instances, we keep all the algorithm parameters as explained previously. This time 30 problems have been generated, where 4 random instances for each configuration were solved. Each row of Table 4.4 contains the average values calculated over 4 random instances. The format of Table 4.4 is similar to that of Table 4.1, with respect to the modified algorithm. The only additional column is n_s, which denotes the number of instances out of 4 for which CPLEX was able to find an integer feasible solution.

Looking at the results in 4.1 we see that the modified algorithm provides good quality solutions (typically with a gap below 5%) in a reasonable amount of time. CPLEX, on the other hand, fails to find an integer feasible solution within the given time limit, as problems grow larger in size.

Table 4.3: Comparison of the original and modified algorithm in terms of the gap

		Original Algorithm		Modified Algorithm	
n[1]	m[2]	$igap$[3]	gap[4]	$\widehat{igap}$[5]	$\widehat{gap}$[6]
50	10	13.51	7.78	13.71	8.63
60	10	7.91	6.91	8.59	6.42
70	10	7.73	4.76	8.16	5.37
80	10	5.47	3.70	6.69	6.10
90	10	3.63	2.44	4.69	2.24
100	10	3.82	3.64	7.00	6.38
50	20	5.01	2.31	8.27	3.89
60	20	3.19	3.10	4.75	3.71
70	20	3.46	2.36	6.26	5.04
80	20	1.30	1.12	4.01	3.47
90	20	2.32	1.91	3.40	3.34
100	20	2.66	2.12	4.09	3.78
50	30	3.89	3.26	6.22	4.94
60	30	3.80	2.49	5.25	4.95
70	30	2.42	2.41	3.56	2.59
80	30	2.49	1.95	4.29	3.29
90	30	2.87	2.29	6.37	5.21
100	30	2.15	2.11	4.62	4.16
50	40	3.45	2.27	5.72	3.77
60	40	3.78	2.45	6.47	5.28
70	40	2.64	2.62	4.49	4.05
80	40	2.31	2.04	4.67	4.09
90	40	2.61	2.07	5.82	5.07
100	40	1.05	1.05	3.24	3.24

[1] number of nodes, [2] number of programs,
[3] initial gap found by the original algorithm,
[4] final gap found by the original algorithm,
[5] initial gap found by the modified algorithm,
[6] final gap found by the modified algorithm

Table 4.4: Computational results for the modified Lagrangean relaxation and decomposition algorithm

n[1]	m[2]	n_L[3]	$\widehat{t}_{sub}$[4]	$\widehat{t}_{FeasY}$[5]	$\widehat{t}_{FeasX}$[6]	$\widehat{igap}$[7]	$\widehat{gap}$[8]	d[9]	n_s[10]
50	50	20.25	10.94	0.32	4.36	6.25	5.41	2.55	4/4
60	50	15.75	16.02	0.41	3.47	5.84	5.35	-0.35	4/4
70	50	11	23.04	0.45	6.68	5.36	5.12	-2.34	4/4
80	50	9	31.09	0.44	4.96	4.33	4.29	-2.19	4/4
90	50	6.25	40.46	0.65	6.78	3.93	3.82	-2.52	1/4
100	50	5.25	51.72	0.72	7.96	4.07	3.82	–	0/4
50	60	17.5	13.27	0.47	3.70	6.86	6.22	0.23	4/4
60	60	12.5	19.66	0.47	4.62	5.33	5.01	-1.99	4/4
70	60	9.25	27.51	0.54	5.85	5.35	4.85	-1.47	4/4
80	60	7	36.86	0.67	7.16	3.98	3.92	-2.14	3/4
90	60	5.75	48.60	0.68	9.40	4.65	4.56	–	0/4
100	60	4.5	62.12	0.69	17.33	3.71	3.63	–	0/4
50	70	15.25	15.81	0.53	3.90	6.70	5.90	-0.40	4/4
60	70	10.5	23.40	0.52	6.59	4.82	4.72	-0.64	4/4
70	70	7.25	31.86	0.61	11.80	5.43	5.27	-1.44	4/4
80	70	6	43.86	0.83	11.75	4.90	4.81	–	0/4
90	70	4.75	56.46	0.75	14.68	4.39	4.31	–	0/4
100	70	4	72.83	0.88	19.48	4.82	4.70	–	0/4
50	80	12	17.93	0.61	7.86	5.57	5.03	-0.59	4/4
60	80	8.75	25.84	1.05	10.23	5.48	4.96	-1.62	4/4
70	80	7	36.42	0.77	8.48	5.11	4.89	–	0/4
80	80	5	49.00	0.79	24.38	4.43	4.43	–	0/4
90	80	4	66.66	0.97	18.51	4.40	4.30	–	0/4
100	80	3	82.40	0.88	21.07	4.07	4.07	–	0/4
50	90	9.75	20.78	0.67	27.61	6.46	5.72	-0.99	4/4
60	90	6.5	29.05	0.75	26.30	5.10	5.06	-1.06	4/4
70	90	4.25	40.61	0.94	42.21	5.38	5.28	–	0/4
80	90	3.5	56.13	1.10	81.74	5.27	4.95	–	0/4
90	90	4	73.68	1.02	18.23	4.28	4.25	–	0/4
100	90	3	95.17	1.21	22.85	4.40	4.35	–	0/4

[1] number of nodes, [2] number of programs, [3] number of iterations,
[4] average computation time (in seconds) to solve a subproblem to optimality,
[5] average computation time (in seconds) to solve *FeasY* to optimality,
[6] average computation time (in seconds) to solve *FeasX* to optimality,
[7] initial gap, [8] final gap, [9] percent difference between best solution found by the algorithm and that of CPLEX, [10] number of instances (out of 4) for which CPLEX was able to find an integer feasible solution within 300 seconds

4.4 Concluding Remarks

In this chapter, we have presented an hybrid Lagrangean relaxation-decomposition algorithm for the resolution of the Video Placement and Routing Problem. The significance of the proposed algorithm lies in its capability to provide a benchmark to measure the quality of the output results. The algorithm proposed in this chapter is different from similar existing algorithms because we achieve the feasible solutions through the use of integer programming techniques and this is the reason that our solution methodology would result in good quality solutions even at the earlier iterations of the algorithm. However, obtaining a lower bound at each step of the algorithm seems to be the main bottleneck as one has to solve a number of integer programs at each step. For this reason, we have also proposed a modification to our algorithm that considerably accelerates its running time. Computational results demonstrate that the algorithm is indeed capable of producing good quality solutions in considerably short running times.

It is clear that obtaining the optimal solution of the model considered here will get harder as the problem sizes increase. In such cases, fast heuristic algorithms or metaheuristics such as tabu search can be of use. However, one must be aware that such approaches are incapable of indicating the quality of the solution found unless additional lower bounding techniques are employed.

Chapter 5

Database Allocation

In this chapter, we deal with the problem of *allocating databases in a distributed computing system*. The chapter begins by a formal definition of the problem and presents an associated integer programming formulation in Section 5.1. Then, illustrating that a relaxation of this problem is the well-known *multidimensional Knapsack Problem*(mKP), we focus on the exact solution of the mKP. In specific, we investigate an important class of valid inequalities for the mKP and study the separation problem in Section 5.2. In Section 5.3, we propose a new scheme for cover inequality separation for the problem and present computational results. The proposed scheme in a branch-and-cut framework along with additional computational results are presented in Section 5.4. Conclusions are stated in Section 5.5.

5.1 Introduction

A distributed computer system is composed of a collection of independent computers that share data, programs, and other resources. Distributed systems, as quoted from Gavish and Pirkul [41], "are well suited to environments where large portions of the data are either used by the location which generates the data or

by locations which are geographically close to the generating locations". Medical, banking, insurance and rental services are examples to such services. These services typically have large databases storing customer data and most of the database operations take place near the physical location of the customer. Therefore, partitioning of the database and allocating the partitions to the computers is a viable beneficial strategy for such services. Consequently, all the transactions related to a customer will be carried out by a local database, which will result in "eliminating the costs and delays which might be introduced by a centralized computer system" [41].

Formally put forward, the problem of allocating databases in a distributed computing system mentioned here consists of allocating customers and databases associated to these customers among computers to minimize telecommunication costs subject to capacity constraints on the computers. The assignment of customers and the assignment of databases can be treated jointly, since it is desirable to assign customers to the nearest processors [85].

