Two Large-Scale Network Design Problems

by

Olga Raskina

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2003

ABSTRACT

Two Large-Scale Network Design Problems

In this work we consider two large-scale network design problems arising in network routing and provisioning. The first problem combines long-term business and design decisions on large-scale telecommunication networks. The problem integrates different elements of network business planning such as capacity expansion, routing and protection and equipment maintenance as well as a complicated pricing model; which altogether make the optimization particularly hard. We propose a scalable optimization algorithm that exploits the structure of the model and produces a fast, high-quality solution. In the second part of this work we consider the maximum concurrent flow problem and analyze one of the first special-purpose approaches, proposed in 1971 by Fratta, Gerla and Kleinrock, and show that it yields a fully polynomial-time approximation scheme.

Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

First and foremost I must express my deep an sincere gratitude to my advisor, Professor Daniel Bienstock for his guidance, support and patience, as well as his keen sense of humor that led me through the years of my study at Columbia University. His help and advice, both in technical and personal issues is something I am going to remember and value for the rest of my life.

I would like to thank the members of the committee: Donald Goldfarb, Clifford Stein, Edward Coffman and Iraj Saniee for valuable remarks that helped to improve this work.

I am grateful to my mentors at Bell Labs, Iraj Saniee and Qiong Wang for their assistance and support without which a big part of this dissertation would not have been possible. I also thank Ramesh Nagarajan, my mentor at Lucent Technologies, for my first exposure to the world outside of academia and for his help and patience that made my first professional experience valuable and unforgettable. I would like to thank the wonderful staff of IEOR department office, who had always provided qualified help with all kinds of problems.

My special thank you to my husband Anton, who has always been there for me and put up with more than I could ask for during this whole ordeal.

I also thank all my friends for making my student years at Columbia memorable

and colorful. I choose not to list all their names, since I do not want to miss any of them.

Last, but not least I want to thank my mother who has been the source of my strength and inspiration through all the years.

Chapter 1

Introduction

In this thesis we deal with two network planning and operation problems. The first problem concerns multiperiod business planning of a telecommunication network while the second is a version of the maximum throughput routing problem.

Evolving technologies in telecommunication networks offer the capability of very large wide-area networks with throughput on the order of terabits (10^{12} bit) per second. These technologies are considered very promising for the next generation of transport networks, as they can satisfy the growing demand for bandwidth that is caused primarily by the explosive growth of web-related services offered over the Internet. The challenge is to react quickly to these increasing bandwidth requirements while maintaining a reliable service. Networks should be designed and operated so as to provide adequate capacity in the areas with growing demand without reducing

revenue. Models of this kind give rise to very difficult optimization problems of constantly increasing size and complexity due to both the growth of network size and traffic throughput, and to increasing concerns about data privacy, reliability, and security. These decision and optimization problems generally require approaches that dynamically update static solutions based on the current data and the state of the network. Despite continuing advances, computer technology is still unable to keep up with the growth in model and data sizes. At the same time, it is evident that empirical and intuitive heuristics can no longer fulfill the need for fast and provably good solutions. Clearly, there is a demand for sophisticated algorithms employing state-of-the art mathematical and computing techniques to answer the growing modeling and optimization needs.

In this work we provide scalable and rigorous algorithms for two difficult large-scale network design problems. The first problem deals with long-term planning and operation of an optical network. The major aspects of such planning involve decisions spread over several time periods, which concern demand pricing, routing and protection as well as network design and capacity expansion. Each of these components gives rise to a difficult optimization problem in its own right; thus the integration of all these components poses a significant challenge to current optimization techniques. We consider network operators who contemplate capacity planning in their networks for a period that spans approximately a decade. A key feature in our model is the ex-

pectation that, during such a period, several generations of transmission technologies will emerge, technologies that will outperform prior ones by improved measures of throughput such as the maximum number of wavelengths, capacity per wavelength, or both.

However, there can be incentives to delay the deployment of new technology to exploit savings from future cost reductions. A dynamic technology environment of this kind immediately poses two interesting problems. First, there is one of timing: when should the operator start to deploy new systems and phase out old technologies? Second, there is one of sizing: what capacity that be deployed on each link during every time period? These two questions have been referred to as the technology substitution problem and capacity expansion problem, respectively. In our model, the two problems are closely interrelated and will be considered together.

The network planning problem above has been discussed extensively in the literature. In many cases, it has been cast as an optimization problem of choosing the capacity configuration strategy that satisfies forecast demands (which are given as inputs) at minimum cost (which includes both the cost of deploying new capacities and the cost of operating old systems); see for example, [5]. While min-cost modeling rationalizes decision-making from an engineering perspective, more comprehensive approaches have been taken in recent studies to integrate capacity planning into the overall business optimization strategy [37], [20]. Under the generalized framework,

instead of fixing forecast demands, the price-demand relationship is taken into consideration and demands are treated as flexible quantities that can be controlled by prices. As decision variables, prices play a dual role. On the one hand prices determine the revenue for each unit of service rendered. On the other hand demands cannot deployed capacity. To put it differently, capacity planning is driven not by the need to meet a fixed demand target, but by the desire of generating more profit. The objective is to choose the prices of network services and network capacity simultaneously, so as to maximize the overall net profit generated by the network. In a multiple period model such as ours, the net profit is referred to as Net Present Value (NPV), and is defined to be the total revenue minus the total cost over the planning horizon, using an appropriate discount factor.

To meet the challenge of solving the generalized multi-period price setting and capacity planning problem, our work develops a set of efficient, scalable algorithms. The overall problem is formulated as a nonlinear optimization model with a concave objective function and linear constraints. While a concave maximization problem is (in theory) solvable using standard techniques, the problem sizes arising in realistic settings makes such models essentially impossible to solve by conventional methods, within anything even remotely resembling a reasonable time frame. The algorithm described in this work provides a considerable improvement in terms of both optimality error and computing time, when compared to state-of-the-art general-purpose

solvers.

The second problem considered in this work is a version of the maximum throughput routing problem, namely the canonical version, called the *maximum concurrent flow problem*. The maximum concurrent flow problem frequently arises in practical applications and has received much attention because of its difficulty.

Many telecommunication network design problems contain a multicommodity flow problem as one of the components. Generally, in a multicommodity flow problem we need to route a set of demands (or traffic requirements) over a given network with prespecified joint capacities on its edges. The minimization version of the problem requires a minimum cost routing, while the feasibility version merely tries to find a routing of the demands such that no edge capacity is violated. It is well known that even the feasibility version is fairly difficult. When the problem turns out to be infeasible it can be significant to determine the extent of the infeasibility. In that case we want to find the maximum throughput of the network, that is the maximum common fraction of the demands that can be routed simultaneously without exceeding any of the network capacities. This is called the maximum concurrent flow problem.

This problem arises in telecommunications in many different variations (directed or undirected edges, restrictions on the routings, etc.). Further, many algorithms for network design rely on solving maximum concurrent flow problems in order to find a feasible routing.

What makes this problem especially noteworthy is that it gives rise to extremely difficult linear programs – instances of a size and type relevant to applications often prove beyond the reach of state-of-the-art linear programming codes, despite recent progress in computer technologies. For this reason there has been sustained research devoted to the development of approximation algorithms. The use of approximation is justified, as a fast, partially accurate result is much more desirable than an algorithm that provides an exact solution but requires an excessive amount of time.

In this work we analyze the convergence of one of the first approximation algorithms developed for this type of problem as well as some enhancements of the algorithm that help to improve performance and simplify implementation. In addition we offer a further refinement of this algorithmic approach as a direction for future research.

This thesis is organized as follows.

In Chapter 2 we present our work on the first problem – multiperiod business planning of a telecommunication network. Section 2.2 reviews several essential optimization concepts used in our work, as well as some essential components of telecommunication network design, such as capacity planning and routing and protection. We also provide a brief review of a recently developed demand-price model used in our design. Section 2.3 introduces the basic underlying model and discusses the major differences between this model and classical network design problems. Section 2.4 gives a broad

overview of the optimization algorithms and the resulting changes in the model. Section 2.5 analysis the complexity of the problem and the performance of some well known commercial optimizers. In Section 2.6 we expand one of the key concept of the problem – the survivability of the network, and give precise formulations of the four different optimization models resulting from applying different protection and integration techniques. Section 2.7 outlines the stages of the algorithm and presents the landmarks of each step. Section 2.8 provides the detailed description of the algorithm and the justification of the upper bound technique. Section 2.9 describes the heuristic used to obtain the initial solution. Section 2.10 presents the summary and the analysis of the extensive experiments conducted with real-life networks and also large generated networks. Section 2.11 concludes with a discussion of the results.

Chapter 3 proceeds with the second part of this work – the maximum concurrent flow problem. In Section 3.1 we introduce the problem and give an overview of the previous work. In Section 3.2 we formulate the problem and introduce some basic notation used in our work. Section 3.3 outlines the algorithm and explains some essential underlying ideas. Section 3.4 describes the algorithm in detail and then proceeds with the complexity analysis. In Section 3.5 we describe and analyze a slightly different version of the algorithm. Finally Section 3.6 presents some ideas for future research.

Chapter 2

Business Planning of Optical Networks

2.1 Introduction

Strategic planning for an optical network involves sophisticated decision making in many respects. While a major component of this task is to minimize cost by optimizing routing and protection of network traffic and by determining network capacity, the business impact of these schemes should also be put into a broad perspective. Specifically, cost reductions resulting from routing and capacity decisions can be translated into pricing advantages that enable the carrier to generate more demand and revenue. Therefore, optimizing pricing and demand should be integrated into the optimization

model. To maximize the total value to the carrier, routing, capacity, and pricing decisions should not be examined individually, but rather be incorporated into an integrated business model and optimized systematically.

In this work we develop efficient algorithms for solving capacity planning problems for long-distance optical transport networks. In our model, a network is formulated as a set of nodes, representing cities and metropolitan areas, connected by links, representing physical routes (right-of-way) owned by the network operator. Communications between a node pair are carried by transmission systems deployed on links that connect the pair. A transmission system is composed of a collection of optical wavelengths, where each wavelength is a carrier of information with a fixed amount of capacity. The capacity of a system can be set at different levels by configuring different numbers of wavelengths, up to the maximum. Therefore, the cost of deploying new capacity involves a fixed investment of installing the initial system and a variable cost of adding wavelengths. Each system or wavelength in use is also associated with a recurring maintenance cost, also defined as the operating expense.

It is natural to plan the capacity expansion over several time periods. During such a span of time, several generations of transmission technologies will emerge. A later technology always outperforms previous ones by increasing the maximum number of wavelengths, capacity per wavelength, or both. Though a newer system may have a higher fixed cost and/or variable cost per wavelength, the magnitude

of capacity improvement usually will outpace the corresponding cost increase. As a result, a fully-loaded new system usually has a lower deployment cost per unit of capacity than the older ones, which makes it an attractive candidate for new deployment. Nevertheless, after a new technology becomes available, the deployment cost of systems and wavelengths decreases over time due to a learning effect.

Implementing the value-based approach in large-scale optical networks poses significant challenges to optimization techniques. In comparison with the min-cost capacity planning model, the new approach introduces a set of pricing variables, whose number is on the order of n^2T, where T is the number of planning periods and n is the number of nodes in the network (and, consequently, the number of node pairs between which the price of network service needs to be determined grows proportional to n^2). Furthermore, since revenue equals price times demand and demand is a function of price, the pricing variables are nonlinear components of the revenue in the objective function. As a result when planning a national or international optical transport networks (which usually contain $50 - 100$ nodes) over a 10-period horizon, applying the generalized model can easily lead to a nonlinear model with tens of thousands, or even millions of variables.

The network aspect of the problem brings yet another set of complexities for optimization. To ensure reliable communications in the presence of possible node and link failure, optical transport networks are designed to have multiple non-overlapping

paths between every node pair. The non-uniqueness of routes leads to questions about which routes should be used to carry demand, and in what quantity. Additional planning is required to specify the rerouting of traffic when a network failure occurs. There are several different restoration models that can be employed in this framework. For example, we can reroute all or some of the commodities to utilize the existing network capacity. Or, we can only allow to reroute the commodities that used the disconnected link or node. Or, some resilient capacities are reserved throughout the network and used only to restore the some or all of the affected traffic. See [7], [31], [1], [2] for the discussions of different models and solution approaches.

These routing and protection decisions need to be modeled in the overall optimization framework. As it will be shown in the following sections, the efficient routing and protection schemes lead to vastly different formulations of the overall optimization model. Formulations of some complicated schemes require the introduction of $O(n^2T)$ additional flow variables. The resulting increase in the problem size makes it necessary to develop efficient algorithms.

We formulate the general problem as a large-scale non-linear integer model. The objective is to maximize the carrier's net present value over an extended period of time with constraints on network topology (right-of-way), technology roadmap, and price-demand relationship. In this model, not only demands and capacities but also end-to-end demand allocations and prices are determined by period, with price being

a nonlinear function of the demand following a certain model to be discussed later. In addition, protection channels are to be allocated according to the specified protection model. Our task is to determine the amount of traffic for each commodity in the network at each time period and also the amount of fibers and wavelengths required. Once the equipment is bought we have an option of maintaining it for any number of periods at some increasing fraction of its initial cost. Furthermore, it can also be retired at any time at no additional charge and once retired will no longer be available at any consecutive period.

The modeling of demand as a nonlinear function of price in the context of telecommunication systems was first proposed in [20]. In that work, the solution methodology could only handle small networks ($5-10$ nodes) which in addition had ring topology. Further, experimentation showed that the methodology in [20] would not scale. In this thesis we expand that model to general networks and create a more precise model of the maintenance cost structure and protection capabilities. In [20] the maintenance cost of any equipment depends only on the current time period. In other words, the cost of maintaining a mile of fiber during any period will be the same for a year old fiber and for a ten years old fiber. However, this is not the case in real life. Equipment tends to wear and hence older fibers will usually require higher maintenance expenses. In our work we view this expenses not as a function of current period, but rather as a function of the current service time. In all fairness, [20] deals mainly with build-

ing a precise economic model and developing accurate parameters. We instead take these parameters as input constants and concentrate on the optimization techniques, suitable for large-scale problems.

2.2 Preliminaries

In this section, we discuss several key concepts of the generalized capacity planning model. The scope of our study is outlined in Figure 2.1 and is explained as follows.

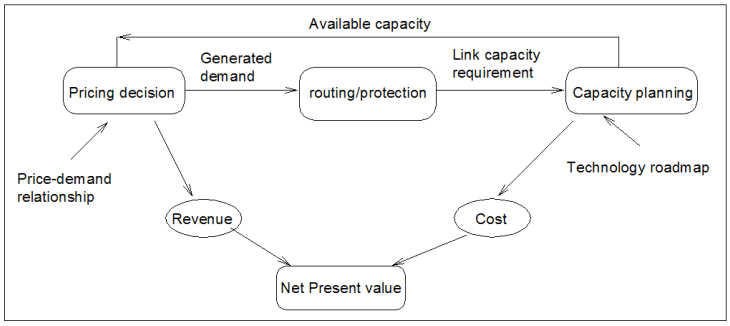

Figure 2.1: Relationship between demands, prices and capacities

Based on the price-demand relationship, a given choice of prices determines corresponding amounts of demands and revenues. The routing and protection decisions then map generated demands to link capacity requirements. These requirements are satisfied by the deployment of transmission systems, whose availability, cost, and capacity are specified by the technology roadmap. Pricing decisions and capacity planning are thus interrelated, as demands generated cannot exceed available capac-

ity.

These three decisions (pricing, routing/protection, and capacity planning) are embedded in the overall optimization framework that seeks to maximize the net present value.

In the following, we discuss each of the three decisions as well as some general optimization techniques and models used in our work.

2.2.1 Price-demand relationship

A modern telecommunication network is determined by continously changing costs and a rapid reflection of the changes on the amount of traffic through the network. To reflect these properties in modeling the demands and prices for the optical networks we first need to introduce the elasticity notion. Let d be the demand and p be the price per unit of the demand. The demand elasticity ϵ for a given time period is defined as the negative ratio of the relative change in the demand to the relative change in the price during that period. That is

$$\epsilon = -\frac{\Delta d/d}{\Delta p/p} \tag{2.2.1}$$

Here Δd is the change in demand and Δp is the relative change in price. Commonly, the demand increases as the price decreases and in that case elasticity ϵ is positive.

Writing

$$\Delta d/d = -\epsilon \frac{\Delta p}{p} \tag{2.2.2}$$

we can see how different values of ϵ affect the demand-price relationship. If $\epsilon > 1$ then any decrease of price gives a larger increase in demand, whereas if $\epsilon < 1$ the same reduction in price leads to a smaller increase in demand. For example, if the price drops by 10% in the first case the demand grows by more than 10% and in the second case by less than 10%.

Taking the limit of (2.2.2) we get

$$\epsilon = -\frac{d'/d}{p'/p} \tag{2.2.3}$$

Let revenue be the product of the demand and price, that is $Rev = pd$, then from (2.2.3) we get the expression for the change in revenue, that is

$$\frac{Rev'}{Rev} = (\frac{\epsilon - 1}{\epsilon})\frac{d'}{d} = -(\epsilon - 1)\frac{p'}{p} \tag{2.2.4}$$

This relationship demonstrates the effect of the elasticity value on the price-demand dependence. In particular, for any value $\epsilon > 1$ the decrease in price causes an increase in the revenue, on the other hand for $\epsilon < 1$ a decrease in price leads to a reduction of the revenue.

For this reason in our work we assume that the elasticity is a constant greater than 1. Then solving (2.2.3) we obtain

$$d = ap^{-\epsilon} \tag{2.2.5}$$

where a is a positive constant. Using (2.2.5) the revenue can be written as a function of demand only, giving

$$Rev = a^{1/\epsilon} d^{1-1/\epsilon} \tag{2.2.6}$$

Under the assumption that $\epsilon > 1$, the revenue Rev is a monotonically increasing and concave function of d. In other words, the network operator derives a higher revenue by reducing the price and therefore increasing demand, but with diminishing return to demand size. This price-demand model was originally developed by Lanning, et. al. for a simplified network model. More details on the demand-price relationship and the computation of the elasticity can be found in [20].

2.2.2 Routing and Protection

The reliability or survivability of a network has received a considerable amount of attention in the network design research and is a vital feature of virtually any telecommunication system.

Any fiber cut or other hardware failure even at a single point can result in the malfunction of the whole system, unless a fast and reliable restoration of the traffic is available. This restoration usually involves rerouting the demands on an alternative link or node-disjoint path. The routing and protection, which can be implemented in many different ways, map end-to-end demands to link capacity requirement and can be implemented in several different ways.

There are two most commonly used restoration strategies – link restoration and path restoration.

- Path protection/restoration

  In path protection each commodity pair has a statically reserved backup path, that is link- or node-disjoint with all working paths. In path restoration each commodity dynamically determines a failed end-to-end path and reroutes the traffic on a backup path.

  - Dedicated-path protection

    In this approach a set of disjoint paths is specified for each node pair, and on each of these paths, some amount of bandwidth is allocated *exclusively* to the pair. The allocated bandwidth satisfies the requirement that if any one of these paths is disconnected by a link or node failure, there will still be enough capacity on remaining paths specified for this pair to carry all traffic. The scheme provides full protection for every commodity against

single path failure, is simple for restoring traffic when failure occurs, but not efficient in bandwidth usage. This model is also referred to as a shared protection on single demand since the same spare capacity allocated for the commodity can be used to restore traffic from any of the paths connecting this pair.

– Shared-path protection

Under this approach, each node pair is assigned a set of working paths and a set of protection paths. Bandwidth on protection paths can be shared among different node pairs that do not have common links or nodes on their working paths. With careful planning, shared protection uses much less redundant bandwidth, as compared to dedicated protection, to fully restore traffic in case of single link failure [14]. Nevertheless, rather than maintaining the same routing during the failure, as is the case with dedicated protection, shared protection usually requires rerouting of traffic based on the location of failure. This approach also improves bandwidth efficiency at the cost of the restoration ability and make the restoration mechanism more complicated.

– Path restoration:

In this scheme all commodities with traffic on the failed link participate in a distributed algorithm that dynamically discovers a restoration route for

each commodity. If for some commodity no feasible route is found, this connection is blocked.

- Link protection/restoration

  Link protection/restoration is the scheme by which traffic is rerouted only around the failed link, regardless of the particular demand pairs that contribute to the flow on that link. The corresponding link backup paths are reserved in advance. The end nodes of the link dynamically determine a failure and reroute all traffic on the protection path. Similarly, in this approach there are three protection schemes.

  - Dedicated-link protection

    Every working link of the network has a predetermined backup path around this link, on with spare capacity allocated exclusively to the traffic on the link

  - Shared-link protection

    The capacity on the restoration path around the link can be used by other backup path.

  - Link restoration

    The end nodes of the link discover the restoration path dynamically and if no feasible path found all connections using the link are blocked.

Generally, path restoration requires more complicated hardware that maintains the set of restoration paths for each commodity pair and also the information of the source and destination of each unit of traffic through every link. On the other hand it has been empirically shown that path restoration requires less total spare capacity [40]. For this reason in this work we consider only path protection/restoration technique. In addition, in our design we require a one hundred percent path restoration strategy to provide a high degree of network survivability. Thus in this work we concentrate on only two approaches, dedicated-path protection and shared-path protection.

