EFFICIENT CONTINUOUS ALGORITHMS FOR COMBINATORIAL OPTIMIZATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Anil Kamath
June 1995

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

Serge A. Plotkin
(Principal Advisor)

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

Rajeev Motwani

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

Andrew Goldberg

Approved for the University Committee on Graduate Studies:

Abstract

The thesis concentrates on design, analysis, and implementation of efficient algorithms for large scale optimization problems with special emphasis on problems related to network flows and routing in high-speed networks. The main focus is on using continuous techniques for solving large combinatorial optimization problems in off-line and online settings.

Some of the major areas of work include:

- Developing and implementing new interior-point techniques for efficiently solving large scale multi-commodity network flow problems. These algorithms are based on solving linear and quadratic programming formulations of the problem.
- Designing fast algorithms based on continuous methods for approximate solution of *minimum cost* multi-commodity flow problem. These techniques are geared towards distributed implementation.
- Application of multi-commodity flow techniques for design and implementation of centralized and distributed algorithms for on-line routing of Virtual Circuits in ATM (Asynchronous Transfer Mode) networks.

Our approach is to design algorithms that have provably good worst or average case performance and demonstrate their efficiency by implementing and evaluating them on real data.

Acknowledgements

First, I would like to thank my advisor Serge Plotkin. He made my stay in Stanford exciting and stimulating. He provided ideas, advice and support throughout my Ph.D. work and this thesis would not have been possible without his guidance.

Next, I would like to thank Rajeev Motwani for teaching excellent courses during my stay at Stanford and also providing me the opportunity for doing some interesting work with him. I also owe thanks to Andrew Goldberg for teaching interesting classes.

I am grateful to Omri Palmon for collaborating on many parts of this thesis. His confidence and insight helped me a lot in my work. I would also like to thank K.G. Ramakrishnan, Narendra Karmarkar and Rainer Gawlick who co-authored some of the results presented here. Ramakrishnan provided the practical and implementational framework for most of the work in this thesis. He provided us with information and data on many interesting real-world problems.

And many thanks to Birdy for being always willing to listen and ask probing questions about my work and tell me about the exciting work in his field.

I was also fortunate to have many interesting and helpful friends in the Computer Science Department, like Luca, Henny, Sanjeev, Julien, Jeffrey to name a few. Finally, I thank my parents, Prabhakar and Poornima, and my sisters, Swati and Sharmila, for their love and support.

This research was supported by NSF Grant CCR-9304971 and ARO Grant DAAH04-95-1-0121.

Contents

Abstract **iv**

Acknowledgements **v**

1 Introduction **1**

2 A Linear Programming method for the Multi-commodity Problem **6**

2.1 Introduction 6

2.2 Linear Programming Formulation 7

2.3 An Overview of the Approximate Dual Projective Algorithm 8

2.4 Complexity of ADP Algorithm for MCNF 12

2.5 Efficient Implementation of ADP Algorithm for MCNF 13

2.6 Concurrent Flow Problem 16

2.7 Computational Experience 16

3 A Quadratic Programming approach to Multi-Commodity Flow **19**

3.1 Introduction 19

3.2 Multi-commodity flow as QP 22

3.3 The Algorithm 23

3.4 Analysis of number of operations per iterations 27

3.4.1 Computing the Newton step using conjugate gradient. 28

3.4.2 Computing the Newton step using matrix inversion and rank one updates. 31

3.5 Starting Point For The Algorithm 33

3.6 Stability 35

3.7 Finding an Exact Solution 37

3.8 Extensions 38
3.8.1 Minimum Cost Multi-commodity Flow. 38
3.8.2 The Concurrent Flow Problem. 41
3.8.3 The Generalized Flow Problem. 45

4 Distributed Algorithm for Multi-commodity Flow 47
4.1 Introduction 47
4.2 Preliminaries and Definitions 49
4.3 The Flow Without Costs 50
4.4 Flow With Costs 53
4.5 Implementation of a single iteration 56

5 Approximate Algorithm for Min-cost Multi-commodity Flow 58
5.1 Introduction 58
5.2 Continuous algorithm 60
5.2.1 The Algorithm 61
5.2.2 Bounding number of phases 62
5.3 Improving The Running Time 66
5.4 Bounding the number of phases of Algorithm-2 69

6 Routing and Admission Control of Virtual Circuits 72
6.1 Introduction 72
6.1.1 Previous work 74
6.2 Centralized On-line Algorithm for SVC Routing 78
6.2.1 Exponential-cost based routing 78
6.2.2 Adding probabilistic assumptions 80
6.2.3 Simulation Results 83
6.3 Distributed Algorithm 90
6.3.1 Simulation Results 91
6.4 Discussion 93

7 Competitive routing under known Holding time Distributions 95
7.1 Introduction 95
7.2 Notation and Definitions 96
7.3 Adaptive Adversary 97

7.4 Admission Control and Routing Algorithm 98
7.5 Exponential Holding Time 101
7.6 Simulation Results 103

Bibliography **105**

List of Tables

2.1 Performance of ADP on *rmfgen* problems 17

2.2 Performance of ADP on the concurrent flow problem 18

2.3 Performance of ADP on *mgrid* problems 18

6.1 NSFNet Traffic Profile 88

List of Figures

2.1 Pictorial View of the Approximate Dual Projective(ADP) Algorithm 9
2.2 MCNF Matrix 14
2.3 Object Oriented Structure of the Software 14

3.1 Comparison of Complexities. 20

6.1 The original AAP routing algorithm 79
6.2 Topology for an existing commercial network. 84
6.3 Simulation results for the centralized algorithms 85
6.4 Determining μ 86
6.5 Comparing EXP with randomized minimum-hop 87
6.6 Topology for NSFNet T3 Backbone. 88
6.7 Comparative performance for Internet topology 89
6.8 Performance of distributed vs centralized schemes. 92

7.1 Comparative performance for Commercial topology 104

Chapter 1

Introduction

Most of the research in computer science deals with developing techniques for processing of discrete objects. Hence not surprisingly most algorithms in the field are based on combinatorial techniques. However, related areas like scientific computation and operations research have developed a rich field of continuous methods that have proved to be both powerful and practical. There are many problems in computer science like multicommodity flow, minimum coloring of perfect graphs or finding the minimum value of submodular set functions, which though combinatorial in nature can be efficiently solved using continuous methods. We distinguish methods to be continuous or combinatorial depending on whether they work on the space of real numbers or in discrete space respectively. There has been a lot of prior work on applying continuous techniques for solution of combinatorial problems. In this thesis, we explore ideas for solving some combinatorial problems arising in networks using continuous methods. We demonstrate how ideas from continuous optimization can be used to efficiently solve some problems that arise in the context of networks.

If we consider the classical problem of flows in networks, there are several known good combinatorial methods for the problem. However, these combinatorial methods do not apply when we introduce simple modifications to the flow problem like introducing multiple commodities or introducing gain or loss factor on edges. The best known algorithms for these problems, in terms of complexity measure, are based on optimizing continuous cost functions defined over the space of real numbers. Thus, the need for continuous techniques becomes apparent for generalized flow and multi-commodity flow problems.

Typically, in solving a combinatorial problem using continuous techniques, one has to first consider a continuous relaxation of the problem and then construct a linear programming(LP) formulation of the relaxed problem. If the LP formulation has certain characteristics, then the solution to the LP relaxation can be rounded to a solution to the combinatorial optimization problem. There are many problems in computer science for which the only known efficient algorithms are based on linear or non-linear optimization.

Consequently, the invention of interior-point techniques for linear and non-linear programming had a big impact on the use of continuous methods for solving discrete problems. The interior-point technique gave good polynomial-time algorithms for a wide-range of combinatorial problems. In the interior-point approach, we define a continuous potential function which is then minimized iteratively over the continuous region of optimization until you reach a point which is close to the optimum combinatorial solution. Many other continuous techniques for combinatorial problems use the idea of a potential function to measure progress and also to define the step at each iteration. Another technique for designing continuous methods for combinatorial optimization involves minimizing the duality gap by looking at the dual of the primal LP formulation. The dual variables then define a cost which is reduced at each step until convergence. This idea of dual costs can be extended to solve on-line optimization problems, to design competitive algorithms that have provably good performance under all possible input sequences.

We shall examine these ideas of continuous optimization in the context of flow problems. We shall first consider a simple linear programming formulation of the multi-commodity flow problem and its variants and examine the computational and complexity issues. We then show how a modified quadratic programming formulation for the problem reduces the complexity in certain cases. We then give approximation algorithms for multi-commodity flow which are distributed and based on potential function minimization. We improve the complexity of this algorithm in the non-distributed case by quantizing our continuous process. Finally, we consider the on-line routing problem which can be seen as the on-line extension of the multi-commodity flow problem and show how the dual costs can be used to provide competitive algorithms.

We begin by considering in Chapter 2, the computational and complexity issues involved in a linear programming based method for solving minimum cost multi-commodity and concurrent flow problems. We use an interior-point algorithm which is based on the Approximate Dual Projective method. We provide complexity and computational results for this algorithm. We also discuss a C++ object oriented implementation of the algorithm that exploits the special structure of the constraint matrix. Implementational details are reported for the algorithm on instances generated by a few standard publicly available generators. The results are given in Chapter 2.

In Chapter 3, we present a new interior-point based polynomial algorithm for the multi-commodity flow problem and its variants. Unlike all previously known interior point algorithms for multi-commodity flow that have the same complexity for approximate and exact solutions, our algorithm improves running time in the approximate case by a polynomial factor. For many cases, the exact bounds are better as well. Instead of using the conventional linear programming formulation for the multi-commodity flow problem, we model it as a quadratic optimization problem which is solved using interior-point techniques. This formulation allows us to exploit the underlying structure of the problem and to solve it efficiently. The algorithm is also shown to have improved stability properties. The improved complexity results extend to minimum cost multi-commodity flow, concurrent flow and generalized flow problems.

In Chapter 4, we present a simple and fast distributed algorithm for approximately solving the minimum-cost multi-commodity flow problem. If there exists a flow that satisfies all of the demands, our algorithm runs in $O(m^2\epsilon^{-2}\log n)$ communication rounds and produces a flow that satisfies $(1-\epsilon)$-fraction of each demand, where m denotes the number of edges in the graph. If there exists a feasible flow of cost B, then our algorithm terminates in $O^*(m^2\epsilon^{-2}\delta^{-2})$ rounds and finds a flow that satisfies $(1-\epsilon)$ fraction of each demand and whose cost is below $(1+\delta)B$, where C and U are maximum cost and maximum edge capacity, respectively. The algorithm is *local* in the sense that in one round each processor makes a local computation and exchanges information with each one of its neighbors. Perhaps the most important property of our algorithm is its resilience to failures. For example, in the case where there are no costs, failure of an edge can delay the convergence of the algorithm by at most $O(m^2\epsilon^{-2}\log n)$ rounds, provided that this failure does not make the problem infeasible.

In Chapter 5, we describe an efficient deterministic approximation algorithm, which given that there exists a multi-commodity flow of cost B that satisfies all the demands, produces a flow of cost at most $(1+\delta)B$ that satisfies $(1-\epsilon)$-fraction of each demand. The algorithm uses techniques similar to the one described in Chapter 4, but is not distributed and has better computational complexity. For constant δ and ϵ, our algorithm runs in $O^*(kmn^2)$ time, which is an improvement over the previously fastest (deterministic) approximation algorithm for this problem due to Plotkin, Shmoys, and Tardos, that runs in $O^*(k^2m^2)$ time.

The last two chapters of the thesis deal with extending some of the continuous techniques for multi-commodity flow to design competitive on-line algorithms for routing. We show how some of these techniques can be used for designing practical routing schemes in high-speed networks.

Emerging high speed Broadband Integrated Services Digital Networks (B-ISDN) are expected to carry traffic for services like video-on-demand and video teleconferencing which will require resource reservation along the path on which the traffic is sent. As a result, such networks will need efficient *routing* and *admission control* algorithms. The simplest approach is to use fixed paths and no admission control. More sophisticated approaches which use state dependent routing and a form of admission control called trunk reservation can be found in the circuit switching literature. However, the circuit switching literature has generally focused on fully connected (complete) networks. The intuition for our routing algorithm comes from viewing the problem as an on-line variant of the multi-commodity flow problem. In the routing context, commodities correspond to virtual circuits, and demands to the requested bandwidth, and capacity to the link bandwidth. Recently, there have been several algorithms for solving approximate multi-commodity flow problems that are based on repeated rerouting of flow on shortest paths based on an exponential length function that corresponds to the dual cost of the edge in a linear programming formulation of the corresponding multicommodity flow problem [67]. These ideas have been used to construct dynamic online routing strategies [5, 7, 6]. In Chapter 6, we outline an algorithm which is an adaptation of a recently discovered theoretical algorithm that is asymptotically best possible with respect to the worst case performance. The main idea behind our algorithm is to improve the performance in the expected instead of the worst case. Although the original theoretically optimum algorithm behaves quite badly in practice [21], we show that our modifications dramatically improve the performance under several classes of traffic loads. We consider both a centralized setting, where routing and admission control decisions are made by a centralized processor, and a *distributed* setting, where decisions are made at the source node, using potentially inaccurate state information. We show that our algorithm can be easily adapted to the distributed setting. We evaluate the performance of our algorithms using extensive simulations on an existing commercial network topology and the NSFNet T3 backbone topology. The simulations show that our modified algorithm outperforms several existing algorithms. The increase in performance is most dramatic in the distributed setting, indicating that our algorithm is able to use outdated information more effectively.

In Chapter 7, we extend our routing method to route circuits for which we do not know the exact holding times but know the distribution on the holding time. We construct an algorithm that can be proven to give competitive performance in the average case. For the special case of Poisson distributions on holding time, we are able to prove a better competitive ratio and also demonstrate that it works well in practice through extensive simulations on realistic topologies.

Chapter 2

A Linear Programming method for the Multi-commodity Problem

2.1 Introduction

The multi-commodity flow problem involves simultaneously shipping several different commodities from their respective sources to their sinks in a single network so that the total amount of flow going through each edge is no more than its capacity. Associated with each commodity is a demand, which is the amount of that commodity that we wish to ship. Given a multi-commodity flow problem, one often wants to know if there is a *feasible* flow, *i.e.* if it is possible to find a flow which satisfies the demands and obeys the capacity constraints. More generally, we might wish to know the maximum percentage z such that at least z percent of each demand can be shipped without violating the capacity constraints. The latter problem is known as the *concurrent flow problem*, and is equivalent to the problem of determining the minimum ratio by which the capacities must be increased in order to ship 100% of each demands. In the minimum cost multi-commodity flow problem, there is associated with each edge a cost per unit flow and the objective then is to find a feasible multi-commodity flow with the minimum cost.

Multi-commodity flow problems arise in several practical applications such as design of communication and computer networks, virtual circuit routing in communication networks, VLSI layout, scheduling, and transportation, and hence has been extensively studied [32, 39, 77, 50, 71, 75, 16, 69, 80]. Though a natural extension of the single commodity flow problem, all known combinatorial algorithms developed for the single commodity case do not readily extend to the multi-commodity flow problem. This is because many of the properties like unimodularity of the constraint matrix in the linear programming formulation and the max-flow min-cut relation do not apply to the multi-commodity flow case. It is therefore ideal for development of solution techniques that use continuous methods.

The MCNF problem has been studied extensively and many special purpose techniques have been developed for solving this problem. Most of the algorithms developed for MCNF are based on linear [46] or non-linear programming [15] as opposed to combinatorial methods. The linear programming approaches to solving MCNF are based on either decomposition [52] or partitioning strategies [46]. In the non-linear programming approach, a barrier function is defined for the capacity constraints and non-linear gradient projection methods are used [15, 22]. In this chapter[1], we examine a variant of the interior-point method for solving a linear programming formulation of multi-commodity network flow problems. We present some computational and complexity results for an Approximate Dual Projective (ADP) variant of the interior point algorithm [40] for solving the minimum cost Multi-commodity Network Flow Problem (MCNF). The ADP algorithm has been found to be effective in solving large scale linear programming problems(LP) [42]. The motivation for this work is to study the performance of the ADP algorithm for solving LPs arising from a linear programing formulation of the MCNF problem.

2.2 Linear Programming Formulation

The MCNF problem is a natural generalization of the minimum cost single commodity network flow problem [22, 45, 52, 63, 64] . The latter concerns a single commodity that must be transported from a source node s to a sink node t. In MCNF, there are k commodities and k source-sink pairs (s_i, t_i). The problem is to simultaneously transport, at minimum total cost, d_i units of the i^{th} commodity from s_i to t_i subject to the constraint that the total flow of the commodities through each edge is bounded by capacity $u(e)$ of the edge. Formally, an instance of the problem consists of a directed graph $G = (\mathcal{V}, \mathcal{E})$, and a set of commodities $\mathcal{K}$, where $|\mathcal{V}| = n$, $|\mathcal{E}| = m$, and $|\mathcal{K}| = k$. For each commodity $i \in \mathcal{K}$ we are given a source-sink pair $(s_i, t_i) \in \mathcal{V} \times \mathcal{V}$ and a demand $d_i \in \mathcal{R}$. Each edge $e \in \mathcal{E}$ has a capacity $u(e)$ and costs $c_i(e)$ for each commodity $i \in \mathcal{K}$.