Gavish and Pirkul [41] and Pirkul [85] have presented integer programming formulations for variations of this problem. The primary difference between the two studies is that the latter assumes that there is a fixed number of computers already installed in the network whereas the former includes decisions pertaining to locating computers on a given set of potential locations. We consider here the problem studied by Pirkul [85] and present the related formulation. However, the same discussion holds for the problem studied by Gavish and Pirkul [41]. Let I be the index set of customer locations and J be the index set of computer locations. Associated with each computer at site $j \in J$, let P_j, V_j and T_j denote the maximum processing capacity, storage capacity and telecommunication capacity, respectively. For each customer $i \in I$, let p_i denote his/her processing requirement, v_i denote the storage requirement for his/her database and t_i denote the communication load per transaction for this customer. Let c_{ij} denote the cost of transaction between sites i and j. For the specific components of this cost structure, the reader is referred to Pirkul [85]. The assignment decisions are made by defining a binary decision variable x_{ij}, which takes the value 1 if customer i is assigned to computer j, and 0 otherwise.

The integer programming formulation of the problem (henceforth denoted by DB) can then be presented as follows:

$$\text{minimize} \sum_{i \in I} \sum_{i \in J} c_{ij} x_{ij} \tag{5.1}$$

s.t.

$$\sum_{j \in J} x_{ij} = 1, \qquad \forall i \in I \tag{5.2}$$

$$\sum_{i \in I} p_i x_{ij} \leq P_j, \qquad \forall j \in J \tag{5.3}$$

$$\sum_{i \in I} v_i x_{ij} \leq V_j, \qquad \forall j \in J \tag{5.4}$$

$$\sum_{i \in I} t_i x_{ij} \leq T_j, \qquad \forall j \in J \tag{5.5}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in I, j \in J \tag{5.6}$$

In DB, constraint (5.2) states that every customer with a corresponding database should be assigned to a single computer. Constraints (5.3), (5.4) and (5.5) are associated with processing, storage and telecommunication capacities for every computer.

Pirkul [85] proves that this problem is $\mathcal{NP}$-Hard and describes a solution algorithm for this problem based on Lagrangean relaxation and subgradient optimization. In specific, if the assignment constraints (5.2) are relaxed in a Lagrangean fashion using a multiplier vector $\lambda = \{\lambda_1, \lambda_2, \ldots\}$, then the resulting problem decomposes over all the computers, giving way in the following subproblem written for each $j \in J$ (henceforth denoted by DB_j):

$$\text{minimize} \sum_{i \in I} (c_{ij} + \lambda_i) x_{ij}$$

s.t.

$$\sum_{i \in I} p_i x_{ij} \leq P_j,$$
$$\sum_{i \in I} v_i x_{ij} \leq V_j,$$
$$\sum_{i \in I} t_i x_{ij} \leq T_j,$$
$$x_{ij} \in \{0,1\}, \quad \forall i \in I$$

Each DB_j is the well-known *multidimensional Knapsack Problem* (mKP). Therefore, efficient solution of the mKP will result in better solution algorithms for the solution of the database allocation problem described above. Based on this motivation, we concentrate on the mKP in the remaining of this chapter.

5.2 The Multidimensional Knapsack Problem

In the remainder of this chapter, we investigate the well-known multidimensional knapsack problem:

$$\max \sum_{i \in N} c_i x_i \tag{5.7}$$

s.t.

$$\sum_{i \in N} a_{ij} x_i \leq b_j, \qquad j \in M \tag{5.8}$$
$$x_i \in \{0,1\}, \qquad i \in N, \tag{5.9}$$

where a_{ij}, b_i and c_i are nonnegative integers. Aside from being utilized in solving database allocation problems, this problem also finds applications in many problems, including cutting stock, loading, delivery in vehicles with multiple compartments, approval voting and management of a remote sensing satellite. The

reader is referred to Chapter 9 of Kellerer et al. [59] and Fréville [36] for a recent survey on the problem.

Exact and heuristic solution approaches are available in the literature for the solution of this problem. For the former, a highly important and widely used class of valid inequalities is the class of *cover inequalities*, explained further in the following.

5.2.1 Cover Inequalities

We will begin this section with an example. Consider the following example of a multidimensional knapsack polyhedron taken from Nemhauser and Wolsey [74], p. 528.

$$K = \{x \in \mathcal{B}^6 : 12x_1 + 2x_2 + 3x_3 + 2x_4 + 4x_5 + 3x_6 \leq 20,$$
$$3x_1 + 8x_2 + 12x_3 + 13x_4 + 20x_5 + 14x_6 \leq 36\}$$

Then, the following system of inequalities give a complete description and are also the facets of the polyhedron K:

$$
\begin{aligned}
(1) \quad & -x_1 \leq 0 \\
(2) \quad & -x_2 \leq 0 \\
(3) \quad & -x_3 \leq 0 \\
(4) \quad & -x_4 \leq 0 \\
(5) \quad & -x_5 \leq 0 \\
(6) \quad & -x_6 \leq 0 \\
(7) \quad & +x_6 \leq 1 \\
(8) \quad & +x_5 \leq 1 \\
(9) \quad & +x_4 \leq 1 \\
(10) \quad & +x_3 \leq 1 \\
(11) \quad & +x_2 \leq 1 \\
(12) \quad & +x_1 \leq 1 \\
(13) \quad & +x_2 + x_5 + x_6 \leq 2 \\
(14) \quad & +x_2 + x_4 + x_5 \leq 2 \\
(15) \quad & +x_2 + x_3 + x_5 \leq 2 \\
(16) \quad & +x_1 + x_5 + x_6 \leq 2 \\
(17) \quad & +x_3 + x_4 + x_5 + x_6 \leq 2 \\
(18) \quad & +x_2 + x_3 + x_4 + 2x_5 + x_6 \leq 3 \\
(19) \quad & +x_1 + x_2 + x_4 + x_5 + x_6 \leq 3 \\
(20) \quad & +x_1 + x_2 + x_3 + x_5 + x_6 \leq 3 \\
(21) \quad & +x_1 + x_2 + x_3 + x_4 + 2x_5 + 2x_6 \leq 4
\end{aligned}
$$

This description has been obtained using PORTA [25]. Among the 21 facets discovered, inequalities numbered (1)-(6) and (7)-(12) are the trivial inequalities, $x_i \geq 0$ and $x_i \leq 1$, respectively. Inequalities (13)-(16) are examples to so-called *cover inequalities*. Among the remaining inequalities, (17)-(20) are the liftings of cover inequalities. As the above example implies, these inequalities may sometimes play an important role in describing the associated polyhedra. We will now provide a formal description of cover inequalities.

Consider the polyhedron $K = \{x \in \mathcal{B}^{|N|} : \sum_{i \in N} a_i x_i \leq b\}$ associated with the unidimensional knapsack problem. The index set $C \subseteq N$ of variables for which $\sum_{i \in C} a_i > b$ holds is called a *cover*. A cover that loses this property when any one

of the indices in it is excluded is called a *minimal cover*. A minimal cover induces a so called *cover inequality* that is valid for the polyhedron K and is given as follows:

$$\sum_{i \in C} x_i \leq |C| - 1 \tag{5.10}$$

Cover inequalities are one of the most important components of the branch-and-cut algorithms designed to solve 0-1 integer programming problems. Recent studies by Gu et al. [44] and [52] provide extensive discussions of available strategic choices for using cover inequalities in the branch-and-cut process for 0-1 programming. These inequalities and extensions have recently been utilized in solving hub location problems ([101], [20]), median cycle location problems ([64]), gas-lift optimization ([23]), and network optimization of a radio mobile telephone system ([34]).

One important question to be answered is, during the solution of a *mKP*, how does one identify cover inequalities, especially those that are violated by a given fractional solution to the problem. This is known as the *separation problem*. Before stepping into this issue for the *mKP*, we will provide below a brief review of the separation problem for the unidimensional knapsack problem.