We consider two different ways of incorporating routing and protection into our model. The status quo approach is to treat the distribution of traffic on different paths as a lower-level, separate problem from the business optimization carried out by our model. Using planning tool specialized for routing and protection design [14], we can obtain the percentages of demand to be carried on each path for every node pair based on some "seed demand". We then apply these fixed percentages to our model as input parameters. We call this method static traffic distribution. The second method does not use any pre-processor to specify the demand fractions, and views traffic distribution as decision variables. Consequently, traffic distribution may vary with time as different amounts of demands are generated in each period. We define this method as dynamic traffic distribution.

In the next section, we first develop a base formulation that assumes dedicated protection and static traffic distribution. We then present formulations of shared protection and dynamic traffic distribution as extensions to the base model.

2.2.3 Capacity Planning

Capacity planning involves decisions on installation and retirement of transmission equipment of different technologies over time. The major decisions to be made in this setting are the amounts of equipment bought, maintained, and retired at every time period. In our model we assume that any amount of the existing equipment can be retired at any time at no additional cost, and once retired it is no longer available at any consequent time period. The equipment of different technologies is characterized by different acquisition costs and amount of wavelengths or capacity.

The cost of deploying a technology depends on technology type, and the timing of installation. Equipment of a later technology type usually has a better capacity/cost ratio than older equipment. The cost of given technology decreases over time, and for a given technology, the deployment cost is lower at a later installation date. Furthermore, an optical transmission system includes amplifiers and regenerators that need to be deployed on the intermediate points on the link between transmission terminals, and the distance between two adjacent points is approximately the same for all links. As a result, links of different lengths require different numbers of these components,

and thus the deployment costs are link-dependent. Likewise, the recurring maintenance costs per period also depends on technology, time, and link. In our models, we assume that for a given piece of equipment, the deployment cost of a technology decreases at a fixed rate over time after it becomes available and the maintenance cost of a system or wavelength is a fraction of its initial installation cost, and increases over time at a constant rate.

2.2.4 Cutting plane algorithms for constrained nonlinear optimization

Let $f(x) \in \mathbb{C}^1$ be a convex function and consider a problem

$$\min_{x \in Q} f(x)$$

for some convex set $Q \in \mathbb{R}^n$. The main idea of the cutting plane method to solve this problem is to generate a series of hyperplanes which approximate $f(x)$ from below and provide a piecewise liner approximation to $f(x)$. This approximation is then optimized over Q, a new approximation point is generated, and the piecewise linear approximation is updated using a set of rules specific for each particular method.

Let $x_1, \ldots x_k \in Q$ be a set of iterates with function values $f(x_i)$ and gradient values $\nabla f(x_i)$. Then the cutting plane model generates a sequence

$$f_i(x) = \max\{f(x_j) + \nabla f(x_j)^T(x - x_j) | j = 1 \ldots i\}$$

Clearly $f_i(x) \leq f(x)\ \forall x \in Q$. A number of ideas for generating the series of iterates $x_1, \ldots x_k \in Q$ have been proposed over the years. Here we review several basic schemes.

Classical cutting-plane algorithm

This is the simplest algorithm in which the next iterate x_{i+1} is taken to be $\text{argmin}\{f_i(x)|x \in Q\}$. It is known to have a very slow convergence [29].

Line search algorithm

Let $x_i^* = \text{argmin}\{f_i(x)|x \in Q\}$ and set

$$x_{i+1} = \alpha_i x_i + (1 - \alpha_i)x_i^*$$

where

$$\alpha_i = \arg\min\{f(\lambda x_i + (1 - \lambda)x_i^*)|0 \leq \lambda < 1\}$$

This method is also rarely used because in a general case the computation of α_i parameter is very computationally costly. However, for a simple $f(x)$ and a small enough n this method proves to be quite effective.

Bundle algorithm

Different versions of this algorithm are described in [19]. The general idea is to set

$$x_{i+1} = \text{argmin}\{f_i(x) + 0.5u_i(x - x_i^+)^2\}$$

where x_i^+ is chosen among $\{x_1 \ldots x_i\}$ and u_i is the penalty parameter. The idea of this method is to update x_i^+ every time there is an adequate decrease in $f(x_{i+1})$ and

keep x_i^+ unchanged otherwise.

Level algorithm

This model was proposed by Lemarechal et al.[21]

At every iteration of the algorithm we define the level parameter l_i for the current function value and the next iterate is computed as

$$x_{i+1} = \text{argmin}\{0.5(x - x_i)^2 | x \in Q, f_i(x) \leq l_i\}$$

Let $f_*(i) = \min_{x \in Q} f_i(x)$ and $f^*(i) = \min_{j \leq 1} f(x_j)$. Define the ith gap as $\Delta(i) = f^*(i) - f_*(i)$, then for some $0 < \lambda < 1$ set

$$l_i = \lambda f^*(i) + (1 - \lambda) f_*(i) = f_*(i) + \lambda \Delta(i)$$

2.2.5 Newton's method for unconstrained nonlinear optimization

In this section we give a brief description of a second-order algorithm for unconstrained minimization of a general convex function $f(x)$.

This method is based on the Taylor series approximation of $f(x)$. Let $f(x) \in \mathbb{C}^2$ for some $f : \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$. Let $p \in \mathbb{R}^n$. Then

$$f(x + p) = f(x) + g(x)^T p + \frac{1}{2} p^T G(x) p + o(\|p\|^2)$$

where

$$g(x) = \nabla f(x)$$

and

$$G(x) = \nabla^2 f(x)$$

Denote

$$Q(p) = g(x)^T p + \frac{1}{2} p^T G(x) p$$

Newton's method proceeds by iteratively optimizing the quadratic approximation $Q(p)$ to $f(x+p)$ at $f(x)$. We have

$$\nabla Q(p) = g(x) + G(x)^T p$$

and assuming that $f(x)$ is strictly convex, that is $G(x)$ is nonsingular we get

$$\nabla Q(p) = 0 \text{ iff } p = -G(x)^{-1} g(x) \tag{2.2.7}$$

Thus the Newton's method iterates by solving (2.2.7) and updating $x \leftarrow x + p$ until $\|g(x)\| \leq \epsilon$ for some predefined tolerance ϵ. Note that in order to solve (2.2.7) we first need to invert the Hessian matrix $G(x)$. A commonly used method for implicitly inverting a Hessian is Cholesky factorization. This method is used to represent a symmetric positive definite matrix A as

$$A = R^T R$$

where R is called Cholessky factor and is a nonsingular upper triangular matrix. The Cholesky factor can for example be computed using elimination. Given the Cholesky factor R the solution to a system of linear equations $Ax = b$ can be found by solving two triangular systems

$$R^T y = b \quad \text{and} \quad Rx = y$$

Computation of a Cholesky factor requires approximately half the work needed to perform Gaussian elimination. Moreover, in practical application there are numerous methods to further speed up the computations. See [3] and [12] for the discussion of these methods.

Newton's method has quadratic convergence when applied to a convex problem. In fact, it is known that, assuming convexity,

$$\|x^n - x^*\| \leq O(\|x^0 - x^*\|^{2^n})$$

where x^* is the optimum of $f(x)$, x^n is the current iterate and x^0 is the starting point. See [3] for the details of the proof. Thus the actual rate of convergence heavily depends on the initial error and whence obtaining a good starting point is essential for this method. The Newton's Method can also be used for optimizing a linearly constrained convex function using a set of sequential unconstrained optimizations. This approach will be described in more details in Sections 2.7 and 2.8.

2.2.6 Standard network design problem

The network design or minimum-cost capacity installation problem is the problem of installing an integer amount of capacities on the links of a given graph such that a given set of demands can be routed simultaneously.

Consider a graph $G = (V, A)$ where V is the set of nodes and A is the set of arcs and a let $d = \{d_k\}$ be the given set of demands (commodities) between the nodes of G. Let K be the number of distinct commodities and for every $1 \leq k \leq K$ let i_k and j_k be the source and destination of demand d_k which in general can be fractional. The capacity on links can be installed in batches of different size.

Let U be the set of different batch types and $M(u)$ is the size (number of capacity units) for each type $u \in U$. The cost of installing a batch of capacity of type u on link $l \in A$ is $c_l(u)$. The batches are installed in integer amounts and any combination of batch types is allowed on any link. The goal is to install the batches of capacity at minimum cost such that all demands are met.

For each commodity k let $P_k = \{p_k^h, 1 \leq h \leq |P_k|\}$ be the set of all paths from i_k to j_k, $f(p_k^h)$ be the amount of flow of commodity d_k on path p_k^h and let $y_l(u)$ be the number of batches of type u installed on link l. The path formulation of the minimum cost capacity installation problem is then

$$\min \sum_{l,u} c_l(u) y_l(u) \tag{2.2.8}$$

$$s.t. \quad \sum_{h} f(p_k^h) \;=\; d_k \qquad \forall 1 \le h \le K$$

$$\sum_{k,:\, l \in p_k^h} f(p_k^h) \;\le\; \sum_{u} M(u) y_l(u) \qquad \forall l \in A$$

$$y_l(u) \in \mathbb{N}^+ \qquad \forall l \in A \; \forall u \in U$$

It is known that the network design problem is NP-hard even for the case when $|U| = 1$ and the generic problem is strongly NP-hard [11] as it contains the set cover problem. This problem has been studied extensively over many years. Magnanti and Mirchandani [25] and Magnanti et al. [26] proposed cut-set inequalities for a single commodity design with multiple batch types where the batch sizes are integer multiples of some basic capacity unit. Bienstock and Gunluk [6] generalize these inequalities to flow-cut-set inequalities for multicommodity design with two batch types, where the size of one batch is a integer multiple of the size of the other batch. Gunluk [10] considers multicommodity problem with two batch types and existing capacity. Chopra et al. [9] proposed inequalities for a single commodity problem on a directed network with two batch types. Pochet and Wolsey [32] analyzed a single link network with arbitrary many batch types and Bienstock et al. [5] study a multicommodity design with a single batch type.

2.3 The core model

Our basic model differs from the standard network design problem in two fundamental aspects. The first and foremost difference is the way the demand values are defined. Unlike the basic problem, in our model the demands are not given in advance. Instead for every pair of nodes we introduce a variable representing the amount of traffic between these nodes. Thus we not only minimize the cost of the equipment on the links, but at the same we also maximize the revenue from the traffic based on the price model defined in the previous section. The second major difference is that we need to solve the problem in the span of multiple time periods. For every time period in the span we again wish to optimize the cost and the revenue of the network. Furthermore at any time period we are allowed to reuse the equipment bought at earlier periods provided we pay the price of maintaining it. We can also retire all or any part of the equipment at any time at no additional cost.

Note that even with these changes the complexity of the problem is at least that of the original network design problem and generally the number of paths in the formulation (2.2.8) can be very large. However, a common approach to survivable network design is to assume a precomputed fixed set of paths for every commodity. This approach can be justified by several reasons. For instance, a set of paths can be predefined by a specific protocol or service used to operate the network, or by an outside provider. For example the OSPF protocol [28] currently employed in

Internet routing assumes a fixed (shortest for some metric) path for every pair of nodes. Further, in practice there is an extensive and sometimes idiosyncratic set of requirements on the paths used to carry traffic. Usually those requirements are non-quantitative and either can not be expressed explicitly as a set of constraints, or would require a large number of binary and integer variables that would make the problem practically intractable.

For example, to satisfy certain requirements of quality of service, the carrier needs to minimize the processing delay incurred at the intermediate nodes. In this case, the candidate paths may be restricted to those that have no more than a certain number of hops between the node pair. The length restriction constraint prevents the demands from taking unacceptably long path in terms of either the number of hops or the actual path length, thus decreasing the probability of a failure of a path component. In addition, to guarantee some degree of network survivability an additional set of path constraints can be imposed. There are several commonly used restrictions utilized in network modeling. For example, $k-$connectivity requires the network to contain at least k node- or link-disjoint path for every commodity pair. Furthermore, concerns about security and privacy may also limit the carrier's choice of candidate paths. See for example [8].

For these reasons in this work we will only look at a given fixed set of paths for every commodity. The algorithm in [14] is used to generate the paths with the required

properties, however any other path generating heuristic can also be employed. For the protection reasons outlined in Section 2.2.2 and specified below the paths for every commodity pair are link-disjoint.

We are now in position to formulate our core model.

2.3.1 Model description

Let $G = (N, A)$ be a given network with $|N| = n$ and $|A| = m$ and let T be the number of discrete time periods.

For any pair of nodes $(i_k, j_k) \in N$ and a time period t let d_k^t be the amount of traffic (demand) between i_k and j_k at t. We will base our model on the path formulation of the standard network design problem. Let P_k be the given fixed set of link-disjoint paths for commodity k, $|P_k| = H_k$.

Thus we also introduce flow variables $f^t(p_k^h)$ which are the amount of commodity d_k^t routed on path $p_k^h \in P_k$, $1 \leq h \leq H_k$.

For any link $l \in A$ and a time period t, let $y_l^t(u) \in \mathbb{N}^+$ be the number of batches of type u bought at t on l, and for any time period $q > t$, let $y_l^{t,q}(u)$ be the amount of batches of type u that were bought at t and kept until q. For the consistency of the notation we let $y_l^{tt}(u) = y_l^t(u)$

We are also given the following constant parameters

- Discount factor h_t. The discount factor at t is usually computed as the inverse of the predicted interest rate at t. The discount factor is used to compute the present value of future income. For a detailed description see [22]. The discount factor is precomputed and used as an input parameter in our model.

- Buying cost $c_l^t(u)$. The cost of buying a batch of type u on link l at time period t. We assume that the buying cost is nonincreasing, that is $c_l^t(u) \geq c_l^{t+1}(u)$ and that the cost per unit of capacity is decreasing as the batch size increases.

- Maintenance fraction $\mu < 1$ and maintenance rate $\alpha > 1$. The cost of maintaining a unit of capacity bought at q until t is defined as $c_l^{q,t} \doteq c_l^t * \mu * \alpha^{t-q-1}$. Thus the maintenance cost is smaller than the buying cost at the first time period and increases as t approaches the time horizon T.

- The batch size $M(u)$ is taken to be the same across the network and the time horizon for every type u.

- Demand-price elasticity $\epsilon_k^t > 1$.

- Demand scaling constant A_k. This constant is usually taken to be equal to the value of the demand when the price $p = 1$; thus A_k can be viewed as the demand potential. We define the demand (i_k, j_k) coefficient at t as $c_k^t \doteq A_k^{1/\epsilon_k^t}$

Using the above constants and the price-demand relationship described in the

previous section we define the revenue at t as

$$Rev_t = \sum_k g_k^t(d_k^t) \tag{2.3.1}$$

where

$$g_k^t(d) = c_k^t d^{1-\frac{1}{\epsilon_k^t}}.$$

The total equipment cost at t is defined as

$$Cost_t = \sum_l \big(c_l^t(u) y_l^t(u) + \sum_{q<t} c_l^{q,t}(u) y_l^{q,t}(u)\big) \tag{2.3.2}$$

which is the cost of buying capacity at t plus the cost of maintaining capacities from the previous periods.

Our objective is then to maximize the net present value, defined as

$$NPV = \sum_t h_t(Rev_t - Cost_t) \equiv F(d, y). \tag{2.3.3}$$

We introduce two types of constraints. The first set is the capacity constraints that are inherited directly from the standard network design problem. We have

$$\sum_{k,h:l\in p_k^h} f^t(p_k^h) \leq \sum_u M(u)\left(y_l^t(u) + \sum_{q<t} y_l^{q,t}(u)\right) \;\; \forall l \in A, \; \forall t$$

$$d_k^t = \sum_P f^t(p_k^h)$$

In addition we restrict the carry over capacity at t not to exceed the capacity at $t-1$

$$y_l^{q,t}(u) \leq y_l^{q,t-1}(u) \;\; \forall l, \;\; \forall q, \;\; \forall t \geq q+1$$

Summarizing all of the above we can now give a complete formulation of the core model.

$$\max \;\; F(d,y) \tag{2.3.4}$$

s.t.

$$\begin{aligned}
\sum_{(k,h:l\in p_k^h} f^t(p_k^h) \;&\leq\; \sum_u M(u)(y_l^t(u) + \sum_{q<t} y_l^{q,t}(u)) && \forall l \in A, \; \forall t \\
d_k^t \;&=\; \sum_h f^t(p_k^h) && \forall k, \;\; \forall t \\
y_l^{q,t}(u) \;&\leq\; y_l^{q,t-1}(u) && \forall \;\; l, \;\; \forall q, \;\; \forall t \geq q+1 \\
y_l^{q,t}(u) \;&\in\; \mathbb{N}^+ && \forall l, \;\; \forall q, \;\; \forall t \geq q
\end{aligned}$$

2.4 Algorithm idea

In this section we will describe the main idea of the algorithm used to solve the above model. The details of the algorithm will be described in Sections 2.7- 2.9.

The optimization problem described in the previous section is a concave maximization problem over the demand variables d and an integer program over the capacity

variables y.

A useful heuristic to handle this type of problems is to relax the integrality constraints first and solve the resulting continuous nonlinear program. After the continuous solution is obtained, the continuous variables are fixed at their current values and the resulting integer problem is solved to optimality using any appropriate method available.

Generally this approach proves to be inefficient for the small values of the integer variables since in that case even a fraction of a unit has a significant contribution to the objective value.

However, in case of telecommunication system the amount of data transmitted over a network is typically very large due to the recent and continuing advances of technology. It is not uncommon to have tens of thousands units of demand between a single node pair. The results of [20] confirm this assumption for ring networks and we expect it to be the case for our model as well. As a result, in our case even a straightforward greedy rounding will produce a solution within a fraction of the lower bound that was obtained by solving the relaxed problem.

In a view of this we adopt the following approach to solve our model:

1. Relax the integrality constraints on the capacity variables and solve the resulting continuous nonlinear problem.

2. Fix the values of the demand and flow variables found in Step 1 and find the

corresponding integer values of the capacity variables.

Note that Step 1 deals with the continuous version of problem (2.3.4). Clearly in that case the optimum solution will only use the capacity batches with the smallest per unit cost. Thus we can reduce the core problem to use only the cheapest capacity on any link at any time period. This is possible since the maintenance fraction and the maintenance rate are the same across all batch types, links and time periods. We define the integrated capacity cost as

$$c_l^t = \min_u \frac{c_l^t(u)}{M(u)},$$

and the core problem becomes

$$\max \sum_t h_t \left(\sum_k g_k^t(d_k^t) - \sum_l (c_l^t y_l^t + \sum_{q<t} c_l^{q,t} y_l^{q,t} \right) \tag{2.4.1}$$

s.t.

$$\sum_{k,h\,:\,l \in p_k^h} f^t(p_k^h) \;\le\; y_l^t + \sum_{q<t} y_l^{q,t} \qquad \forall l \in A,\ \forall t$$

$$d_k^t \;=\; \sum_h f^t(p_k^h) \qquad \forall k,\ \ \forall t$$

$$y_l^{q,t} \;\le\; y_l^{q,t-1} \qquad \forall\ l,\ \ \forall q,\ \ \forall t \ge q+1 \tag{2.4.2}$$

$$y_l^{q,t} \;\ge\; 0 \qquad \forall l,\ \ \forall q,\ \ \forall t \ge q \tag{2.4.3}$$

In what follows we will be looking only at relaxed core problem (2.4.1) and its

extentions. In the subsequent section we will describe two models based on this formulation.

2.5 Difficulty of the core model

Having described a core model in the previous section, we will now pause and consider how one should tackle problems of this sort – a more specific model will be given in the following section.

In this work we will develop an algorithm approach for optimization problems derived from the core model. Our algorithms, as we will show, are fast and experimentally scale well. Our algorithms also take advantage of essentially combinatorial features of the core model. Thus a natural question would be to ask whether standard, e.g. commercial solvers cannot already tackle our problems. In this Section we take up this issue, to explain why the answer to this question is a clear 'no'.

Problem 2.4.1 is a hard nonlinear optimization problem with a steep objective function and a large number of variables even for a moderate size network. For example a regular network with 50 nodes and 70 links considered over 14 time periods can result in a formulation with approximately 60000 variables for the complicated protection model, and a larger network with 150 nodes and 800 links over the same time horizon will already have over one million of variables for the same model. The large number of variables is due to the presence of a decision variable for every path

Set	Nodes	Links	Vars	Const	Nonz	Paths (min)	Paths (max)
snet1	14	22	3584	2310	17472	2	3
snet2	38	48	12782	2940	96222	2	3
snet3	47	55	20909	5775	196623	2	4
snet4	50	64	23870	6720	223272	2	3
snet5	70	94	43680	9870	431382	2	3

Table 2.1: Statistics on real-life models

of every possible commodity, at every time period.

In order to evaluate the difficulty of the problem we conducted experiments with several commercially available solvers on networks of different size. Table 2.1 shows the statistics on the first set of problems arising from real-life data sets. Here the last two columns show the minimum and maximum number of paths per node pair.

Our testing focused on two well-known nonlinear optimizers: LOQO [39] and SNOPT [13]. Both are excellent general-purpose systems designed to handle any nonlinearity in both objective and constraints. Table 2.2 shows the performance comparison of the solvers on the test networks.