A multi-commodity flow f consists of flows $f_i(e)$ of the i^{th} commodity on edge e. Flow

[1] This chapter represents joint work with K.G. Ramakrishnan and N. Karmarkar [38].

f has to satisfy the following constraints.

$$
\begin{array}{llll}
(2.1) & f_i(e) & \geq & 0 \quad \forall e \in \mathcal{E}\,,\ i \in \mathcal{K} \\
(2.2) & \sum_{i \in \mathcal{K}} f_i(e) & \leq & u(e) \ \forall e \in \mathcal{E}
\end{array}
$$

$$
(2.3) \quad \sum_{wv \in \mathcal{E}} f_i(e)(wv) - \sum_{vw \in \mathcal{E}} f_i(e)(vw) = \begin{cases} 0 & v \neq s_i \text{ and } v \neq t_i \\ -d_i & v = s_i \\ d_i & v = t_i \end{cases} \quad \forall v \in \mathcal{V}; \forall i \in \mathcal{K} \quad .
$$

The objective is to minimize the total cost of the flow f. The objective function is given by

$$\min \sum_{i \in \mathcal{K}} \sum_{e \in \mathcal{E}} c_i(e) f_i(e).$$

2.3 An Overview of the Approximate Dual Projective Algorithm

In this section, we give a brief overview of the Approximate Dual Projective (ADP) algorithm for linear programming. For a complete description of the algorithm please see [42]. As the name implies, ADP method is an approximation to the projective algorithm of Karmarkar [40] applied to the dual. The projective algorithm has very nice stability properties, since the iterates of the projective algorithm always stay on the central trajectory [13]. The ADP method makes discrete approximations to the continuous central trajectory. Asymptotically, after a large number of iterations, the discrete approximations converge to the central trajectory. The discrete approximation consists of two steps; the objective step, and the reciprocal-estimates-improvement (REI) step. Figure 2.1 depicts the ADP method. In the figure, the optimal vertex is shown as a solid bullet at the top, and the central trajectory is shown as a double solid line. The objective step is the dual affine scaling step [1, 42] that improves the objective function. The REI step has the effect of bringing the iterate closer to the central trajectory. The name reciprocal estimates improvement comes from the fact that on the central trajectory of the dual polytope, the reciprocals of slack variables are exact primal feasible solutions up to a multiple [42]. Thus if we take a step in the dual polytope that improves the primal feasibility of the reciprocals of slacks, then this necessarily implies that the step brings the iterate closer to the central trajectory. The approach

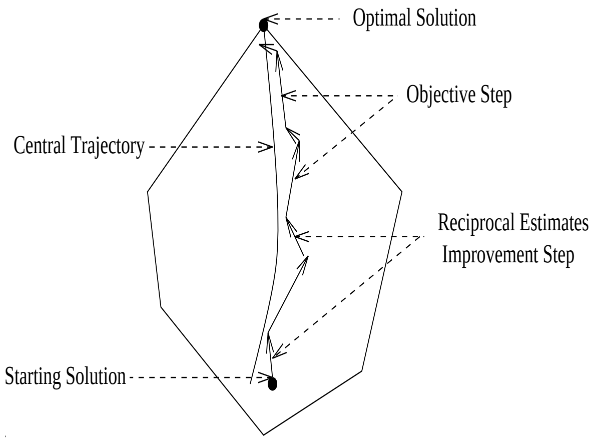

Figure 2.1: Pictorial View of the Approximate Dual Projective(ADP) Algorithm

of improving reciprocal estimates is an interesting interpretation of the traditional way of centering, i.e., that of minimizing the potential function $-\sum \ln s_i$, where s_i is the dual slack variable. In fact, the linear systems solved by the newton-method-based minimization of the potential function and the REI step are identical. The ADP algorithm is similar to the BLCA algorithm of Todd [76], the affine scaling algorithms of Gonzaga [30],Barnes, Chopra, and Jensen [12], and many others. All these algorithms use affine scaling with centering on the primal problem. Recently, dual affine scaling algorithm with centering was studied by Hipolito [31].

Generally the ADP method consists of two phases. In Phase 1, the dual phase, the dual of the LP problem is solved; and in phase 2, a primal solution is obtained, starting with the estimates from the converged dual solution. At the end of execution of both phases, a primal-dual pair with a small relative duality gap is obtained.

We now give a brief overview of the ADP method. Consider the LP problem, with upper bounds in standard primal form:

$$\min \; c^T x \tag{2.4}$$

subject to

$$Ax = b \tag{2.5}$$

$$0 \le x \le u \; . \tag{2.6}$$

Here $A \in \mathcal{R}^{m\times n}$, $c, x, u \in \mathcal{R}^n$, and $b \in \mathcal{R}^m$. Introducing dual variables y and z, corresponding to 2.5 and 2.6 respectively, the dual of 2.4–2.6 becomes

$$\max\ b^T y + u^T z \tag{2.7}$$

subject to

$$A^T y + z \leq c \tag{2.8}$$

$$z \leq 0\ . \tag{2.9}$$

Defining

$$s_2 \ = \ -z \tag{2.10}$$

and introducing slack variables s_1 in 2.8, the dual 2.7–2.9 becomes

$$\max\ b^T y - u^T s_2 \tag{2.11}$$

subject to

$$A^T y - s_2 + s_1 = c \tag{2.12}$$

$$s_1, s_2 \geq 0\ . \tag{2.13}$$

We can now develop a variant of the dual projective algorithm for solving 2.11–2.13. The scaling matrix D is defined in partitioned form as

$$D \ = \ \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix}\ ; \tag{2.14}$$

where

$$D_1^{-1} \ = \ \text{diag}\ (s_1) \tag{2.15}$$

and

$$D_2^{-1} \ = \ \text{diag}\ (s_2)\ . \tag{2.16}$$

Let Δy, Δs_1, and Δs_2 be the ascent directions used in the algorithm. Given below are the formulas for Δy, Δs_1, and Δs_2. The derivation of these formulas is identical to the derivation in [1] except for the treatment of upper bounds on the primal variables. The derivation of these formulas in the presence of upper bounds can be found in [41].

$$\begin{aligned}
(2.17)\ \Delta y &= (A\tilde{D}^2A^T)^{-1}(b - AD_1^2D_e^2u) \\
(2.18)\Delta s_1 &= -D_e^2D_2^2A^T(A\tilde{D}^2A^T)^{-1}(b - AD_1^2D_e^2u) - D_e^2u \\
(2.19)\Delta s_2 &= D_e^2D_1^2A^T(A\tilde{D}^2A^T)^{-1}(b - AD_1^2D_e^2u) - D_e^2u
\end{aligned}$$

where

$$\begin{aligned}
(2.20)D_e^{-2} &= D_1^2 + D_2^2 \\
(2.21)\ \tilde{D}^2 &= D_1^2 - D_1^2D_e^2D_1^2\ .
\end{aligned}$$

The ADP algorithm is given below.

APPROXIMATE DUAL PROJECTIVE ALGORITHM: DUAL PHASE

1. Given an initial interior starting point y^0, s_1^0, s_2^0 and parameters α, and ϵ, set the iteration counter $i = 0$; switch_criterion = false.

2. **WHILE** switch_criterion is false **DO**

   2.1 // Objective function improving step.

   a) Compute ascent directions Δy, Δs_1, and Δs_2 using equations 2.14 to 2.21.

   b) // compute maximum step length permissible to keep s_1 and
   // s_2 nonnegative.
   $\beta = \min\left(\min_{\Delta s_{1k}<0} -\frac{s_{1k}^i}{\Delta s_{1k}},\ \min_{\Delta s_{2m}<0} -\frac{s_{2m}^i}{\Delta s_{2m}}\right)$

   c) // Compute a fraction of the maximum step length.
   $\beta = \alpha\beta$

   d) // Update the solution.
   $y^{i+1} = y^i + e\Delta y$
   $s_1^{i+1} = s_1^i + \beta\Delta s_1$
   $s_2^{i+1} = s_2^i + \beta\Delta s_2$
   $i = i + 1$

2.2 // Check if relative improvement in objective function is less than ϵ
IF $((b^T y^i - u^T s_2^i) - (b^T y^{i-1} - u^T s_2^{i-1}) < \epsilon |(b^T y^{i-1} - u^T s_2^{i-1})|)$
THEN
switch_criterion = true
END_IF

2.3 // Reciprocal estimates step.

a) Apply the reciprocal estimates to the current iterate and obtain direction vectors Δy, Δs_1, and Δs_2.

b) reexecute steps 2.1.b to 2.1.d to update the solution.

END_WHILE

We can make the following observations about this algorithm.

- The reciprocal estimates applied in step 2.3.a is based on an elegant mathematical relationship between the dual slacks and the primal variables of the LP. The relationship is that, on the central trajectory [13], the reciprocals of dual slacks are primal feasible, up to a scalar multiple. This property is exploited in ADP to perform approximate centering using the reciprocal estimates improvement (REI) algorithm. The details of this algorithm can be found in [42].

- The termination criterion in step 2.2 is the normal termination criterion. This step checks whether the relative improvement in the objective function value is less than ϵ.

- The computationally intensive step is the solution of the positive definite linear system of equations encountered in evaluating the ascent direction. For this step, we have used the preconditioned conjugate gradient method, with a dynamically adaptive preconditioner.

2.4 Complexity of ADP Algorithm for MCNF

The computationally intensive step in each iteration of the ADP algorithm is the solution of a linear system of equations

$$AD^2A^T x = b, \tag{2.22}$$

where A is the constraint matrix of MCNF, given by constraints $2.2 - 2.3$ and D is a diagonal scaling matrix. In our implementation, we use Preconditioned Conjugate Gradient (PCG) to solve the linear system. The Conjugate Gradient (CG) method is an iterative method which has been shown to be effective in solving large sparse positive definite linear systems. The CG method is guaranteed to converge in m steps, where m is the rank of the matrix, and each step involves multiplying the matrix by a vector [53].

Consider an instance of the MCNF problem, with k commodities, n vertices and m edges. The LP formulation of this instance has ke variables and $kn + m$ constraints. The linear system 2.22 has rank $kn+m$ and so the CG is guaranteed to terminate in $O(kn+m)$ steps. The constraint matrix A has $O(km)$ non-zeros, and hence one CG step which involves computing AD^2A^Tx can be done in $O(km)$ operations. Thus the number of operations per ADP iteration is $O(km(kn + m))$. The ADP variant of the interior point method for LP has been shown to terminate in $O(\sqrt{p}L)$ iterations where p is the number of variables in the primal problem and L is the number of bits required to encode the problem [31]. Hence the overall complexity of ADP algorithm for MCNF can be shown to be $O(km(kn+m)\sqrt{kmL})$.

Better complexity for solving MCNF problem has been reported by Vaidya [77]. Vaidya's method uses fast matrix multiplication with direct factorization that uses the special nature of the matrix to solve the linear system [39]. However, for problem sizes of practical interest fast matrix multiplication is inefficient. This is because the constant multiplicative factors in the complexity measure for fast matrix multiplication are large. For sparse graphs, the complexity of our method matches the best known complexity as reported in [77].

2.5 Efficient Implementation of ADP Algorithm for MCNF

The constraint matrix in the LP formulation of MCNF has a block of conservation constraints for each commodity coupled by the capacity constraints on the edges, as shown in Fig. 2.2. We have efficiently implemented the ADP algorithm, by exploiting the special nature of the constraint matrix in the MCNF problem. We store the constraint matrix implicitly by just storing a single copy of the incidence matrix of the underlying graph. All matrix operations on the constraint matrix are carried out by appropriate indexing in the incidence matrix. The capacity constraints are not stored since their structure is known.

We implemented the ADP algorithm for MCNF in an object-oriented fashion. We define a C++ class called MCF-matrix with member functions to do various operations on the

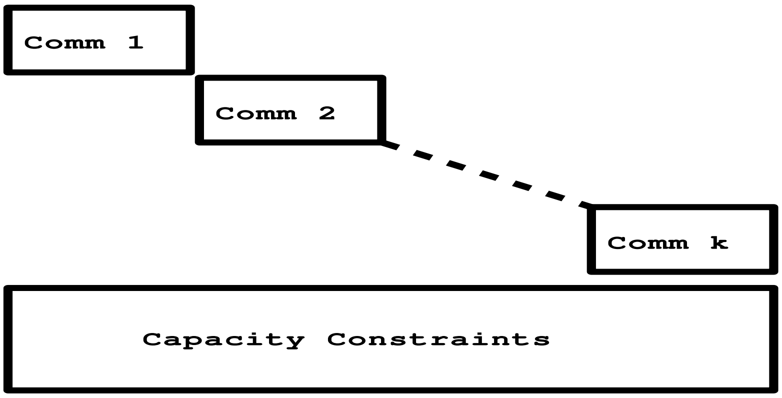

Figure 2.2: MCNF Matrix

matrix object. The software structure of our implementation is outlined in Figure 2.3. The

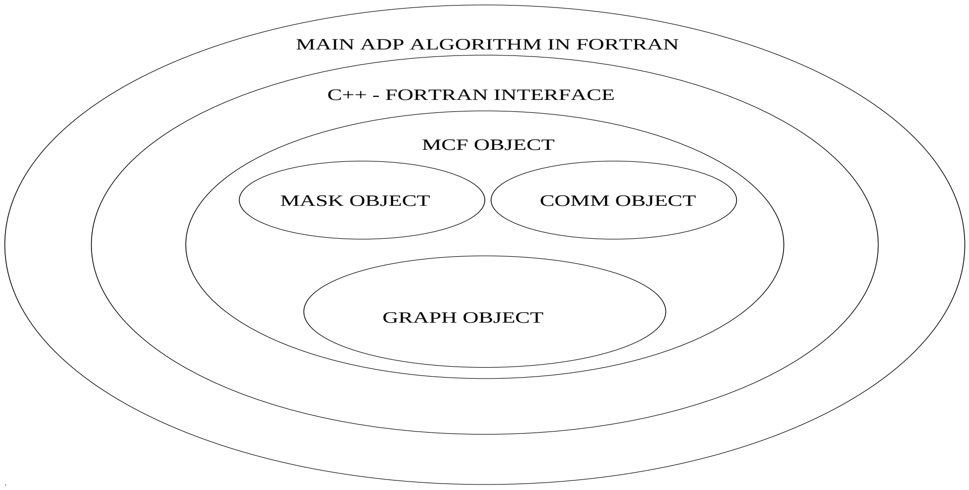

Figure 2.3: Object Oriented Structure of the Software

outermost layer in the figure represents the main ADP algorithm. The implementation of the main algorithm is still in FORTRAN-77 but it is matrix-transparent i.e., it has no knowledge of how and where the constraint matrix is stored. The interface to the C++ object is through the C++-Fortran interface as shown in the middle layer. The kernel of the software is the MCF-matrix object.

This object has three subclasses, namely, the network object, the mask object and the commodities object. A pseudo C++ class definition is given below.

```
class MCF_matrix{
        network graph;
        mask maskobj;
        int commodities;
        public:
        vector ax(const vector x);
        vector atx(const vector x);
        vector adatx(const vector d,const vector x);
        matrix adat(const vector d);
        ⋮
        };
```

The subclass *network* encodes the incidence matrix of the graph. The *mask* subclass is used for computing preconditioners. The information about commodities is stored in the scalar *commodities*. The publicly available member functions *ax* , *atx* , *adatx* are called by the C++-Fortran interface routines and return a vector object; for example, *ax* takes a vector x and multiplies it with the expanded MCF constraint matrix and returns the resulting vector. The function *adat* returns a matrix object. Additional member functions perform other related operations.

The motivation of this object-oriented implementation was to achieve compact storage, ease of vectorization, reduction in data access, and data transparency. Since we store only a single copy of the incidence matrix, the storage required for representing the constraint matrix is reduced. The reduction in storage can be substantial for problems with large number of commodities. Each edge in the network subclass of our object representation corresponds to a vector of length $2k$ in the expanded constraint matrix, where k is the number of commodities. Thus, all matrix operations are readily vectorizable and the number of data accesses required to fetch the constraint matrix is reduced.. This data-hiding principle of object-oriented programming is central to our implementation. The main ADP algorithm is oblivious to the matrix representation and hence the same software can be reused for efficient implementation of other LPs by changing the object definition. For instance, by doing minor modifications to the object. we have reused the ADP software to solve concurrent flow problem (CFP) which is discussed below.