5.2.2 The Separation Problem

The separation problem for cover inequalities in the context of 0-1 integer programming was introduced by Crowder et al. [30]. Let $X^* = \{x_1^*, x_2^*, \ldots\} \in [0,1]^n$ be an arbitrary feasible solution to $\sum_{i \in N} a_i x_i \leq b$, where a_i's are nonnegative integers. Deciding whether X^* satisfies all of the possible cover inequalities of K is known as the *cover separation problem (CSP)*. The CSP can be formulated as the following 0-1 integer linear program:

$$z = \min \sum_{i \in N} (1 - x_i^*) z_i \tag{5.11}$$

s.t.

$$\sum_{i \in N} a_i z_i \geq b + 1 \tag{5.12}$$

$$z_i \in \{0, 1\}, \qquad i \in N \tag{5.13}$$

If $z < 1$, then the set $C = \{i : \overline{z}_i = 1\}$ induces a cover inequality of the form (5.10), where $\overline{z}_i$ is an optimal solution to the CSP. Otherwise, X^* satisfies all the possible cover inequalities.

Most studies on this subject indicate that the exact solution of the separation problem is costly in practice and usually resort to a greedy type algorithm to obtain approximate solutions. For more details, the reader may refer to Wolsey [98] and Nemhauser and Wolsey [74]. Recently, Kellerer et al. [59] presented computational results on solving the separation problem.

5.2.3 Lifting

Although cover inequalities are a well-known class of valid inequalities for the *mKP*, they in general are not tight and have to be strengthened. This can be accomplished through a process called *lifting*. Given an ordering of variables $\{x_1, x_2, \ldots, x_k, \ldots\}$ to be lifted with $k \in N \setminus C$, the inequality

$$\sum_{i \in C} x_i + \sum_{k \in N \setminus C} \alpha_k x_k \leq |C| - 1 \tag{5.14}$$

is called a *lifted cover inequality*, where α_k are the corresponding lifting coefficients. For each variable to be lifted, the maximum value of the lifting coefficient is calculated as $\alpha_k = |C| - 1 - z_k$, with z_k being the optimal solution value to the following knapsack problem:

$$z_k = \max \sum_{i \in C} x_i + \sum_{k=1}^{k-1} x_k \tag{5.15}$$

s.t.

$$\sum_{i \in N} a_i x_i + \sum_{k=1}^{k-1} a_k x_k \leq b - a_k \tag{5.16}$$

$$x_i \in \{0, 1\}, \qquad \forall i \tag{5.17}$$

It is clear that the values of the lifting coefficients depend on the order of the variables to be lifted. Hence, different lifting sequences result in different lifted cover inequalities.

5.3 An Exact Separation Procedure for mKP

We now extend the previously discussed separation procedure given for the unidimensional knapsack problem to the *mKP*. In this case, each constraint $j \in M$ of the *mKP* is associated a set of cover inequalities denoted by C_j. Given a fractional solution $X^* = \{x_1^*, x_2^*, \ldots\}$ to the *mKP* (usually the solution to the corresponding LP-relaxation), we refer to the problem of identifying whether X^* violates a cover inequality in $\cup_{j \in M} C_j$ or concluding that it satisfies all the possible ones as the *generalized cover separation problem* (*GCSP*). We formulate the *GCSP* by the following 0-1 integer linear programming formulation:

$$z = \min \sum_{i \in N^f} (1 - x_i^*) z_i \tag{5.18}$$

s.t.

$$\sum_{i \in N^f} a_{ij} z_i \geq b'_j + 1 - R(1 - y_j), \qquad j \in M \tag{5.19}$$

$$\sum_{j \in M} y_j \geq 1 \tag{5.20}$$

$$z_i \in \{0, 1\}, \qquad i \in N^f \tag{5.21}$$

$$y_j \in \{0, 1\}, \qquad j \in M \tag{5.22}$$

where $b'_j = b_j - \sum_{i \in N^1} a_{ij}$ with $N^1 = \{i : x_i^* = 1\}$ and $N^f = \{i : 0 < x_i^* < 1\}$. The parameter R used in constraint (5.19) is a sufficiently large constant (which may be chosen as $R = max_{j \in M} b_j + 1$). In this formulation, the additional binary variable y_j is used to check the violation of cover inequalities in the set C_j.

Given an optimal solution $\overline{Z} = \{\overline{z}_1, \overline{z}_2, \ldots\}$ to the integer linear programming formulation with value $z < 1$, the set $C = N^1 \cup \{i : \overline{z}_i = 1\}$ induces a cover inequality given by (5.10). Note that we only include in the formulation variables $i \in N^f$, since one can easily fix $\overline{z}_i = 1$ for $i \in N^1$ and $\overline{z}_i = 0$ for $i \in N \setminus \{N^1 \cup N^f\}$. This also reduces the size of the binary variables in the formulation, thereby facilitating its solvability.

The separation procedure consists of identifying the cover inequality via the $GCSP$ that is violated by the fractional solution X^* and appending it to the formulation. The augmented formulation is resolved and cuts are appended in a similar and an iterative manner until no violated cover inequalities are found. We refer to this procedure as *generalized cover separation* (GCS).

As it has been already pointed out, the 0-1 programming model above is very straightforward, and it is a simple exercise to extend the unidimensional knapsack version to this generalized case. However, the computational experimentation results presented in the next section clearly demonstrate that, with currently available mathematical programming software, there are benefits to reap in using

it for solving the separation problem.

5.3.1 Computational Results

In this section, we describe our computational experience with the proposed procedure on test problems. The generalized cover separation procedure proposed for the mKP has been implemented in C and all the tests are performed on a Sun UltraSPARC 12x400 MHz with 3 GB RAM, using CPLEX 9.0 as the optimization package.

We compare the proposed method with a simple and a straightforward separation procedure in which violated covers are identified using a greedy algorithm (henceforth denoted by GR). To the best of our knowledge, there are no other heuristic procedures previously proposed to separate cover inequalities for the mKP. At every iteration of the algorithm, we try to identify a violated cover inequality for each constraint $j \in M$. More specifically, for each constraint $j \in M$, the variables of the mKP are put in increasing order of the ratios $(1 - x_i^*)/a_{ij}$ and the variable with the smallest ratio is set equal to 1 while keeping the rest at zero. Then, constraint j is checked as to whether this solution causes a violation. If there is a violation, then a cover inequality for this constraint is identified consisting of this single variable. Otherwise, the variable with the second lowest ratio is raised to 1, and the process is repeated until a violating solution is found for constraint j. Among all the cover inequalities identified as a result of scanning all $|M|$ constraints, the one with the maximum violation is appended to the LP-relaxation of the problem and the resulting problem is solved to optimality. This concludes a single iteration. The procedure continues in an iterative manner until no violated cover inequalities are found. For each constraint, the GR has a time complexity of $\mathcal{O}(|N|log|N|)$ to sort the variables and $\mathcal{O}(log|N|)$ for the binary search. Therefore, as a result of scanning all the constraints, the GR has an overall time complexity of $\mathcal{O}(|M||N|log|N|) + \mathcal{O}(|M|log|N|)$ to identify a single violated cover inequality at each iteration.

The performance of the algorithm was tested on both randomly generated

instances and instances taken from the literature. For the former group, a batch of fifty multidimensional 0-1 integer programming problems were generated pseudo-randomly with the following specifications: The number of constraints (m) range between 5 and 1000. That of variables (n) range between 20 and 2000. There are 10 different combinations of these dimension parameters as may be seen in the first column of Table 5.1. For each combination, linear relaxations of five pseudo-randomly generated problems are solved. All objective function coefficients and constraint coefficients have values uniformly distributed between 0 and 100. The right hand side constants are computed using the formula $b_i = 10np_1 + 0.5p_2 \sum_{j=1}^{n} a_{ij}$, $\forall i = 1, \ldots, m$. Here, p_1 and p_2 are pseudo-random variates uniformly distributed between 0 and 1. This formula was chosen to provide variability in tightness among constraints, and avoid the possibility of having a constraint with right hand side equal to zero.

It may seem counter-intuitive to solve a rather complex integer programming formulation to separate valid inequalities for a seemingly simpler formulation of the *mKP*. Therefore, in order to investigate whether the separation problem is easier to solve than the original multidimensional knapsack problem, we have used state-of-the-art optimization software CPLEX 9.0 to solve the instances considered here using the formulation defined by (5.7)-(5.9). A time limit of 3 hours (10800 seconds) is imposed on CPLEX.

The results of the computational experiments on random instances are presented in Table 5.1. In this table, the two columns under the heading *Avg. cover* present the average number of cover inequalities found by *GR* and *GCS*, respectively. Each entry in these columns is calculated over five instances. The next two columns present the maximum number of cover inequalities found by the two procedures. The two columns under the heading T_{avg} indicate the unit time required by the corresponding algorithm (in seconds) to identify a violated cover inequality per cover and to solve the LP-relaxation, which is calculated by dividing the total solution time to the total number of cover inequalities found. The last two columns, T_C and n_C, show the average time required to solve the instances and the number of problems that could be solved to optimality out of the total number of problems within the 3 hour time limit, respectively.