The empty cells indicate that the solver was unable to handle the corresponding instance, which in most cases means no convergence in a reasonable time frame and

Solver	snet1	snet2	snet3	snet4	snet5
LOQO	30	600	720	1000	–
SNOPT	90	–	–	–	–

Table 2.2: Performance comparison (LOQO and SNOPT)- running time (sec)

in some cases insufficient memory for the solver to run, even on a machine with ample resources. These tests were conducted on 336 Mhz UltraSPARC machine with 1.7Gb of RAM. What is more, in all of the cases where the solvers were not able to converge either no feasible solution was found, or the optimality gap was unacceptably large.

This test also indicates that although one of the solvers was able to solve the problem on the moderate size networks, it still shows a poor performance in terms of the running time.

Altogether it is evident that the problem cannot be handled by these two commercial optimizers and there is an apparent need for an algorithm that is effective on our models.

Of course, it is impossible to test *every* available solver, but the above experiments lead us to expect that general-purpose solvers will all be ineffective, in particular when run on much larger problem instances than those in Table 2.1. One of the reasons underlying our belief concerns the use of Newton's method (or a similar second-order method) to handle the highly nonlinear objective. This, in turn, requires a good

Set	Nodes	Links	Vars	Const	Nonz	Paths (min)	Paths (max)
bnet1	100	500	745500	406000	5299700	3	6
bnet2	150	900	1659000	889350	15340500	3	7
bnet3	200	1500	4336500	1850100	40267500	3	9
bnet4	250	2000	6746250	2852500	69618500	3	10
bnet5	300	4500	18053700	5558700	311100300	4	12

Table 2.3: Larger networks

Cholesky factorization algorithm for large sparse matrices, but may nevertheless exact too high a computational price. To explore this difficulty, we also tested CPLEX [16] which is one of the most efficient linear, integer and quadratic solver available on the market. In terms of nonlinear programming, CPLEX can only handle (convex) quadratic programs – but CPLEX also has one of the fastest Cholesky implementations. In order to test CPLEX we replaced the original objective with its second order Taylor series approximation and generated a large set of data described below.

Table 2.3 shows the statistics on the five large networks generated for this test.

In order to build the second order approximation for these tests we replaced the original objective function 2.4.1 with its piece-wise linear approximation built at t

Solver	bnet1	bnet2	bnet3	bnet4	bnet5
CPLEX	2341	7982	14324	18943	–

Table 2.4: CPLEX performance - running time (sec)

uniformly distributed points. The resulting objective function was then used to solve the optimization problem over all time periods for each commodity separately using CPLEX linear solver. We used the solution obtained by this method to evaluate the Hessian for the second order Taylor series approximation of the objective function. Note that the purpose of these tests was not to attempt to find a good solution by optimizing a series of second order approximations, but rather to get an understanding of the nature of problem difficulty and thus we optimized only one quadratic approximation for each test case.

Table 2.4 shows the performance of CPLEX on the quadratic relaxation of the problem, when run on the problems in Table 2.3. It is evident that the CPLEX efficiency deteriorates rapidly as the network size and density increase. Thus clearly a state-of-the-art implementation of Newton's method, by itself, is unlikely to improve the performance of a standard nonlinear optimization procedure.

Consequently, our work focuses on an algorithm that exploits the specific structure of the problem constraints and the objective function. Our algorithm will essentially decompose the problem into a set of smaller problems and will project out a subset

of variables; thus drastically decreasing the size of the matrices tackled in Newton's method.

2.6 Protection models

The core model described in the previous section allows us to design a network that has enough capacity to accommodate the required amount of traffic between different node pairs. However this design does not take into account the reliability of the resulting network. Therefore we need to expand this model to accommodate the protection/restoration techniques outline in Section 2.2.2.

In other words, we need to allocate additional amount of capacity on some alternative paths for every demand, such that in case of a single or multiple link failure there is enough slack capacity to restore hundred percent of the affected traffic. In what follows we provide the detailed description of the two previously described approaches to reliability requirements.

2.6.1 Shared protection on single demand (SD)

In this model we assume that for any commodity not more than one of the commodity paths can fail at a time. In case of a failure the affected portion of the demand has to be rerouted through one or more new paths that are link-disjoint with all other paths.

In this case the additional capacity allocated on the new paths can be used to restore flows from any path of the working set. In principle, this condition can be relaxed to allow the restoration capacity to be also allocated on the remaining (functioning) paths of the working set. However, in this case this capacity can not be shared across all the paths of the set and we do not consider this option in our work.

Let P_k be the given fixed set of link-disjoint paths for demand d_k and $Q \subset P_k$ be the working subset used to carry the primary flows of d_k. Let $f(p)$ be the flow carried on some path $p \in Q$ and let f be the maximum of these flows, that is $f = \max_{p \in Q} f(p)$. Then the amount of the protection capacity required to restore any primary flow equals f. This condition can also be viewed as sending an additional f units of flow on some paths of $P_k \setminus Q$. In order to formulate this model exactly, one would need to consider all possible subsets of P which can result in a very large amount of variables and constraints even for a moderate size network. Instead, we restrict the set of restoration paths to be just a singleton. With this additional constraint the model formulation becomes more compact and much less complicated. We replace the demand constraint $d_k^t = \sum_h f^t(p_k^h)$ with a set of constraints

$$f_k^t \;\geq\; f^t(p_k^h) \;\; \forall p_k^h \in P_k \tag{2.6.1}$$

$$d_k^t = \sum_h f^t(p_k^h) - f_k^t \tag{2.6.2}$$

2.6.2 Shared protection across demands (MD)

This model is a generalization of the previous formulation to allow sharing of protection capacities across the multiple demands. Consider a network in Figure 2.2 illustrating shared protection on single demand between nodes i and j. Denote the set of paths P_{ij} for this demand as P.

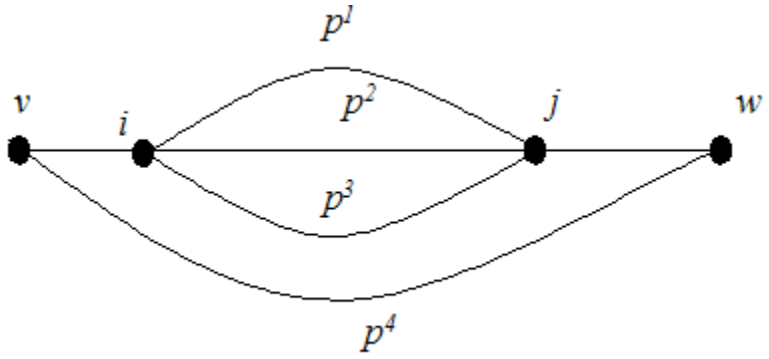

Figure 2.2: Protection example

Let $f(1) = 2$, $f(2) = 4$, $f(3) = 4$, and $d = d_{ij} = 6$. Therefore 2 units of capacity are installed on p^1 and 4 units on each p^2 and p^3. Either p^2 or p^3 can be used for restoration, however if either of them is used to restore $f(p^1)$ the protection capacity will not be used completely. Therefore even if this path fails we can use the existing protection capacity to restore the flow from some other failure. Moreover,

assuming that only one link can fail at a time the entire set of restoration capacities can be utilized by another demand. Consider again the example in Figure 2.2. Let $p^5 = v - i - p^3 - j - w$ and suppose $d_{vw} = f(p^4) = 4$ with p^4 being a primary path. Then if p^3 is the restoration path for d_{ij} the existing protection capacity on p^3 can also be used to restore $f(p^4)$ in case p^4 fails.

The only obstacle here is that it is not known in advance which of the paths will be used for protection. Consequently, in order to be able to share protection across the demands we need to specify the restoration capacity explicitly.

In order to do that we again consider the given link-disjoint set path P and for every pair paths $p, s \in P$ introduce a variable $f(p, s)$ representing the part of $f(p)$ that will be routed on path s in case p fails. In other words, $f(p, s)$ corresponds to the protection capacity required on s to restore the traffic from p. Note that unlike the previous model, here we allow the primary flow to be rerouted through more than one additional path as well as some of the working paths. Thus the assumption that only one link can fail at a time is crucial for our formulation. With this information in hand we can now introduce the set of constraints to formulate the model

1. As in the core model again the demand is equal to the sum of the primary flows

$$d_k^t = \sum_h f^t(p_k^h) \tag{2.6.3}$$

2. Every primary flow has to be restored completely

$$f^t(p_k^h) \;=\; \sum_{h \neq h'} f^t(p_k^h, p_k^{h'}) \tag{2.6.4}$$

3. The capacity on any link l has to be sufficient to meet all primary flows plus the protection flows, taking into account capacity sharing across the demands.

   Let z_l^t be the total protection flow on link l at t.

   Then the capacity constraint becomes

$$\sum_{k,h:l \in p_k^h} f^t(p_k^h) + z_l^t \;\leq\; y_l^t + \sum_{q<t} y_l^{q,t} \quad \forall l \in A,\ \forall t \tag{2.6.5}$$

   Consider a failure of some link e. For any demand that has e in one of its working paths the flow routed on that path has to be restored. Consequently, the protection capacity on l has to be at least the sum of all restoration flows from e on l. Let z_{el}^t represent the protection capacity required on l in case e fails. Then

$$z_{el}^t \;=\; \sum_{k,\, h:e \in p_k^h,\, h':\, l \in p_k^{h'}} f^t(p_k^h, p_k^{h'}) \tag{2.6.6}$$

   where the summation is taken over all demands which can be affected by the failure of e and that can use l in one of the restoration paths. In other words, the summation is taken over all the demands that have disjoint paths using l and e.

Therefore, since we assume that only one link can fail at a time and all the protection capacity can be shared, the total protection capacity required on l is then the maximum over all possible failures

$$z_l^t = \max_e z_{el}^t \tag{2.6.7}$$

Our goal is then to maximize the net present value over (2.6.3)-(2.6.7) and (2.4.2).

2.6.3 Flows on the paths

Generally, telecommunication system design has three major components that contribute to maximization of revenue under given capacity and routing constraints.

1. Strategic planning: decide long-term investment in network systems, manage capital expenditure on strategical deployment of network capacity.

2. Marketing and sales: set price, acquire customers.

3. Network operation: allocation of existing capacity to customers, traffic engineering and routing. This stage is targeting to make the maximum use of existing capacity. In other words, at this stage the decisions about network usage are made. In particular one determines the routing of all the demands, that is the fraction of each demand routed on each of its paths.

The design optimization can be made sequentially over each of the stages. In that case each part is optimized separately based on the solution of the previous stages. This approach allows a relatively easy solution at the expense of the quality of the result. On the other hand, integrating all three parts together creates a much harder optimization problem, but can provide a better overall solution. However, the design stages are usually performed by different teams and the integration is not always easy, if possible at all.

Therefore we would like to compare different approaches to determine whether it is worthwhile to go through the costly integration.

In this work we consider two options. The first option is to optimize the low-level traffic distribution of stage 3 first and use the solution to solve 1 and 2 together. The second option is to integrate all three parts of the process.

Note that the models defined in the previous section reflect the second approach. We now describe the changes to that model that take into account the solution of stage 2. Recall that stage 3 determines how the demands are routed over the network. That is, for every demand d_k in addition to a fixed set of paths P_k we are given a fixed fraction or percentage of the demand that has to be routed on this path.

For every $p_k^h \in P_k$ we are given a percentage constant r_k^h representing the fraction of d_k that has to be routed on p_k^h. For the SD protection model

$$\sum_h r_k^h = 1 + \max_h\{r_k^h\}, \ \ \forall k$$

as defined in section (2.6.1). Using the percentage constants we then can represent the flow variables via the demand variables as

$$f^t(p_k^h) = r_k^h d_k^t$$

and the optimization problem in this case becomes

$$\max F(d, y) \tag{2.6.8}$$

$$s.t.$$

$$\sum_{k,h:l\in p_k^h} r_k^h d_k^t \ \leq \ y_l^t + \sum_{q<t} y_l^{q,t} \qquad \forall l \in A, \ \forall t \tag{2.6.9}$$

$$y_l^{q,t} \ \leq \ y_l^{q,t-1} \qquad \forall l, \ \ \forall q, \ \ \forall t \geq q+1 \tag{2.6.10}$$

$$y_l^{q,t} \ \geq \ 0 \qquad \forall l, \ \ \forall q, \ \ \forall t \geq q \tag{2.6.11}$$

For the MD model, in addition to the percentage constant r_k^h for each path $p_k^h \in P_k$ we are also given a set of constants $r_k^{h,h'}$ for each pair of paths $p_k^h, p_k^{h'} \in P_k$, such that

$$\sum_h r_k^h = 1 \ \ \forall k$$

$$\sum_{h\neq h'} r_k^{hh'} = r_k^h \ \ \forall q \ \ \forall k$$

as defined in Section 2.6.2. Using this constants we rewrite problem 2.4.1 as

$$\max \; F(d, y)$$

$$s.t.$$

$$\sum_{k,h:l\in p_k^h} r_k^h d_k^t + z_l^t \;\leq\; y_l^t + \sum_{q<t} y_l^{q,t} \forall l \in A,\ \forall t \tag{2.6.12}$$

$$y_l^{q,t} \;\leq\; y_l^{q,t-1} \qquad \forall l,\ \ \forall q,\ \ \forall t \geq q+1 \tag{2.6.13}$$

$$z_{el}^t \;=\; \sum_{k,\, h:e\in p_k^h,\, h':l\in p_k^{h'}} r_k^{hh'} d_k^t \tag{2.6.14}$$

$$z_l^t \;=\; \max_e z_{el}^t \tag{2.6.15}$$

$$y_l^{q,t} \;\geq\; 0 \tag{2.6.16}$$

Fixing the percentages of the demands over the paths reduces the size of the problem and makes the optimization process faster and easier. In addition, this model allows the network operator to specify his own requirements and constraints on the distribution of the demands as opposed to choosing the best allocation from the profit point of view.

In what follows we propose the algorithms and bounds for each of the four resulting models and evaluate the implications of the integration.

2.7 Algorithm outline

Recall from the Section (2.4) the two major solution stages:

1. Relax the integrality constraints on the capacity variables and solve the resulting continuous nonlinear problem

2. Fix the values of the demand and flow variables found in step 1 and find the appropriate values of the capacity variables.

Before we move to the main algorithm in Step 1 let us first describe the idea behind step 2. Recall from (2.3.3) that the objective is a concave function of the demand variables and a linear function of the capacity variables. Therefore after the demand values are fixed the objective becomes linear. Moreover, since the set of paths is given and fixed for every demand pair, the values of the flow variables provide the total load on every link. The problem in step 2 becomes

$$\max \sum_t h_t(-Cost_t) \equiv \min \sum_t \sum_l \sum_u h_t(c_l^t(u) y_l^t(u) + \sum_{q<t} c_l^{q,t}(u) y_l^{q,t}(u)) \tag{2.7.1}$$

$$s.t.$$

$$\sum_u M(u)(y_l^t(u) + \sum_{q<t} y_l^{q,t}(u))) \geq L_l^t \qquad \forall l \in A,\ \forall t \tag{2.7.2}$$

$$y_l^{q,t}(u) \leq y_l^{q,t-1}(u) \qquad \forall l,\ \ \forall q,\ \ \forall t \geq q+1 \tag{2.7.3}$$

$$y_l^{q,t}(u) \in \mathbb{N}^+ \qquad \forall l,\ \ \forall q,\ \ \forall t \geq q \tag{2.7.4}$$

Where L_l^t is the load on link l at t computed as

$$L_l^t = \sum_{k,h:l \in p_k^h} f^t(p_k^h)$$

for the SD protection model and as

$$L_l^t = \sum_{k,h:l \in p_k^h} f^t(p_k^h) + z_l^t$$

for the MD model.

It is easy to see that problem (2.7.1) is a combination of separate optimization problems for every link, of a form

$$\min \sum_{u} \sum_{t} h_t(c^t(u)y^t(u) + \sum_{<t} c^{qt}(u)y^{qt}(u)) \tag{2.7.5}$$

$$s.t.$$

$$\sum_{u} M(u)(y^t(u) + \sum_{q<t} y^{qt}(u)) \;\geq\; L^t \qquad \forall t \tag{2.7.6}$$

$$y^{qt}(u) \;\leq\; y^{q,t-1}(u) \qquad \forall q,\;\; \forall t \geq q+1 \tag{2.7.7}$$

$$y^{qt}(u) \;\in\; \mathbb{N}^+ \qquad \forall q,\;\; \forall t \geq q \tag{2.7.8}$$

Note that for $t = 1$ the problem 2.7.5 is a regular knapsack problem. For $t > 1$ it becomes a multiperiod knapsack problem and can be readily solved by a standard integer programming solver. The size of each problem depends on the number of time periods different batch types and usually does not exceed $300 - 400$ variables and constraints. Note that in addition the constraint matrix is sparse, thus allowing to obtain an optimal solution in a matter of seconds.

2.7.1 The nonlinear optimization algorithm

The algorithm in Step 1 is primarily based on the well-known active set method [3]. Consider a general nonlinear problem

$$\min \quad f(x)$$

$$s.t. \quad Ax \leq b$$

1. The active set algorithm starts with a feasible solution which provides a set $\mathcal{A}$ of tight or *active* constraints.

2. Ignore the rest of the constraints and optimize the problem only over the current set of active constraints.

3. If the new solution violates any of the remaining constraints, compute the step length to the constraint boundary and update the set of active constraints. Otherwise, check the Lagrange multipliers and drop the corresponding constraint from the set.

4. If no constraints can be dropped, stop. Otherwise, goto Step 2.

Under an appropriate nondegeneracy assumption, this algorithm terminates in a final number of steps ([3], [23]). In order to carry out Step 2 we usually have to invert the submatrix corresponding to the active constraints. Let $B_{\mathcal{A}}$ be a basis of $\mathcal{A}$ and $N_{\mathcal{A}}$ be the remaining columns.

Set

$$x_B = (B_{\mathcal{A}})^{-1}(b - N_{\mathcal{A}}x_N) \equiv \mathcal{B}x_N + d$$

and optimize

$$\bar{f} \equiv f(\bar{x})$$

where $\bar{x} \equiv \mathcal{B}x_N + d, x_N$.

In general, it is not known in advance which variables will be projected out; each time Steps 2 and 3 are performed the subset of the projected variables and therefore $\bar{f}$ may change significantly. This property requires a general optimization algorithm in Step 2 that can turn out to be both inefficient and hard to implement. In our algorithm we explore the specific structure of the constraints and the objective function with the result that the same set of variables is projected at each step. Thus, only minor changes to the structure of $\bar{f}$ occur at any step of the algorithm. More specifically, the values of the demand and flow variables are chosen such that the entire set of capacity variables can be projected out every time Step 2 of the algorithm is carried out. Further, since the objective function (2.3.3) is a linear function of the capacity variables, the nonlinear structure of $\bar{f}$ remains unaffected throughout the algorithm. In addition we utilize the properties of $\bar{f}$ to obtain a tight upper bound at no additional computational expense.

2.8 The algorithm

In this section we will describe the optimization algorithm in detail for the SD model with fixed percentages and then discuss the adjustments and extensions of the algorithm necessary to apply it to the other three models.

2.8.1 Shared protection on single demand with fixed percentages (SDP)

Let $d = \{d_k^t\} \in \mathbb{R}^{n_d}$ and $y = \{y_l^{q,t}\} \in \mathbb{R}^{n_y}$, where $n_d = \frac{Tn(n-1)}{2}$ and $n_y = \frac{mT(T+1)}{2}$

Then the constraints can be rewritten as

$$Ad + By \leq 0 \tag{2.8.1}$$

where the entries of A are the corresponding values of the percentages for the link capacity constraints (2.6.9) and 0 for the maintenance (2.6.10) and nonnegativity (2.6.11) constraints and the entries of B are accordingly 1 in (2.6.9) and $0 - 1$ in (2.6.10) and (2.6.11).

Let d^0 be some given values of the demand variables. Then the corresponding optimal values of the capacity variables y^0 can be easily determined by solving a set of linear programs. Let E^0 be the resulting subset of constraints (2.8.1) that are satisfied as equalities at (d^0, y^0) and I^0 be the remaining constraints. Thus E^0 is the set of all tight or *active* constraints.

Definition 1 *A* region *is the set of all values of d that induce the same subset of active constraints.*

Let B_{E^0} be the submatrix of B consisting of the tight constraints. Assume that d^0 is such that it produces a unique optimum set of y^0 , that is B_{E^0} is a square nonsingular matrix. This can always be achieved by a small perturbation of d^0.

Any such d^0 then belongs to the interior of the unique corresponding region. Let $\bar{d}^0$ be some other set of demand values such that there are exactly two constraints i and j such that $i \in E^0,\ i \notin \bar{E}^0$ and $j \in \bar{E}^0,\ j \notin E^0$. Then d^0 and $\bar{d}^0$ induce two neighboring regions and their common boundary is defined as $E^0 \cup \{j\}$. The dimension of this boundary is then $n_d - 1$.