2.6 Concurrent Flow Problem

Consider a directed graph $G = (V, E)$. Given any multi-commodity flow f which satisfies conservation and demand constraints but not necessarily the capacity constraints, we can define for each edge $e \in E$ a congestion factor $\lambda_f(e) = f(e)/u(e)$ which is the ratio of total flow to capacity on that edge. The congestion λ_f for the flow is defined to be the maximum of these edge congestions. If the capacities are scaled by λ_f, then f becomes a feasible multi-commodity flow. The concurrent flow problem is to find the lowest possible congestion factor λ^*. The LP formulation of the problem is stated as follows.

$$f_i(e) \geq 0 \quad \forall e \in \mathcal{E}\,,\, i \in \mathcal{K} \tag{2.23}$$

$$\sum_{i \in \mathcal{K}} f_i(e) - \lambda u(e) \leq 0 \ \forall e \in \mathcal{E} \tag{2.24}$$

$$\sum_{wv \in \mathcal{E}} f_i(e)(wv) - \sum_{vw \in \mathcal{E}} f_i(e)(vw) = \begin{cases} 0 & v \neq s_i \text{ and } v \neq t_i \\ -d_i & v = s_i \\ d_i & v = t_i \end{cases} \quad \forall v \in \mathcal{V}; \forall i \in \mathcal{K} \quad . \tag{2.25}$$

The objective is to minimize the congestion factor λ.

CFP was first proposed by Shahrokhi and Matula [70]. They consider the special case of unit capacities and demands. Leighton et al., [49] later proposed an algorithm for solving the general CFP with arbitrary capacities and demands. This algorithm is an approximation algorithm which uses combinatorial techniques for finding an $\epsilon - approximate$ solution. Computational results for an implementation of this algorithm were reported in [51].

2.7 Computational Experience

In this section, we describe the computational results for MCNF using our object oriented code for ADP and compare it with CPLEX-NETOPT [17]. All runs were carried out on a *Silicon Graphics Machine*, having a Risc-based architecture with 100MHZ clock. The ADP algorithm is implemented in FORTRAN and C++. The computational experiments were conducted on instances generated by two different generators. The first set of instances were generated using the *rmfgen* generator, described in [28]. An instance of the problem generated by *rmfgen*, denoted by $a.b.k$ has b layers of grids of size $a \times a$, and k commodities to be routed. In the instances we generated, the costs were generated randomly between 1 and 100 and the capacities on the grid edges ranged from 1 to 100. The demands varied randomly between 1 to 1000. The MCNF problems generated by *rmfgen* seem to be simple: we observed in our experiments that the sum of the optimal solutions to each of the single commodity flow problems was close to the optimal solution of the MCNF problem.

Problem	Rows in A	Columns in A	ADP Time	CPLEX-NETOPT Time
16.8.4	17664	47360	283s	685s
16.8.8	25856	85248	505s	1253s
16.8.16	42240	161024	1441s	2300s
16.16.4	35584	96000	313s	2933s
16.16.8	51968	172800	1620s	5582s
16.16.16	84736	326400	6363s	11000s
16.32.4	71424	193280	1580s	12807s
16.32.8	104192	347904	10596s	23553s
16.32.16	169728	657152	19834s	44001s

Table 2.1: Performance of ADP on *rmfgen* problems

To generate difficult problems, we developed the *mgrid* generator. This generator creates a square 2-dimensional grid graph. Each edge in the grid is bidirectional and has a capacity associated with it in each direction. For each commodity we randomly choose a source on the boundary of the grid and the corresponding sink is chosen randomly on the opposite boundary. The demand for each commodity is generated independently at random. We solve a concurrent flow problem (CFP), in which we determine the maximum multiple μ of the demand vector that one can satisfy subject to the capacity constraints. The solution, μ, to the CFP problem is then used to scale the demands, thereby creating a feasible MCNF problem. We then randomly assign costs per unit flow for each commodity in each edge and solve the resulting MCNF problem. For the instances that we solved, the cost range was 1 to 1000 and the capacities ranged from 1 to 1000.

The performance of our ADP method and a comparison with the corresponding CPLEX-NETOPT times, for instances generated by *rmfgen* and *mgrid* are given in tables 2.1 , 2.3 respectively. The computational results for solving the CFP problems that arise in the

Grid Size	Comm.	Rows	Columns	ADP Time	CPLEX NETOPT
20	5	3520	9121	82s	515s
50	5	22300	58801	878s	2221s
100	5	89600	237601	8818s	$>$ 594533s

Table 2.2: Performance of ADP on the concurrent flow problem

mgrid generator are reported in table 2.2 . All times reported are user times in seconds. Asymptotically, ADP performs better than the CPLEX-NETOPT.

Grid Size	Comm.	Rows	Columns	ADP Time	CPLEX NETOPT
20	5	3520	9120	147s	408s
50	5	22300	58800	1740s	67956s
100	5	89600	237600	13102s	$>$ 556803s

Table 2.3: Performance of ADP on *mgrid* problems

The results show how the interior-point method for linear programming can be effectively used to efficiently solve large instances of multi-commodity flow problems.

Chapter 3

A Quadratic Programming approach to Multi-Commodity Flow

3.1 Introduction

In this chapter [1], we present a new approach to the multi-commodity flow problem that is also based on the interior point method. Traditional interior-point based algorithms for this problem like the one described in Chapter 2 solve a linear programming (LP) formulation of the problem. The LP formulation has two types of constraints, one to enforce conservation and the second for capacity. The conservation constraints are independent over commodities and the capacity constraints over edges. In this chapter, we try to use the separability of the constraints to design an efficient algorithm for the problem. We retain the relatively simpler capacity constraints in the linear form, but define a quadratic objective function to enforce flow conservation. The resulting quadratic programming problem (QP) can then be solved efficiently using interior-point techniques. For a multi-commodity flow problem that has n nodes, m edges and k commodities, the new approach has an improved complexity when $m < n^{0.9}k^{0.9}$. Moreover since in our formulation the region of optimization is defined by simple separable linear conditions, we can find a good starting point to the algorithm and hence improve the running time by a polynomial factor whenever we need an approximate

[1] This chapter represents joint work with Omri Palmon [34].

our result	exact solution[a]	ϵ approximation[a]
V-89 [77] FMM[b]	$k^{2.5}m^{0.5}n^2L^c$	same
V-89 [77] w.o. FMM	$k^3m^{0.5}n^{2.5}L^c$	same
M-92 [62]	$k^{2.5}m^{1.5}nL^c$	same
R-95 [68] [d]		$kmn\epsilon^{-2}$
PST-91 [67] (rand.)		$km^2\epsilon^{-2}$
PST-91 [67]		$k^2m^2\epsilon^{-2}$
KPP-94 [36]		$kmn^2\epsilon^{-6}$
Our result		
Variant 1	$k^{2.5}m^{1.5}nL^e$	$k^{2.5}m^{1.5}n\log\epsilon^{-1}$
Variant 2 FMM	$(k^{0.5}m^{2.7}+km^{1.5}n^{1.2}+km^{2.5})L^e$	$(k^{0.5}m^{2.7}+km^{1.5}n^{1.2}+km^{2.5})\log\epsilon^{-1}$
Variant 2 w.o. FMM	$(k^{0.5}m^3+km^{1.5}n^{1.5})L^e$	$(k^{0.5}m^3+km^{1.5}n^{1.5})\log\epsilon^{-1}$

[a]These are asymptotic complexities without log factors
[b]Fast Matrix Multiplication
[c]L not specified.
[d]without costs.
[e]$L = m\log m + \log DU$.

Figure 3.1: Comparison of Complexities.

solution to the problem.

We present two variants of our algorithm. Both variants compute an exact solution in $O(k^{0.5}m^{0.5}L)$ interior point iterations, where $L = m\log m + \log(DU)$, D is the biggest demand and U the biggest capacity . The main step in each interior-point iteration involves solving a linear system of equations. The first variant computes a single iteration in $O(k^2mn)$ time and the second requires $O(m^{2.5} + k^{0.5}n^{1.5}m)$ time. Using fast matrix multiplication the complexity of computing a single iteration in the second variant can be improved to $O(m^{2.2} + k^{0.5}m^2 + k^{0.5}mn^{1.2} + kn^2)$. The fastest previously known multi-commodity flow algorithm was due to Vaidya [39, 77], and achieves a complexity bound of $O(k^{2.5}m^{0.5}n^2L\log(nDU))$, with fast matrix multiplication techniques and $O(k^3m^{0.5}n^3L\log(nDU))$ without fast matrix multiplication. The complexities per iteration are $O(n^2k^2)$ when using fast matrix multiplication, and $O(n^{2.5}k^{2.5})$ when not. Even though fast matrix multiplication has better asymptotic bounds it is considered to be impractical for any reasonable sized problems. Hence we report the complexities with and without fast matrix multiplications. Our algorithm improves on the previously best known complexity bound when $m < n^{0.9}k^{0.9}$. For example, when the multi-commodity flow problem has $k = \Theta(n)$ commodities and $m = \Theta(n)$ edges our algorithm is faster than the best previously known algorithm by a factor of $\Theta(n)$. The same results apply to the minimum cost multi-commodity flow and the concurrent flow problems.

Most practical applications of the multi-commodity flow problem do not need the exact solution but only need to solve it up to a finite precision. When this precision is small, relative to the problem size, our algorithm is better than previous algorithms by a polynomial factor, in all cases. The improvement in running time is possible because we can compute a good starting point for our algorithm. In previous formulations of multi-commodity flow as an LP problem the polytope is defined by both capacity and conservation constraints. Because of the way these constraints interleave it does not seem possible to compute a good starting point for these algorithms. Thus their asymptotic complexities for computing both approximate and exact solutions are the same. On the other hand, the region of optimization in our formulation is defined by only simple separable capacity constraints. Hence it is easier to compute a good starting point from which we can converge faster to an approximate solution. To get the complexities of our algorithms for computing ϵ-approximate solutions, one can replace L (which is at least nk) by $\log nDU\epsilon^{-1}$. For example, in the case described above ($k = \theta(n)$ and $m = \theta(n)$), our complexity is an improvement by a factor of $\theta(n^3/\log \epsilon^{-1})$ over previously known results. Moreover, previously known interior point algorithms for multi-commodity flow [39] required to work with $O(L)$ bits. If b bits of precision are needed, then our algorithm needs to work with $O\left(b + \log(nDU)\right)$ bits of precision.

The algorithm extends to the generalized flow problem and the generalized multi-commodity flow problem. Our complexity bound for the generalized multi-commodity flow is the same (up to log factors) as for the multi-commodity flow problem. For the single commodity generalized flow problem, our running time is $O\left(m^{1.5}n^2\log(nUD\gamma)\right)$ (where γ is the biggest ratio between two generalized flow coefficients) which matches the best previously known result in [77] and [62].

Section 3.2 will describe the formulation of the multi-commodity flow as a QP problem. Section 3.3 will describe the algorithm. Section 3.4 will describe methods to efficiently solve the linear system that arises in the formulation of the problem. Section 3.5 will describe how to obtain the starting point to the algorithm. Section 3.6 will prove the numerical stability of the algorithm. Section 3.7 will show how an exact solution is obtained. Section 3.8 will describe how to extend the algorithm to solve the minimum cost, concurrent flow and generalized multi-commodity flow problems.

3.2 Multi-commodity flow as QP

An instance of the *multi-commodity flow problem* consists of a directed graph $G = (V, E)$, a non-negative capacity $u(e)$ for every edge $e \in E$, and a specification of k commodities, numbered 1 through k, where the specification for commodity i consists of a source-sink pair $s_i, t_i \in V$ and a non-negative demand d_i. We will denote the number of nodes by n, and the number of edges by m. We assume that $m \geq n$, and that the graph G is connected and has no parallel edges. Let U be the biggest capacity and D the biggest demand (all capacities and demands are greater than 1).

In this section, we shall formulate multi-commodity flow as a QP problem. The problem will have $m(k+1)$ variables. Define the index set $K \stackrel{def}{=} \{1, \ldots, k, s\}$. Let x_e^l be the flow of commodity $l = 1, \ldots, k$ on edge e and let x_e^s be the unused capacity (slack variable) for edge e. A feasible multi-commodity flow solution has to obey conservation and capacity constraints.

The capacity constraints are written as linear equations. Define A as an $m(k+1) \times m$ matrix constructed from $k+1$ blocks of identity matrices of dimension $m \times m$:

$$A \quad \stackrel{def}{=} \quad (I_{m\times m} \;\; I_{m\times m} \quad \cdots I_{m\times m}) \,.$$

Let $\bar{u} \in \mathbb{R}^E$ be the vector of capacities. The capacity constraints can then be written as

$$Ax = \bar{u} \qquad x \geq 0.$$

The conservation constraints are enforced through the objective function. We will define a convex quadratic objective function (positive semi-definite form) which has value 0 if and only if the flow variables obey conservation. Denote the vector of flow variables for the lth commodity by $x^l \in \mathbb{R}^E$. For every $v \in V$ define the excess of the lth commodity at vertex v to be

$$Ex_v(x^l) = \sum_{w:wv \in E} x_{wv}^l - \sum_{w:vw \in E} x_{vw}^l .$$

The conservation constraints are enforced by computing the minimum of the quadratic function

$$E(x) \stackrel{def}{=} \sum_{v \in V, l \leq k} \left(Ex_v(x^l) - h^l(v)\right)^2 ,$$

where $h^l(s_l) = d_l$, $h^l(t_l) = -d_l$ and $h^l(v) = 0$ otherwise. To write the matrix formalization, we will define B, an $n \times m$ matrix that calculates the excesses of the flow as

$$B_{u,e=vw} = \begin{cases} -1 & u = v \\ 1 & u = w \\ 0 & \text{otherwise.} \end{cases}$$

Now define Q as an $m(k+1) \times m(k+1)$ matrix composed of $(k+1) \times (k+1)$ blocks of size $m \times m$ each:

$$Q \stackrel{def}{=} \begin{pmatrix} B^T B & 0 & \cdots & 0 & 0 \\ 0 & B^T B & & \vdots & \vdots \\ \vdots & & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & B^T B & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix}$$

Define $c^l_{vw} = h_l(v) - h_l(w)$ for $l \neq s$ and $c^s_{vw} = 0$, then the corresponding QP problem can be written as

$$\min \frac{1}{2} E(x) = c^T x + \frac{1}{2} x^T Q x \qquad s.t.\ Ax = \bar{u} \ \ x \geq 0.$$

A solution to the QP problem that minimizes $E(x)$ subject to the capacity constraints gives a solution to the multi-commodity flow problem. We shall use a primal-dual algorithm to solve this QP problem. The algorithm introduces dual variables, y_e corresponding to the capacity constraints for every edge e and the dual slack variables z^l_e for every edge e and $l \in K$. The primal-dual feasible polytope W is defined as

$$W = \left\{ (x, y, z) \,\middle|\, \begin{array}{c} x \geq 0 \ , \ \ z \geq 0 \ , \ \ Ax = \bar{u} \ , \\ -Qx + A^T y + z - c = 0 \end{array} \right\}.$$

3.3 The Algorithm

In this section, we show how to solve the resulting QP problem using interior point techniques. The interior point method, is an iterative method which computes a sequence of points in the interior of the polytope defined by the constraints. The value of the objective function will be improved in each iteration. The algorithm described in this chapter is the same as in [61] but has been adapted to exploit the underlying structure of the multi-commodity flow problem.

In the feasible set W, we will define the central path $\rho(\mu)$ for every $\mu > 0$ as the point that minimizes

$$E(x) - \mu \sum_{l \in K e \in E} \log x_e^l$$

subject to the capacity constraints. As μ tends to zero, $\rho(\mu)$ converges to the optimal solution. The point $\rho(\mu)$ is completely characterized by the following Karush-Kuhn-Tucker conditions:

$$\begin{array}{ll} \forall e \in E \;\; \forall l \in K & x_e^l > 0 \quad z_e^l > 0 \\ \forall e \in E \;\; \forall l \in K & x_e^l z_e^l = \mu \\ & Ax = \bar{u} \\ & -Qx + A^T y + z - c = 0. \end{array}$$

The algorithm constructs a series of approximations to the central path until the current point is sufficiently close to the optimal solution. At each step, given an approximation to a point on the central path $\rho(\mu)$ we will compute an approximation to $\rho(\hat{\mu})$ where $\hat{\mu} = (1 - \delta/\sqrt{mk})\mu$ (δ is a constant). The approximation will be calculated using the Newton direction associated with the Karush-Kuhn-Tucker conditions. The Newton direction $(\Delta x, \Delta y, \Delta z)$ will be approximately computed from the following equations

$$\begin{array}{lrcl} \forall e \; \forall l & \tilde{z}_e^l \Delta x_e^l + \tilde{x}_e^l \Delta z_e^l & = & x_e^l z_e^l - \hat{\mu} \\ & A \Delta x & = & 0 \\ & -Q\Delta x + A^T \Delta y + \Delta z & = & 0 \end{array} \tag{3.1}$$

where $\tilde{x}$ and $\tilde{z}$ are approximations to x and z respectively.