Table 5.1: Statistics for a sample of 50 randomly generated instances

	Avg. cover		Max. cover		T_{avg}		$CPLEX$	
m x n[1]	GR[2]	GCS[3]	GR[4]	GCS[5]	GR[6]	GCS[7]	T_C[8]	n_C[9]
5x20	2.20	7.20	5	13	0.02	0.03	0.01	5/5
10x20	1.20	9.60	2	22	0.04	0.03	0.01	5/5
25x100	2.80	5.60	5	8	0.10	0.10	0.28	5/5
50x100	1.40	6.20	4	12	0.21	0.14	0.89	5/5
125x500	1.80	9.20	4	26	2.61	1.44	93.67	5/5
250x500	1.80	2.80	3	5	5.16	3.23	505.24	5/5
250x1000	2.00	4.20	4	8	11.75	6.64	3353.80	5/5
500x1000	2.00	4.00	4	6	23.58	13.09	3326.60	4/5
500x2000	1.20	4.80	5	8	82.26	28.17	8512.40	2/5
1000x2000	1.60	7.20	4	15	145.01	57.88	6785.40	2/5

[1] size of the problem,
[2] average number of cover inequalities produced by the greedy algorithm,
[3] average number of cover inequalities produced by the proposed procedure,
[4] maximum number of cover inequalities produced by the greedy algorithm,
[5] maximum number of cover inequalities produced by the proposed procedure,
[6] average computation time required by the greedy algorithm,
[7] average computation time required by the proposed procedure,
[8] average computation time required by CPLEX, [9] number of instances (out of 5) for which CPLEX was able to find the optimal solution in 3 hours

As Table 5.1 shows, the number of cover inequalities identified by *GCS* is clearly superior to that of *GR*. We additionally note that the *GCS* identified a total of 304 violated cover inequalities over all instances whereas *GR* produced 90 of these. That is, *GR* missed about 70% of the violated cover inequalities produced by *GCS*. In addition, the unit time required by the *GCS* becomes superior to that of *GR* as the instances grow bigger in size. Thus, we can conclude that the *GCS* is quite efficient considering the size of the problems handled and the gain acquired in terms of the number of cover inequalities produced. The last two columns indicate that the time spent for cover generation is only a small fraction of the time required by CPLEX. This implies that the *GCS* is not computationally expensive as compared to CPLEX.

The performance of both algorithms on instances taken from the OR-Library [5] are given in Table 5.2. The first column of the table presents the name of the group of instances, where each group contains 30 problems. The average number of covers and the reported solution times are calculated as the average

Table 5.2: Statistics for the OR-Library instances

		Avg. cover		Max. cover		T_{avg}		$CPLEX$	
$Instance$[1]	m x n[2]	GR[3]	GCS[4]	GR[5]	GCS[6]	GR[7]	GCS[8]	T_C[9]	n_C[10]
mknapcb1	5x100	2.20	5.77	5	14	0.05	0.06	20.58	30/30
mknapcb2	5x250	2.33	4.33	6	8	0.11	0.10	532.69	30/30
mknapcb3	5x500	2.30	5.10	6	12	0.22	0.17	6702.10	17/30
mknapcb4	10x100	0.40	1.70	2	7	0.20	0.13	164.47	30/30
mknapcb5	10x250	0.57	1.13	2	3	0.34	0.23	10506.07	2/30
mknapcb6	10x500	0.33	1.20	1	3	1.10	0.36	10848.13	0/30

[1] name of the problem set, [2] size of the problem,
[3] average number of cover inequalities produced by the greedy algorithm,
[4] average number of cover inequalities produced by the proposed procedure,
[5] maximum number of cover inequalities produced by the greedy algorithm,
[6] maximum number of cover inequalities produced by the proposed procedure,
[7] average computation time required by the greedy algorithm, [8] average computation time required by the proposed procedure, [9] average computation time required by CPLEX,
[10] number of instances (out of 5) for which CPLEX was able to find the optimal solution in 3 hours

of 30 problems for each instance group. The remaining columns of this table are same as those of Table 5.1.

For the problems presented in Table 5.2, we also note that *GCS* identified a total of 577 violated cover inequalities over all instances whereas *GR* produced 244 of these, indicating that *GR* missed about 58% of the violated cover inequalities produced by *GCS*. Similar to the previous result, the last two columns of Table 5.2 indicate that the separation problem is indeed much more easier to solve than the *mKP*. In fact, as instances grow larger in size, the results demonstrate that CPLEX was not able to find the integer optimal solution for most of the problems.

5.4 A Branch-and-Cut Framework for the mKP

Cover inequalities, along with other types of valid inequalities such as Gomory cuts, flow cover inequalities, etc. are generally implemented in many commercial software, including CPLEX. It is not, however, clear as to how, for instance, CPLEX generates such cuts nor how CPLEX manages the generated cuts throughout the branch-and-cut tree. Nevertheless, CPLEX provides the

user with the option of specifying the frequency of cover inequality generation in the branch-and-bound (B&B) tree. These are automatic generation, moderate generation and aggressive generation.

In this section, we report our results in embedding the proposed procedure to separate cover inequalities in a branch-and-cut framework. To this purpose, we use CPLEX's framework, which allows one to use any custom separation routine to be implemented through callback functions. The aim of the computational analysis of this section is to see whether the proposed procedure results in any improvements compared to CPLEX's own cover inequality generation procedure. We emphasize that our focus is not to devise a branch-and-cut algorithm to solve multidimensional knapsack problems, but rather to see the effect of the most violated cover inequality generation in the process of solving the *mKP*.

We perform the experiments on small and medium sized instances taken from the OR-Library [5], as well as randomly generated problems. We report the results in Tables 5.3, 5.4 and 5.5. Using the proposed procedure, several rounds of violated cover inequality generation is performed at each node of the branch-and-cut tree. The results pertaining to this procedure is indicated under the column E in the tables. We also employ lifting to each of the generated inequalities, which is indicated under the column $E(L)$ in the tables. The computation of the lifting coefficients are computed in an exact manner. Different lifting sequences have been considered in the experiments, such as nonincreasing and/or nondecreasing order of variables in terms of their fractional solution values and reduced costs. It turned out that a lifting sequence in the nonincreasing order of the variables yielded the least number of B&B nodes required to optimally solve a number of test problems. Therefore, the same sequence has been utilized in the computational experiments reported below.

Since we are comparing an "external" procedure (i.e., the proposed procedure) with an "internal" separation procedure (i.e. CPLEX's native separation procedure), we can not consider the solution time as a basis for comparisons. We do, however, consider the number of nodes in the branch and bound tree required to solve the instances to optimality as an indicator of the performance of both

procedures.

Tables 5.3 and 5.4 are related to the results obtained using the problems from the OR-Library [5] in the file `mknap2.txt`. In each table, we report the number of B&B nodes required to solve each problem to optimality (Nd) and the number of cover inequalities identified (C), for each procedure. In performing the comparisons, we use CPLEX's cover generation using automatic and aggressive generation options, which are denoted by $CPLEX(AU)$ and $CPLEX(AG)$ in the tables, respectively. We have also switched of all other cuts that CPLEX generates, such as flow covers or Gomory cuts, in order to see the effect of cover inequality generation alone.

Table 5.5 is related to the results of randomly generated problems, with the number of constraints ranging from 25 to 50 and that of variables ranging from 100 to 200. For each configuration, 5 instances have been generated randomly as explained in Section 5.3.1. We additionally provide the average values for each problem configuration in the table.

Looking at the results given in Tables 5.3, 5.4 and 5.5, we see that a branch-and-cut algorithm with the proposed procedure is in general capable of solving problems to optimality, using less number of B&B nodes than that of CPLEX. In some cases, CPLEX is dominant. However, one must be aware that these comparisons are not performed on a completely fair setting since CPLEX also has an internal cut manager that decides on which node to generate any type of cuts, whereas our procedure generates a cover inequality at every node (if violated). Therefore, we conjecture that, once implemented in CPLEX as a native cut generator in cooperation with its own cut manager, the proposed procedure will be very effective in solving such problems.