More generally

Definition 2 *For any region R defined by a set E of tight constraints, and for any* $S \subseteq I$, $E \cup S$ *is a boundary of R of dimension* $n_d - |S|$

Consider the following example. Let G be a single link network, i.e. $n = 2$, $m = 1$, with $t = 2$. Let $\epsilon^1 = \epsilon^2 = 1.5$, $a = 1$, $c^1 = 10$, $c^2 = 9$, $\mu = 0.3$

Then the problem formulation is

$$\begin{aligned} & \max (d^1)^{1/3} + (d^2)^{1/3} - 10y^1 - 9y^2 - 3y^{1,2} \\ & s.t. \\ d^1 & \leq y^1 \\ d^2 & \leq y^{1,2} + y^2 \\ y^{1,2} & \leq y^1 \\ y^1, y^2, y^{1,2} & \geq 0 \end{aligned}$$

and $n_d = 2$, $n_y = 3$.

Clearly for any values of d^1, d^2 such that $d^1 < d^2$ the active constraints are

$$\begin{aligned} d^1 &= y^1 \\ d^2 &= y^{1,2} + y^2 \qquad (E_1) \\ y^{1,2} &= y^1 \end{aligned}$$

and for any values of d^1, d^2 such that $d^1 > d^2$ the active constraints are

$$\begin{aligned} d^1 &= y^1 \\ d^2 &= y^{1,2} + y^2 \qquad (E_2) \\ y^2 &= 0 \end{aligned}$$

These two sets of demand values define two regions E_1 and E_2. Let $S = \{y^2 = 0\}$. Then $E_1 \cup S$ is the boundary of E_1 and E_2. Thus, the set of demands is split into two regions by the line $d^1 = d^2$. Note that adding a different constraint to E_1 or E_2 will define the $d^1 = 0$ or $d^2 = 0$ boundaries. See Figure 2.3

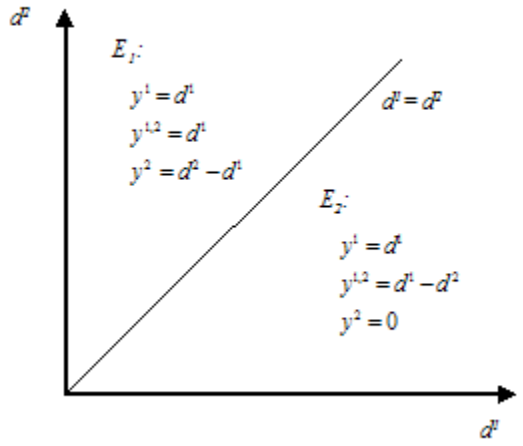

Figure 2.3: An example of the region set

In the interior of the region the values of y can be represented as

$$y = -B_{E^0}^{-1} A d \tag{2.8.3}$$

Thus in the interior of a region the values of the capacity variables are uniquely defined by the values of the demand variables, therefore in every region we can project the space of all variables onto the space of the demand variables. The projected

objective function then becomes

$$f^0(d) \equiv \sum_k h_t\left(g_k^t(d_k^t)\right) - C^T B_{E^0}^{-1} A d \tag{2.8.4}$$

where $C = \{c_l^{q,t}\}$

Region Projection (RP) Algorithm Outline

1. Let d^0 be a starting point in the interior of some region R^0 and $f^0(d)$ be the projected objective function. Let $i = 0$.

2. **Region Optimization.** Find $\max_{d \in R^i} f^i(d)$

   (a) Let $\mathcal{A} = \emptyset$

   (b) Find d^* - the maximum of $f^i(d)$ subject to the constraints in $\mathcal{A}$

   (c) If $d^* \in R^i$ stop, $\max_{d \in R^i} f^i(d)$ is obtained. Otherwise determine the steplength α to the region boundary. Let C be the constraints defining the boundary. Reset $\mathcal{A} \leftarrow \mathcal{A} \cup C$ and goto step (b).

3. Using (2.8.3) find the values of the capacity variables y^* corresponding to d^*.

4. Let $g = \nabla F(d^*, y^*)$ and let $(\bar{d}, \bar{y})$ be the optimum of g subject to the original constraints (2.6.9)-(2.6.11)

5. Obtain the optimum steplength β and set $(d^{i+1}, y^{i+1}) = \beta(d^*, y^*) + (1-\beta)(\bar{d}, \bar{y})$

6. If the improvement $F(d^*, y^*) - F(d^{i+1}, y^{i+1})$ is less than $\epsilon F(d^*, y^*)$ for some tolerance ϵ stop. Otherwise, goto step 2.

It is easy to draw a parallel between the above algorithm and the general active set method described in the previous section. In particular, step 2 of the RP algorithm corresponds to the optimization of $\bar{f}$ in the general active set method in case the current solution violates the constraint boundary, whereas step 5 of the RP algorithm is associated with dropping a constraint from the active set. Note however that in our algorithm we allow to remove more than one constraint from $\mathcal{A}$, in fact the entire set of active constraints may change after step 5 of the RP algorithm is performed. This in turn may affect the convergence rate of the algorithm if an exact solution is desired. Nevertheless this algorithm proves to be effective when a reasonable optimality gap is acceptable.

The constant ϵ in the stopping criterion in step 6 is usually taken to be a fraction of percent. We introduce this criterion in order to prevent tailing off when the solution is close to the optimum.

Region Optimization

We will now describe our implementation of step 2 of the RP algorithm.

1. Using (2.8.3) we project the original set of constraints (2.8.1) onto the space of the demand variables. Thus the constraints become

$$A^p d \leq 0$$

where $A^p = A - BB_{E^0}^{-1} A d^0$

2. Note that when $\mathcal{A} = \emptyset$, $f^i(d)$ is a separable function and hence can be easily optimized. Let d^* be the optimum solution. Set $\hat{d} \leftarrow d^0$.

3. Find the steplength α to the region boundary as

$$\alpha = \min_i \{1, \frac{-a_i^{pT} d^*}{a_i^{pT}(d^* - \hat{d})} | a_i^{pT}(d^* - \hat{d}) < 0\}$$

4. If $\alpha = 1$ exit. The optimum value d^* does not violate the region boundary and the region maximum is found.

5. Set $\hat{d} \leftarrow \hat{d} + \alpha(d^* - \hat{d})$

6. Let C be the set of new tight constraints. Set $\mathcal{A} \leftarrow \mathcal{A} \cup C$. Note that C might contain more than one constraint in case of a tie in step 3.

7. Optimize $f^i(d)$ subject to the constraints in $\mathcal{A}$. For this step we use an optimization method based on the null space representation of $\mathcal{A}$. Recall that the our constraint set is

$$\mathcal{A}d = 0$$

Let Z be a basis of the nullspace of $\mathcal{A}$. Recall that $\mathcal{A} \in \mathbb{R}^{n_d \mathrm{X}\, k}$ where k is the number of the active constraints. Hence $Z \in \mathbb{R}^{n_d x(n_d-k)}$. Then any vector d in the nullspace of $\mathcal{A}$ can be represented as a linear combination of the basis vectors as $d = Zv$ where $v \in \mathbb{R}^{n_d-k}$.

We then proceed by optimizing $f^i(Zv)$ using standard Newton's method.

8. Let v^* be the optimum solution. Set $d^* \leftarrow Zv^*$ and goto 3.

Bounds

We can now describe the process of obtaining a tight upper bound for the problem (2.6.8)

Recall from the description of the RP that every time step 2 with $\mathcal{A} = \emptyset$ is carried out, the global optimum of a region function is found. We show that this optimum, which may actually be located outside the given region, can be used as an upper bound on $F(d, y)$, using the fact that $F(d, y)$ is a continuous concave function and so

are all $f^i(d)$.

Let $F(d)$ be the projection of $F(d, y)$ onto the space of the demand variables. Then

$$F(d) = \{f^i(d) | d \in R^i\}$$

is a continuous and concave function of d. Recall from (2.8.4) that all region functions $f^i(d)$ have the same nonlinear part. Thus, $F(d)$ can be defined as the sum of a smooth concave nonlinear function and a piece-wise linear function. Since $F(d)$ is also smooth and concave, the piece-wise linear function is concave as well. This also follows from the fact that the values of the capacity variables in every region are determined as an optimal solution to a minimum cost LP, which takes the demand values as input right-hand-side parameters. It is well known that in this case the objective function of the LP is a convex function of these parameters. Recall that in the original profit function and therefore in all the projected functions the cost of the capacities is subtracted from the revenue. Thus, the piece-wise linear function resulting from the capacity cost is concave.

Therefore, for any region its region function is dominated by the function of any neighboring region, and since the functions of any two regions intersect only along the regions' common boundary we have proved

Theorem 1 *For any two regions R^i and R^j*

$$f^j(d) \leq f^i(d) \; \forall d \in R^j$$

Therefore the global optimum of any region function obtained at step 2 of the RP algorithm provides a valid upper bound on the optimum value of $F(d, y)$. Note that this bound is obtained at no additional computational expense and we will show in Section 2.10 that this is in fact a tight bound.

2.8.2 Shared protection across demands with fixed percentages (MDP)

In this model we have a set of supplementary variables $\{z_t^l\}$ representing the maximum protection load on link l and a set of constraints linking these variables with the demands. Clearly the values of $\{z_t^l\}$ are uniquely defined by the values of $\{d_k^t\}$, moreover these auxiliary variables do not affect the objective value and used only to determine the set of demands that defines the protection capacity on the given link. Thus we can again project the space of all variables onto the space of the demand variables and use the algorithm described in the previous section.

2.8.3 Shared protection on single demand without percentages (SDN)

In this model in addition to the demand and capacity variables we also have flow variables and the constraints reflecting the flow-demand dependence. As in the previous model the values of the capacity variables are uniquely defined by the flow on the link. Furthermore, we have an additional set of constraints linking demand and flow variables. Note that the values of the flow variables uniquely define the values of the demand variables. Thus we redefine the region concept to capture the flow-demand relationship. Let $f = \{f^t(p_k^h)\} \in \mathbb{R}^{n_f}$ where $n_f = n_d \sum_k |P_k|$

Definition 3 *A* region *is the set of all values of f that induce the same subset of active constraints.*

The nonlinear optimization algorithm then proceed in the same manner by projecting out the capacity and demand variables and sequentially optimizing the regions defined by the flow variables.

2.8.4 Shared protection across demands without percentages (MDN)

For this model we utilize the ideas described in the previous subsection. Using (2.6.3)-(2.6.4) we can immediately project out $\{d_k^t\}$ and $\{f^t(p_k^h)\}$ variables. Let

$f = \{f^t(p_k^h, p_k^{h'})\} \in \mathbb{R}^{n_{f_p}}$ where $n_{f_p} = n_d \sum_k \frac{|P_k|(|P_k|-1)}{2}$ and redefine regions using Definition 3 The optimization algorithm then proceeds in the same manner as for the previous model.

2.8.5 Bounds

Recall that the bound for the SDP model is based on the properties of the original and projected objective functions, specifically the fact that all these functions are concave and any two projected functions intersect along a single hyperplane. It is easy to see that these properties are maintained in the other three models as well and hence the same approach to obtaining a valid bound can be utilized. fi

2.9 Starting point

Recall from section (2.2.5) that the actual convergence rate of the Newton method and therefore that of the RP algorithm relies on the quality of the starting point. On order to obtain a good initial point we start with a first order heuristic. The central idea of the heuristic is optimize each commodity separately, thus replacing the large problem with $O(n^2)$ smaller problems. Clearly, for $T = 1$, that is for a single time period, optimizing one commodity at a time will provide an overall optimal solution. On the other hand, in the multi-period problem we have a certain

amount of capacity maintained from the previous period, that has to be shared by different commodities. One way to resolve the sharing problem would be to split the existing capacity in the same proportion it was originally used. However, recall that the parameters of nonlinear terms of the objective function differ not only between different commodities, but also between different periods of the same commodity. Thus in general, this approach will not provide an optimal solution, nevertheless it turns out to be a reasonable starting point in our experiments.

2.9.1 Fixed percentages

We again start with the description of the algorithm for the simplest model, namely the SD model with fixed percentages. The single commodity problem for this instance is

$$\max \sum_t h_t \left(g_k^t(d_k^t) - \sum_l \left(c_l^t y_l^t + \sum_{q<t} c_l^{q,t} y_l^{q,t} \right) \right)$$

$$s.t.$$

$$\begin{aligned}
\sum_{q:l\in p^q} r^q d_k^t &\le y_l^t + \sum_{q<t} y_l^{q,t} && \forall l \in A,\ \forall t \\
y_l^{q,t} &\le y_l^{q,t-1} && \forall l,\ \ \forall q,\ \ \forall t \ge q+1 \\
y_l^{q,t} &\ge 0 && \forall l,\ \ \forall q,\ \ \forall t \ge q
\end{aligned}$$

Consider the example in Figure 2.4. Let d be the commodity i, j with three given

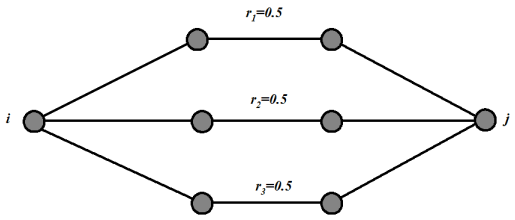

Figure 2.4: Original network for SDP model

paths and the requirement of 50% of d over each path. Again denote the set of paths P_{ij} as P. Note that the capacity on every path at every time period is uniquely defined by the value of d at that period. Moreover, the capacity is the same on every link of a path. Then our problem is equivalent to solving the same problem over the compressed network $\bar{G}$ in Figure 2.5 with the cost of the accumulated link

$$\bar{c}^{\,t} = \sum_{h} r^h \Big(\sum_{l \,:\, l \in p^h} c_l^t \Big) \tag{2.9.1}$$

and the percentage requirement

$$\bar{r} = \sum_{h} r^h \tag{2.9.2}$$

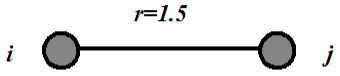

Figure 2.5: Compressed network $\bar{G}$ for SDP model

We can now simplify the single commodity instance as

$$\max \sum_t h_t \left(g_k^t(d_k^t) - \bar{c}^{\,t} y^t + \sum_{q<t} \bar{c}^{\,q\,t} y^{\,q\,t} \right)$$

$$s.t.$$

$$\bar{r} d_k^t \;\leq\; M(y^t + \sum_{q<t} y^{\,q\,t}) \qquad \forall t \qquad (2.9.3)$$

$$y^{\,q\,t} \;\leq\; y^{\,q,t-1} \qquad \forall q,\;\; \forall t \geq q+1 \qquad (2.9.4)$$

$$y^{\,q\,t} \;\geq\; 0 \qquad \forall q,\;\; \forall t \geq q \qquad (2.9.5)$$

This is a much smaller problem with only $O(T^2)$ variables and can be easily solved by any of the cutting plane methods described in section (2.2.4).

For the MD model we utilize the same approach of reducing the set of paths to a single link. However the cost model in this instance is more complicated due to the structure of the capacity usage.

In order to formulate this model we introduce an additional constant.

Consider the example in Figure 2.6

We are given three commodities

1. d_{34} with 100% on primary path c and 100% on protection path $d - e$

2. d_{12} with 100% on primary path a and 100% on protection path $b - e$

3. d_{13} with 100% on primary path $a - b$ and 100% on protection path e

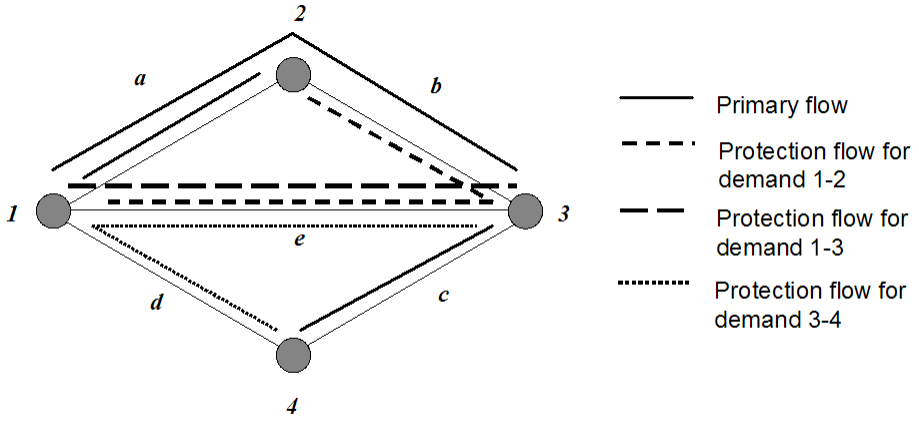

Figure 2.6: Original graph for the MD model

The protection capacity on link e can be shared by demands d_{34} and d_{12} or d_{34} and d_{13}, but not d_{13} and d_{12} since they share link a on their primary paths. Thus the protection capacity on link e will be defined by the maximum of d_{34} and $d_{12} + d_{13}$ since in case link a fails both d_{12} and d_{13} will be rerouted through link e. In other words we can divide the set of the commodities into two groups $\{d_{12}, d_{13}\}$ and $\{d_{34}\}$ that share the capacity on link e.

Denote g_l the number of groups on link l. In our example $g_l = 2$, $g_b = g_d = 1$. The number of groups can be easily precomputed. Using this constant we then define the cost $\bar{c}^t$ as

$$\bar{c}^{\,t} = \sum_{l} \sum_{h\,:\,l \in p^h} \left(r^h c_l^t + \frac{\sum_{h' \,:\, l \in p^{h'}} r_{hh'} c_l^t}{g_l} \right) \tag{2.9.6}$$

Thus the cost of the protection capacity is equally split between the groups that

share the protection capacity on this link.

2.9.2 No percentages

If the percentages are not given, then the proportion of the demand on each path is not known, however the load on each link of any path is still the same and the network in Figure 2.6 becomes a compressed network with 2 nodes (the demand nodes) and H_k links.

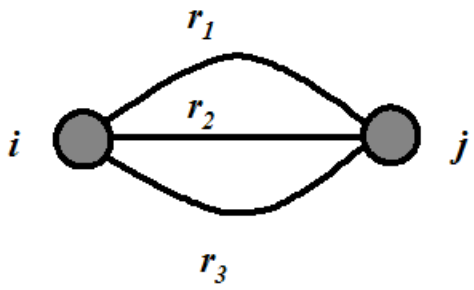

Figure 2.7: Compressed network $\bar{G}$ for SDN model

Then the SD model can be formulated as

$$\max \sum_t h_t \left(g_k^t(d_k^t) - \sum_{h=1}^{H_k} \left(\bar{c}_h^t y_h^t + \sum_{q<t} \bar{c}_h^{q\,t} y_h^{q\,t} \right) \right) \tag{2.9.7}$$

$$s.t.$$

$$\begin{aligned}
f^t(p^h) &\leq M(y_h^t + \sum_{q<t} y_h^{qt})) && 1 \leq h \leq H_k,\ \forall t \\
y_h^{qt} &\leq y_h^{q,t-1} && 1 \leq h \leq H_k,\ \ \forall q,\ \ \forall t \geq q+1 \\
f^t(p_h) &\leq f^t && 1 \leq h \leq H_k \\
d_k^t &= \sum_{p_h \in P(d)} f^t(p_h) - f^t && \forall t \\
y_h^{qt} &\geq 0 && \forall 1 \leq h \leq H_k,\ \ \forall q,\ \ \forall t \geq q
\end{aligned}$$

where

$$\bar{c}_h^t = \sum_{l\,:\,l \in p^h} c_l^t$$

This problem is slightly larger than the instance with percentages, nevertheless since the number of paths is usually small (< 10) it remains an easy problem for the first-order method.

Note that for the single commodity case there is no essential difference between the SD and MD models. Thus we can use model (2.9.7) with costs

$$\bar{c}_h^t = \sum_{l\,:\,l \in p^h} \left(r^h c_l^t + \frac{\sum_{h'\,:\,l \in p^{h'}} r^{hh'} c_l^t}{g_l} \right)$$

The solution to model (2.9.7) provides the values of the demand and flows on each path for the given commodity. Recall that for the MD model we in addition require to specify the values of the protection flows explicitly, that is for each path p_k we need the values of the protection flows $f^t(p^h, p^{h'})$ for every path $p^{h'},\ h \neq h'$ and the values of the primary flows $f^t(p^h)$ for every path p^h such that

$$f^t(p^h) + \max_{h' \neq h} f^t(p^h, p^{h'}) = f^{t*}(p^h)\ \forall t,\ 1 \leq h \leq H_k \tag{2.9.8}$$

where $f^{t*}(p^h)$ is the optimal solution to (2.9.7)

Note, however, that any values of $f^t(p^h, p^{h'})$ and $f^t(p^h)$ that satisfy (2.9.8) will provide a solution of the same cost. More precisely, let $f^{t*} = \max_h f^{t*}(p^h)$. Clearly there are always at least two paths that carry the flow of value f^{t*}. Let $p^{h(t)}$ be one of these paths and let

$$\begin{aligned} f^t(p^h) &= f^{t*}(p^h) \forall h \neq h(t) \\ f^t(p^{h(t)}) &= 0 \\ f^t(p^h, p^{h'}) &= 0 \qquad \forall h' \neq h, h(t) \\ f^t(p^h, p^{h(t)}) &= f^t(p^h) \end{aligned}$$

Obviously these values of $f^t(p^h, p^{h'})$ and $f^t(p^h)$ satisfy (2.9.8).