Now we can state the algorithm (compare to algorithm 3.1 in [61]):

Monteiro-Adler Algorithm for QP

Initialize Set $j = 0$. Compute a starting point (x^0, y^0, z^0) and the corresponding parameter μ^0.

Compute while $\mu^j > 0.9\epsilon^2(m(k+1)n)^{-1}$ do

1. Set $\mu^{j+1} = (1 - \delta/\sqrt{mk})\mu^j$.

2. Compute $(\Delta x, \Delta y, \Delta z)$ solving equations (3.1).
3. Set
$$(x^{j+1}, y^{j+1}, z^{j+1}) = (x^j, y^j, z^j) - (\Delta x, \Delta y, \Delta z)$$
and set $j = j + 1$.

Round If $E(x) \leq 0.9\epsilon^2(m(k+1)n)^{-1}$ round the solution, otherwise the problem has no feasible solution.

A good starting point is important for proving good complexity bounds in the approximate case and one such point is presented in section 3.5. The Newton direction is efficiently computed by solving a system of linear equations using one of the procedures described in Section 3.4. The rounding procedure is outlined in theorem 3.3.2.

In order to state the theorems that prove the convergence result for this algorithm, we need the following definitions:

Definition 3.3.1 *A point $w = (x, y, z) \in W$ will be called a θ approximation to the central path $\rho(\cdot)$ at μ if*
$$\sum_{e \in E, l \in K} (x_e^l z_e^l - \mu)^2 \leq \theta^2 \mu^2.$$

Note that the definition of the central path implies that $\rho(\mu)$ is the only point that satisfies $x_e^l z_e^l = \mu$ for all e and l.

Definition 3.3.2 *A vector $\tilde{v}$ will be called a γ approximation to a vector v if for each i we have*
$$\frac{|\tilde{v}_i - v_i|}{\tilde{v}_i} \leq \gamma.$$

We will also denote by $\|v\|_u$ the Euclidean norm of the vector v normalized by the vector u, i.e.
$$\|v\|_u \stackrel{def}{=} \left(\sum_i \left(\frac{v_i}{u_i} \right)^2 \right)^{1/2}.$$

Using this notation, we can now state the following main theorem that proves that our algorithm can compute approximations to the central path $\rho(\mu)$ for decreasing values of μ.

Theorem 3.3.1 *Let $\theta = \delta = \gamma = 0.1$. Let $w = (x, y, z) \in W$ and $\mu > 0$ such that w is a θ approximation to the central path at μ. Assume that $\tilde{x}$ and $\tilde{z}$ are γ approximations to x and z. Let $\hat{\mu} = \mu(1 - \delta/\sqrt{mk})$. Consider the point $\hat{w} = w - \Delta w$ where $\Delta w = (\Delta x, \Delta y, \Delta z)$ solves equations (3.1). Then we have*

1. $\hat{w} \in W$
2. $\hat{w}$ *is a* θ *approximation to the central path at* $\hat{\mu}$
3. $\|\hat{x} - x\|_x \leq 0.28$ *and* $\|\hat{z} - z\|_z \leq 0.28$.

Proof: Theorem 3.3.1 is exactly theorem 4.1 in [61]. ∎

The above theorem implies the convergence of our quadratic optimization method.

The next theorem will provide us with a technique for rounding the solution of the quadratic optimization problem to obtain an ϵ-approximate solution. An ϵ-approximate solution to the multi-commodity flow problem is a flow that obeys capacity constraints and for each commodity l, satisfies at least $(1 - \epsilon)$ fraction of the total demand d_l.

Theorem 3.3.2 *Assume that there is a solution to the multi-commodity flow problem. Let $w = (x, y, z) \in W$ be a θ approximation to the central path at $\mu \leq (1+\theta)^{-1}\epsilon^2(m(k+1)n)^{-1}$. Then we can obtain an ϵ-approximate solution to the multi-commodity flow problem in $O(kmn)$ time.*

Proof: Since there is a solution to the multi-commodity flow problem the optimal value of $E(\cdot)$ is 0. Hence current dual value is non-positive. Thus (see section 2 in [61] for theoretical background)

$$E(x) = (\text{primal value}) \leq (\text{primal value}) - (\text{dual value}) = (\text{primal dual gap}) =$$

$$\sum_{e,l} x_e^l z_e^l \leq m(k+1)(1+\theta)\mu \leq \epsilon^2 n^{-1}.$$

Hence for each commodity l, we have

$$\sum_{v \in V} \left(Ex_v(x^l) - h^l(v)\right)^2 \leq \epsilon^2 n^{-1}$$

which implies

$$|Ex_{s_l}(x^l) + d_l| + |Ex_{t_l}(x^l) - d_l| + \sum_{v \in V \setminus \{s_l, t_l\}} |Ex_v(x^l)| \leq \epsilon \leq \epsilon d_l$$

Now solving a single commodity max flow with current flow values as capacities will result in a feasible flow that satisfies at least $(1-\epsilon)$ fraction of the demand for each commodity. ■

We shall now state the main complexity result for our algorithm. The result follows from various theorems that we prove in the following sections.

Theorem 3.3.3 *We can find an ϵ approximation to the multi-commodity flow problem (or determine that no solution exists) in $O(m^{0.5}k^{0.5}\log(n\epsilon^{-1}DU))$ iterations. Each iteration can be done in either $O(k^2mn)$ or $O(m^{2.5}+k^{0.5}mn^{1.5})$ time. The complexity of each iteration can be improved to $O(m^{2.2}+k^{0.5}m^1n^{1.2}+k^{0.5}m^2+kn^2)$ using fast matrix multiplication.*

Proof: Using theorem 3.3.2 and theorem 3.5.1 that is proved in section 3.5, we can bound the number of iterations by $O(m^{0.5}k^{0.5}\log(n\epsilon^{-1}DU))$. From theorems 3.4.5 and 3.4.9, that are proved in the next section, the overall complexity of the algorithm can be estimated. ■

3.4 Analysis of number of operations per iterations

The dominant factor in the complexity of interior-point algorithms comes from the computation of the Newton step which involves solving the linear system defined in (3.1). In this section, we will describe two ways for using the underlying structure of the problem to efficiently compute the Newton step. In subsection 3.4.1, we will outline a procedure to solve the equations using conjugate gradient [29]. In subsection 3.4.2, we will outline a procedure to solve the equations using matrix inversion and rank one updates.

Denote by $X, Z, \tilde{X}, \tilde{Z}$ the diagonal matrices with $x, z, \tilde{x}, \tilde{z}$ on the diagonal respectively. Denote by $\bar{1}$ the vector of ones. A symbol with superscript l will denote the part associated with commodity l (or slack). Using the temporary variables $v^l \in \mathbb{R}^E$ and D^l a diagonal $m \times m$ matrix for every $l \in K$, a generic procedure that uses the matrix structure for solving the linear system (3.1) is outlined below. (compare to the equations on the bottom of page 47 in [61])

GENERIC PROCEDURE:

1. Compute the temporary variables v^l and D^l for every $l \in K$

$$D^l = \tilde{Z}^l \tilde{X}^{l^{-1}}$$

$$v^l = (X^l Z^l \bar{1} - \hat{\mu} I_{m(k+1)\times m(k+1)})\tilde{X}^{l^{-1}}\bar{1}.$$

2. Using D^l and v^l we compute Δy such that

$$\left((D^s)^{-1} + \sum_{l\in K\setminus\{s\}} \left(D^l + B^T B\right)^{-1} \right) \Delta y \;\; = \;\; -\left((D^s)^{-1} v^s + \sum_{l\in K\setminus\{s\}} \left(D^l + B^T B\right)^{-1} v^l \right).$$

3. From Δy, v^l and D^l we compute Δx and Δz

$$\forall l \in K\setminus\{s\} \;\; \Delta x^l = \left(D^l + B^T B\right)^{-1} (\Delta y + v^l)$$

$$\Delta x^s = (D^s)^{-1} (\Delta y + v^s)$$

$$\forall l \in K \qquad \Delta z^l = v^l - D^l x^l.$$

The following claim will be used extensively in the rest of the section.

Claim 3.4.1 *Given vectors $x \in \mathbb{R}^E$ and $y \in \mathbb{R}^V$ we can calculate Bx and $B^T y$ in $O(m)$ operations.*

Proof: The proof follows immediately from the sparseness of B. ■

3.4.1 Computing the Newton step using conjugate gradient.

The dominant factor in the complexity of Generic Procedure is in computing Δy in Step 2. It involves solving the linear system,

$$\left(\sum_{l\in K\setminus\{s\}} \left(D^l + B^T B\right)^{-1} + (D^s)^{-1} \right) \Delta y = b$$

where

$$b = -\left((D^s)^{-1} v^s + \sum_{l\in K\setminus\{s\}} \left(D^l + B^T B\right)^{-1} v^l \right).$$

The problem is equivalent to finding $a^l, l \in K$ such that

$$a^1 + \cdots + a^k + a^s = b$$

$$\forall l \in K \setminus \{s\} \quad \left(D^l + B^T B\right) a^l = D^s a^s = \Delta y.$$

We will find these a^l's using conjugate gradient method. For a description of the conjugate gradient method see pages 516–526 in [29].

The procedure for solving the equation is the following:

VARIANT-1:

1. Compute v^l and D^l for all $l \in K$ in the way they are defined. Compute b, using conjugate gradient to compute each of the terms $\left(D^l + B^T B\right)^{-1} v^l$.

2. Define

$$\tilde{D} = \begin{pmatrix} I & I & \cdots & I \\ -D^s & D^1 & & \\ \vdots & & \ddots & \\ -D^s & & & D^k \end{pmatrix}$$

$$\tilde{B} = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & B^T B & & \\ \vdots & & \ddots & \\ 0 & & & B^T B \end{pmatrix}$$

   Compute $b' \in \mathbb{R}^{m(k+1)}$ as

$$b' = \left(I + \tilde{D}^{-1}\tilde{B}\right)^T \tilde{D}^{-1} \begin{pmatrix} b^T\, 0 \cdots 0 \end{pmatrix}^T.$$

   where I is $I_{m(k+1)\times m(k+1)}$.

3. Compute $\bar{a} = (a^s, a^1, \ldots, a^k)$ that solves the following equation using conjugate gradient

$$\left(I + \tilde{D}^{-1}\tilde{B}\right)^T \left(I + \tilde{D}^{-1}\tilde{B}\right) \bar{a} = b'.$$

   where I is $I_{m(k+1)\times m(k+1)}$.

4. Compute $\Delta y = D^s a^s$.

5. Compute Δx^l and Δz^l, using conjugate gradient to compute Δx^l.

To estimate the complexity of the procedure we need the following claims.

Claim 3.4.2 *Given $y \in \mathbb{R}^E$ and D, a diagonal $m \times m$ matrix, $(D + B^T B)^{-1} y$ can be computed in $O(mn)$ operations.*

Proof: The solution x should satisfy

$$\left(I_{m\times m} + D^{-\frac{1}{2}} B^T B D^{-\frac{1}{2}}\right)\left(D^{\frac{1}{2}} x\right) = D^{-\frac{1}{2}} y.$$

The matrix $I_{m\times m} + D^{-\frac{1}{2}} B^T B D^{-\frac{1}{2}}$ is self adjoint and positive definite, hence we can use the conjugate gradient method. Since $rank\, B = n$, the conjugate gradient method will converge in n iterations. It follows from claim 3.4.1 that each iteration (i.e. multiplying a vector by the matrix) can be done in $O(m)$ operations. ■

Claim 3.4.3 *Given $x \in \mathbb{R}^{m(k+1)}$ we can compute $\tilde{D}^{-1}x$ and $\tilde{B}x$ in $O(km)$ operations.*

Proof: We can look at the matrix $\tilde{D}$ as a $(k+1) \times (k+1)$ matrix where each entry is a $m \times m$ diagonal matrix. As a $(k+1) \times (k+1)$ matrix, computing $\tilde{D}^{-1}x$ takes only $O(k)$ "operations" (this is a diagonal matrix with two rank one updates) where each "operation" involves multiplying an m vector by a diagonal $m \times m$ matrix. Hence computing $\tilde{D}^{-1}x$ takes $O(km)$ time.

The complexity of computing $\tilde{B}x$ follows directly from claim 3.4.1. ■

Claim 3.4.4 *Given $y \in \mathbb{R}^{m(k+1)}$ using conjugate gradient to find the x such that*

$$\left(I_{m(k+1)\times m(k+1)} + \tilde{D}^{-1}\tilde{B}\right)^T \left(I_{m(k+1)\times m(k+1)} + \tilde{D}^{-1}\tilde{B}\right) x = y$$

will converge in $O(kn)$ iterations.

Proof: Since the matrix is self adjoint and positive definite the conjugate gradient method will find a solution. To estimate the number of iterations observe that the matrix can be written as the identity matrix plus three rank n matrices. Hence the claim follows. ■

Theorem 3.4.5 *The complexity of the procedure* VARIANT-1 *is $O(k^2 mn)$.*

Proof: The bottleneck for computing one iteration is step 3 which can be computed in $O(k^2mn)$ time, because it is involves $O(nk)$ iterations of conjugate gradient, each with complexity of $O(mk)$.

Step 1 can be done in $O(kmn)$ time by claim 3.4.2. Step 2 can be done in $O(km)$ time by claim 3.4.3. Step 5 can be done in $O(kmn)$ time by claim 3.4.2. ∎

3.4.2 Computing the Newton step using matrix inversion and rank one updates.

In the following procedure, we will show how to efficiently solve the equations using matrix inversions. We will use a standard technique first proposed in [40] to maintain approximate inverses of the matrices involved. The approximation to the inverse matrix is updated using rank one update, when an entry in x or z changes significantly. The convergence of the algorithm is still assured since in theorem 3.3.1, $\tilde{x}$ and $\tilde{z}$ need only be approximations to x and z.

We will use C^l and C to denote

$$\forall l \in K \setminus \{s\} \quad C^l \stackrel{def}{=} \left(I_{n\times n} + BD^lB^T\right)^{-1}$$

$$C \stackrel{def}{=} \left((D^s)^{-1} + \sum_{l\in K\setminus\{s\}} \left(D^l + B^TB\right)^{-1} \right)^{-1}.$$

The matrices C^l, C and D^l will be updated after each update in $\tilde{x}$ and $\tilde{z}$ rather than being computed in each iteration (compare step 5 to section 5 in [61]). Let r denote a constant integer that will be set later to minimize the complexity, i be the iteration counter and let $\gamma = 0.1$. The procedure is the following:

VARIANT-2:

1. If $i \bmod r = 1$ do the following:

   (a) Set $\tilde{x} = x$ and $\tilde{z} = z$. Compute D^l.

   (b) Calculate explicitly for every $l \in K \setminus \{s\}$

   $$C^l = \left(I_{n\times n} + BD^{l^{-1}}B^T\right)^{-1}$$

   using standard matrix inversion techniques and

   $$\left(D^l + B^TB\right)^{-1} = D^{l^{-1}} - D^{l^{-1}}B^TC^lBD^{l^{-1}}$$

using the Sherman-Morrison-Woodbury formula.

(c) Calculate explicitly using results from previous step and standard matrix inversion techniques

$$C \stackrel{def}{=} \left((D^s)^{-1} + \sum_{l \in K \setminus \{s\}} \left(D^l + B^T B\right)^{-1} \right)^{-1}.$$

2. Compute v^l for every $l \in K$ in the way they are defined.

3. Using the Sherman-Morrison-Woodbury formula compute for $l \in K \setminus \{s\}$

$$\left(D^l + B^T B\right)^{-1} v^l = D^{l^{-1}} v^l - D^{l^{-1}} B^T C^l B D^{l^{-1}} v^l.$$

4. Using the results of the previous step compute

$$\Delta y = -C \left((D^s)^{-1} v^s + \sum_{l \in K \setminus \{s\}} \left(D^l + B^T B\right)^{-1} v^l \right).$$

Compute Δx^l and Δz^l as in Step 3 of the Generic Procedure. Use the Sherman-Morrison-Woodbury formula as in step 3 when needed.

5. Set $x = x - \Delta x$ and $z = z - \Delta z$. For every $e \in E$ and $l \in K$ if either $\left|x_e^l - \tilde{x}_e^l\right| / \tilde{x}_e^l > \gamma$ or $\left|z_e^l - \tilde{z}_e^l\right| / \tilde{z}_e^l > \gamma$ set $\tilde{x}_e^l = x_e^l$ and $\tilde{z}_e^l = z_e^l$, and update D^l, C^l and C using rank one updates.