5.5 Conclusions

In this chapter, we have proposed an exact separation procedure to separate violated cover inequalities for the multidimensional knapsack problem that is based

Table 5.3: Statistics for the Branch-and-Cut implementation - ORLibrary Instances 1

			E		$E(L)$		$CPLEX(AU)$		$CPLEX(AG)$	
$problem$[1]	m[2]	n[3]	Nd[4]	C[5]	Nd[6]	C[7]	Nd[8]	C[9]	Nd[10]	C[11]
flei.dat	10	20	529	511	**384**	372	708	0	801	30
hp1.dat	4	28	31	285	30	259	**12**	12	**12**	12
hp2.dat	4	35	45	395	23	153	**15**	4	**15**	4
pb1.dat	4	27	22	192	22	192	**12**	12	**12**	12
pb2.dat	4	34	57	283	72	425	**22**	12	**22**	12
pb4.dat	2	29	17	143	5	64	**4**	6	**4**	6
pb5.dat	10	20	377	403	**293**	288	789	0	733	30
pb6.dat	30	40	75	230	**60**	187	166	0	169	78
pb7.dat	30	37	432	1407	**322**	1030	650	90	650	90
pet2.dat	10	10	**0**	10	**0***	10	7	3	7	3
pet3.dat	10	15	**5**	21	**5**	21	8	15	8	15
pet4.dat	10	20	**4**	19	**4**	19	7	6	7	6
pet5.dat	10	28	**1**	16	**1**	16	9	4	9	4
pet7.dat	5	50	58	518	58	518	**35**	15	**35**	15
sent01.dat	30	60	61	210	**55**	158	90	0	93	17
sent02.dat	30	60	363	1389	**254**	957	696	90	696	90
weing1.dat	2	28	2	21	2	19	2	6	2	6
weing2.dat	2	28	0	13	0	13	0	4	0	4
weing3.dat	2	28	6	19	**0**	8	9	6	9	6
weing4.dat	2	28	2	100	2	125	**0**	5	**0**	5
weing5.dat	2	28	0	23	0	23	0	6	0	6
weing6.dat	2	28	4	16	4	16	**0**	3	**0**	3
weing7.dat	2	105	5	24	5	24	**4**	5	**4**	5
weing8.dat	2	105	8	117	**4**	122	21	6	21	6
Average			**87.67**	**265.21**	**66.88**	**209.13**	**136.08**	**12.92**	**137.88**	**19.38**

[1] problem name, [2] number of constraints, [3] number of variables,
[4] number of B&B nodes required to solve the problem to optimality using the proposed procedure,
[5] number of cover inequalities produced by the proposed procedure,
[6] number of B&B nodes required to solve the problem to optimality using the proposed procedure with lifting,
[7] number of cover inequalities produced by the proposed procedure with lifting, [8] number of B&B nodes required to solve the problem to optimality using CPLEX with automatic cover generation,
[9] number of cover inequalities produced by CPLEX with automatic cover generation, [10] number of B&B nodes required to solve the problem to optimality using CPLEX with aggressive cover generation,
[11] number of cover inequalities produced by CPLEX with aggressive cover generation

Table 5.4: Statistics for the Branch-and-Cut implementation - ORLibrary Instances 2

			E		$E(L)$		$CPLEX(AU)$		$CPLEX(AG)$	
problem	m	n	Nd	C	Nd	C	Nd	C	Nd	C
weish01.dat	5	30	**3**	15	4	14	15	5	15	5
weish02.dat	5	30	5	30	5	29	5	6	5	6
weish03.dat	5	30	0	10	0	8	0	4	0	4
weish04.dat	5	40	0	4	0	3	0	3	0	3
weish05.dat	5	30	0	2	0	2	0	2	0	2
weish06.dat	5	40	8	41	8	31	**5**	8	**5**	8
weish07.dat	5	40	6	18	**4**	20	7	8	7	8
weish08.dat	5	40	5	26	**3**	18	9	5	9	5
weish09.dat	5	40	0	3	0	3	0	0	0	0
weish10.dat	5	50	**19**	145	26	99	24	15	24	15
weish11.dat	5	50	**2**	49	**2**	37	7	15	7	15
weish12.dat	5	50	2	39	**1**	32	**1**	3	**1**	3
weish13.dat	5	50	3	37	**0**	50	6	9	6	9
weish14.dat	5	60	**0**	35	**0**	48	3	11	3	11
weish15.dat	5	60	**2**	12	**2**	11	5	5	5	5
weish16.dat	5	60	6	11	6	11	**5**	8	**5**	8
weish17.dat	5	60	**2**	25	**2**	25	4	4	4	4
weish18.dat	5	70	**1**	45	**1**	39	9	8	9	8
weish19.dat	5	70	**7**	37	**7**	37	16	12	16	12
weish20.dat	5	70	3	19	4	24	**1**	5	**1**	5
weish21.dat	5	70	10	29	10	29	**0**	8	**0**	8
weish22.dat	5	80	**5**	34	**5**	34	11	5	11	5
weish23.dat	5	80	**7**	55	9	41	13	8	13	8
weish24.dat	5	80	**2**	10	**2**	10	4	5	4	5
weish25.dat	5	80	11	19	**4**	13	10	13	10	13
weish26.dat	5	90	**12**	91	**12**	28	14	10	14	10
weish27.dat	5	90	0	3	0	3	0	3	0	3
weish28.dat	5	90	**0**	17	**0**	13	4	1	4	1
weish29.dat	5	90	3	8	3	8	0	3	0	3
weish30.dat	5	90	0	1	0	1	0	1	0	1
Average			**4.13**	**29.00**	**4.00**	**24.03**	**5.93**	**6.43**	**5.93**	**6.43**

Table 5.5: Statistics for the Branch-and-Cut implementation - Random Instances

		E		$E(L)$		$CPLEX(AU)$		$CPLE(AG)$	
m x n	No.	Nd	C	Nd	C	Nd	C	Nd	C
25x100	1	229	283	163	251	166	75	166	75
25x100	2	81	117	52	82	69	55	69	55
25x100	3	41	76	23	43	55	62	55	62
25x100	4	39	43	27	45	30	21	30	21
25x100	5	232	410	507	619	429	75	429	75
Average		**124.40**	**185.80**	**154.40**	**208.00**	**149.80**	**57.60**	**149.80**	**57.60**
50x100	1	11	28	3	9	5	7	5	7
50x100	2	559	609	319	376	606	150	606	150
50x100	3	804	1175	609	696	913	150	913	150
50x100	4	0	2	0	2	0	1	0	1
50x100	5	29	34	3	9	13	9	13	9
Average		**280.60**	**369.60**	**186.80**	**218.40**	**307.40**	**63.40**	**307.40**	**63.40**
25x200	1	210	450	79	129	157	67	157	67
25x200	2	31	37	48	50	27	35	27	35
25x200	3	38	51	36	52	25	18	25	18
25x200	4	2905	5761	997	1211	2106	75	2106	75
25x200	5	230	273	163	159	269	75	269	75
Average		**682.80**	**1314.40**	**264.60**	**320.20**	**516.80**	**54.00**	**516.80**	**54.00**
50x200	1	548	811	54	82	235	69	235	69
50x200	2	12232	16379	10973	12452	11825	150	11825	150
50x200	3	1076	1524	1277	1350	1383	150	1383	150
50x200	4	794	1224	961	1234	623	150	623	150
50x200	5	22	50	16	34	50	20	50	20
Average		**2934.40**	**3997.60**	**2656.20**	**3030.40**	**2823.20**	**107.80**	**2823.20**	**107.80**

on solving a single 0-1 integer programming formulation at each iteration. The procedure proves to be simple and quite efficient whilst the implementation can be done quite easily, requiring no specialized algorithm. We then implemented the proposed procedure in a branch-and-cut framework. Comparing with a state-of-the-art exact solver, the results demonstrate that the proposed procedure is able to solve instances requiring less computational effort. We therefore conclude that the proposed algorithm may be a viable alternative in separating cover inequalities for the multidimensional knapsack problem.

Chapter 6

Conclusions

In this dissertation, we have investigated several problems arising in electronic content distribution and proposed models and exact solution algorithms for their solution. The first section of this chapter reports main research findings and contributions of this research. We then point out several research avenues that may be considered in future research.

6.1 Summary of Research Contributions

We further subdivide this section according to the different types of problems studied in this dissertation.