2.10 Implementation and numerical results

In order to evaluate our algorithm and the bound we generated 5 data sets based on different real-life networks. In addition a set of larger networks was created to test the boundaries of our method. Before we proceed with the descriptions of the data sets and the computational results let us first give some details on the implementation of some parts of the algorithm. The Newton method in step 7 of the RP algorithm was implemented using the CLAPACK [15] library for sparce matrix operations including Cholesky factorization. Recall from Section 2.7 that the second part of the algorithm involves solving a series of a multiperiod knapsack problems. We use CPLEX integer solver to solve these problems to optimality within a fraction of time required by the nonlinear optimization algorithm. The resulting integer solution was in all cases within a fraction of a percent of the nonlinear solution due to the reasons described in Section 2.4. For this reason we do not provide these results here.

The real-world problems were solved on a 336 Mhz UltraSPARC machine with 1.7Gb of RAM and the test with larger networks were conducted on 1.89Ghz Xeon with 3Gb of RAM.

2.10.1 Real-world problems

Table 2.5 shows the statistics of the first set of data. Here the last two columns show the minimum and maximum number of paths per node pair. The small number of

paths used is due to the sparsity of the real-life networks. The paths were generated using the SPIDER code[14]. The time horizon T was taken to be 14 in all test sets. Note that this set is identical to the set used in Section 2.5

Recall from Section 2.3 that there are six main data parameters defining the problem data for every model, whose values we describe next.

The maintenance fraction μ and the maintenance rate α were taken to be equal to 0.05 and 1.01 correspondingly.

The number of batch types u, the size of each batch $M(u)$ and the initial cost per unit of capacity are reflected in Table 2.7. The discount factor h_t was taken to be 0.86 for every time period $t \geq 2$. The parameters were chosen to approximate the real data patterns and behavior as closely as possible.

The initial buying costs $c_l^1(u)$ were also picked to reflect the real cost structure of the networks based on the installation and mileage costs of the equipment. The buying costs for all the subsequent time periods $t = 2 \ldots T$ were computed as $c_l^t(u) = \gamma c_l^{t-1}(u)$ for a given constant $\gamma < 1$.

Table 2.6 shows the range of the remaining parameters.

All possible combinations of these parameters were considered (a total of 27) and 50 random sets were generated for each combination. The data corresponding to each parameter was independently and uniformly distributed with mean shown in table 2.6. Here we take $A = \sum_k A_k$. In all runs the code was terminated when the total

Set	#Nodes	#Links	#Paths (min)	#Paths(max)
snet1	14	22	2	3
snet2	38	48	2	3
snet3	47	55	2	4
snet4	50	64	2	3
snet5	70	94	2	3

Table 2.5: Statistics on real-life networks

$\bar{A}$	$\bar{\gamma}$	$\bar{\epsilon}$
50000	0.95	1.3
500000	0.9	1.4
5000000	0.85	1.5

Table 2.6: Data parameters

improvement in the last 10-20 steps did not exceed 0.05%.

In what follows we present the detailed numerical analysis of the first algorithm and the summary of the results for the other three models. In our experiments the behavior of all models follows the same patterns with only slight variations due to the different size of the models.

Table 2.8 illustrates the average and boundary values of the performance of the

u	1	2	3	4	5	6	7
$M(u)$	198	198	198	762	762	1536	1536
$\frac{c^1(u)}{M(u)}$	0.65	0.625	0.42	0.17	0.16	0.1	0.08

Table 2.7: Capacity costs

algorithm on different parameter groups. Table 2.9 shows the comparison of the running time. In order to single out the pattern for each group of parameters, the data was averaged on the other two groups.

Table 2.8 clearly shows that the problem becomes easier to solve as the elasticity ϵ and the yearly reduction rate γ parameters decrease. On the other hand the scaling constant A does not affect the performance. This observation supports the natural assumption that the problem becomes harder to solve as the demand-price curve that is essentially defined by the elasticity becomes steeper. In addition, smaller values of γ induce large decrease in cost from one year to another and helps to reduce the problem degeneracy, thus allowing a faster convergence.

Table 2.10 compares the average optimality gap and the running time for the solution obtained by the starting heuristic with the final solution. The data is averaged over all parameter groups. This table illustrates the quality of the heuristic which in most cases provides a good starting solution. The running time of the heuristic is approximately half of the overall running time. Thus for a case where only a rough

net	Gap	$\bar{A}$			$\bar{\gamma}$			$\bar{\epsilon}$		
	(%)	50000	500000	5000000	0.9	0.8	0.7	1.3	1.4	1.5
	Ave	0.00081	0.00077	0.00082	0.001	0.00078	0.00062	0.00098	0.00075	0.00065
snet1	Max	0.0085	0.0082	0.0088	0.0098	0.0081	0.0068	0.0096	0.0082	0.0071
	Min	0	0	0	0	0	0	0	0	0
	Ave	0.006	0.0061	0.0061	0.008	0.006	0.0041	0.0086	0.0062	0.004
snet2	Max	0.051	0.052	0.051	0.063	0.051	0.039	0.071	0.053	0.04
	Min	0	0	0	0	0	0	0	0	0
	Ave	0.019	0.018	0.016	0.044	0.017	0.009	0.039	0.018	0.01
snet3	Max	0.078	0.076	0.075	0.088	0.076	0.064	0.089	0.075	0.065
	Min	0	0	0	0	0	0	0	0	0
	Ave	0.019	0.023	0.021	0.031	0.021	0.011	0.033	0.021	0.012
snet4	Max	0.11	0.125	0.14	0.25	0.13	0.1	0.21	0.12	0.096
	Min	0	0	0	0	0	0	0	0	0
	Ave	0.021	0.024	0.025	0.041	0.023	0.012	0.033	0.022	0.013
snet5	Max	0.193	0.198	0.21	0.29	0.2	0.176	0.28	0.196	0.18
	Min	0.011	0.014	0.009	0.018	0.013	0.007	0.019	0.014	0.007

Table 2.8: Performance comparison - SDP model

net	$\bar{A}$			$\bar{\gamma}$			$\bar{\epsilon}$		
	50000	500000	5000000	0.95	0.9	0.85	1.3	1.4	1.5
snet1	15	15	14	16	15	15	16	16	14
snet2	45	47	45	47	46	44	46	46	44
snet3	100	100	99	102	99	99	100	100	98
snet4	150	152	151	153	151	148	153	152	148
snet5	213	212	214	215	213	213	214	214	212

Table 2.9: Average running time (sec) - SDP model

	snet1	snet2	snet3	snet4	snet5
Heur. gap (%)	0.0017	0.016	0.041	0.049	0.056
Final gap (%)	0.00079	0.006	0.018	0.021	0.023
Heur. time (sec)	7	26	47	77	103
Final time (sec)	15	46	100	152	213

Table 2.10: Heuristic performance - SDP model

Model	Gap(%)	snet1	snet2	snet3	snet4	snet5
SDN	Ave	0.32	0.35	0.48	0.47	0.65
	Min	0.22	0.17	0.2	0.21	0.4
	Max	0.44	0.61	0.88	0.85	1.13
MDP	Ave	0.27	0.29	0.33	0.33	0.41
	Min	0.16	0.18	0.16	0.18	0.2
	Max	0.39	0.43	0.52	0.59	0.64
MDN	Ave	1.19	1.77	2.01	2.2	2.3
	Min	0.82	1.16	1.29	1.56	1.45
	Max	1.56	2.47	2.89	3.08	3.89

Table 2.11: Performance summary - SDN, MDP, MDN models

estimate of the solution is required, it can be obtained much faster and improved later if needed.

The summary performance of the other three models on the same data sets is reflected in tables 2.11 and 2.12. Table 2.11 shows the average and boundary optimality gaps for SD model without percentages (SDN) and MD model with (MDP) and without (MDN) percentages. Table 2.12 shows the average running time for the models.

Generally the problems become harder to solve as the problem size grows. Nev-

Model	snet1	snet2	snet3	snet4	snet5
SDN	10	79	119	108	284
MDP	11	80	126	130	304
MDN	13	93	142	158	332

Table 2.12: Running time (sec) summary - SDN, MDP, MDN models

Model	snet1	snet2	snet3	snet4	snet5
SD	1.5	1.7	1.6	1.8	1.85
MD	1.6	1.9	1.86	1.91	1.9

Table 2.13: The effect of fixing percentages

Model	snet1	snet2	snet3	snet4	snet5
SDP/MDP	1.92	1.79	1.98	2.01	2.02
SDN/MDN	2.4	2.34	2.22	2.3	2.19

Table 2.14: Model comparison (%)

ertheless it is evident that the algorithm can handle large instances of the models without substantial loss of the solution quality. The additional numerical results supporting this claim will be shown in the following subsection. However before we move to the more detailed exploration of the scalability of our algorithm let us first compare the performance of the different models from the business solution perspective. Recall that the goal of the optimization algorithm is to create a reliable and survivable network. SD and MD models were designed to provide a different degree of survivability, with higher level available at the expense of receiving a lower overall profit. In addition a different solution technique, precomputing the demand routing was proposed, allowing to simplify the model and the algorithm once more at the expense of the profit. Table 2.13 shows the decrease in the objective value when fixed percentages are used and Table 2.14 demonstrates the average increase in total profit when the MD model is used instead of the SD model. The average improvement is about 2% of the entire business profit for all cases considered. Thus a weaker survivability model offers a considerable increase in company profit consequently providing the grounds for a serious analysis of the survivability and profit tradeoff. On the other hand the decrease in the profit from using fixed percentages is also about 2% on average. However it is evident from tables 2.11 and 2.12 that reducing the size of the problem by using fixed percentages does not provide a considerable decrease in the running time or noticeable improvement of the algorithm performance. This in

Solver	snet1	snet2	snet3	snet4	snet5
LOQO	0	0	0	0	–
SNOPT	0	–	–	–	–
Our Code	0.0008	0.006	0.018	0.021	0.023

Table 2.15: Performance comparison (LOQO and SNOPT) - Average optimality gap (%)

Solver	snet1	snet2	snet3	snet4	snet5
LOQO	30	600	720	900	–
SNOPT	90	–	–	–	–
Our Code	15	46	100	152	213

Table 2.16: Performance comparison (LOQO and SNOPT)- running time (sec)

turn implies that it is most likely not worthwhile to utilize this technique.

In order to evaluate our algorithm we compared it against LOQO [39] and SNOPT [13] Table 2.15 and 2.16 show the performance comparison of our algorithm and solvers on the real-life networks. Note that some of these results were also presented and analysed in Section 2.5.

It is evident that even though the comparison is biased toward our code as it is tuned up specifically to solve this problem, the existing solvers are still not able to handle even a reasonable size network and in cases where LOQO was able to solve

Set	#Nodes	#Links	#Paths (min)	#Paths(max)
bnet1	100	500	3	6
bnet2	150	900	3	7
bnet3	200	1500	3	9
bnet4	250	2000	3	10
bnet5	300	4500	4	12

Table 2.17: Statistics on the generated networks

the problem, our algorithm significantly outperforms it in terms of running time with only a slight increase of the optimality gap.

2.10.2 Generated problems

In this section we analyze the performance of our algorithm on the generated set of large dense networks. Table 2.17 shows the statistics on the five networks generated to test the scalability of the algorithm.

Since these cases were generated to test the quality of the algorithm on the large data set only one set of parameters was chosen for trials. Namely, $\bar{A} = 50000$, $\bar{\epsilon} = 1.5$, $\bar{\gamma} = 0.95$. Recall that these values of the parameters proved to be the hardest for the optimization algorithm. For the same reason the performance was tested on the largest model, i.e. MD without percentages.

Solver	bnet1	bnet2	bnet3	bnet4	bnet5
CPLEX	2341	7982	14324	18943	–
Our Code	1943	2457	2691	2893	3103

Table 2.18: Performance comparison (CPLEX) - running time (sec)

The remaining parameters were taken the same as in the previous section. Similarly to the previous section, 50 random data sets were generated for each network.

We compared our algorithm with CPLEX [16] which is on of the most efficient linear, integer and quadratic solver available on the market. CPLEX also has one of the best Cholesky factorization algorithms built in it. In order to compare our algorithm with CPLEX we replace the original objective with its second order Taylor series approximation. The idea here is to ensure that our implementation of the Newton method is not the bottleneck of the performance, and that the active set method considerably reduces the complexity of the problem.

Table 2.18 shows the comparison of our algorithm with the performance of CPLEX on the quadratic relaxation of the problem. It is evident that the CPLEX efficiency deteriorates rapidly as the network size and density increase, while our algorithm shows stable and scalable performance.

Finally, tables 2.19 and 2.20 illustrate the optimality gap and the running time on all the networks.

Gap (%)	bnet1	bnet2	bnet3	bnet4	bnet5
Ave	2.91	2.99	3.15	3.26	3.51
Max	4.56	4.87	4.91	5.1	5.08
Min	1.71	1.82	1.83	1.93	1.96

Table 2.19: Performance summary - Optimality gap (%)

bnet1	bnet2	bnet3	bnet4	bnet5
1943	2457	2691	2893	3103

Table 2.20: Average running time (sec) summary

2.11 Discussion

The previous section demonstrates several important conclusions which we will summarize in this section.

- Algorithmic inferences

  The numerical results indicate that the approach proposed in this work leads to a viable algorithm that provides a high quality solution within a reasonable timeframe and scales well with the problem size. Note that the specific form of the objective function is not essential for the course of the algorithm. The idea of projecting out the capacity variables and working only with the demand or flow variables can thus be extended to different pricing and cost models

as long as the resulting objective can be formulated as a separable concave function. Further, this approach might prove efficient for other network design problems with liner or concave objective functions. The essential detail of the approach is the unique dependency of the link capacity and flows (demands), thus many network design problem that poses this characteristic might benefit from a similar algorithm.

- Network Planning inferences

  From the design point of view we confirmed our initial assumption that a higher level of protection will result in a lower profit value. The shared protection on the other hand leads to a more complicated model and as a result requires longer running time to achieve the similar solution quality. In addition we have demonstrated that the integration the routing problem into the optimization model can result in a higher profits as opposed to sequential solution. Thus even though the integration is not currently employed in existent schemes due to its complexity and difficulty, it is definitely worthwhile to explore this option as it is evident that an adequate algorithm can be developed.

Overall we conclude that our results provide justification for further, more extensive study of different protection and integration approaches as well as the expansion of our technique to more general planning, design and pricing problems.

Chapter 3

Maximum Concurrent Flow Problem

3.1 Introduction

The *maximum concurrent flow problem* frequently arises in practical applications and has received much attention due to its difficulty.

The input to the problem is a graph with capacities on its edges and a set of multi-commodity demands to be routed. The objective is to find a largest value $\Gamma^* > 0$ such that a fraction Γ^* of every demand can be simultaneously routed without exceeding the available capacities (possibly $\Gamma^* > 1$). In other words, we want to find a feasible flow with maximum throughput. A precise definition is given in Section 3.2. An

equivalent problem is the *minimum congestion problem*: here we must simultaneously route 100% of every demand so that the maximum load of any edge – the ratio of total flow to capacity – is minimized. In the literature both problems are sometimes regarded as one and the same and are jointly referred to as the "maximum concurrent flow problem", but in this paper we will work with the maximum throughput version.

Despite large advances in computer technology and optimization software, even state-of-the-art linear optimizers are often proven ineffective on very large problems. Furthermore, the scalability of these codes comes into question. As an example, a concurrent flow problem on 400 nodes, 1740 arcs and 5000 commodity pairs which yields a model with 160143 rows, 689056 columns and 2067177 nonzeros is solved by CPLEX in 9434 sec. on a modern computer, whereas the same problem on 700 nodes, 3120 arcs and 5000 comodity pairs yields a model with 481326 rows, 2131038 columns and 6393176 nonzeros. This larger model already requires 455541 seconds (see [4]). With more extensive experiments it can be shown that the running time using CPLEX (dual) grows cubically with the number of columns (on concurrent flow models). Moreover, during a very large fraction of the running time the optimality error is rather large (more than one percent, say).

This situation has prompted a long line of research on alternative algorithms for the maximum concurrent flow problem, in particular, approximation algorithms. Given the nature of the problem, in a sense an ideal approximation algorithm would

be one whose precision can be controlled. Given $\epsilon > 0$, a *fully polynomial polynomial-time approximation scheme* is one that estimates Γ^* within relative error ϵ, and whose worst-case complexity grows polynomially in ϵ^{-1} and on the size of the graph.

The first such algorithm was given by Shahrokhi and Matula [60] in the late 80's, and it applied to the minimum congestion problem. [60] introduced a "potential" function that exponentially penalizes the load on any edge, so that the approximate minimization of the potential function yields a flow whose maximum load is provably near-optimal. To carry out the approximate optimization of the potential function [60] used what amounts to a Frank-Wolfe scheme (see [44], [46], [47]). The overall algorithm was proved to have a worst-case complexity bound that grows proportional to ϵ^{-7} (as well as polynomially on the size of the graph). The Sharokhi-Matula algorithm can also be viewed as a Lagrangian relaxation approach, where the capacity constraints are relaxed, and the violation of these constraints is penalized using very special multipliers. These are exponential multipliers, which, roughly stated, arise in the Frank-Wolfe scheme as the gradient of the potential function.

The Sharokhi-Matula results catalyzed an intense and still active research effort. [54] has improved the running time of Sharokhi-Matula algorithm. The work in [54] was later extended ([56]) and improved ([59]). See also [50], [51], [57], [58], [53]. This research primarily showed how to better use an exponential potential function, still in the Frank-Wolfe Lagrangian relaxation framework, to obtain faster algorithms

for the minimum congestion problem, and generalizations. A logarithmic "sliding barrier" function, applied to the minimum congestion problem, is discussed in [61]. The analysis in [62] shows how the exponential potential function naturally arises in the derandomization of probabilistic algorithms; the algorithms for the maximum concurrent flow problem described in [52] and [45] are related to the ideas in [62].

The fastest of these algorithms have running time bounded by ϵ^{-2} times a low-order polynomial on the size of the graph and number of commodities. Altogether, the algorithms in [59], [52] and [45] yield the best bounds for the maximum concurrent flow problem, but precisely which is "best" depends in a complex manner on the number of vertices, edges and commodities. Experimental testing of these algorithms has yielded implementations that are substantially more effective than commercial linear programming codes, see [41], [49].

The Sharokhi-Matula algorithm was for a long time considered the first $\epsilon-$approximation algorithm to exploit the idea of a potential function that penalizes the edge load. However, in 1971, Fratta, Gerla and Kleinrock [48] proposed a general algorithmic scheme towards the solution of various optimization problems arising in telecommunications, among them that of finding a feasible multicommodity flow, which is tantamount to solving a maximum concurrent flow problem. The approach in [48], which is quite different from those found in the references cited above, involves a simple idea for increasing the throughput of a feasible flow while using a rational barrier function

to prevent flows from exceeding capacities, together with a Frank-Wolfe procedure that reduces the barrier function. A partial convergence proof was presented in [48]. Although the flow deviation method is well-known and frequently used in the telecommunications community, it is poorly known in the algorithms community.

In this work we show that the flow deviation method, properly implemented, yields an algorithm that solves the maximum concurrent flow problem to relative error ϵ by solving $O(\epsilon^{-2}m^3k^2 + m^3k^2 \log m)$ minimum-cost flow computations, where m and k are, respectively, the number of edges and commodities. The algorithm we describe contains the critical algorithmic ideas in [48] together with some refinements designed to achieve the ϵ^{-2} performance. Without these refinements, the algorithm in [48] can be shown to converge in $O(\rho^2 m^4 \epsilon^{-2} k^2)$ shortest path computations, where ρ is the *width*, which is defined as follows. Given a convex set $P \in \mathbb{R}^n$ and the set of m inequalities $Ax \leq b$, the width of P relative to $Ax \leq b$ is defined by $\rho = \max_i \max_{x \in P} a_i x / b_i$ [58].

3.2 Definitions

Consider a graph G with n vertices, and a capacity $u_e > 0$ for each edge e. Suppose we are given a set of k *commodities*, where for $1 \leq j \leq k$, commodity j consists of a pair s_j, t_j of vertices and a *demand amount* $d_j > 0$. For $1 \leq j \leq k$, let P_j denote the set of paths between s_j and t_j, and for any edge e let $P_{e,j}$ denote the subset of

P_j consisting of those paths that contain e. The *maximum concurrent flow problem* is the linear program

$$\Gamma^* = \max \quad \gamma$$

(TF) s.t.

$$\sum_{j=1}^{k} \sum_{p \in P_{e,j}} x_p \leq u_e \quad \forall\, e, \tag{3.2.1}$$

$$\sum_{p \in P_j} x_p = \gamma d_j, \quad 1 \leq j \leq k, \tag{3.2.2}$$

$$0 \leq x_p \qquad \forall p. \tag{3.2.3}$$

This definition and formulation apply whether the graph is directed or not. From the point of view of linear programming, this path-based formulation is not the most efficient – a flow formulation is more compact. Note that for any commodity j a linear program over the constraints (3.2.2), (3.2.3) reduces to a shortest path problem. Given $\epsilon > 0$, an ϵ-approximate solution to **TF** is a flow $\hat{x}$ with throughput $\hat{\gamma}$, feasible for **TF**, such that $\hat{\gamma} \geq (1-\epsilon)\Gamma^*$, i.e. $\hat{\gamma}$ approximates Γ^* within relative error at most ϵ.