Claim 3.4.6 *The complexity of step 1 is $O(kn^3 + km^2 + m^3)$.*

Proof: Because of the structure of B as vertex-edge adjacency matrix each entry of $BD^{l^{-1}}B^T$ can be computed by at most one subtraction depending on whether there is an edge between the two vertices that define the entry. Hence computing each matrix C^l in step 1b can be done in $O(n^3)$ time. Using the same argument, we can prove that each entry of $D^{l^{-1}} B^T C^l B D^{l^{-1}}$ can be computed in constant time using only 4 entries in C^l. All other estimates are standard. ∎

Claim 3.4.7 *Each update in step 5 can be done in $O(m^2)$ time.*

Proof: Since each update in step 5 is a simple rank one update, we can prove the claim using the Sherman-Morrison-Woodbury formula. ■

Claim 3.4.8 *Assuming that $r \leq \sqrt{mk}$, the total number of updates done in step 5 in the r iterations between two applications of step 1 is $O(r^2)$.*

Proof is the same as in [39].

Theorem 3.4.9 *The average complexity of* VARIANT-2 *is $O(m^{2.5} + k^{0.5}mn^{1.5} + kn^2)$.*

Proof: The average complexity of step 1 is $O\left((kn^3 + m^3 + km^2)/r\right)$. The complexity of steps 3 and 4 is $O(m^2 + kn^2)$. The average complexity of step 5 is $O(rm^2)$ (using claims 3.4.7 and 3.4.8). Choosing r to balance these terms we get the result. ■

Theorem 3.4.10 *Assuming usage of fast matrix multiplication procedure that inverts an $n \times n$ matrix in $O(n^{2.4})$ time the average complexity of the procedure* VARIANT-2 *is $O(m^{2.2} + k^{0.5}m^2 + k^{0.5}mn^{1.2} + kn^2)$.*

3.5 Starting Point For The Algorithm

In this section, we provide a good starting point for our algorithm that enables us to prove a good complexity for our approximation algorithm. The idea behind the choice of the point is outlined below. The algorithm can be shown to converge rapidly if we start at a point close to the central path. As μ converges to infinity the central path converges to the center of the polytope (the minimizer of the barrier function). Since our constraints are separable it is very easy to calculate this center by splitting the capacity equally between the commodities. Near the center of the polytope, the value of μ changes rapidly on the central path and we can show a polynomial bound on μ, for which our starting point is a θ-approximation. We can set $w^0 = (x, y, z)$ as follows:

$$x_e^l = x_e^s = x_e = \frac{u(e)}{k+1} \quad \text{denote} \quad \bar{x} = (x_e)_{e \in E}$$

$$y_e = -\frac{\mu}{x_e} + \left[B^T B \bar{x}\right]_e$$

$$z = c + Qx - A^T y$$

Theorem 3.5.1 *We can find μ such that w^0 is a θ approximation to the central path $\rho(\cdot)$ at μ with $\log \mu = O(\log(\theta^{-1} nUD))$.*

Proof: As mentioned above the values of x are chosen to minimize the barrier function (regardless of the objective function). The values of y are chosen such that

$$z_e^l = c_e^l + \left[B^T B\bar{x}\right]_e - y_e = c_e^l + \frac{\mu}{x_e} = c_e^l + \frac{\mu(k+1)}{u(e)} \quad \forall l \in K \setminus \{s\}$$

$$z_e^s = -y_e = \frac{\mu}{x_e} - \left[B^T B\bar{x}\right]_e$$

The values of z are set by the feasibility of the dual problem.

The proof of the theorem follows from the next two claims.

Claim 3.5.2 *If $\mu \geq 2nUD$ then (x, y, z) is primal and dual feasible.*

Proof: The only non-obvious thing to check is $z \geq 0$. Now

$$z_e^l = c_e^l + \frac{\mu(k+1)}{u(e)} \geq -D + \mu\frac{k+1}{U} \geq 0$$

$$z_e^s = \frac{\mu}{x_e} - \left[B^T B\bar{x}\right]_e \geq \frac{\mu(k+1)}{u} - 2n\frac{U}{k+1} \geq 0$$

■

Claim 3.5.3 $\sum_{e\in E, l\in K} (z_e^l x_e^l - \mu)^2 \leq nkD^2U^2 + 4mU^4n^2$

Proof: The distance from the central path can be bounded as

$$\sum_{e\in E, l\in K\setminus\{s\}} (z_e^l x_e^l - \mu)^2 = \sum_{e\in E, l\in K\setminus\{s\}} (c_e^l x_e^l)^2 \leq nkD^2U^2$$

$$\begin{aligned}\sum_{e\in E} (z_e^s x_e^s - \mu)^2 &= \sum_{e\in E} (\mu - x_e \left[B^T B\bar{x}\right]_e - \mu)^2 \\ &\leq m\left(\frac{U}{k+1}\cdot 2n\frac{U}{k+1}\right)^2 \leq 4mU^4n^2.\end{aligned}$$

■

The theorem now follows since we proved that for μ which is polynomial in θ^{-1}, n, d and U the chosen point is a feasible point and a θ approximation to the central path at μ. ■

3.6 Stability

In this section, we will bound the precision needed in performing each arithmetic operation.

Claim 3.6.1 *A vector x is the global optimum solution of*

$$\min c^T x + \frac{1}{2} x^T Q x - \sum_i \mu_i \ln x_i$$

$$s.\ t.\ \ Ax = b \quad x \geq 0$$

if and only if there exist vectors z and y such that the following Karush-Kuhn-Tucker conditions are met:

$$\forall i \quad x_i z_i = \mu_i$$

$$Ax = b \quad x \geq 0$$

$$-Qx + A^T y + z - c = 0 \quad z \geq 0.$$

Proof: Using standard primal-dual arguments. ∎

Claim 3.6.2 *Let $w = (x, y, z)$ be a θ approximation to the central path $\rho(\cdot)$ at μ then $\log z_e^l / x_e^l$ is $O\left(\log((1+\theta)\mu n U)\right)$ and $\log x_e^l / z_e^l$ is $O\left(\log((1+\theta)\mu^{-1} U)\right)$ for every edge e and commodity l.*

Proof: Set

$$\mu_e^l = x_e^l z_e^l \in [(1-\theta)\mu, (1+\theta)\mu]. \tag{3.2}$$

Using claim (3.6.1) we know that x is the minimizer of

$$E(x) - \sum_{e \in E\ l \in K} \mu_e^l \ln x_e^l \tag{3.3}$$

$$\text{s.t. } \forall e \in E \qquad \sum_{l \in K} x_e^l = u(e).$$

For a given $e = vw \in E$ and $l, j \in K$ define

$$\begin{aligned} p(t) \ &\stackrel{def}{=}\ \left(Ex_v(x^l) - h^l(v) - t\right)^2 + \left(Ex_w(x^l) - h^l(w) + t\right)^2 \\ &+\ \left(Ex_v(x^j) - h^j(v) + t\right)^2 + \left(Ex_w(x^j) h^j(w) - t\right)^2 \\ &-\ \mu_e^l \ln(x_e^l + t) - \mu_e^j \ln(x_e^j - t) \end{aligned}$$

From the fact that x minimizes the potential function defined in 3.3 we can deduce

$$\begin{aligned} 0 = \frac{\partial}{\partial t} p(t)\Big|_{t=0} &= 2\left(-Ex_v(x^l) + Ex_w(x^l) + Ex_v(x^j) - Ex_w(x^j)\right) \\ &- \mu_e^l \frac{1}{x_e^l} + \mu_e^j \frac{1}{x_e^j} \end{aligned}$$

Using the bounds in (3.2) we get

$$\begin{aligned} \frac{1}{x_e^l} &\leq \frac{1+\theta}{x_e^j} + \frac{2}{\mu(1+\theta)}\left(\left|Ex_v(x^l)\right| + \left|Ex_w(x^l)\right| + \left|Ex_v(x^j)\right| + \left|Ex_w(x^j)\right|\right) \\ &\leq \frac{1+\theta}{x_e^j} + \frac{2}{\mu(1+\theta)} 4nU \end{aligned}$$

Since for every e there exists $j \in K$ such that $x_e^j \geq u(e)/(k+1) \geq 1/(k+1)$ we get that for every $e \in E$ and $l \in K$

$$\frac{1}{x_e^l} \leq (1+\theta)(k+1) + \frac{2}{\mu(1+\theta)} 4(nU + D).$$

The claim follows now from the facts $(1-\theta)\mu \leq x_e^l z_e^l \leq (1+\theta)\mu$ and $x_e^l \leq U$. ■

Claim 3.6.3 *In each iteration, the logarithms of the condition numbers of the matrices involved are* $O\left(\log(UDn\mu)\right)$.

Proof: Since the matrix $D^l + B^T B$ is symmetric and positive definite we can estimate

$$\kappa_2(D^l + B^T B) = \frac{\max_{\|x\|_2=1} x^T(D^l + B^T B)x}{\min_{\|x\|_2=1} x^T(D^l + B^T B)x} \leq \frac{n + \max_e z_e^l/x_e^l}{\min_e z_e^l/x_e^l}$$

and the claim about this matrix follows from the previous claim. For the matrix

$$\left((D^s)^{-1} + \sum_{l \in K\setminus\{s\}} (D^l + B^T B)^{-1}\right)^{-1}$$

note that it is the sum of symmetric positive definite matrices and hence the claim follows also for this matrix. All the other matrices that appear in the computations can be bounded in the same way. ■

Theorem 3.6.4 *To compute a solution with b bits of precision, the computations need to be done with* $O\left(b + \log(nUD)\right)$ *bits of precision.*

Proof: Using the theorem 3.3.2 we know that if only b bits of precision in the solution are needed we need to continue the algorithm until $\log \mu^{-1}$ becomes $O(b + \log(nUD))$. The theorem now follows from the previous claim. ■

There are two conditions that should be satisfied at each iteration, namely, $Ax = u$ and $z = c + Qx - A^T y$. When working with infinite precision the conditions are maintain because of the definitions of $\Delta w = (\Delta x, \Delta y, \Delta z)$. When working with finite precision in order to avoid accumulation of rounding errors we should calculate the following values in the end of each iteration again:

$$x_e^s = u(e) - \sum_{l \in K \setminus \{s\}} x_e^l$$

$$z = c + Qx - A^T y$$

The variables resulting from these corrections can be shown to obey central path conditions since the only errors are those introduced by rounding.

3.7 Finding an Exact Solution

If we find an ϵ-approximate solution then for sufficiently small ϵ, we can round to an exact solution using the technique described in [79]. The conditions under which the technique is guaranteed to give an exact solution is stated in the following theorem, which is proved in [79].

Theorem 3.7.1 *If there exists an optimum solution to a quadratic programming problem in n variables, in which every non zero variable is at least ϵ then an ϵ^2/n-approximate solution can be rounded to an exact optimum solution in $O(n)$ time.*

To use this result we will prove the following theorem:

Theorem 3.7.2 *Given a multi-commodity flow problem which has a solution, we can find $\epsilon > 0$ such that $\log \epsilon^{-1} = O(m \log m + \log UD)$ and such that there exists a solution in which every non zero flow is at least ϵ.*

Proof: Formulate the multi-commodity flow problem as a linear programming feasibility problem, where a variable is associated with every path between a source and a sink of a commodity. We will denote by x_p the variable corresponding to the flow on the path p. The constraints are:

$$\forall p \quad x_p \geq 0$$

$$\forall i \leq k \qquad \sum_{p:p \text{ from } s_i \text{ to } t_i} x_p = d_i$$

$$\forall e \in E \quad \sum_{p:e \in p} x_p \leq u(e)$$

Using the theory of linear programming, we know that every linear optimization problem has a vertex solution which is defined by a solution of a linear system which is a sub-matrix of the constraints matrix. Using Cramer's rule we can give a lower bound on a non-zero entry in the solution by giving an upper bound on the determinant of a sub-matrix of the constraints matrix.

The constraints matrix is exponential in size, but it is easy to check that all the rows associated with the equalities of the type $x_p \geq 0$ do not contribute to the determinant. Hence every subdeterminant is bounded by the determinant of a $(m+k) \times (m+k)$ matrix with entries in $\{0, 1\}$. Hence the theorem follows. ∎

Note also that for a given flow solution, we can always use the zero vectors for the dual variables (this can be proved either directly from the equations, or intuitively, by noting that the optimal value on the polytope of the objective function is also the global optimal value).

Theorem 3.7.3 *An exact solution to the multi-commodity flow problem can be obtained by computing an ϵ-approximate solution where* $\log \epsilon^{-1} = O(m \log m + \log U + \log D)$.

3.8 Extensions

In this section, we show that our results extend to other variants like minimum cost multi-commodity flow, concurrent flow and generalized multi-commodity flow.

3.8.1 Minimum Cost Multi-commodity Flow.

The minimum cost multi-commodity flow instance is a multi-commodity flow instance with costs $p(e) \geq 0$ on the edges e. The optimum flow is the solution to the multi-commodity flow problem which minimizes the cost function $P(x) = \sum_e p(e) \sum_l x_e^l$. Denote by $\bar{p} \in \mathbb{R}^E$ the vector of costs and let P be the maximum cost. In order to find an ϵ-approximate solution to the problem we will use our algorithm on the following objective function:

$$F_\epsilon(x) \stackrel{def}{=} \frac{PUm}{\epsilon} E(x) + P(x).$$

Theorem 3.8.1 *The results that we obtained for the multi-commodity flow problem (complexity, starting point, stability and exact solution) also hold for the minimum cost multi-commodity flow problem with additional log of the ratio between biggest and smallest cost in the number of iterations.*

To prove the theorem we will state several claims that will show how we can use the results for the multi-commodity case for the case of flow with costs.

Claim 3.8.2 *Assume that p^* is the minimum cost of a solution to the multi-commodity flow problem and F_ϵ^* is the optimum value of F_ϵ. Also, assume that for a given pre-multi-flow x, we have $F_\epsilon(x) \leq F_\epsilon^* + \epsilon$, then*

$$E(x) \leq 2\epsilon$$

$$P(x) \leq p^* + \epsilon.$$

Proof: Let x^* be the minimizer of F_ϵ and $\tilde{x}$ the minimum cost flow, then

$$F_\epsilon(x^*) \leq F_\epsilon(\tilde{x}) = P(\tilde{x}) = p^*.$$

Thus

$$P(x) \leq F_\epsilon(x) \leq F_\epsilon(x^*) + \epsilon \leq p^* + \epsilon.$$

Using the same inequalities

$$\frac{PUm}{\epsilon} E(x) \leq p^* + \epsilon.$$

hence

$$E(x) \leq \epsilon \left(\frac{p^*}{PUm} + \frac{\epsilon}{PUm} \right) \leq 2\epsilon.$$

■

Claim 3.8.3 *The complexity of computing a single iteration for the minimum cost multi-commodity flow problem is the same as that for the feasibility problem.*

Proof: Since the only change from the original problem was changing the linear part of the objective function the algorithm stays exactly the same. The linear part does not affect the computation of a single iteration. ■

Claim 3.8.4 *We can find a starting point $w = (x, y, z)$ and μ such that w is a θ approximation to the central path $\rho(\cdot)$ at μ with $\log \mu = O(\log(\theta^{-1} nUDP\epsilon^{-1}))$.*

Proof: Following the same idea as in section 3.5 we will choose the following starting point for a given μ :

$$x_e^l = x_e^s = x_e = \frac{u(e)}{k+1} \qquad \text{denote} \quad \bar{x} = (x_e)_{e \in E}$$

$$y_e = -\frac{\mu}{x_e} + \frac{PUm}{\epsilon} \left[B^T B\bar{x}\right]_e$$

$$z = \frac{PUm}{\epsilon} c + p + \frac{PUm}{\epsilon} Qx - A^T y$$

Calculating the z's explicitly gives (note that slack variables are not multiplied by the prices):

$$\begin{aligned} z_e^l &= \frac{PUm}{\epsilon} c_e^l + p(e) + \frac{PUm}{\epsilon} \left[B^T B\bar{x}\right]_e - y_e = \frac{PUm}{\epsilon} c_e^l + p(e) + \frac{\mu}{x_e} \\ &= \frac{PUm}{\epsilon} c_e^l + p(e) + \frac{\mu(k+1)}{u(e)} \qquad \forall l \in K \setminus \{s\} \end{aligned}$$

$$z_e^s = -y_e = \frac{\mu}{x_e} - \frac{PUm}{\epsilon} \left[B^T B\bar{x}\right]_e$$

In order to check the feasibility of the starting point we need to check if $z > 0$. The z's can be estimated as (compare to claim 3.5.2)

$$z_e^l = \frac{PUm}{\epsilon} c_e^l + p(e) + \frac{\mu(k+1)}{u(e)} \geq -\frac{DPUm}{\epsilon} + \frac{\mu(k+1)}{U} \qquad \forall l \in K \setminus \{s\}$$

$$z_e^s = \frac{\mu}{x_e} - \frac{PUm}{\epsilon} \left[B^T B\bar{x}\right]_e \geq \frac{\mu(k+1)}{U} - \frac{PUm}{\epsilon} 2n \frac{U}{k+1}.$$

Hence if $\mu \geq \max(\epsilon^{-1} DPU^2 m, U^3 Pm\epsilon^{-1} 2n)$ then (x, y, z) is primal and dual feasible.