6.1.1 Content Distribution Networks

Content Distribution Network (CDN) is a new technology aimed at increasing the effectiveness of the Internet by improving the response time and reducing the loads of the servers. In this dissertation, we have identified a new problem in such networks to which we refer as Content Distribution Network Design Problem (CDNDP). The main contributions of this dissertation in the context of CDNs

can be summarized as follows:

- Two new problems in designing CDNs are described, respectively for networks with single and multiple servers. The complexity of the problems are investigated.
- Integer programming models are proposed for the two problems identified which can be used to optimally locate proxy servers, decide on which objects should be stored in each server and assign of clients to suitable proxies simultaneously. Furthermore, the objective functions of the models are suitable representations of the way CDNs operate. Our approach seems to be one of the first to consider all three decision problems jointly and utilizing such an objective function.
- For the single server CDN design problem, exact and heuristic solution algorithms are offered. The former is based on a linearization and Benders' decomposition. The latter is a greedy-like procedure. Computational results are provided so as to demonstrate the efficiency of these algorithms.

Based on the computational results we conclude that the exact solution algorithm proposed for the single server CDN design problem seems computationally effective for reasonable sized networks (such as Virtual Private Networks or Multimedia Networks). For very huge networks such as the Internet itself, however, the greedy algorithm appears to be capable of providing near-optimal solutions in fairly short computing times.

6.1.2 Video on Demand Services

In a part of this dissertation, we studied the problem of Video Placement and Routing (VPRP) that in the context of Video-on-Demand systems. Video on Demand (VoD) is a service that provides tens to hundreds of videos (programs) to hundreds to thousands of clients through a network. The VPRP arises as a problem in the local groups of VoD systems to properly balance the load on the network. Research contributions on this problem can be summarized as follows:

- Although previous studies commented on the complexity of the VPRP, it was never investigated. We have proved here that this problem is $\mathcal{NP}$-Hard.

- A solution algorithm for the problem is devised based on Lagrangean relaxation, decomposition and subgradient optimization. Computational results demonstrate that the algorithm outperforms a commercial software in obtaining solutions within given time limits. The main motivation for developing such a solution procedure is due to the fact that such algorithms are rare in the literature, especially for general graphs. In contrast, heuristic solution procedures are offered for the solution of similar problems. However, such procedures are typically incapable of indicating the quality of the solution produced. The algorithm devised in this study differs with respect to most of the existing algorithms in that whenever the algorithm is stopped without reaching the optimal solution, it is able to indicate the the quality of the best solution obtained.

- In contrast to its counterparts in the literature, the algorithm proposed here brings novelty in obtaining feasible solutions at each iteration. Usually, heuristics are used for this purpose in similar algorithms, to obtain fairly good solutions at every iteration in short computing times. Here, we take a reverse approach and show that good quality feasible solutions can be obtained even in the very early iterations of such algorithms by the use of integer programming formulations.

6.1.3 Multidimensional Knapsack Problem

Finding applications in database allocations in distributed computer networks, we studied in this dissertation the well-known multidimensional Knapsack Problem (mKP). Focusing on the exact solution of this problem, we have investigated the effect of the widely-used and well-known cover inequalities in the solution process. Research contributions to this problem may be stated as follows:

- A novel exact separation procedure was proposed that is able to find the

most violated cover inequality with respect to a given fractional solution. The separation procedure lies in solving a binary program at each iteration.

- The separation procedure was implemented in a cutting plane and a branch-and-cut framework. Computational results regarding the the former implementation demonstrates that the algorithm is superior to a heuristic separation procedure that is offered in the literature for this problem. On the other hand, the latter implementation suggests that generation of violated cover inequalities coupled with applications of intelligent lifting in branch-and-cut algorithms can significantly reduce the computational effort in solving mKPs to optimality.

6.2 Further Research Issues

Similar to the previous section, we also subdivide this section where we provide prospects for further research.

6.2.1 Content Distribution Networks

Problems arising in content distribution networks are usually of a very-large scale nature. Therefore, solution algorithms that will be developed for such problems need to be scalable, i.e. they need to be capable of accommodating very large scale data. In addition, such problems typically need to be solved in short computing times. Solution algorithms based on heuristic procedures for such problems seem to be a promising research issue, as they run very fast and able to handle problems of huge sizes. Therefore, we propose as a further research issue, the investigation of metaheuristic techniques (such as tabu search) for the resolution of the problems described in Chapter 3. Although the greedy heuristic offered in this chapter seems to produce near-optimal solutions, one may benefit from expectedly better solutions offered through more intelligent metaheuristic techniques.

There are also many open optimization problems for content delivery and

caching that needs further investigation. Some of these in the context of OR/MS are pointed out by Datta et al. [31]. Within the scope of this study, several extensions may consist of issues such as proxy-sizing (the problem of determining the optimal capacities of the proxies) and pricing (the problem of finding the price for the CDN to charge to its customers). In defining the CDNDP in Chapter 3, we have not taken into account any bandwidth limitations as we assumed to be working on networks with such a characteristic (such as *national networks* or *intranets*). However, large networks spanning multiple countries usually have a limited bandwidth capacity. Introducing a bandwidth constraint into the proposed model for SCDNP would surely make the solution much more difficult. Furthermore, as it would modify the current structure of the model, it would require a new solution algorithm. This is offered as a further research avenue. In addition, the CDNDP may very well be extended to take into account a type of constraint aimed at achieving an acceptable performance from the CDN customers perspective (such as a Quality of Service constraint). This, again, would alter the special structure of the SCDNP model and would require investigation different solution strategies.

A very interesting issue in content distribution is Peer-to-Peer (P2P) networks. These networks have a decentralized structure in that all the individual users are inter-connected and serve both as a proxy and a client. The basic idea in P2P networks is that each user pair may exchange content without the need for a centralized structure as in CDNs. They are very suitable for sharing files of large sizes (such as music or videos) and are an alternative way to distribute content. These networks bear very interesting problems of their own and may be investigated in future research. For now, we refer the reader to Koo et al. [62] for an introduction to problems in P2P networks and research directions.

6.2.2 Video on Demand Services

Although being a well-studied problem for over 10 years, VoD services still pose challenging and interesing optimization problems now that new networking technologies are being developed. In this research, we assumed the VoD service to

be implemented on a conventional cable network. However, VoD services may benefit from new technologies such as Multi Protocol Label Switching (MPLS) and Passive Optical Networks (PONs). It would be interesting to consider VoD implementations in combination with such novel network technologies.

Another issue that may be considered in VoD systems is providing multiple services on the same network (such as both audio and video). In this case, the provider would have to store different kinds of media at each server and therefore would need to consider various charging schemes as different services would have differing costs. Such an extension would be an interesting generalization of the problem considered in this study.

6.2.3 Multidimensional Knapsack Problems

The *mKP* is one of the well-known specially structured integer programming problems. The contributions made in this dissertation in the context of an exact solution algorithm suggest further topics for research. In specific, investigation of new classes of valid inequalities for the *mKP* and implementation of these in a branch-and-cut algorithm may be useful in solving the problem. Surely, the new classes of valid inequalities would also require separation routines. A recent study by Gabrel and Minoux [37] is an important attempt in this direction of research, who investigated so-called extended cover inequalities for the *mKP*.

One other topic of research that may be considered is to develop an efficient and a faster way of identifying violated cover inequalities and their extensions in solving the *mKP*. Oğuz and Bektaş [78] demonstrated that it is possible to separate violated cover inequalities in polynomial time through dynamic programming. Similar improvements can be made for the inequalities investigated by Gabrel and Minoux [37].

Bibliography

[1] Akamai. http://www.akamai.com.

[2] Digital island. http://www.sandpiper.net.

[3] Inktomi. http://www.inktomi.com.

[4] Mirror image. http://www.mirror-image.com.

[5] OR-Library. Available at http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/mknapinfo.html.

[6] Speedera. http://www.speedera.com.

[7] The economic impacts of unacceptable web-site download speeds. Technical report, Zona Research, April 1999. Available at http://also.co.uk/e-hosting.html (Accessed 24.06.2004).

[8] The Need for Speed II. Zona market bulletin, Zona Research, April 2001. Available at http://www.websiteoptimization.com/speed/1/ (Accessed 24.06.2004).

[9] W.P. Adams and H.D. Sherali. A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.

[10] 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, April 2004.