We will say that a flow vector x is $feasible$ if there exists a nonnegative value $\gamma = \gamma(x)$ such that (x, γ) is satisfies (3.2.1 - 3.2.3), in which case we will also say that x has *throughput* γ. Given a flow vector x, we will say x has *load*

$$\lambda(x) \doteq \max_e \lambda_e(x) \tag{3.2.4}$$

where for an edge e,

$$\lambda_e(x) \doteq \frac{\sum_{j=1}^{k} \sum_{p \in P_{e,j}} x_p}{u_e} \tag{3.2.5}$$

is called the *load* of x on e. Thus x is feasible iff $\lambda(x) \leq 1$, and we will say x is *strictly* feasible if $\lambda(x) < 1$.

Consider a problem of the form

$$\min \{F(x) : x \in Q\} \tag{3.2.6}$$

where $Q \subseteq R^n$ is convex and $F()$ is convex and differentiable over Q. The classical Frank-Wolfe procedure solves this problem by generating a sequence $\{x^t\}$, $t = 1, 2, \cdots$ contained in Q. At iteration t, the procedure solves the linear program

$$\min \left\{ [\nabla F(x^t)]^T v : v \in Q \right\} \tag{3.2.7}$$

with solution $y^t \in Q$, and sets $x^{t+1} = (1 - \sigma^t)x^t + \sigma^t y^t = x^t + \sigma^t(y^t - x^t)$ where $0 \leq \sigma^t \leq 1$ is chosen so as to minimize $F((1 - \sigma^t)x^t + \sigma^t y^t)$. σ^t is called the stepsize. The Frank-Wolfe method is closely related to Danzig-Wolfe decomposition, and to steepest-descent methods for unconstrained optimization.

For completeness, we state the *minimum congestion problem.* Given a graph and multicommodity demands as in the maximum concurrent flow problem, the minimum congestion problem is the linear program

$$\Lambda^* = \min \quad \lambda$$

s.t.

$$\sum_{j=1}^{k} \sum_{p \in P_{e,j}} x_p - u_e \lambda \leq 0 \quad \forall\, e, \tag{3.2.8}$$

$$\sum_{p \in P_j} x_p = d_j \quad 1 \leq j \leq k, \tag{3.2.9}$$

$$0 \leq x_p \qquad \forall p. \tag{3.2.10}$$

A simple analysis shows that $\Lambda^* = 1/\Gamma^*$. In what follows, we will deal exclusively with the maximum concurrent flow problem.

3.3 The central idea

In this section we describe the main idea in [48].

Consider an instance of the maximum concurrent flow problem, and let $\hat{x}$ be strictly feasible with throughput $\hat{\gamma}$. To fix ideas, suppose that $\lambda(\hat{x}) = 1/2$. Then $2\hat{x}$ is feasible and has throughput $2\hat{\gamma}$. In general, $y = \frac{1}{\lambda(\hat{x})}\hat{x}$ is feasible and has throughput $\frac{1}{\lambda(\hat{x})}\hat{\gamma} > \hat{\gamma}$ (but also $\lambda(y) = 1$). Suppose that by using an appropriate algorithmic construct we obtain, from y, a feasible vector z, still with throughput $\frac{1}{\lambda(\hat{x})}\hat{\gamma}$ but with loads that are singnificantly lower than those of y. Then we would have a skeletal outline for an algorithm:

OUTLINE

A1. Let $\hat{x}$ be strictly feasible, with throughput $\hat{\gamma}$.

B1. Let $y = \frac{1}{\lambda(\hat{x})}\hat{x}$.

C1. Find a feasible z with throughput $\frac{1}{\lambda(\hat{x})}\hat{\gamma}$ and $\lambda(z)$ substantially smaller than $\lambda(y)$.

Reset $\hat{x} \leftarrow z$ and $\hat{\gamma} \leftarrow \frac{1}{\lambda(\hat{x})}\hat{\gamma}$, and go to **B1**.

Thus, each execution of **B1** increases throughput while **C1** "spreads out" the flow. Next we describe a concrete implementation of **C1**.

Let $\psi : [0,1) \rightarrow [0,+\infty)$ satisfy the properties

(i) In the range $0 \leq t < 1$, $\psi(t)$ is increasing, continuous and convex,

(ii) $\psi(t) \rightarrow +\infty$ as $t \rightarrow 1^-$.

Then one way to accomplish **C1** is to define, for any strictly feasible x,

$$\Psi(x) \;=\; \sum_e \psi(\lambda_e(x)) \tag{3.3.1}$$

and to choose z as the solution to

$$\min\left\{\Psi(x) \,:\, x \text{ a feasible flow with } \gamma(x) = \frac{\hat{\gamma}}{\lambda(\hat{x})}\right\} \tag{3.3.2}$$

In fact, by property (ii) of ψ the optimal solution to the optimization problem (3.3.2) "should" have maximum load smaller than $\lambda(\hat{x})$. Note, however, that in view of our

construction of y in step **B1**, $\Psi(y)$ is undefined (because $\lambda(y) = 1$), and in general, it may be the case that $\Psi(x)$ is undefined for any $x \in \frac{1}{\lambda(\hat{x})}\hat{\gamma}P$. Thus, we must amend Step **B1** slightly: to obtain y, we scale up $\hat{x}$ by a factor slightly smaller than $\frac{1}{\lambda(\hat{x})}$. Further, it may not be strictly necessary to minimize Ψ – rather, we should decrease it from the value $\Psi(y)$. Our algorithmic scheme is now:

Basic Algorithm

A2. Initialization. Let $\hat{x}$ be strictly feasible with throughput $\hat{\gamma}$.

B2. Magnification. Set $y = \mu\hat{x}$, where $1 < \mu < \frac{1}{\lambda(\hat{x})}$ is a parameter.

C2. Barrier reduction. Find a feasible z with throughput $\mu\hat{\gamma}$ such that $\Psi(z)$ is sufficiently smaller than $\Psi(y)$. If no such z exists, STOP. Otherwise, reset $\hat{x} \leftarrow z$ and $\hat{\gamma} \leftarrow \mu\hat{\gamma}$, and go to **B2**.

The algorithm implicit in Steps **A2 - C2** is the main contribution in [48]; found in pp. 112(bott.)–114. It significantly differs from algorithms found in the references cited above. Other ingredients in [48] are:

1. The barrier function $\psi(x) = \frac{x}{1-x}$,
2. A Frank-Wolfe procedure to carry out the barrier reduction step **C2**,
3. A choice for the parameter μ in **B2** that provides a sufficiently rapid increase in throughput, and

4. A provably good choice for the starting point $\hat{x}$ in **A2**.

The particular implementation we describe below differs from that in [48] in two aspects:

- Our Frank-Wolfe iterations are minimum-cost flow problems, as opposed to shortest path computations, and
- In **C2** we may carry out several Frank-Wolfe iterations (i.e. several gradient steps) whereas [48] uses a single iteration.

If **A2 - C2** is implemented using shortest path computations only and single Frank-Wolfe iterations in **C2**, the resulting algorithm can be shown to converge to an ϵ-approximate solution in $O(\rho^2 m^5 \epsilon^{-3} k)$ shortest paths computations, where ρ is the width parameter [58]. Finally, we note that in [48] the term "flow deviation method" appears to refer to the Frank-Wolfe procedure itself; however it has since become synonymous with the entire scheme **A2 - C2**.

3.4 The algorithm

In this section we describe our implementation of **A2 - C2**. We will use the same notation as in Section 3.2. We let q be the smallest positive integer such that $2^{-q} \leq \epsilon$.

Algorithm FD

Step 0. (Initialization.) To each edge e of the graph assign the length u_e^{-1}. For $1 \leq j \leq k$, let z^j denote the flow that carries d_j units along the shortest path between s_t and t_j under this metric, and let z denote the multicommodity flow $(z^1, z^2, \cdots, z^k)$. Set $x^0 = \frac{1}{2\lambda(z)} z$ and $t = 0$.

Step 1. Set $y^t = \left(\frac{1}{2\lambda(x^t)} + \frac{1}{2}\right) x^t$.

Step 2. (Frank-Wolfe procedure.) Write $v^0 = y^t$ and $\gamma = \gamma(y^t)$.

For $h = 0, 1, \cdots$ **Do:**

A.h For $1 \leq j \leq k$, let $w^{h,j}$ denote the solution to the minimum-cost flow problem where we send γd_j units of flow from s_j to t_j and each edge e has cost $[\nabla \Psi(v^h)]_e$ and capacity u_e. Let $w^h = (w^{h,1}, \cdots, w^{h,k})$ denote the resulting multicommodity flow.

B.h Set $v^{h+1} = (1 - \sigma_h) v^h + \sigma_h w^h$ where $0 \leq \sigma_h \leq 1$ is chosen so as to minimize $\Psi((1 - \sigma_h) v^h + \sigma_h w^h)$.

C.h If $\Psi(v^h) - \Psi(v^{h+1}) < \frac{\Psi(v^h)^2}{128(\Psi(v^h)^3 + m)k^2}$, exit loop: set $t \leftarrow t + 1$, $x^t \leftarrow v^{h+1}$ and go to **Step 3**.

End

Step 3. If $\lambda(x^t) \geq 1 - \frac{\epsilon}{2m}$, or if $\Psi(x^t) \geq m 2^{q+2}$, **terminate** algorithm.

Otherwise to **Step 1**.

End.

3.4.1 Analysis of Algorithm FD

Given $\gamma > 0$ denote

$$\Psi^*(\gamma) \;=\; \min\{\Psi(x) \,:\, x \text{ feasible with } \gamma(x) = \gamma\}. \tag{3.4.1}$$

In order to show the correctness of the algorithm, we first state the following theorem whose proof is deferred:

Theorem 2 *Consider an execution of the* **For** *loop in* **Step 2**, *with input* y^t. *Suppose the loop exits at iteration* h. *Then*

$$\Psi(v^h) \;\leq\; 2\Psi^*(\gamma(y^t)). \tag{3.4.2}$$

Pending the proof of Theorem 2, the following sequence of lemmas establish the correctness and workload of the algorithm.

Lemma 3.4.1 *Let* r *be a nonnegative integer. (i) Suppose* $\gamma < (1-2^{-(r+1)})\Gamma^*$. *Then* $\Psi^*(\gamma) < 2^{r+1}m$. *(ii) Suppose* $(1-2^{-r})\Gamma^* \leq \gamma$. *Then* $2^r - 1 \leq \Psi^*(\gamma)$.

Proof. (i) Consider x^* feasible with throughput Γ^*. Then $u = \frac{\gamma}{\Gamma^*}x^*$ has throughput γ and satisfies $\Psi(u) \leq m\Psi(\frac{\gamma}{\Gamma^*}) < 2^{r+1}m$. (ii) is proved in a similar way. ■

Lemma 3.4.2 *Let* $0 < \tau < 1$, *and suppose that an iterate* x^t *in* **Step 3** *satisfies* $\lambda(x^t) \geq 1 - \frac{\tau}{2m}$. *Then*

$$\gamma(x^t) \;\geq\; (1-\tau)\Gamma^* \tag{3.4.3}$$

Proof. Assume by contradiction that (3.4.3) does not hold. Let x^* be feasible with throughput Γ^*. Then $u = \frac{\gamma(x^t)}{\Gamma^*} x^*$ has throughput $\gamma(x^t)$, and satisfies

$$\lambda(u) \;<\; 1 - \tau. \tag{3.4.4}$$

As a result

$$\Psi(u) \;<\; m(\tau^{-1} - 1). \tag{3.4.5}$$

Given the value of $\lambda(x^t)$, we also have

$$\Psi(x^t) \;\geq\; \frac{2m}{\tau} - 1, \tag{3.4.6}$$

which together with (3.4.5) contradicts Theorem 2. ■

Lemmas 3.4.1(i) and 3.4.2 imply:

Corollary 3.4.3 *Upon termination of Algorithm FD, we have a feasible flow with ϵ-optimal throughput.*

Now we turn to the complexity of Algorithm FD. First we have:

Lemma 3.4.4 *The vector x^0 has throughput at least $\frac{\Gamma^*}{2m}$.*

Proof. This result is implied by weak linear programming duality, but a direct proof follows. Denote by L^* the sum, over all commodities, of the shortest path lengths in **Step 0**. If x is any multicommodity flow we have

$$m\lambda(x) \;\geq\; L^*. \tag{3.4.7}$$

Clearly $\lambda(z) \leq L^*$. Consequently,

$$\frac{1}{\lambda(z)} \geq \frac{1}{m\lambda(x)}. \tag{3.4.8}$$

Thus $\frac{1}{\lambda(z)} \geq \frac{\Gamma^*}{m}$, and by construction of x^0, the result follows. ■

In what follows, for integral $0 \leq r$ we will denote by *Phase r* of the algorithm the set of those iterations t with $(1-2^{-r})\Gamma^* \leq \gamma(x^t) < (1-2^{-(r+1)})\Gamma^*$. Note that a given Phase r might be empty, and that the algorithm might perform iterations of Phase r with $r > q$.

Lemma 3.4.5 *(i) For $0 < r \leq q$, the number of iterations t in Phase r is $O(m)$. (ii) The number of iterations in Phase 0 is $O(m \log m)$. (iii) The total number of iterations in Phases $q+1, q+2, \cdots$ is $O(m)$.*

Proof. By Lemma 3.4.2, for $r \geq 0$ if $\gamma(x^t) < (1-2^{-(r+1)})\Gamma^*$ we have $\lambda(x^t) < 1 - \frac{2^{-r}}{4m}$. Consequently,

$$\begin{aligned} \gamma(x^{t+1}) = \gamma(y^t) &= \frac{1 - 1/2(1-\lambda(x^t))}{\lambda(x^t)}\gamma(x^t) && (3.4.9)\\ &= \frac{1}{2}(1 + \frac{1}{\lambda(x^t)})\gamma(x^t) && (3.4.10)\\ &> (1 + \frac{2^{-r}}{8m})\gamma(x^t). && (3.4.11) \end{aligned}$$

(i) Suppose $0 < r$. By definition we started Phase r with throughput at least $(1-2^{-r})\Gamma^*$. Thus, (3.4.11) implies that this Phase will perform $O(m)$ iterations, as desired.

(ii)This follows as (i), using Lemma 3.4.4.

(iii) Consider an iteration t during Phase r with $r > q$. Since the algorithm has not yet terminated, we can replace (3.4.11) with the stronger condition $\gamma(y^t) > (1+\frac{2^{-q}}{8m})\gamma(x^t)$, and again we obtain that there are altogether at most $O(m)$ iterations in Phases $q+1, q+2, \cdots$. ■

The next lemmas analyze the complexity of each execution of **Step 2**.

Lemma 3.4.6 *Consider an iteration t of* **Step 1**, *and let e be any edge. Then* $1 - \lambda_e(y^t) \geq \frac{1-\lambda_e(x^t)}{2}$.

Proof. We have that $\lambda_e(y^t) = \frac{1+\lambda(x^t)}{2\lambda(x^t)}\lambda_e(x^t)$. Since by definition $\lambda_e(x^t) \leq \lambda(x^t)$, the result follows. ■

Lemma 3.4.7 *Consider an iteration t of* **Step 1** *during Phase r. Then* $\Psi(y^t) \leq O(m2^{\min\{r,q\}})$.

Proof. Suppose first that $r > 0$. Then by Theorem 2 and respectively, Lemma 3.4.1 (for the case $r \leq q$) and the second termination criterion in **Step 3** (for the case $r > q$) we have that $\Psi(x^t) \leq O(m2^{\min\{r,q\}})$. To obtain the desired result, we will show $\Psi(y^t) = O(\Psi(x^t))$. To see this, consider the contributions of an edge e to $\Psi(x^t)$ and to $\Psi(y^t)$. These are, respectively, $\frac{\lambda_e(x^t)}{1-\lambda_e(x^t)}$ and $\frac{\lambda_e(y^t)}{1-\lambda_e(y^t)}$. By Lemma 3.4.6, the denominator in the second expression is at least half of that in the first. To compare the numerators, note that since this is an iteration during Phase $r > 0$,

$\lambda(x^t) \geq 1 - 2^{-r} \geq 1/2$ and consequently $\lambda_e(y^t) = \frac{1+\lambda(x^t)}{2\lambda(x^t)}\lambda_e(x^t) \leq \frac{3}{2}\lambda_e(x^t)$. Hence $\Psi(y^t) = O(\Psi(x^t))$, as desired.

Suppose instead that $r = 0$. We always have (again by Theorem 2 and by the choice of x^0) that $\Psi(x^t) = O(m)$. If $\lambda(x^t) \geq 1/2$ the Lemma follows as in the previous paragraph. If instead $\lambda(x^t) < 1/2$, then each edge e will satisfy $\lambda_e(y^t) \leq 3/4$ and thus $\Psi(y^t) = O(m)$. ■

Lemma 3.4.8 *Let $0 \leq r$. The number of iterations of* **A.h-C.h** *in an execution of* **Step 2** *during Phase r is $O(2^{2\min\{r,q\}}m^2k^2)$.*

Proof. We will show that if more than $O(2^{2\min\{r,q\}}m^2k^2)$ iterations h achieve

$$\Psi(v^{h+1}) - \Psi(v^h) \quad < \quad -\frac{\Psi(v^h)^2}{128(\Psi(v^h)^3 + m)k^2}, \tag{3.4.12}$$

then we will reach a value of Ψ smaller than Ψ^*, a contradiction. Thus, consider an iteration h where (3.4.12) holds. Suppose first that

$$\Psi(v^h)^3 \quad \geq \quad m. \tag{3.4.13}$$

In this case, the recursion (3.4.12) can be abstracted as

$$z_{h+1} - z_h \quad \leq \quad -c\frac{1}{z_h k^2},$$

where $c > 0$ is a constant. This recursion has the property that it reduces z_h by a factor of 2 in $O(z_h^2k^2)$ iterations. Thus, using Lemma 3.4.7, we obtain that there are at most $O(2^{2\min\{r,q\}}m^2k^2)$ iterations of **A.h-C.h** where (3.4.13) holds.

In the remainder of the proof we handle the iterations with $\Psi(v^h)^3 < m$. If $r > 0$ then using Lemma 3.4.1(ii) we conclude that each iteration satisfying (3.4.12) decreases Ψ by $\Omega(1/(mk^2))$, and consequently there are at most $O(m^{5/3}k^2)$ such iterations (a tighter analysis is possible but not needed). This concludes the proof if $r > 0$. Finally, if $r = 0$ then just as in the previous line we conclude that in at most $O(m^{5/3}k^2)$ iterations we obtain $\Psi(v^h) \leq 1$. By Lemma 3.4.4, $\gamma(x^0) \geq \frac{\Gamma^*}{2m}$, and a variation of the analysis in Lemma 3.4.1 shows that any time during Phase 0, $\Psi^* \geq \frac{1}{2m}$. We conclude that there are at most $O(\frac{1}{1/2m}) = O(m)$ iterations h with $\Psi(v^h) < 1$. ■

Corollary 3.4.9 *The total number of Frank-Wolfe iterations* **A.h** *over the course of Algorithm FD is* $O(k^2m^3\epsilon^{-2} + m^3 \log m)$.

In the next section we prove Theorem 2. In Section 3.4.3 we show how to replace the line-search in step **B.h** with a provably good stepsize rule that requires $O(1)$ time.

3.4.2 Proof of Theorem 2

Consider an iteration h of the Frank-Wolfe procedure, corresponding to input vector y^t. For simplicity, write $\gamma = \gamma(y^t)$. For $0 \leq \sigma \leq 1$, write

$$g(\sigma) \doteq \Psi(\,(1-\sigma)v^h + \sigma w^h) = \Psi(\,v^h + \sigma(w^h - v^h)). \tag{3.4.14}$$

Then g is convex, $g(0) = \Psi(v^h)$,

$$g'(0) = [\nabla\Psi(v^h)]^T \cdot (w^h - v^h), \text{ and} \tag{3.4.15}$$

$$g(\sigma) = g(0) + g'(0)\sigma + \frac{1}{2}g''(\alpha)\sigma^2, \tag{3.4.16}$$

where $0 < \alpha < \sigma$ (2nd order Taylor expansion). By choice of w^h in **A.h**, the right-hand side of (3.4.15) cannot decrease if we replace w^h with any other feasible flow with throughput γ. In particular, if we use a flow w^* with throughput γ and such that $\Psi(w^*) = \Psi^*(\gamma)$, we obtain

$$g(\sigma) - g(0) \leq [\nabla\Psi(v^h)]^T \cdot (w^* - v^h)\sigma + \tfrac{1}{2}g''(\alpha)\sigma^2. \tag{3.4.17}$$

which, since Ψ is convex, implies:

$$g(\sigma) - g(0) \leq (\Psi^*(\gamma) - \Psi(v^h))\sigma + \frac{1}{2}g''(\alpha)\sigma^2. \tag{3.4.18}$$

Next we will bound the quadratic term in (3.4.18). We have

$$g(\alpha) = \sum_e \psi(\lambda_e(v^h) + \sigma[\lambda_e(w^h) - \lambda_e(v^h)]), \tag{3.4.19}$$

and consequently

$$g''(\alpha) = \sum_e \frac{(\lambda_e(w^h) - \lambda_e(v^h))^2}{(1 - \lambda_e(v^h) - \alpha[\lambda_e(w^h) - \lambda_e(v^h)])^3} \tag{3.4.20}$$

Since both v^h and w^h are feasible, $(\lambda_e(w^h) - \lambda_e(v^h))^2 \leq k^2$. Further, $\frac{1}{(1-x)^3} < 8(\frac{x}{1-x})^3 + 8$ for all $0 \leq x < 1$. Thus, from (3.4.20) we get that

$$g''(\alpha) \leq 8(k^2[\Psi(v^h + \alpha(w^h - v^h)]^3 + m). \tag{3.4.21}$$

If σ is chosen so that

$$\Psi(v^h + \sigma(w^h - v^h)) \leq \Psi(v^h), \tag{3.4.22}$$

then (3.4.21) and the convexity of Ψ imply $g''(\alpha) \leq 8k^2(\Psi(v^h)^3 + m)$, and substituting in (3.4.18) we obtain:

$$\begin{aligned} g(\sigma) - g(0) &= \\ \Psi(v^h + \sigma(w^h - v^h)) - \Psi(v^h) &\leq (\Psi^*(\gamma) - \Psi(v^h))\sigma + 8k^2(\Psi^3(v^h) + m)\sigma^2 \end{aligned} \tag{3.4.23}$$

Inequality (3.4.23) holds for any $0 \leq \sigma \leq 1$ that satisfies (3.4.22). In Step **B.h** we choose $\sigma = \sigma_h$ so that the left-hand side of (3.4.23) is minimized. Instead, let $0 \leq \mu < 1$ be the argument that minimizes the quadratic Q on the right-hand side of (3.4.23) (note: a simple check verifies the stated bounds on μ). We would like to argue that $\Psi(v^h + \sigma_h(w^h - v^h)) - \Psi(v^h) \leq Q(\mu)$. This will follow if we can argue that μ is such that (3.4.23) holds at $\sigma = \mu$; or, in other words, that $\sigma = \mu$ satisfies (3.4.22).