Estimating the distance of w from the central path at μ gives (compare to claim 3.5.3)

$$\sum_{e \in E, l \in K \setminus \{s\}} (z_e^l x_e^l - \mu)^2 = \sum_{e \in E, l \in K \setminus \{s\}} \left(x_e^l \left(\frac{PUm}{\epsilon} c_e^l + p(e) \right) \right)^2 \leq 2m^3 U^3 P^2 D^2 \epsilon^{-2}$$

$$\begin{aligned} \sum_{e \in E} (z_e^s x_e^s - \mu)^2 &= \sum_{e \in E} (\mu - x_e \frac{PUm}{\epsilon} \left[B^T B\bar{x}\right]_e - \mu)^2 \\ &\leq m \left(\frac{U}{k+1} \cdot \frac{PUm}{\epsilon} 2n \frac{U}{k+1} \right)^2 \leq 4m^3 U^6 n^2 P^2 \epsilon^{-2} \end{aligned}$$

Hence we can find a feasible starting point which is a θ approximation to the central path at μ where μ is a polynomial in n, U, D, P and ϵ^{-1} . ■

Claim 3.8.5 *The stability results proved for the multi-commodity flow problem hold for the minimum cost multi-commodity flow problem.*

Proof: Following the same proof as in section 3.6 proves the claim. An additional term of $\log P$ should be added to the number of bits to be used. ■

3.8.2 The Concurrent Flow Problem.

A concurrent flow problem involves finding the minimum factor λ such that the demands can be satisfied with capacities scaled to $\lambda u(e)$. In order to find an ϵ-approximate solution to the concurrent flow problem we will define for $\epsilon > 0$ the following objective function

$$G_\epsilon(x,t) \stackrel{def}{=} E(x) - \epsilon t$$

We will find an approximate solution to the problem by minimizing $G_\epsilon(x,t)$ subject to

$$\forall e \in E \quad \sum_{l \in K} x_e^l + tu(e) = u(e) \quad t \geq 0 \quad x_e^l \geq 0.$$

To ensure that a solution exists we will multiply each capacity by kD.

Theorem 3.8.6 *The results that we obtained for the multi-commodity flow problem (complexity, starting point, stability and exact solution) apply to the concurrent multi-commodity flow problem as well.*

To prove the theorem we will state and prove several claims that will show how to use the same methods obtained for the multi-commodity flow problem to the concurrent flow problem.

Claim 3.8.7 *Assume that $\epsilon \leq 1$. Suppose that $\tilde{t}$ is the optimal value of the concurrent flow problem and that G_ϵ^* is the optimal value of G_ϵ subject to the constraints. Assume that for a given pre-flow x and parameter t we have $G_\epsilon(x,t) \leq G_\epsilon^* + \epsilon^2$, then*

$$t \geq \tilde{t} - \epsilon$$

$$E(x) \leq 2\epsilon$$

Proof: Let $(\tilde{x}, \tilde{t})$ be the solution for the concurrent flow problem. Then

$$G_\epsilon^* \leq G_\epsilon(\tilde{x}, \tilde{t}) = -\epsilon\tilde{t}$$

hence

$$-\epsilon t \leq G_\epsilon(x,t) \leq G_\epsilon^* + \epsilon^2 \leq -\epsilon\tilde{t} + \epsilon^2$$

which implies the first inequality. In addition to that

$$E(x) = G_\epsilon(x,t) + \epsilon t \leq G_\epsilon^* + \epsilon^2 + \epsilon t$$

Note that always $G_\epsilon^* \leq 0$, $t \leq 1$, $\epsilon \leq 1$, hence

$$E(x) \leq 2\epsilon$$

■

Remark: Note that since we scaled the capacities by kD to ensure the existence of a solution, an ϵ-approximation to our problem is an ϵkD approximation to the original concurrent flow problem.

Claim 3.8.8 *The asymptotic complexity of one iteration in optimizing G_ϵ is the same as in the multi-commodity flow problem.*

Proof: The algorithms described in section 3.4 are can be seen as solving the following linear system:

$$\begin{pmatrix} Z & 0 & X \\ A & 0 & 0 \\ -Q & A^T & I \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} XZe - \hat{\mu}e \\ 0 \\ 0 \end{pmatrix}$$

In the concurrent-flow problem we have two additional variables t and z_t and the matrix to solve is the following:

$$\begin{pmatrix} Z & 0 & X & 0 & 0 \\ 0 & 0 & 0 & z_t & t \\ A & 0 & 0 & \bar{u} & 0 \\ -Q & A^T & I & 0 & 0 \\ 0 & u^T & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t \\ \Delta z_t \end{pmatrix} = \begin{pmatrix} XZe - \hat{\mu}e \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Hence the linear system for the concurrent flow case is the same as the multi-commodity flow case, except a constant number of rank one updates. Using the Sherman-Morrison formula, it then follows that the complexity of solving the linear system is a constant multiple of that for the multi-commodity flow case. ■

Claim 3.8.9 *We can find a starting point $w = (x, y, z, t)$ and μ such that w is a θ approximation to the central path $\rho(\cdot)$ at μ with $\log \mu = O(\log(\theta^{-1} nUD\epsilon^{-1}))$.*

Proof: As in section 3.5 we will use as a starting point the point that minimizes the barrier function regardless of the objective function. By differentiating the barrier function we get that the starting primal-dual point (x, y, z, t) for a given μ is:

$$t = \frac{1}{m(k+1)+1}$$

$$x_e^l = x_e^s = x_e = (1-t)\frac{u(e)}{k+1} \quad \text{denote} \quad \bar{x} = (x_e)_{e \in E}$$

$$y_e = -\frac{\mu}{x_e} + \left[B^T B\bar{x}\right]_e$$

$$z_e^l = c_e^l + \left[B^T B\bar{x}\right]_e - y_e = c_e^l + \frac{\mu}{x_e} = c_e^l + \frac{\mu(k+1)}{u(e)(1-t)} \qquad \forall l \in K \setminus \{s\}$$

$$z_e^s = -y_e = \frac{\mu}{x_e} - \left[B^T B\bar{x}\right]_e$$

$$\begin{aligned} z_t &= -\epsilon - \sum_{e \in E} u(e) y_e = -\epsilon + \mu \sum_{e \in E} \frac{u(e)}{x_e} - \left[B^T B\bar{x}\right]_e \\ &= -\epsilon + \mu m \frac{k+1}{1-t} - \left[B^T B\bar{x}\right]_e = -\epsilon + \mu(m(k+1)+1) - \left[B^T B\bar{x}\right]_e \end{aligned}$$

To ensure that the point (x, y, z, t) is primal-dual feasible we have to check that $z > 0$. Estimating z can be done by (compare to claim 3.5.2):

$$z_e^l = c_e^l + \frac{\mu(k+1)}{u(e)(1-t)} \geq -D + \frac{\mu(k+1)}{U}$$

$$z_e^s = \frac{\mu}{x_e} - \left[B^T B\bar{x}\right]_e \geq \frac{\mu(k+1)}{U} - 2n\frac{U}{k+1}$$

$$z_t = -\epsilon + \mu(m(k+1)+1) - \left[B^T B\bar{x}\right]_e \geq -1 + \mu(m(k+1)+1) - 2n\frac{U}{k+1}$$

Hence if $\mu \geq \max(UD, 2U^2 n)$ then (x, y, z, t) is primal and dual feasible.

To bound the distance from the central path we can calculate (compare to claim 3.5.3):

$$\sum_{e \in E, l \in K \setminus \{s\}} (z_e^l x_e^l - \mu)^2 = \sum_{e \in E, l \in K \setminus \{s\}} (c_e^l x_e^l)^2 \leq nkD^2U^2$$

$$\begin{aligned}\sum_{e\in E}(z_e^s x_e^s-\mu)^2 &= \sum_{e\in E}(\mu - x_e\left[B^TB\bar{x}\right]_e-\mu)^2\\ &\leq m\left(\frac{U}{k+1}\cdot 2n\frac{U}{k+1}\right)^2\leq 4mU^4n^2\end{aligned}$$

$$\begin{aligned}(tz_t-\mu)^2 &= (-t\epsilon + t\mu(m(k+1)+1)-t\left[B^TB\bar{x}\right]_e-\mu)^2\\ &= (-t\epsilon - t\left[B^TB\bar{x}\right]_e)^2\leq\left(\epsilon+2n\frac{U}{k+1}\right)^2\leq 5n^2U^2\end{aligned}$$

Hence for μ polynomial in n, D, U and θ we know that (x, y, z, t) is a θ approximation to the central path at μ. ∎

Claim 3.8.10 *We can find an approximate solution to the concurrent flow problem with the same asymptotic complexity as in the multi-commodity flow problem.*

Proof: Given $\epsilon > 0$, in order to find an ϵ approximation to the concurrent flow problem we will find the approximate optimal of $G_\epsilon(\cdot)$. Claim 3.8.7 proves that this will give an approximate solution to the concurrent flow problem. Combining claim 3.8.9, theorem 3.3.1 and theorem 3.3.2 (adjusted to G_ϵ instead of the original quadratic objective function) assures us that an ϵ approximate solution will be calculated in $O(\sqrt{mk}\log(nDU\epsilon^{-1}))$ iterations. Claim 3.8.8 assures us that the complexity of each iteration is the same. ∎

Claim 3.8.11 *We can find an exact solution to the concurrent flow problem with the same asymptotic complexity as the multi-commodity flow problem.*

Proof: Following the same technique as in the multi-commodity case, we will show that the number of bits needed to compute an exact solution is $O(m\log m + \log UD)$ which is sufficient to prove the claim.

To bound the number of bits needed to describe an exact solution we will follow the same proof as in section 3.7, adjusted to the concurrent flow problem. We will associate a variable with every path from a source and a sink of a commodity. In addition slack variables $s(e)$, and a variable t will be defined. The linear constraints defined for each edge will be:

$$\forall e\in E,\quad \sum_{p:e\in p} x_p + s(e) + tu(e)\leq u(e)$$

Estimating the solution in the same way, we have to bound the determinant of an $m+k\times m+k$ matrix in which all entries are 0 or 1, except possibly the column containing the capacities. Hence the claim follows. ∎

3.8.3 The Generalized Flow Problem.

For the definition of the generalized flow problem see [25]. In this problem, flows moving on edges have different gains or losses. To solve the generalized multi-commodity flow problem using our algorithm, all we have to change are the entries in B which instead of being 0 and 1, will now depend on the gain/loss factor on the edges. Let γ be the ratio between the biggest and smallest generalized flow coefficients. The next two theorems can be easily verified:

Theorem 3.8.12 *Each iteration of our algorithm, as applied to the generalized multi-commodity flow problem has the same complexity as for the original multi-commodity flow problem. However, for finding the exact solution there is an additional $O(\log\gamma)$ factor in the number of iterations and for finding an approximate solution there is an additional factor of $n\log\gamma$.*

Proof: Since complexity proofs used only the sparseness of B and not the fact that all entries are either 0 or 1, the complexity of a single iteration remains the same in the generalized flow problem. In the estimation of the starting point, μ will depend polynomially on γ, adding a $\log\gamma$ factor to the running time. Also, in converting a pre-flow to a flow, we can gain or lose as much as γ^n factor. Hence, to satisfy $1-\epsilon$ fraction of all the demands, we may need more accurate solutions, giving an additional term of $n\log\gamma$ in the number of iterations. ∎

Theorem 3.8.13 *We can solve the single commodity generalized flow problem without using fast matrix multiplication in $O\left(n^2m^{1.5}\log(nUD\gamma)\right)$.*

Proof: We will use the conjugate gradient variant to compute each iteration and hence the complexity of each iteration is $O(mn)$. To estimate the convergence criterion for rounding to the exact solution, we have to bound the largest sub-determinant of the matrix,

$$\begin{pmatrix} Q & A \\ A & 0 \end{pmatrix}.$$

In our case, the matrix is

$$\begin{pmatrix} B^TB & 0 & I \\ 0 & 0 & I \\ I & I & 0 \end{pmatrix}.$$

Since the log of the biggest subdeterminant of $B^T B$ is $O(n \log n\gamma)$ and the number of variables is $2m$ the overall complexity of our algorithm is $O(mn \cdot m^{0.5} \cdot (n \log n\gamma + \log DU))$ and the theorem follows. ■

Chapter 4

Distributed Algorithm for Multi-commodity Flow

4.1 Introduction

In this chapter[1], we present a simple and fast distributed approximation algorithm for a variant of the *minimum-cost concurrent multi-commodity flow problem.* One can ask the question, why we need an approximation algorithm, when the problem can be written as a linear program which can be solved exactly using interior point methods. Unfortunately, the fact that the number of variables is large causes these methods to generate rather slow (large computational complexity) algorithms. In particular, as seen in Chapter 3, the fastest interior-point min-cost multi-commodity flow algorithm [34, 39, 77, 78] relies on fast matrix multiplication subroutine and takes $O(k^{3.5}n^3m^{.5}\log(nDU))$ time for the multi-commodity flow problem with integer demands and at least $O(k^{2.5}nm^{1.5}\log(n\epsilon^{-1}DU))$ time to find an approximate solution, where D is the largest demand, and U is the largest edge capacity. Moreover, since this algorithm involves repeated matrix multiplications, it does not seem appropriate for efficient implementation in the distributed setting.

Significantly better running times were achieved by approximation algorithms. In particular, if there exists a feasible solution, then the algorithm due to Leighton, Makedon, Plotkin, Stein, Tardos, and Tragoudas [49] will produce an ϵ-approximation to the feasible flow in $O^*(k^2mn\epsilon^{-2})$ time. A flow that both satisfies at least $(1-\epsilon)$ fraction of each demand and whose cost is within $(1+\epsilon)$ of the optimum can be obtained by the algorithm due to

[1]This chapter represents joint work with Omri Palmon and Serge Plotkin [35].

Plotkin, Shmoys and Tardos [67] in $O^*(k^2m^2\epsilon^{-2}\log C)$ time, where C is the maximum edge cost. The running times of these algorithms can be improved by a factor of k by use of randomization. Radzik [68] showed a deterministic technique to improve by a factor of k, the complexity of the Leighton et al. algorithm. Later Karger and Plotkin [34] modified the algorithm in [67], to give a deterministic algorithm for the minimum cost multi-commodity flow which has an improved complexity of $O^*(kmn\epsilon^{-3}\log C)$. All these algorithms are based on repeated computations of minimum-cost flows or shortest paths, and hence are not very easy to implement in a distributed setting. (Especially in networks with dynamic topology.) The best performance one can hope to achieve by implementing the algorithm in [67], in a static network is $O^*(kmn\epsilon^{-2}\log C)$ rounds. This significantly exceeds the number of rounds of the algorithm presented in this chapter for large k and constant ϵ, δ.

If there exists a feasible flow, *i.e.* flow that satisfies all the demands, then in $O^*(m^2\epsilon^{-2})$ communication rounds, our algorithm finds a flow that satisfies at least $(1-\epsilon)$-fraction of each demand, where m is the number of edges in the network.[2] If there exists a feasible flow of cost B, then our algorithm finds a flow that satisfies $(1-\epsilon)$ fraction of the demands and whose cost is below $(1+\delta)B$ in $O^*(m^2\epsilon^{-2}\delta^{-2}\log(CU))$ rounds, where C and U are maximum edge cost and maximum edge capacity, respectively.

A communication round consists of each processor performing a computation on its local data and sending the results to all of its neighbors. Our algorithm is *local* in a sense that it does not depend on any global computations like shortest paths or min-cost flow, that were the main subroutines of the previously known efficient combinatorial multi-commodity flow algorithms [47, 49, 67]. The only previously known local approximate concurrent flow algorithm, due to Awerbuch and Leighton [8, 9], does not address the min-cost problem.

The main advantages of a "local" algorithm is ease of implementation in a distributed system where centralized control is either infeasible or very expensive. Moreover, our algorithm is resilient to failures: an edge failure (or, as a matter of fact, and change in the network topology) that does not make the problem infeasible, can delay the convergence of the algorithm by at most the number of rounds stated above. The only two nodes needed to be notified about the failure are the nodes adjacent to the failed edge. This makes it unnecessary to run the complicated and costly "global reset" algorithms [2] after every failure/reappearance of an edge. Another interesting property of our algorithm is that it can be modified to run forever, maintaining an approximate solution to the flow, automatically

[2] We say that $f(n) = O^*(g(n)$ if $f(n) = O(g(n)\log^k n)$ for some constant k.

adjusting it after every edge failure or demand change.