[11] P. Avella, M. Boccia, R. Canonico, D. Emma, A. Sforza, and G. Ventre. Web cache location and network design in VPNs. In *Proceedings of INOC (International Network Optimization Conference)*, Evry/Paris, France, October 27-29 2003.

[12] P. Backx, T. Lambrecht, B. Dhoedt, F. DeTurck, and P. Demeester. Optimizing content distribution through adaptive distributed caching. *Computer Communications*, 28(6):640–653, 2005.

[13] I.D. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 661–670, January 2001.

[14] S.A. Barnett and G.J. Anido. A cost comparison of distributed and centralized approaches to video-on-demand. *IEEE Journal on Selected Areas in Communications*, 14(6):1173–1183, 1996.

[15] N. Bartolini, F. Lo Presti, and C. Petrioli. Optimal dynamic replica placement in content delivery networks. In *Proceedings of the 11th IEEE International Conference on Networks (ICON2003)*, pages 125–130, 2003.

[16] H.S. Bassali, K.M. Kamath, R.B. Hosamani, and L. Gao. Hierarchy-aware algorithms for CDN proxy placement in the Internet. *Computer Communications*, 26:251–263, 2003.

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

[18] J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[19] C.C. Bisdikian and B.V. Patel. Cost-based program allocation for distributed multimedia-on-demand systems. *IEEE Multimedia*, pages 62–72, Fall 1996.

[20] N. Boland, M. Krishnamoorthy, A.T. Ernst, and J. Ebery. Preprocessing and cutting for multiple allocation hub location problems. *European Journal of Operational Research*, 155:638–653, 2004.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *Proceedings of IEEE INFOCOM'99*, volume 1, pages 126–134, New York, March 1999.

[22] A.J. Cahill and C.J. Sreenan. VCDN: A content distribution network for high quality video distribution. In *Proceedings of Information Technology & Telecommunications Conference (IT&T)*, October 2003.

[23] E. Camponogara and P.H.R. Nakashima. Solving a gas-lift optimization problem by dynamic programming. *European Journal of Operational Research*, 2005. Forthcoming.

[24] S. Choi and Y. Shavitt. Proxy location problems and their generalizations. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)*, 2003.

[25] T. Christof and A. Löbel. PORTA: POlyhedron Representation Transformation Algorithm. Available online at http://www.zib.de/Optimization/Software/Porta/, 2004.

[26] W.W. Chu. Optimal file allocation in a multiple computer system. *IEEE Transactions on Computers*, 18(10):865–889, 1969.

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

[28] G. Cornuejols, G.L. Nemhauser, and L.A. Wolsey. The uncapacitated facility location problem. In P.B. Mirchandani and R.L. Francis, editors, *Discrete Location Theory*, chapter 3, pages 119–171. John Wiley & Sons, 1990.

[29] E. Cronin, S. Jamin, C. Jin, A.R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. *IEEE Journal on Selected Areas in Communications*, 20(7):1369–1382, 2002.

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

[31] A. Datta, K. Dutta, H. Thomas, and D. VanderMeer. World Wide Wait: a study of Internet scalability and cache-based approaches to alleviate it. *Management Science*, 49(10):1425–1444, October 2003.

[32] A. Datta, K. Dutta, H. Thomas, and D. VanderMeer. World Wide Wait: a study of Internet scalability and cache-based approaches to alleviate it. *Management Science Electronic Companion Pages*, 2003. (Available at http://mansci.pubs.informs.org/).

[33] Ö. Erçetin and L. Tassiulas. Market-based resource allocation for content delivery in the Internet. *IEEE Transactions on Computers*, 52(12):1573–1585, December 2003.

[34] M. Fischetti, G.R. Jacur, and J.J.S. González. Optimisation of the interconnecting network of a UMTS radio mobile telephone system. *European Journal of Operational Research*, 144(56-67), 2003.

[35] M.L. Fisher and D.S. Hochbaum. Database location in computer networks. *Journal of the Association for Computing Machinery*, 27(4):718–735, 1980.

[36] A. Fréville. The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research*, 155:1–21, 2004.

[37] V. Gabrel and M. Minoux. A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems. *Operations Research Letters*, 30:252264, 2002.

[38] L.-L. Gao and E.P. Robinson Jr. A dual-based optimization procedure for the two-echelon uncapacitated facility location problem. *Naval Research Logistics*, 39:191–212, 1992.

[39] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Franciso, California, 1979.

[40] B. Gavish. Topological design of computer communication networks - The overall design problem. *European Journal of Operational Research*, 58:149–172, 1992.

[41] B. Gavish and H. Pirkul. *Management of Distributed Data Processing*, chapter Allocation of data bases in distributed computing systems, pages 215–231. North-Holland, 1982.

[42] D. Ghose and H.J. Kim. Scheduling video streams in video-on-demand systems: A survey. *Multimedia Tools and Applications*, 11:167–195, 2000.

[43] D. Ghosh, I. Murthy, and A. Moffett. File allocation problem: comparison of models with worst case and average communication delays. *Operations Research*, 40(6):1074–1085, November-December 1992.

[44] Z. Gu, G.L. Nemhauser, and M.W.P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs: computation. *INFORMS Journal on Computing*, 10(4):427–437, 2000.

[45] M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[46] K. Hosanagar, R. Krishnan, J. Chuang, and V. Choudhary. Pricing and resource allocation in caching services with multiple levels of QoS. In *Proceedings of International Conference on Information Systems (ICIS 2002)*, 2002. Available at http://opim.wharton.upenn.edu/workpapers/storage/05-02-01.pdf (accessed 15.08.2005).

[47] K. Hosanagar, R. Krishnan, M. Smith, and J. Chuang. Optimal pricing of content delivery network (CDN) services. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, pages 205–214, January 2004.

[48] Y.-F. Huang and C.-C. Fang. Load balancing for clusters of VOD servers. *Information Sciences*, 164:113–138, 2004.

[49] R.-H. Hwang and P.-H. Chi. Fast optimal video placement algorithms for hierarchical video-on-demand systems. *IEEE Transactions on Broadcasting*, 47(4):357–366, December 2001.

[50] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of Internet instrumentation. In *Proceedings of IEEE INFOCOM'00*, pages 295–304, March 2000.

[51] X. Jia, D. Li, X. Hu, W. Wu, and D. Du. Placement of web-server proxies with consideration of read and update operations on the Internet. *The Computer Journal*, 46(4):378–390, 2003.

[52] E.L. Johnson, G.L. Nemhauser, and M.W.P. Savelsbergh. Progress in linear programming-based algorithms for integer programming: an exposition. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[53] K.L. Johnson, M.S. Carr, J.F.and Day, and M.F. Kaashoek. The measured performance of content distribution networks. *Computer Communications*, (24):202–206, 2001.

[54] J. Kangasharju. *Internet Content Distribution*. PhD Thesis, L'Universite de Nice - Sophia Antipolis, April 2002.

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

[56] J. Kangasharju, K.W. Ross, and J. Roberts. Performance evaluation of redirection schemes in content distribution networks. *Computer Communications*, 24(2):207–214, 2001.

[57] M. Karlsson, C. Karamanolis, and M. Mahalingam. A framework for evaluating replica placement algorithms. Technical Report HPL-2002, HP Laboratories, Available at http://www.hpl.hp.com/personal/Magnus Karlsson, July 2002.

[58] M. Karlsson and M. Mahalingam. Do we need replica placement algorithms in content delivery networks? In *Proceedings of the International Workshop on Web Content Caching & Distribution (WCW)*, pages 117–128, August 2002.

[59] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[60] Y.K. Kim, J.Y. Kim, and S.S. Kang. A tabu search approach for designing a non-hierarchical video-on-demand network architecture. *Computers and Industrial Engineering*, 33(3-4):837–840, 1997.

[61] J.G. Klincewicz and H. Luss. A dual-based algorithm for multiproduct uncapacitated facility location. *Transportation Science*, 21:198–206, 1987.

[62] S.G.M Koo, C.S.G. Lee, and K. Kannan. Using P2P to Distribute Large-volume Contents Research Problems, Solutions and Future Directions. In *Proceedings of the 9th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2005)*, July 2005. Available online at http://min.ecn.purdue.edu/ koo/publications/problems.pdf (Accessed 15.08.2005).

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

[64] M. Labbé, G. Laporte, I.R. Martín, and J.J.S. González. Locating median cycles in networks. *European Journal of Operational Research*, 160:457–470, 2005.