But note that for $0 \leq \sigma$ small enough (3.4.22) holds by choice of w^h ($w^h - v^h$ is a descent direction). If (3.4.23) holds for all $0 \leq \sigma$ we are done. Otherwise (3.4.22) holds for some closed interval $[0, \tilde{\sigma}]$ and does not hold outside the interval. Since (3.4.23) holds with equality at 0, and $Q'(0) < 0$, clearly $\mu < \tilde{\sigma}$, as desired.

Thus, from (3.4.23) we get (after a calculation to minimize the quadratic)

$$\begin{aligned} \Psi(v^{h+1}) - \Psi(v^h) &= \\ \Psi(v^h + \sigma_h(w^h - v^h)) - \Psi(v^h) &\leq -\frac{(\Psi^*(\gamma) - \Psi(v^h))^2}{32k^2(\Psi^3(v^h) + m)}. \end{aligned} \tag{3.4.24}$$

If $\Psi(v^h) > 2\Psi^*(\gamma)$ we therefore obtain

$$\Psi(v^{h+1}) - \Psi(v^h) \leq -\frac{\Psi^2(v^h)}{128k^2(\Psi^3(v^h) + m)}. \tag{3.4.25}$$

The theorem is proved. ■

3.4.3 Choosing the stepsize

A key step in Algorithm FD is the choice of the stepsize σ_h in Step **B.h**. We expect that from the point of view of computational effectiveness, it will be best to actually carry out a numerical line-search to estimate σ_h. In the rest of this section, we show how to instead choose a value for σ_h in $O(1)$ time which nevertheless preserves the theoretical properties of the algorithm.

One way to choose σ so as to minimize the quadratic $Q(\sigma)$ introduced in the proof of Theorem 2, while assuming $\Psi(v^h) > 2\Psi^*(\gamma)$, or in other words, to minimize

$$\hat{Q}(\sigma) \doteq -\frac{\Psi(v^h)}{2}\sigma + 8k^2(\Psi^3(v^h) + m)\sigma^2 \tag{3.4.26}$$

whose decrease was used in the proof of Theorem 2 to bound the decrease in Ψ itself.

We have to show that if we alter Algorithm FD to use this stepsize ($= \frac{\Psi(v^h)}{32k^2(\Psi^3(v^h)+m)}$) we still have a correct algorithm with the same complexity bound.

In fact, we proved that as long as $\Psi(v^h) > 2\Psi^*(\gamma)$, choosing σ so as to minimize $\hat{Q}(\sigma)$ guarantees a decrease in Ψ of at least $\frac{\Psi^2(v^h)}{128k^2(\Psi^3(v^h)+m)}$. Thus the validity of the algorithm is preserved. Further, Lemma 3.4.8 still applies (consider the first line in its proof). This shows that the complexity bound for Algorithm FD still applies.

3.4.4 Comments on Algorithm FD

In [48], the procedures and analysis are not pitched towards provably good approximation – this field existed only in embryonic form in 1971. This is the primary cause for the differences between Algorithm FD and that in [48]. Nevertheless, the critical idea in [48] is the key ingredient in Algorithm FD.

(i) The fact that the Frank-Wolfe iterations are minimum-cost flow problems is only needed in the paragraph immediately following (3.4.20) so that we can guarantee $\lambda(w^h) \leq 1$. If we were to replace the minimum-cost flow calls with shortest path calls, then instead we could only claim $\lambda(w^h) \leq \rho$, where ρ is the *width* ([58] – in the context of this paper, the width of the problem is upper bounded by the ratio of the largest demand d_j to the smallest capacity u_e). This results in an increase in our complexity estimate by a factor of ρ^2. The "trick" of using minimum-cost flow calls to avoid the dependence on ρ is not

new: it is used in [58] and [50]. Recent algorithms for the maximum concurrent flow problem avoid the dependence on ρ while only resorting to shortest path calls ([52], [45]). We conjecture that a more refined version of our algorithm will achieve the same behavior.

(ii) The algorithmic outline in [48] uses a single Frank-Wolfe iteration in each **Step 2**. The algorithm described above can be adapted so as to use single Frank-Wolfe iterations in each **Step 2**; it will be shown in Section 3.5 that the dependence on ϵ of the adapted algorithm also grows as $O(\epsilon^{-2})$. This adaptation entails a finer tuning of the multiplier in **Step 1**.

(iii) The magnification factor $\frac{1-1/2(1-\lambda(x^t))}{\lambda(x^t)}$ used in **Step 1** is as in [48], except that the choice of the parameter $1/2$ is ours ([48] only states that "a proper tolerance" should be used). In **Step 0** our choice for flow z is precisely the same as the choice in the initialization step in [48]. We start with $x^0 = \frac{1}{2\lambda(z)}z$, rather than with $\frac{z}{\lambda(z)}$, so that we can guarantee that $\Psi(x^0) = O(m)$ (as opposed to $O(m^2)$).

(iv) Our proof of Theorem 2, up to inequality (3.4.18), follows fairly closely a similar analysis in [48](pp. 128 - 130). The latter part of our proof follows [48] somewhat less closely. In particular, [48] does not analyze the quadratic term in (3.4.18), which is key to the overall complexity estimate. Finally, our analysis prior to the proof of Theorem 2 is new.

(v) Algorithm FD is easily generalized to general packing linear programs [58].

(vi) [48] points out that [43] contains some ideas similar to those in the Flow Deviation method, although in less developed form. Courant [42] has been credited with one of the earliest uses of "potential" functions in the solution of systems of equations.

(vii) Some pointers for the reader who decides to tackle [48]. Here we use the notation from [48]. (a) The algorithm in pp. 113-114 is geared towards finding a feasible flow with throughput 1. This includes the exit condition in Step 1. (b) Step 2, and the definition of the FD operator in page 110 inductively show the for $n > 0$ we always have $\sigma_n < 1$ and consequently $RE_{n+1} > RE_n$. (c) There is an error in the paper for the case $n = 0$: the exit condition should be $\sigma_0/RE_0 \leq 1$.

3.5 Single Frank-Wolfe iterate analysis

In this section we provide the analysis of the original version of the algorithm that uses a single Frank-Wolfe iteration in each **Step 2**. First we will show how to tune the coefficient in **Step 1** and the termination criteria in **Step 3** and then adjust the corresponding lemmas to adopt these modifications. The updated FD algorithm now takes form

Algorithm FD'

Step 0. (Initialization.) To each edge e of the graph assign the length u_e^{-1}. For $1 \leq j \leq k$, let z^j denote the flow that carries d_j units along the shortest path between s_t and t_j under this metric, and let z denote the multicommodity flow $(z^1, z^2, \cdots, z^k)$. Set $x^0 = \frac{1}{2\lambda(z)} z$ and $t = 0$.

Step 1. Set $y^t = \hat{\alpha} x^t$,where $\hat{\alpha} = \frac{1-\alpha(1-\lambda(x^t))}{\lambda(x^t)} x^t$ and $\alpha = \frac{am^2k^2}{((1-\lambda(x^t))2^{-\min\{r,q\}}+m^2)}$.

Step 2. (Frank-Wolfe iteration.) Write $v = y^t$ and $\gamma = \gamma(y^t)$.

A For $1 \leq j \leq k$, let w^j denote the solution to the minimum-cost flow problem where we send γd_j units of flow from s_j to t_j and each edge e has cost $[\nabla\Psi(v)]_e$ and capacity u_e. Let $w = (w^1, \cdots, w^k)$ denote the resulting multicommodity flow.

B Set $\bar{v} = (1-\sigma)v + \sigma_h w$ where $0 \leq \sigma \leq 1$ is chosen so as to minimize $\Psi((1-\sigma)v + \sigma w)$.

C Set $t \leftarrow t+1$, $x^t \leftarrow \bar{v}$.

Step 3. If $t > a\frac{m^2}{\epsilon}k^2$, where $a > 0$ is some constant, and either $\lambda(x^t) \geq 1 - \frac{\epsilon}{2m}$, or $\Psi(x^t) \geq m2^{q+2}$, **terminate** algorithm. Otherwise go to **Step 1**.

Note that the only change in **Step 2** is a single application of Frank-Wolfe iteration and that **Step 3** is adjusted to accommodate this modification. The rest of the FD algorithm remains unaffected. We know describe the necessary changes to several lemmas in our analysis. We should specifically emphasize that the termination criteria

in **Step 3** is chosen such that the result and the proof of Theorem 2 are not affected. We however will need to modify lemmas 4.6 - 4.9 to analyze the new complexity results and link the termination criteria of FD and FD' algorithms.

Lemma 3.5.1 *(**4.6**′) (i) For $0 < r \leq q$, the number of iterations t in Phase r is $O(2^{2\min\{r,q\}}m^4k^2)$. (ii) The number of iterations in Phase 0 is $O(m \log m)$. (iii) The total number of iterations in Phases $q+1, q+2, \cdots$ is $O(2^{2\min\{r,q\}}m^4k^2)$.*

Proof. Write $\epsilon_r = 2^{-r}$. By Lemma 3.4.2, for $r \geq 0$ if $\gamma(x^t) < (1 - 2^{-(r+1)})\Gamma^*$ we have $\lambda(x^t) < 1 - \frac{\epsilon_r}{4m}$. Consequently,

$$\begin{aligned}
\gamma(x^{t+1}) > \gamma(y^t) &> \frac{1-\alpha(1-\lambda(x^t))}{\lambda(x^t)}\gamma(x^t) && (3.5.1)\\
&> (1+\frac{(1-\alpha)(1-\lambda(x^t))}{\lambda(x^t)})\gamma(x^t) && (3.5.2)\\
&> (1+\frac{(1-\alpha)(1-\lambda(x^t))}{\lambda(x^t)})\gamma(x^t) && (3.5.3)\\
&> (1+\frac{2^{-3r}}{4m^4k^2})\gamma(x^t). && (3.5.4)
\end{aligned}$$

(i) Suppose $0 < r$. By definition we started Phase r with throughput at least $(1 - 2^{-r})\Gamma^*$. Thus, (3.5.4) implies that this Phase will perform $O(2^{2r}m^4k^2)$ iterations, as desired. (ii)This follows as (i), using Lemma 3.4.4. (iii) Consider an iteration t during Phase r with $r > q$. Since the algorithm has not yet terminated, we can replace (3.5.4) with the stronger condition $\gamma(y^t) > (1 + \frac{2^{-q}}{8mk^2})\gamma(x^t)$, and again we obtain that there are altogether at most $O(2^{2q}m^4k^2)$ iterations in Phases $q+1, q+2, \cdots$. ■

The next lemmas analyze the complexity of each execution of **Step 2**.

Lemma 3.5.2 *(4.7′)* $\Psi(y^t) \leq \Psi(x^t) + \frac{2^{-\min\{r,q\}}}{amk^2}$

Proof. The increase of the potential on an edge e is

$$\frac{\lambda_e(\hat{\alpha}x^t)}{1-\lambda_e(\hat{\alpha}x^t)} - \frac{\lambda_e(x^t)}{1-\lambda_e(x^t)} = \frac{\lambda_e(x^t)(1-\hat{\alpha})}{(1-\lambda_e(x^t))(1-\hat{\alpha}\lambda_e(x^t))} \leq \frac{\lambda(x^t)(1-\hat{\alpha})}{(1-\lambda(x^t))(1-\hat{\alpha}\lambda(x^t))}.$$

Therefore

$$\Psi(\hat{\alpha}x^t) - \Psi(x^t) \leq m\frac{\lambda(x^t)(1-\hat{\alpha})}{(1-\lambda(x^t))(1-\hat{\alpha}\lambda(x^t))} = m\frac{1-\alpha}{\alpha(1-\lambda(x^t))} = \frac{2^{-\min\{r,q\}}}{mk^2}$$

■

Lemma 3.5.3 *(4.8′) Consider an iteration t of* **Step 1** *during Phase r. Then* $\Psi(y^t) \leq O(m2^{\min\{r,q\}})$.

Proof. Suppose first that $r > 0$. Then by Theorem 2 and respectively, Lemma 3.4.1 (for the case $r \leq q$) and the second termination criterion in **Step 3** (for the case $r > q$) we have that $\Psi(x^t) \leq O(m2^{\min\{r,q\}})$. By Lemma 3.5.2 $\Psi(y^t) \leq \Psi(x^t) + \frac{\epsilon}{m}$ and the result follows.

Suppose instead that $r = 0$. We always have (again by Theorem 2 and by the choice of x^0) that $\Psi(x^t) = O(m)$ and the proof follows in the same manner as in Lemma 3.4.7

Lemma 3.5.4 *(4.9′) For any $t > a\frac{m^2}{\epsilon}k^2$, where $a > 0$ is some constant,* $\Psi(\bar{v}) \leq 2\Psi^*(\gamma(y^t))$.

Proof. First we will show that if more than $O(2^{2\min\{r,q\}}m^2)$ iterations achieve

$$\Psi(\bar{v}) - \Psi(v) \;<\; -\frac{\Psi(v)^2}{128(\Psi(v)^3+m)k^2}, \tag{3.5.5}$$

then we will reach a value of Ψ smaller than Ψ^*, a contradiction. Thus, consider an iteration where (3.5.5) holds. Suppose first that

$$\Psi(v)^3 \;\geq\; m. \tag{3.5.6}$$

In this case, using Lemma 3.4.1 and Lemma 3.5.2 and with the appropriate choice of parameter a the recursion (3.5.5) can be abstracted as

$$z_{t+1} - z_t \;\leq\; -c\frac{1}{z_t k^2},$$

where c is a constant. This recursion has the property that it reduces z_t by a factor of 2 in $O(k^2 z_t^2)$ iterations. Thus, using Lemma 3.5.3, we obtain that there are at most $O(2^{2\min\{r,q\}}m^2k^2)$ iterations of **Step 1** – **Step 3** where (3.5.6) holds.

Next we handle the iterations with $\Psi(v)^3 < m$. If $r > 0$ then using Lemma 3.4.1(ii) we conclude that each iteration satisfying (3.5.5) decreases Ψ by $\Omega(1/(mk^2))$, and consequently there are at most $O(m^{5/3}k^2)$ such iterations. This concludes the proof if $r > 0$. If $r = 0$ then just as in the previous line we conclude that in at most $O(m^{5/3}k^2)$ iterations we obtain $\Psi(v) \leq 1$. By Lemma 3.4.4, $\gamma(x^0) \geq \frac{\Gamma^*}{2m}$, and a variation of the analysis in Lemma 3.4.1 shows that any time during Phase 0, $\Psi^* \geq \frac{1}{2m}$. We conclude that there are at most $O(\frac{1}{1/2m}) = O(m)$ iterations with $\Psi(v) < 1$.

Thus, after at most $a\frac{m^2}{\epsilon}k^2$ iterations of **Step 1** – **Step 3** we will achieve $\Psi(\bar{v}) - \Psi(v) < -\frac{\Psi(v)^2}{128k^2(\Psi(v)^3+m)}$ and therefore by Theorem 2 $\Psi(\bar{v}) \leq 2\Psi^*(\gamma(y^t))$. In the remainder of the proof we will handle iterations with $t > a\frac{m^2}{\epsilon}k^2$. Let t be the first iteration such that after execution of **Step 1** $\Psi(y^t) \geq 2\Psi^*(\gamma(y^t))$. Then

$\Psi(\bar{v}) - \Psi(v) < -\frac{\Psi(v)^2}{128(\Psi(v)^3+m)k^2}$ and by Lemma 3.5.2 every iteration increases $\Psi(x^t)$ by at most $\frac{2^{-\min\{r,q\}}}{mk^2}$. Therefore $\Psi(y^t) \leq 2\Psi^*(\gamma(y^t)) + \frac{2^{-\min\{r,q\}}}{mk^2}$ and one execution of **Step 2** will achieve $\Psi(\bar{v}) \leq 2\Psi^*(\gamma(y^t))$. ■

Corollary 3.5.5 *(**4.10′**) The total number of Frank-Wolfe iterations* **A** *over the course of Algorithm FD is* $O(k^2m^4\epsilon^{-2} + m^4k^2 \log m)$.

Thus we have demonstrated that using a single Frank-Wolfe iteration in **Step 2** we can achieve the same degree of convergence by tuning the magnification factor in **Step 1**

3.6 Future research

In this section we propose some ideas for further enhancement of the algorithm. [55] demonstrated that any Dantzig-Wolf type algorithm for a general fractional packing problem with high probablity will requier at least $\Omega(\epsilon^{-2})$ iterations. Therefore, in order to improve the dependency on ϵ we need to move away from the Dantzig-Wolf framework and develop a different approach to generating a sequience of approximate

solutions. In what follows we present an approach that can potentially lead to a decrease in the running time dependency on ϵ.

Recall that the central operation of the algorithm is to sequentially minimize the first-order approximation of the potential function over the set of multicommodity constraints. The length of the step taken towards the optimum of the gradient essentially reflects the "quality" of the approximation and produces a new vector within the neighborhood where the approximation is accurate. The natural modification of this scheme would be to replace the first order approximation with a tighter one, thereby expanding the neighborhood and allowing for longer steps. This in turn will require a smaller number of iterations and thus will lead to a faster convergence of the algorithm. The obvious drawback of this approach is the requirement to optimize a more complex, nonlinear function over the same set of multicommodity constraints. Further, the nonlinearity of the function will no longer allow for a simple decomposition of the resulting optimization problem. However, if a method to solve this problem efficiently is developed, this approach will lead to an overall improvement of the algorithm performance.

In what follows we consider the second order Taylor approximation of the potential function and offer the analysis of the complexity of the resulting algorithm provided an efficient method to solve the quadratic multicommodity flow problem is known.

3.6.1 Notations

For some flow f and some constant $\theta > 0$ let f_θ be any feasible flow such that $\lambda_e(f_\theta) = \lambda_e(f) + \theta t_e$, for some vector $t = \{t_e\}_e$ with $|t_e| \leq 1$. In other words, f_θ belongs to θ-neighborhood of f with respect to $\lambda_e(f)$.

Let $g = f + \theta t$ where t is as defined above for some θ-neighborhood of f. Then

$$\Psi(g) = \Psi(f) + \theta \nabla \Psi(f) t + 0.5\theta^2 t H(\Psi(f)) t + \frac{\theta^3}{6} \Psi'''(f + \alpha t) t^3$$

where $\alpha < \theta$.

Denote

$$Q(t) \doteq \Psi(f) + \theta \nabla \Psi(f) t + 0.5\theta^2 t H(\Psi(f)) t$$

and

$$E(t) \doteq \frac{\theta^3}{6} \Psi'''(f + \alpha t) t^3$$

Then the quadratic multicommodity flow problem in θ-neighborhood of f is defined as

$$\min \{Q(t) \,:\, g \text{ a feasible flow such that } g = f + \theta t\} \tag{3.6.1}$$

Here we define a feasible flow as a flow that satisfies all the constraints except for the joint capacity constraint. Note that the form of $Q(t)$ does not allow us to decompose (3.6.1) into single-commodity problems.

Let f be a flow such that $\Psi(f) \gg \Psi^*$, then there is a point $\bar{f}$ in the θ-neighborhood of f such that $\Psi(\bar{f}) \ll \Psi(f)$. Moreover, if the approximation error $E(t)$ in this

neighborhood does not exceed the possible decrease in the potential function, we can conclude that there is a vector t, such that $Q(t) \ll Q(0)$.

3.6.2 Algorithm

We can now state the new algorithm

Algorithm FDQ

Step 0. (Initialization.)