The algorithm presented in this chapter is very simple. The idea is to maintain a *preflow*, which is flow that obeys capacity constraints but does not obey flow-conservation constraints. We refer to the difference between the flow into a node and the flow out of the node as the *excess* at this node. The algorithm starts with zero flow, which is a valid preflow. Each iteration consists of two phases. In the first phase, each node distributes its excess among its adjacent edges. Then, independently for each edge, the flow through the edge is updated in a way that reduces a certain non-linear function of the excesses assigned to this edge. Intuitively, this operation "spreads" the excesses throughout the graph, eventually obtaining a solution with very small excesses.

Our algorithm is in some sense similar to the one in [8]. There are two main differences that allowed us to get improved performance. First, we use a different local optimization criteria. Our criteria can be viewed as an adaptation to the excess-minimization context of the exponential "length function", introduced by Shahrokhi and Matula [71]. Second, we use preflow instead of explicitly maintaining queues. Roughly speaking, if one interprets flow as the "amount of flow in unit time", then the fact that a node has excess implies that each time unit there is more flow coming in than going out. The algorithm in [8] explicitly maintains the number of flow units queued at each node and tries to minimize the size of these queues. Instead, our algorithm minimizes the excess, which corresponds to the *rate of growth* of the queues.

4.2 Preliminaries and Definitions

As defined in previous chapters, an instance of the *multi-commodity flow problem* consists of an undirected graph $G = (V, E)$, a non-negative capacity $u(vw)$ for every edge $vw \in E$, and a specification of k commodities, numbered 1 through k, where the specification for commodity i consists of a source-sink pair $s_i, t_i \in V$ and a non-negative demand d_i. Also, for notational convenience, we arbitrarily direct each edge. If there is an edge directed from v to w, this edge is unique by assumption, and we denote it by vw.

A multi-commodity flow f consists of a function $f_i(vw)$ on the edges of G for every commodity i, which represents the *flow* of commodity i on edge vw. If the flow of commodity i on edge vw is oriented in the same direction as edge vw, then $f_i(vw)$ will be positive, otherwise it will be negative. The signs only serve to indicate the direction of the flows.

Given flow $f_i(vw)$, the *excess* $\text{Ex}_i(f, v)$ of commodity i at node v is defined by:

$$\text{Ex}_i(f, v) \stackrel{def}{=} d_i(v) - \sum_{w:vw\in E} f_i(vw),$$

where $d_i(s_i) = d_i$, $d_i(t_i) = -t_i$ and $d_i(v) = 0$ otherwise. This definition of excess is slightly different from the earlier definition given in Chapter 3.

A flow where all excesses are zero is said to satisfy the *conservation constraints.* A flow satisfies *capacity constraints* if $\sum_i |f_i(vw)| \le u(vw)$ for all edges vw. A flow that satisfies both the capacity and the conservation constraints is *feasible.* During the description of our algorithm it will be convenient to work with *preflow*, which is a flow that satisfies capacity constraints but does not necessarily satisfy the conservation constraints. An ϵ-approximation to the multi-commodity flow feasibility problem is a feasible flow that satisfies at least $(1-\epsilon)$ fraction of the demands for each commodity.

4.3 The Flow Without Costs

In this section we present the distributed multi-commodity flow algorithm. If a feasible solution satisfying all the demands exists, the algorithm will produce a flow that satisfies at least $(1-\epsilon)$ fraction of each demand.

The Algorithm The algorithm starts with zero flow, which is a valid preflow. The algorithm maintains for every node pair v, w such that $vw \in E$ or $wv \in E$, commodity i, and edge $vw \in E$ an excess $Ex_i(f, vw)$.[3] Excess of commodity i at node v is the sum of $Ex_i(f, vw)$ for all w. Each iteration consists of two phases. In the first phase, each node recomputes $Ex_i(f, vw)$, effectively distributing its excess among its edges. Then, independently for each edge, the flow through the edge is updated in a way that reduces the potential function defined below. Intuitively, this operation "spreads" the excesses throughout the graph, eventually obtaining a solution with very small remaining excesses.

Given a flow f, we define the following potential function:

$$\Phi(f) \stackrel{def}{=} \sum_{vw\in E} \sum_{i=1}^{k} \exp\left(\alpha \frac{Ex_i(f, vw)}{d_i}\right) + \exp\left(\alpha \frac{Ex_i(f, wv)}{d_i}\right), \tag{4.1}$$

[3]Note that, in general, $Ex_i(f, vw) \neq Ex_i(f, wv)$.

where

$$\alpha \stackrel{def}{=} 4\epsilon^{-1} m \log(mk).$$

The algorithm starts from the zero flow and proceeds in iterations, where each iteration consists of two phases. The first phase computes the excess $Ex_i(f,v)$ at each node and distributes it equally among the edges incident to this node by setting $Ex_i(f,vw) = Ex_i(f,v)/\delta(v)$, where $\delta(v)$ is the degree of v. The second phase calculates the flow update $\Delta f_i(vw)$ on each edge vw for each commodity i where the goal is to minimize the potential function $\Phi(f)$. In fact, since this function can be written as a sum of functions where each one depends on the flow on a single edge, the algorithm minimizes the following function independently for each edge vw:

$$\sum_{i=1}^{k} \exp\left(\alpha \frac{Ex_i(f,vw) - \Delta f_i(vw)}{d_i}\right) + \exp\left(\alpha \frac{Ex_i(f,wv) + \Delta f_i(vw)}{d_i}\right). \tag{4.2}$$

In order to ensure convergence to feasible flow, the minimization is done under the following constraints:

$$\begin{aligned} \sum_{i=1}^{k} |f_i(vw) + \Delta f_i(vw)| &\leq u(vw) \\ |f_i(vw) + \Delta f_i(vw)| &\leq d_i \end{aligned}$$

The new flow is computed by setting $f_i(vw) = f_i(vw) + \Delta f_i(vw)$. In the rest of this section we will assume that the minimization is executed as described above. In fact, one can minimize a much simpler expression, thus improving the complexity of each iteration. We will address this issue in Section 4.4.

Analysis of the algorithm We will show that existence of a feasible solution that satisfies all of the demands implies that each iteration reduces the potential function. We will concentrate on proving the reduction due to the second phase in each iteration, since the first phase can not increase the potential function. The following theorem shows that sufficiently small value of $\Phi(f)$ implies that the sum of the excesses of any commodity does not exceed an ϵ-fraction of the demand of this commodity. Note that such a preflow can be easily converted into a flow that satisfies $(1-\epsilon)$-fraction of each demand.

Theorem 4.3.1 *If the value of the potential function $\Phi(f) \leq \exp(\alpha \frac{\epsilon}{m})$ then for each commodity i, we have $\sum_v |Ex_i(f,v)| \leq \epsilon d_i$.*

Theorem 4.3.2 *For each iteration* $\Phi(f+\Delta f) \leq \Phi(f)$. *Furthermore, if there exists a feasible solution to the problem then in each iteration*

$$\Phi(f+\Delta f) \leq \Phi(f)\left(1 - \frac{\log^2 \frac{\Phi(f)}{mk}}{16\alpha^2}\right).$$

Proof: As we have noted above, the first phase of each iteration can not increase the potential function. Obviously, the second phase can not increase the potential function value. It remains to prove that the decrease due to the second phase is large.

Let f^* denote a feasible solution that satisfies all of the demands such that $|f^*(vw)| \leq d_i$ for every commodity i and edge vw. Let $\tilde{f} \stackrel{def}{=} f^* - f$. Observe that flow update defined by $\sigma\tilde{f}$ for any $\sigma \in [0, \alpha^{-1}]$ satisfies the optimization constraints. Hence the reduction in the potential function can be bounded by computing the effect of updating the current flow by $\sigma\tilde{f}$.

We will use the fact that $e^{x+t} \leq e^x + te^x + t^2e^x$ for $|t| \leq 1$. Thus, if $\left|\alpha\sigma\tilde{f}_i(vw)\right| \leq d_i$ for all i and vw, we have:

$$\Phi\left(f+\sigma\tilde{f}\right) \leq \tag{4.3}$$

$$\sum_{i=1}^{k}\sum_{vw\in E} \exp\left(\alpha\frac{Ex_i(f,vw) - \sigma\tilde{f}_i(vw)}{d_i}\right) + \exp\left(\alpha\frac{Ex_i(f,wv) + \sigma\tilde{f}_i(vw)}{d_i}\right) \tag{4.4}$$

$$\leq \Phi(f)$$

$$-\sum_{i=1}^{k}\sum_{vw\in E} \sigma\alpha\frac{\tilde{f}_i(vw)}{d_i}\left[\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right) - \exp\left(\alpha\frac{Ex_i(f,wv)}{d_i}\right)\right] \tag{4.5}$$

$$+\sum_{i=1}^{k}\sum_{vw\in E} \sigma^2\alpha^2\left(\frac{\tilde{f}_i(vw)}{d_i}\right)^2\left[\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right) + \exp\left(\alpha\frac{Ex_i(f,wv)}{d_i}\right)\right]. \tag{4.6}$$

We will evaluate each one of the terms in the above expression separately. To evaluate (4.5) we decompose $\tilde{f}$ into cycles and paths from positive to negative excesses. Notice that the cycles do not affect the value of (4.5). Moreover, each flow path of value F that starts with edge vv' and ends with $u'u$, contributes

$$\sigma\alpha F \cdot \left(\exp\left(\alpha\frac{Ex_i(f,vv')}{d_i}\right) - \exp\left(\alpha\frac{Ex_i(f,uu')}{d_i}\right)\right).$$

Hence, (4.5) is equal to:

$$\sigma\sum_i\sum_{vw\in E} \alpha\frac{Ex_i(f,vw)}{d_i}\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right) \geq \sigma\Phi(f)\log\left(\frac{\Phi(f)}{mk}\right),$$

where the inequality follows from the fact that the sum has mk terms and $\sum y_i \log y_i$ is convex. To evaluate expression (4.6) we use the fact that $\left|\tilde{f}_i(vw)\right| \leq 2d_i$ and thus

$$\sum_i \sum_{vw\in E} \sigma^2\alpha^2 \left(\frac{\tilde{f}_i(vw)}{d_i}\right)^2 \exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right) \leq 4\sigma^2\alpha^2\Phi(f).$$

Combining our bounds on (4.5) and (4.6) we get from (4.3) that for $\sigma \leq \alpha^{-1}$:

$$\Phi\left(f+\sigma\tilde{f}\right) \leq \Phi(f) - \sigma\Phi(f)\log\frac{\Phi(f)}{mk} + 4\sigma^2\alpha^2\Phi(f).$$

Thus taking

$$\sigma = \frac{\log\frac{\Phi(f)}{km}}{8\alpha^2}$$

we get

$$\Phi\left(f+\sigma\tilde{f}\right) \leq \Phi(f)\left[1 - \frac{\log^2\frac{\Phi(f)}{mk}}{16\alpha^2}\right].$$

This completes the proof of the theorem. ▮

Remark: Note that the way we proved the decrease in the potential function is by expressing a lower bound on this decrease as a sum of quadratic functions in Δf and then showing that there exists a flow that achieves a large decrease. We will use this fact in Section 4.5, where we present an efficient implementation of each iteration.

Theorem 4.3.3 *After $O(m^2\epsilon^{-2}\log mk)$ iterations, we have $\sum_v |Ex_i(f,v)| \leq \epsilon d_i$ for each commodity i.*

Proof: Theorem 4.3.2 implies that as long as $\Phi(f) \geq 2m^2k^2$, the value of the potential function is halved in $O(\alpha^2/\log^2\Phi(f))$. Thus, the number of iterations until the potential function falls to $2m^2k^2$ is bounded by $O\left(\alpha^2/\log(2m^2k^2)\right) = O(m^2\epsilon^{-2}\log mk)$. The claim follows by substituting the value of α into the claim of Theorem 4.3.1. ▮

4.4 Flow With Costs

In this section, we show how to modify the algorithm described in Section 4.3 to approximately solve the minimum cost multi-commodity flow problem. More precisely, we consider the problem where every edge vw has an associated cost $c(vw)$, and the goal is to find a flow f that satisfies at least a $(1-\epsilon)$ fraction of each demand and whose cost $\sum_{i,vw}|f_i(vw)|c(vw) \leq (1+\delta)B$, where B is a given budget and $\delta > 0$ is some given parameter.

In this section we will assume that B is given and that there exists a feasible flow of cost at most B that satisfies all the demands. Note that by doing bisection search, one can easily extend our approximation algorithm to the case where B is unknown in advance. Details will appear in the full paper.

In order to take the costs into account, we will use the following potential function, which is a modification of the function used in the previous section.

$$\begin{aligned}\Phi(f) \quad \stackrel{def}{=} \quad & \sum_{vw\in E}\sum_{i=1}^{k}\left[\exp\left(\alpha\frac{Ex_i(f,vw)}{d_i}\right)+\exp\left(\alpha\frac{Ex_i(f,wv)}{d_i}\right)\right] \\ & +\frac{\gamma}{(1+\delta)B}\sum_i\sum_{vw\in E}c_i(vw)|f_i(vw)|\end{aligned}$$

where $\gamma = m^2k^2e^{4/\delta}/\delta$ and $\alpha = \frac{m}{\epsilon}\log\gamma$. To simplify notation, we will use Φ_1 to denote the first term and Φ_2 to denote the second. Also, we will write Φ instead of $\Phi(f)$ where it will not cause confusion.

Observe that we can express $\Phi(f)$ as a sum of terms such that each term depends only on the flow on a single edge and is independent of the flows on the other edges. Thus, the distributed algorithm presented in Section 4.3 can be based on $\Phi(f)$ instead of the potential function given by (4.1). In other words, the only difference will be the minimization step, which will update the flow on each edge in order to minimize the corresponding term in $\Phi(f)$ instead of the corresponding term in (4.1).

As before, it is easy to see that the algorithm ensures that Φ does not increase. Roughly speaking, our analysis consists of two parts. First we show that a sufficiently small value of $\Phi(f)$ implies that we have good approximation to the MFCB problem. Then we prove that $\Phi(f)$ decreases sufficiently fast.

Theorem 4.4.1 *If preflow f obeys capacity constraints and $\Phi(f) \leq \gamma(1+\delta)$, then the cost of f is at most $(1+3\delta)B$ and the total excess of commodity i is bounded by ϵd_i.*

Proof: We observe that $\Phi^c(f) \leq m^2k^2$ implies that $\Phi(f) \leq m^2k^2$ and hence the flow is ϵ-approximate. Moreover we also have

$$\frac{\gamma}{B}\sum_i\sum_{e\in E}c_i(e)f_i(e) \leq m^2k^2$$

and hence

$$\sum_i \sum_{e \in E} c_i(e) f_i(e) \leq B(1+\delta).$$

∎

Observe that using the standard technique of decomposing preflow into paths from sources to sinks, from positive to negative excesses, and into cycles, a preflow that satisfies the condition of the above theorem can be easily transformed into a flow of cost at most $(1+\delta)B$ that satisfies $(1-\epsilon)$ fraction of each demand. This transformation is done differently in the sequential and distributed setting and we defer the details to the full paper.

Theorem 4.4.2 *The algorithm that minimizes the potential function $\Phi(f)$ solves MCFB problem in $O(\frac{m^2}{\epsilon^2\delta^2}\log^2(mk\delta^{-1})\log(mCU))$ iterations where C is the maximum cost and U is the maximum capacity associated with any edge in the graph.*

Proof: Let f^* denote the optimum flow of cost at most B that satisfies all the demands. Like in the proof of Theorem 4.3.2, we will use the fact that the decrease in the total potential function at each iteration is at least as large as the decrease obtained by setting the new flow to be $(1-\sigma)f + \sigma f^*$. Let $\Delta f = \sigma(f^* - f)$. Then, for $\sigma \leq 1/\alpha$, the decrease in the total potential function is at least:

$$\Phi(f) - \Phi(f+\Delta f) \geq \sigma\Phi_1(f)\log\frac{\Phi_1}{mk} - 4\sigma^2\alpha^2\Phi_1(f) + \sigma\Phi_2(f) - \sigma\frac{\gamma}{(1+\delta)}. \tag{4.7}$$

We will distinguish three cases, according to the relative values of Φ_1 and Φ_2. First consider the case where $\Phi_1 \geq \gamma\delta/2$ and $\Phi_2 \leq \Phi_1$. Using the fact that $\log\frac{\Phi_1}{mk} \geq 4/\delta$, we get that for

$$\sigma = \frac{\log\frac{\Phi_1}{mk}}{8\alpha^2}$$

the decrease is bounded by:

$$\frac{\log^2\frac{\Phi_1}{mk}}{32\alpha^2}\Phi.$$

The number of iterations of this type is at least $O(\alpha^2/\log(mk))$.