[65] 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.

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

[67] Y.-W. Leung and E.W.M. Wong. An incentive charging scheme for video-on-demand. *Journal of the Operational Research Society*, 52:55–63, 2001.

[68] B. Li, M.J. Golin, G.F. Italiano, X. Deng, and K. Sohraby. On the optimal placement of web proxies in the Internet. In *Proceedings of IEEE INFOCOM'99*, volume 3, pages 1282–1290, New York, March 1999.

[69] Y. Li and M.T. Liu. Optimization of performance gain in content distribution networks with server replicas. In *Proceedings of the 2003 Symposium on Applications and the Internet (SAINT'03)*, 2003.

[70] T.D.C. Little and D. Venkatesh. Prospects for interactive video-on-demand. *IEEE Multimedia*, 1(3):14–24, Autumn/Fall 1994.

[71] D. McDaniel and M. Devine. A modified Benders' partitioning algorithm for mixed integer programming. *Management Science*, 24(3):312–319, 1977.

[72] P.B. Mirchandani. The p-median problem and generalizations. In P.B. Mirchandani and R.L. Francis, editors, *Discrete Location Theory*, chapter 2, pages 55–117. John Wiley & Sons, 1990.

[73] I. Murthy and D. Ghosh. File allocation involving worst case response times and link capacities: model and solution procedure. *European Journal of Operational Research*, 67:418–427, 1993.

[74] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1999.

[75] Nortel Networks. Content delivery network solutions. White Paper. Available at http://www.nortelnetworks.com (retrieved May 2004).

[76] T. Nguyen, C.T. Chou, and P. Boustead. Resource optimization for content distribution networks in shared infrastructure environment. In *Proceedings of the Australian Telecommunications Networks Applications Conference (ATNAC 2003)*, 2003. Available at http://atnac2003.atcrc.com/ORALS/NGUYEN-resource.pdf.

[77] T.V. Nguyen, C.T. Chou, and P. Boustead. Provisioning content distribution networks over shared infrastructure. In *Proceedings of the 11th IEEE International Conference on Networks (ICON2003)*, pages 119–124, 2003.

[78] O. Oğuz and T. Bektaş. A fast algorithm for cover inequality separation. Technical report, Bilkent University, 2005. (Under revision for Operations Research Letters).

[79] C.A.S. Oliveira and P.M. Pardalos. A survey of combinatorial optimization problems in multicast routing. *Computers and Operations Research*, 2004. In press.

[80] I. Ouveysi, L. Sesana, and A. Wirth. *Operations Research / Management Science at Work: Applying Theory in the Asia Pacific Region*, volume 43 of *International Series in Operations Research and Management Science*, chapter The video placement and routing problem, pages 53–71. Kluwer Academic Publishers, 2002.

[81] I. Ouveysi, K.-C. Wong, S. Chan, and K.T. Ko. Video placement and dynamic routing algorithms for video-on-demand networks. In *Proceeding of the IEEE Globecom'98 Conference*, volume 2, pages 658–663, 1998.

[82] M. Padberg. The Boolean quadric polytope: some characteristics, facets and relatives. *Mathematical Programming*, 45(1):139–172, 1989.

[83] J. Pan, Y.T. Hou, and B. Li. An overview of DNS-based server selections in content distribution networks. *Computer Networks*, 43:695–711, 2003.

[84] G. Peng. CDN: Content Distribution Network. January 2003. Available at http://citeseer.ist.psu.edu/peng03cdn.html.

[85] H. Pirkul. An integer programming model for the allocation of databases in a distributed computer system. *European Journal of Operational Research*, 26:401–411, 1986.

[86] F. Plastria. Formulating logical implications in combinatorial optimisation. *European Journal of Operational Research*, 140(2):338–353, 2002.

[87] L. Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *Proceedings of IEEE INFOCOM'01*, volume 3, pages 1587–1596, Anchorage, AK, USA, April 2001.

[88] P. Radoslavov, R. Govindan, and D. Estrin. Topology informed Internet replica placement. *Computer Communications*, 25:384–392, 2002.

[89] J. Ryoo and S.S. Panwar. File distribution in networks with multimedia storage servers. *Networks*, 38(3):140–149, 2001.

[90] S. Saroiu, K.P. Gummadi, R.J. Dunn, S.D. Gribble, and H.M. Levy. An analysis of Internet content delivery systems. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[91] E.L.F. Senne, L.A.N. Lorena, and M.A. Pereira. A branch-and-price approach to p-median location problems. *Computers and Operations Research*, 2004. In press.

[92] A. Tamir. An $\mathcal{O}(PN^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59–64, 1996.

[93] A. Vakali and G. Pallis. Content delivery networks: status and trends. *IEEE Internet Computing*, 7(6):68–74, November-December 2003.

[94] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. The potential costs and benefits of long-term prefetching for content distribution. *Computer Communications*, 25(4):367–375, 2002.

[95] D.C. Verma. *Content Distribution Networks: An Engineering Approach*. John Wiley & Sons, 1st edition, 2001.

[96] C.-F. Wang, B.-R. Lai, and R.-H. Jan. Optimum multicast of multimedia streams. *Computers and Operations Research*, 26:461–480, 1999.

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

[98] L.A. Wolsey. *Integer Programming*. John Wiley & Sons, 1998.

[99] 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.

[100] 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*, pages 560–569. Springer-Verlag, 2003.

[101] H. Yaman and G. Carello. Solving the hub location problem with modular link capacities. *Computers and Operations Research*, 32:3227–3245, 2005.

[102] M. Yang and Z. Fei. A model for replica placement in content distribution networks for multimedia applications. In *Proceedings of IEEE International Conference on Communications (ICC '03)*, volume 1, pages 557 –561, 2003.

Appendix A

Linearization of the MCDNP model

The integer programming formulation proposed for the MCDNP in Chapter 3 is cubic due to the objective function. In what follows, we provide a linearization that will allow the resulting formulation to be solved using any mixed integer programming software.

The linearization of the formulation of MCDNP can be done as follows. After simple substitutions, the objective function given in (3.9) is reduced to the following:

$$\text{minimize} \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} \sum_{s \in S} \sum_{k \in K} (b_k d_{ik} (c_{ij} x_{ij} + c_{js} (x_{ij} t_{js} - x_{ij} t_{js} z_{jk}))) \quad \text{(A.1)}$$

We define the binary variable $\delta_{ijs} = x_{ij} t_{js}$, which indicates whether client i is connected to proxy server j, which is connected to the origin server s or not. We further define another binary variable β_{ijks} as $\beta_{ijks} = x_{ij} t_{js} z_{jk}$. This variable takes the value 1 when client i is connected to proxy server j holding the object k, which in turn is connected to the origin server s. The linearization is given by

the following proposition.

Proposition 9 *The simplified objective function given in (A.1) can be linearized using the following constraints:*

$$\delta_{ijs} \geq x_{ij} + t_{js} - 1, \qquad \forall i \in I, j \in J, s \in S \tag{A.2}$$

$$x_{ij} + t_{js} + z_{jk} \geq 3\beta_{ijks}, \quad \forall i \in I, j \in J, s \in S, k \in K \tag{A.3}$$

where $\delta_{ijs} = x_{ij}t_{js}$ *and* $\beta_{ijks} = x_{ij}t_{js}z_{jk}$.

Proof We provide a proof for constraint (A.2). The proof for constraint (A.3) is similar. In (A.2), δ_{ijs} is a binary variable since $\delta_{ijs} = x_{ij}t_{js}$. Consider the case where $x_{ij} = t_{js} = 1$. Then, $\delta_{ijs} = 1$. The remaining cases are $x_{ij} = 1$, $t_{js} = 0$ or $x_{ij} = 0$, $t_{js} = 1$ or $x_{ij} = 0$, $t_{js} = 0$. For each of these cases, $\delta_{ijs} = 0$, since the objective is to minimize and the coefficient of δ_{ijs} in the objective function is positive. □

We can now construct the integer linear programming formulation of the MCDNP as $\{Minimize \sum_{j\in J} f_j y_j + \sum_{i\in I}\sum_{j\in J}\sum_{s\in S}\sum_{k\in K}(b_k d_{ik}(c_{ij}x_{ij} + c_{js}(\delta_{ijs} - \beta_{ijks}))) : s.t.(3.10) - (3.17), (A.3), (A.2)\}$.