Step 1. Set $y^t = \frac{1-1/2(1-\lambda(x^t))}{\lambda(x^t)} x^t$. **Step 2.** Write $v^0 = y^t$ and $\gamma = \gamma(y^t)$.

For $h = 0, 1, \cdots$ **Do:**

A.h Let w^h denote the solution to (3.6.1) with $f \equiv v^h$ for some θ to be defined later.

B.h Set $v^{h+1} = w^h$.

C.h If $\Psi(v^h) - \Psi(v^{h+1}) < \frac{2}{7k^{1.5}(4\sqrt{\Psi}+\sqrt{m})}$, exit loop: set $t \leftarrow t+1$, $x^t \leftarrow v^{h+1}$ and go to **Step 3**.

End

Step 3. If $\lambda(x^t) \geq 1 - \frac{\epsilon}{2m}$, or if $\Psi(x^t) \geq m2^{q+2}$, **terminate** algorithm. Otherwise go to **Step 1**.

End.

Note that the only difference of FD and FDQ algorithms is the optimization problem in Step B.h and the termination criteria in C.h. Thus the only additional analysis required for the FDQ algorithm is the number of iterations in **Step 2** and the validity of its termination criteria.

3.6.3 Analysis

Denote $\delta = \frac{1}{k^{1.5}(4\sqrt{\Psi}+\sqrt{m})}$ and let f^* be such that $\Psi(f^*) = \Psi^*(\gamma)$ and $\Psi(f) > 2\Psi^*$. Let $\Psi(f) - \Psi(f_\theta) = \delta$.

Consider the example in Figure 3.1. This example illustrates the main idea of our approach, which is the dependence of the decrease in the potential function on the length of the step θ.

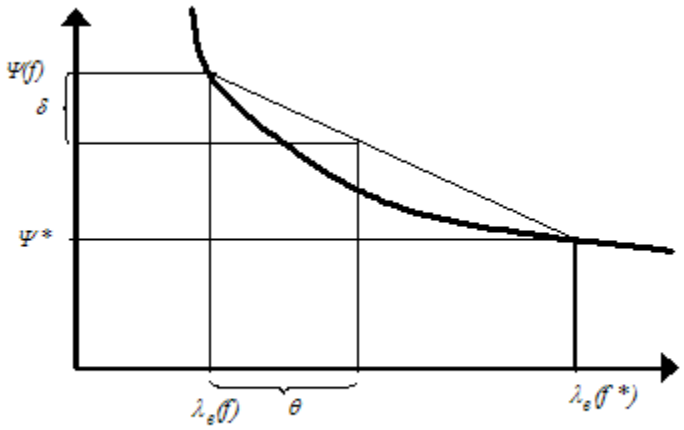

Figure 3.1: Potential function decrease in θ-neighborhood

More precisely, for every edge e

$$\frac{\theta}{|\lambda_e(f) - \lambda_e(f^*)|} = \frac{\delta}{\Psi(f) - \Psi^*}$$

Since both f and f^* are feasible flows,$|\lambda_e(f) - \lambda_e(f^*)| \leq k$, hence

$$\theta \leq \frac{\delta}{k^{0.5}(\Psi(f) - \Psi^*)} \leq \frac{2\delta}{k^{0.5}\Psi(f)}$$

or

$$\theta \leq \frac{1}{10k^{0.5}\Psi(f)^{1.5}}$$

if $\Psi(f) > 2m$ and

$$\theta \leq \frac{1}{10k^{0.5}m^{1.5}}$$

otherwise.

Lemma 3.6.1 *Let f be a feasible flow and g be a flow such that $\lambda(f) \geq 3/5$ and $|\lambda_e(g) - \lambda_e(f)| = \theta$ for every edge e, where $\theta = \frac{1}{10k^{0.5}\Psi(f)^{1.5}}$ if $\Psi(f) > 2m$ and $\theta = \frac{1}{10k^{0.5}m^{1.5}}$ otherwise. Then*

$$\Psi(g) \leq 3\Psi(f)$$

Proof: Denote $x_e \equiv \lambda_e(f)$, $y_e \equiv \lambda_e(g)$, $x \equiv \lambda(f)$, $y \equiv \lambda(g)$, then for any edge e such that $\lambda_e(g) > \lambda_e(f)$

$$\frac{\psi(\lambda_e(g))}{\psi(\lambda_e(f))} = \frac{y_e(1 - x_e)}{x_e(1 - y_e)} = \frac{(x_e + \theta)(1 - x_e)}{x_e(1 - x_e - \theta)} =$$

$$= 1 + \frac{\theta}{x_e(1 - x_e - \theta)}$$

which is clearly an increasing function of x_e for any $x_e > 3/5$. Then for any edge e

$$\frac{\psi(\lambda_e(g))}{\psi(\lambda_e(e))} \leq \frac{\psi(\lambda(g))}{\psi(\lambda(f))}.$$

Hence

1. If $\Psi(f) > 2m$, we have $\theta = \frac{1}{10k^{0.5}\Psi(f)^{1.5}} \leq (\frac{1-x}{10x})^{1.5}$. For any $x \geq 3/5$

$$(\frac{1-x}{10x})^{1.5} \leq (1-x)/2.$$

   Hence we get

$$1 + \frac{\theta}{x(1 - x - \theta)} \leq 1 + \frac{(1-x)/2}{x((1-x) - (1-x)/2)} = 1 + \frac{1}{x} \leq 3$$

2. If $\Psi \leq 2m$, then $\frac{x}{1-x} \leq 2m$, or $x \leq 1 - \frac{1}{2m}$. Then

$$1 + \frac{\theta}{x(1 - x - \theta)} \leq 1 + \frac{\frac{1}{10m^{1.5}}}{x(\frac{1}{2m} - \frac{1}{10m^{1.5}})} \leq 1 + \frac{1}{3/5(2m^{0.5} - 1)} \leq 1 + \frac{5}{5m^{0.5}} \leq 3$$

Then

$$\Psi(g) = \sum_e \frac{\lambda_e(g)}{1 - \lambda_e(g)} \leq \sum_e 3\frac{\lambda_e(f)}{1 - \lambda_e(f)} = 3\Psi(f)$$

□

Observation 1 *If at some point of the algorithm we achieve $\lambda(f) \leq 3/5$ then we can terminate* **Step 2** *without disrupting the course of the algorithm. This follows from the fact that in this case the termination criteria in* **Step 3** *is not met, and hence the proof of Lemma 3.4.2 is not affected.*

We can now bound the error term $E(t)$ for the quadratic approximation of $\Psi(f)$.

Lemma 3.6.2 *For* $\theta = \frac{1}{10k^{0.5}\Psi(f)^{1.5}}$ *if* $\Psi(f) > 2m$ *and* $\theta = \frac{1}{10k^{0.5}m^{1.5}}$ *otherwise, we have*

$$E(t) \leq \frac{5}{7}\delta$$

where $\delta = \frac{1}{k^{1.5}(4\sqrt{\Psi(f)}+\sqrt{m}}$)

Proof: Let $v = f + \alpha p$. We have

$$\Psi^{'''}(v)t^3 = \sum_e \frac{t_e^3}{(1-\lambda_e(v))^4} \leq \sum_e \frac{1}{(1-\lambda_e(v))^4} \tag{3.6.2}$$

$\frac{1}{(1-x)^4} \leq 8(\frac{x}{1-x})^4 + 8$ for all $0 \leq x < 1$. Thus, we get

$$\Psi^{'''}(v) \leq 8([\Psi(v)]^4 + m). \tag{3.6.3}$$

Using Lemma 3.6.1 we obtain

$$\begin{aligned} 1/6\Psi^{'''}(\lambda(v))t^3 &\leq \theta^3 * 8/6([\Psi(v)]^4 + m) \\ &\leq \theta^3 * 8/6([3\Psi(f)]^4 + m) \\ &= 108\theta^3\Psi(f)^4 + 8/6\theta^3 m \end{aligned} \tag{3.6.4}$$

Then if $\Psi(f) > 2m$,

$$108\theta^3\Psi(f)^4 + 8/6\theta^3 m \leq \frac{108\Psi(f)^4}{1000k^{1.5}\Psi(f)^{4.5}} + \frac{8m}{1000k^{1.5}\Psi(f)^{4.5}} \leq\leq \frac{1}{7k^{1.5}\Psi(f)^{0.5}} \leq 5/7\delta$$

If $\Psi(f) < 2m$ then

$$108\theta^3\Psi(f)^4 + 8/6\theta^3 m \leq \frac{108m^4}{1000k^{1.5}m^{4.5}} + \frac{8m}{1000k^{1.5}m^{4.5}} \leq \frac{1}{7k^{1.5}m^{0.5}} \leq 5/7\delta$$

□

Thus we have also proved the following result.

Theorem 3 *Consider an execution of the* **For** *loop in* **Step 2**, *with input* y^t. *Suppose the loop exits at iteration* h. *Then*

$$\Psi(v^h) \leq 2\Psi^*(\gamma(y^t)). \tag{3.6.5}$$

We now analyze the complexity of Step 2 of the new algorithm.

Lemma 3.6.3 *Let* $0 \leq r$. *The number of iterations of* **A.h-C.h** *in an execution of* **Step 2** *during Phase* r *of the algorithm FDQ is* $O(2^{1.5\min\{r,q\}}m^{1.5}k^{1.5})$.

Proof. We will show that if more than $O(2^{1.5\min\{r,q\}}m^{1.5}k^{1.5})$ iterations h achieve

$$\Psi(v^{h+1}) - \Psi(v^h) < -2/7\delta, \tag{3.6.6}$$

then we will reach a value of Ψ smaller than Ψ^*, a contradiction.

Thus, consider an iteration h where (3.6.6) holds. Suppose first that

$$2\Psi(v^h) \geq m. \tag{3.6.7}$$

In this case, the recursion (3.6.6) can be abstracted as

$$z_{h+1} - z_h \leq -c\frac{1}{z_h^{0.5}k^{1.5}},$$

where $c > 0$ is a constant. This recursion has the property that it reduces z_h by a factor of 2 in $O(z_h^{1.5}k^{1.5})$ iterations. Thus, using Lemma 3.4.7, we obtain that there are at most $O(2^{1.5\min\{r,q\}}m^{1.5}k^{1.5})$ iterations of **A.h-C.h** where (3.6.7) holds.

In the remainder of the proof we handle the iterations with $2\Psi(v^h) < m$. If $r > 0$ then using Lemma 3.4.1(ii) we conclude that each iteration satisfying (3.6.6) decreases Ψ by $\Omega(1/m^{0.5}k^{1.5})$, and consequently there are at most $O(m^{1.5}k^{1.5})$ such iterations. This concludes the proof if $r > 0$. Finally, if $r = 0$ then just as in the previous line we conclude that in at most $O(m^{1.5}k^{1.5})$ iterations we obtain $\Psi(v^h) \leq 1$. By Lemma 3.4.4, $\gamma(x^0) \geq \frac{\Gamma^*}{2m}$, and a variation of the analysis in Lemma 3.4.1 shows that any time during Phase 0, $\Psi^* \geq \frac{1}{2m}$. We conclude that there are at most $O(\frac{1}{1/2m}) = O(m)$ iterations h with $\Psi(v^h) < 1$. ■

Corollary 3.6.4 *The total number of iterations* **A.h** *over the course of Algorithm FD is* $O(m^{2.5}k^{1.5}\epsilon^{-1.5}T)$ *where* T *is the complexity to optimize the quadratic multi-commodity flow problem in* θ*-neighborhood of* f*.*

Thus it is evident that the algorithm developed in 1971 by Fratta, Gerla and Kleinrock not only shows a performance comparable with the most recent algorithms, but also has a strong potential for further improvement.

Bibliography

[1] D. Alevras, M. Grotschel, R. Wessaly, Capacity and survivability models for telecommunications networks, *ZIB preprint SC 97-24*, ZIB, Berlin, June 1997.

[2] D. Alevras, M. Grotschel, R. Wessaly, Cost-efficient network synthesis from leased lines, *Annals of Operations research*, 76, pp. 1-20, 1998.

[3] N. Bakhvalov, N. Zhidkov, G. Kobelkov, Numerical methods, *Physmathlit*, 2000.

[4] D. Bienstock, Potential function methods for approximately solving linear programming problems: Theory and Practice, *CORE Lecture Series*, 2001.

[5] D. Bienstock, O. Gunluk, S. Chopra and C.Y. Tsai, Mininum cost capacity installation for multicommodity flows, *Mathematical programming*, 81, pp. 177-199, 1998.

[6] D. Bienstock, O. Gunluk, Capacitated network design - polyhedral structure and computation, *Mathematical programming*.

[7] D. Bienstock, G. Muratore, Strong inequalities for capacitated survivable network design problems, *Mathematical programming*, 89, no. 1, Ser. A, pp 127-147, 2000.

[8] D. Bienstock, I. Saniee, ATM network design: traffic models and optimization-based heuristics, *Telecommunication systems*, 16:3,4, pp. 399-421, 2001.

[9] S. Chopra, I. Gilboa, S.T. Sastry, Source sink flows with capacity installation in batches, *Discrete Applied Mathematics*, 85, pp. 165192, 1998.

[10] O. Gunluk, A branch-and-cut algorithm for capacitated network design problems, *Mathematical programming* 86, 17-39, 1999.

[11] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, *Freeman*, 1979.

[12] P. Gill, W. Murray, M.Wright, Numerical linear algebra and optimization, *Addison-Wesley*, 1991.

[13] P. Gill, W. Murray, M.Saunders, User's guide for SNOPT 5.3

[14] R. D. Davis, K. Kumaran, G. Liu, I. Saniee, SPIDER: A simple and flexible tool for design and provisioning of protected lightpaths in optical networks, *Bell Labs Technical Journal*, January-June 2001.

[15] CLAPACK library, *www.netlib.org/clapack*

[16] ILOG CPLEX, *www.ilog.com/products/cplex*

[17] L. Fadeev, V. Fadeev, Numerical methods in linear algebra, *Physmathgiz*, 1960.

[18] J. Kelley, The cutting-plane method for solving convex programs, *Journal of SIAM*, 8, pp. 703-712, 1960.

[19] K. Kiwiel, An aggregate subgradient method for nonsmooth convex minimization, *Mathematical programming*, 27, pp. 320-341, 1983.

[20] S. G. Lanning, D. Mitra, Q. Wang, M. H. Wright, Optimal planning for optical transport networks, *Philosophical Transactions Royal Society London A*, Vol.358, No.1773, pp.2183-2196, August 2000.

[21] C. Lemarechal, A. Nemirovskii, Y. Nesterov, New variants of bundle methods, *Mathematical programming*, 69, pp. 111-147, 1995.

[22] D. Luenberger, Investment science, *New York : Oxford University Press*, 1998.

[23] D. Luenberger, Introduction to linear and nonlinear programming, *Addison-Wesley*, 1989.

[24] A. Malcev, Linear algebra, *Nauka*, 1970.

[25] T.L. Magnanti, P. Mirchandani, Shortest paths, single origin-destination network design, and associated polyhedra, *Networks*, 23, pp. 103121, 1993

[26] T.L. Magnanti, P. Mirchandani, R. Vachani, Modeling and solving the two-facility capacitated network loading problem, *Operations Research*, 43, pp. 142157, 1995

[27] G. Mohan, S. Murthy, A. Somani, Efficient algorithms for routing dependable connections in WDM optical networks, *IEEE/ACM Transactions on Networking*, vol. 9, no. 5, pp. 553-566, 2001.

[28] J. T. Moy, OSPF: Anatomy of an Internet Routing Protocol, *Addison-Wesley*, 1998.

[29] A. Nemirovskii, D. Yudin, Problem complexity and method efficiency in optimization, *Wiley-Interscience*, 1983.

[30] C. Papadimitriou, K. Steiglitz, Combinatorial optimization, *Dover*, 1998.

[31] U. Paul, P. Jonas, D. Alevras, M. Grotschel, R. Wessaly, Survivable mobile phone network architectures: models and solution methods, *ZIB preprint SC 96-48*, ZIB, Berlin, December 1996.

[32] Y. Pochet, L.A. Wolsey, Integer knapsack and flow covers with divisible coefficients: Polyhedra, optimization, and separation, *Discrete Applied Mathematics*, 59, pp. 5774, 1995

[33] S. Ramamurthy, B. Makherjee, Survivable WDM mesh networks,Part I - Protection, *IEEE INFOCOM*, vol. 2, pp. 744-751, 1999.

[34] S. Ramamurthy, B. Makherjee, Survivable WDM mesh networks,Part II - Restoration, *IEEE ICC*, vol. 3, pp. 2023-2030, 1999.

[35] I. Saniee, Optimal routing design in self-healing communications networks, *Int. Trans. Op. Res.*, vol 3, no. 2, pp. 187-195 1996.

[36] A. Schrijver, Theory of linear and integer programming, *Wiley-Interscience*, 1999.

[37] M. Sridharan, A. Somani, M. Salapaka, Approaches for capacity and revenue optimization in survivable WDM networks, *Journal of High Speed Networks*, no. 2, pp. 109-125 2001.

[38] M. Sridharan, M. Salapaka, A. Somani, Operating mesh-survivable WDM transport networks, *SPIE International Symposium on SPIE Terabit Optical Netwroking: Terabit Optical Networking*, pp. 113-123 2000.

[39] R. Vanderbei, LOQO User's manual, *Statistics and Operations Research, Technical report No SOR-97-08*, 1997.

[40] J. Veerasamy, S. Venkatesan, J. C. Shah, Spare capacity assignment in telecom networks using path restoration, *MASCOTS*, pp. 370-375 1995.

[41] D. Bienstock, Approximately solving large-scale linear programs. I. Strengthening lower bounds and accelerating convergence, CORC Report 1999-1, Columbia University. (Extended abstract: *Proc 11th. Ann. ACM-SIAM Symposium on Discrete Algorithms*, January 2000).

[42] R. Courant, Variational methods for the solution of problems of equilibrium and vibration, *Bull. Amer. Math. Soc.* **49** (1943), 1 – 43 .

[43] S.C. Dafermos and F.T. Sparrow, The traffic assignment problem for a general network, *Journal of Research of the National Bureau of Standards - B*, **73B** (1969).

[44] G.B. Danzig and P. Wolfe, The decomposition algorithm for linear programming, *Econometrica* **29** (1961), 767 – 778.

[45] L. Fleischer, Approximating fractional multicommodity flow independent of the number of commodities, *SIAM J. Disc. Math.* **3** (2000) 505 – 520.

[46] A.V. Fiacco and G.P. McCormick, *Nonlinear Programming: Sequential Unconstrained Optimization Techniques*, Wiley, New York (1968).

[47] M. Frank and P. Wolfe, An algorithm for quadratic programming, *Naval Res. Logistics Quarterly* **3** (1956), 149 – 154.

[48] L. Fratta, M. Gerla and L. Kleinrock, The flow deviation method: an approach to store-and-forward communication network design, *Networks* **3** (1971), 97 – 133.

[49] A.V. Goldberg, J.D. Oldham, S. Plotkin and C. Stein, An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow, in *Integer Programming and Combinatorial Optimization* (Bixby, Boyd, Rios-Mercado, eds.) **1412** (1998) 338 - 352.

[50] M.D. Grigoriadis and L.G. Khachiyan Fast approximation schemes for convex programs with many blocks and couping constraints, *SIAM Journal on Optimization* **4** (1994) 86 – 107.

[51] M.D. Grigoriadis and L.G. Khachiyan An exponential-function reduction method for block-angular convex programs, *Networks* **26** (1995) 59 – 68.

[52] N. Garg and J. Könemann, Faster and simpler algorithms for multicommodity flow and other fractional packing problems, *Proc. 39th Ann. Symp. on Foundations of Comp. Sci.* (1998) 300 – 309.

[53] D. Karger and S. Plotkin, Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows, *Proc. 27th Ann. ACM Symp. on Theory of Computing* (1995) 18 – 25.

[54] P. Klein, S. Plotkin, C. Stein and E. Tardos, Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts, *Proc. 22nd Ann. ACM Symp. on Theory of Computing* (1990),310 – 321.

[55] P. Klein and N. Young, On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms.

[56] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos and S. Tragoudas, Fast approximation algorithms for multicommodity flow problems, *Proc. 23nd Ann. ACM Symp. on Theory of Computing* (1991), 101-111.

[57] T. Leighton and S. Rao, An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms, *Proc. FOCS 29* (1988), 422 – 431.

[58] S. Plotkin, D.B. Shmoys and E. Tardos, Fast approximation algorithms for fractional packing and covering problems, *Proc. 32nd Annual IEEE Symp. On Foundations of Computer Science*, (1991), 495 – 504.

[59] T. Radzik, Fast deterministic approximation for the multicommodity flow problem, *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms* (1995), 486 – 492.

[60] F. Shahrokhi and D.W. Matula, The maximum concurrent flow problem, *Journal of the ACM* **37** (1991), 318 – 334.

[61] J. Villavicencio and M.D. Grigoriadis, Approximate Lagrangian decomposition using a modified Kamkarkar logarithmic potential, in *Network Optimization*

(Pardalos and Hager, eds.) Lecture Notes in Economics and Mathematical Systems **450** Springer-Verlag, Berlin (1995), 471 – 485.

[62] N. Young, Randomized rounding without solving the linear program, in The maximum concurrent flow problem, *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms* (1995), 170 – 178.