The second type of iteration occurs when $\Phi_1 \geq \gamma\delta/2$ and $\Phi_2 \geq \Phi_1$. Using the same value of σ as above, the decrease per iteration can be bounded in this case by $\Omega(\Phi/\alpha^2)$. Since in this case $\gamma \leq \Phi \leq O(\gamma mCU)$, the number of such iterations is bounded by $O(\alpha^2\log(mCU))$.

The third type of iteration occurs when $\Phi_1 \leq \gamma\delta/2$. We will bound the number of such iterations for $\Phi \in [(1+\eta)\gamma, (1+2\eta)\gamma]$ for $\eta \geq \delta$. When Φ is in this range, then

$\Phi_2 \geq (1+\eta/2)\gamma$. Observe that the first term in expression (4.7) is always nonnegative. For $\eta \leq 1$, since

$$\Phi_2 - \gamma/(1+\delta) \geq \eta\Phi_2/3$$

by choosing

$$\sigma = \frac{\eta}{8\delta\alpha^2}$$

we see that the decrease in the potential function is at least $\Omega(\eta^2/(\alpha^2\delta)\Phi)$. Hence, the number of iterations of this type for $\Phi \in [(1+\eta)\gamma, (1+2\eta)\gamma]$ is bounded by $O(\alpha^2\delta/\eta)$. Thus, the total number of such iterations for the range $\Phi \in [(1+\delta)\gamma, 2\gamma]$ is bounded by $O(\alpha^2)$. For the case where $\Phi_1 \leq \gamma\delta/2$ and $\Phi \geq 2\gamma$, choice of $\sigma = 1/(8\alpha^2\delta)$ implies that the decrease is at least $\Omega(\alpha^2\delta)$, and hence the total number of such iterations is bounded by $O(\alpha^2\delta\log(mCU))$.

Observe that if we are not in any of the types of iterations addressed above, then $\Phi \leq (1+\delta)\gamma$. Since Φ monotonically decreases, the claim of the theorem follows by summing up the bound on the number of iterations of each type and applying Theorem 4.4.1. ∎

4.5 Implementation of a single iteration

The previous sections described the algorithms in terms of iterations, where the main part of each iteration is minimization of a set of independent functions of flow, one for each edge. In this section we will address the question of how to perform this minimization efficiently.

The main idea is to minimize a quadratic approximation to the potential function instead of the function itself. The proofs of Theorems 4.3.2 and 4.4.2 rely on the fact that the decrease in Φ as a function of the flow update Δf, can be bounded by a sum of independent quadratic functions in Δf (one per edge), as long as the following constraints are satisfied for every commodity i and edge vw:

$$\begin{aligned} |\Delta f_i(vw)| &\leq d_i/\alpha \\ |f_i(vw) + \Delta f_i(vw)| &\leq d_i \\ \sum_i |f_i(vw) + \Delta f_i(vw)| &\leq u(vw). \end{aligned} \tag{4.8}$$

Moreover, these proofs show that there exists a Δf that satisfies the above constraints and for which the value of the sum of these quadratic functions is sufficiently large. Hence, instead of minimizing the potential function itself, we can achieve at least as large a decrease by maximizing these quadratic functions over the region defined by (4.8).

The advantage of working with the quadratic approximation is that it is relatively easier to optimize than the original potential function that includes exponentials. Observe that the quadratic functions that we need to maximize are concave. The quadratic optimization can be done in $O(k \log k)$ time per edge.

Chapter 5

Approximate Algorithm for Min-cost Multi-commodity Flow

5.1 Introduction

Since multi-commodity flow approaches based on interior point methods for linear programming lead to high asymptotic bounds on running time, recent emphasis was on designing fast combinatorial *approximation* algorithms. If there exists a flow of cost B that satisfies all the demands, the goal of an (ϵ, δ)-approximation algorithm is to find a flow of cost at most $(1+\delta)B$ that satisfies $(1-\epsilon)$ fraction of each demand.

Approximation algorithms for various variants of multi-commodity flow can be divided according to whether they are based on relaxing the capacity [70, 47, 49, 67] or conservation constraints [8, 9]. In particular, the algorithm in [67] is based on relaxing the capacity and budget constraints. In other words, the algorithm starts with a flow that satisfies the demands but does not satisfy either the capacity or budget constraints. It repeatedly reroutes commodities in order to keep demands satisfied while reducing the amount by which the current flow overflows the capacities or overspends the budget.

Awerbuch and Leighton [8, 9] proposed several algorithms for multi-commodity flow without costs that are based on relaxing the conservation constraints. More precisely, these algorithms do not try to maintain a flow that satisfies all of the demands. Instead, they maintain a *preflow* like in the single commodity flow algorithms of Goldberg and Tarjan [26, 27]. These algorithms repeatedly adjust the preflow on an edge-by-edge basis, with the goal of maintaining capacity constraints while making preflow closer to a flow.

Though the algorithms in [8, 9] have very good complexity, the main problem of extending them to the min-cost case is the fact that these algorithms are based on minimizing (at each step) a function that defines the penalties for not satisfying the conservation constraints. The reason these algorithms are fast is that this function is *separable*, i.e. it is a sum of non-linear terms where each term depends on a single edge, and thus this function can be minimized on an edge-by-edge basis. A natural extension of this approach to the min-cost case is to introduce an additional penalty term that depends on the cost of the current preflow. Unfortunately, the resulting function is *non-separable* since this term has to be non-linear for the technique to work. This, in turn, leads to an algorithm that has to perform a very slow global optimization at each step. In fact, this optimization does not seem to be much easier than solving the original problem. We saw in Chapter 4, one way of overcoming this problem. This gave us an algorithm which also has the nice property of being distributed. If we do not need the algorithm to be distributed, we can extend some of the ideas developed there to give a faster approximation algorithm.

The main result of this chapter[1] is an $O^*(kmn^2 \log(CU))$ deterministic approximation algorithm[2], where C and U denote the maximum cost and the maximum capacity, respectively; the constant in the big-O depends on ϵ^{-1} and δ^{-1}. Our algorithm combines ideas from the feasibility multi-commodity algorithm of Awerbuch and Leighton [9] and the min-cost flow algorithm described in chapter 4. As in [9], we relax the conservation constraints. In addition, we relax the budget constraint. Our algorithm maintains a preflow that always satisfies capacity constraints; the preflow is gradually adjusted through local perturbations in order to make it as cheap and as close to flow as possible. The main innovation lies in the pair of two functions – one separable (can be minimized on an edge-by-edge basis) and one not that are used to guide the flow adjustment at each step. Essentially, the flow adjustment is guided by the separable function that is *redefined at each step*, while the progress is measured by the other function, which is non-separable.

There are several important differences between our algorithm and the algorithms of Awerbuch and Leighton in [8, 9] that allow us to treat the min-cost case. Their algorithms explicitly maintain the number of flow units queued at each node and try to minimize the size of the queues by moving flow units from one queue to another. In contrast, our algorithm is based on maintaining the preflow and the corresponding excess at each node.

[1]This chapter represents joint work with Omri Palmon and Serge Plotkin [36].
[2]We say that $f(n) = O^*(g(n))$ if $f(n) = O(g(n) \log^c n)$ for constant c.

Use of preflow instead of explicit queues required development of different techniques to prove the convergence. Our techniques are based on trying, at each step, to minimize a certain separable function that can be computed on an edge-by-edge basis, while measuring the progress by another, non-separable, function. We believe that these techniques might be useful for analyzing future preflow-based multi-commodity flow algorithms.

Our algorithm can be viewed as a generalization of the algorithm in [9] to the min-cost case, and as such preserves some of the advantages of that algorithm. In particular, the edge-by-edge adjustment of the flow, done by our algorithm, is inherently parallel. The min-cost multi-commodity flow algorithm in [67] is based on repeated computation of shortest paths. Although this computation can be done in NC, it requires n^3 processors, making it impractical. For the case where a linear number of processors is available (one for each variable, *i.e.* commodity-edge pair), the algorithm in [67] runs in $O^*(kmn \log CU)$ time for constant ϵ and δ. In contrast, our algorithm runs in $O^*(mn)$ time. We believe that the inherent parallelism in our algorithm, combined with locality of reference, will lead to efficient implementation on modern superscalar computers.

Section 5.2 presents the continuous version of the algorithm and proves fast convergence assuming that at each iteration the algorithm computes an exact minimization of a quadratic function for each edge. In Section 5.3 we show how to reduce the amortized time per iteration by using a rounding scheme that allows us to compute approximate minimization at each iteration. Section 5.4 presents a proof that this rounding does not increase the total number of iterations.

5.2 Continuous algorithm

In this section we present the (slow) "continuous" version of the min-cost multi-commodity flow algorithm. The improvement in running time due to rounding is addressed in the next section. If a feasible solution of cost $B/(1+\delta)$ satisfying all the demands exists, the algorithm produces an $(O(\epsilon), O(\delta))$-preflow. We present here the algorithm for directed case; the undirected case can be treated similarly. We assume without loss of generality that the minimum node degree is $\frac{m}{2n}$.

5.2.1 The Algorithm

The algorithm starts with zero flow, which is a valid preflow. For every node pair v, w such that $vw \in E$ or $wv \in E$ and commodity i, it maintains an excess $Ex_i(f, vw)$.[3] The excesses $Ex_i(f, vw)$ are maintained such that for every commodity i at node v, the excess $Ex_i(f, v)$ of the commodity at that node is equal to the sum of $Ex_i(f, vw)$ for all w such that $vw \in E$ or $wv \in E$. Given a flow f, we define the following potential function:

$$\Phi = \Phi_1 + \gamma \Phi_2 \tilde{\Phi}_1$$

where

$$\Phi_1 = \sum_{i, vw \in E \text{ or } wv \in E} \exp\left(\alpha \frac{Ex_i(f, vw)}{d_i^*}\right),$$
$$\Phi_2 = \frac{1}{B} \sum_{e \in E} c(e) f(e).$$

We use d_i^* to denote the original demand of commodity i. $\tilde{\Phi}_1$ is the current approximation to the current value of Φ_1 and should satisfy:

$$(1 + \delta)^{-1} \Phi_1 \leq \tilde{\Phi}_1 \leq (1 + \delta) \Phi_1. \tag{5.1}$$

As we show below, the best performance is achieved for

$$\begin{aligned} \alpha &= 8\epsilon^{-1} m \log(2mk), \\ \gamma &= \alpha\epsilon/(2m). \end{aligned}$$

Roughly speaking, Φ_1 measures how close the current preflow is to a flow and Φ_2 measures how close the cost of the current flow is to the budget B. Unfortunately, it is not clear how to use Φ to directly measure the progress of the algorithm, since it is redefined each time Φ_1 changes by more than a constant factor. Instead, we define another potential function:

$$\Psi = \log \frac{\Phi_1}{2mk} + \frac{\gamma}{1 + \delta} \Phi_2.$$

The algorithm starts from the zero flow and proceeds in phases. In the beginning of a phase, we choose $\sigma = \frac{\delta m}{16\alpha^2 n} \Psi_0$, where Ψ_0 is the value of potential function Ψ in the beginning of the phase. Each phase proceeds in iterations, where each iteration consists of the

[3] Note that, in general, $Ex_i(f, vw) \neq Ex_i(f, wv)$.

following steps:

ALGORITHM 1: (SINGLE ITERATION)

1. Add capacity $\sigma u(vw)$ to each edge $vw \in E$. Change demand of each commodity i: $d_i = d_i + \sigma d_i^*$, where d_i^* is the original demand of commodity i. Update the value of excesses accordingly.

2. Find the increment Δf in the flow vector f that minimizes Φ under the constraint $0 \leq \Delta f_i(vw) \leq \sigma d_i^*$ and $\sum \Delta f_i(vw)$ is less then the available capacity. Update $f = f + \Delta f$.

3. Recompute new excess $Ex_i(f, v)$ at each node and distribute it equally among the edges incident to this node by setting $Ex_i(f, vw) = Ex_i(f, v)/\delta(v)$, where $\delta(v)$ is the degree of v.

4. Update the current value of $\tilde{\Phi}_1$, if necessary.

A phase is terminated either after $1/\sigma$ iterations or when the value of Ψ falls below $\Psi_0/2$. In the end of a phase, we scale down all the flows, the demands, and the capacities (and hence all the excesses) by a factor of $1 + r\sigma$, where r is the number of iterations in this phase. Note that in the end of each phase the capacities and the demands are equal to the original capacities and the demands. Moreover, since we scale flows and capacities by the same factor, the preflow in the end of each phase satisfies the original capacity constraints.

The second step is the heart of the algorithm. Since function Φ is separable, *i.e.* can be written as a sum of functions where each one depends on the flow on a single edge, the algorithm computes change in flow Δf by minimizing the following function independently for each edge $vw \in E$:

$$\sum_{i=1}^{k} \left[\exp\left(\alpha \frac{Ex_i(f, vw) - \Delta f_i(vw)}{d_i^*} \right) + \exp\left(\alpha \frac{Ex_i(f, wv) + \Delta f_i(vw)}{d_i^*} \right) \right] + \frac{\gamma}{B} \tilde{\Phi}_1 c(vw) \Delta f_i(vw).$$

5.2.2 Bounding number of phases

In this section we show a bound on the number of phases of ALGORITHM 1. Initially, we assume that each phase is executed exactly as stated above, i.e. the minimization in step 2 and the excess rebalancing at step 3 are computed exactly. First we show that sufficiently small value of Ψ implies that the current flow is close to optimum.

Lemma 5.2.1 *If $\Psi \leq (1+\delta)\gamma$, then the current preflow is $(2\epsilon, 3\delta)$-optimum.*

Proof: Assume that the current preflow is not $(2\epsilon, 3\delta)$-optimum. There can be two cases: either there exists an edge vw with excess $Ex_i(f, vw) > \epsilon d_i^*/m$ or the current cost of the flow exceeds $(1+3\delta)B$. In the second case, $\Phi_2 > (1+\delta)$, and hence the fact that $\Phi_1 \geq 2mk$ implies that $\Psi > (1+\delta)\gamma$. If there exists an edge vw with excess $Ex_i(f, vw) > \epsilon d_i^*/m$,

$$\Phi_1 > \exp(\alpha\epsilon/m) \geq (2m)^8 k^8.$$

Thus:

$$\log \frac{\Phi_1}{2mk} \geq 7\log(2mk) = 1.75\alpha\epsilon/(2m) > (1+\delta)\gamma$$

when $\delta < 0.75$. ∎

Observe that Ψ might increase during an iteration. On the other hand, the rescaling of the demands, flows and capacities at the end of each phase halves Ψ. The main point of the following lemma is to show that this reduction is significantly larger than the total sum of increases in a single phase.

Lemma 5.2.2 *The increase in Φ during a single iteration is bounded from above by:*

$$4\sigma^2\alpha^2\frac{n}{m}\Phi_1 + \frac{\gamma}{1+\delta}\sigma\tilde{\Phi}_1.$$

We defer the proof of this lemma. Instead, we prove that it implies convergence in a small number of phases. Lemma 5.2.2, together with the definition of Ψ gives a bound on the increase in Ψ:

Lemma 5.2.3 *The increase $\Delta\Psi$ in Ψ during a single iteration is bounded by $4\sigma^2\alpha^2\frac{n}{m} + \gamma\sigma$.*

Proof: Let $\Delta\Phi_1$ and $\Delta\Phi_2$ denote the change in Φ_1 and Φ_2 during one iteration, respectively. Denote the total increase in Φ during one iteration by $\Delta\Phi = \Delta\Phi_1 + \gamma\tilde{\Phi}_1\Delta\Phi_2$. Lemma 5.2.2 implies that:

$$\begin{aligned}
\Delta\Psi &\leq \log\frac{\Phi_1 + \Delta\Phi_1}{\Phi_1} + \frac{\gamma}{1+\delta}\Delta\Phi_2 \leq \frac{1}{\Phi_1}(\Delta\Phi_1 + \gamma\Delta\Phi_2\frac{\Phi_1}{1+\delta}) \\
&\leq \frac{1}{\Phi_1}(\Delta\Phi_1 + \gamma\tilde{\Phi}_1\Delta\Phi_2) = \frac{\Delta\Phi}{\Phi_1} \\
&\leq 4\sigma^2\alpha^2\frac{n}{m} + \sigma\frac{\tilde{\Phi}_1}{\Phi_1}\frac{\gamma}{1+\delta} \leq 4\sigma^2\alpha^2\frac{n}{m} + \sigma\gamma
\end{aligned}$$

Note that we used the bounds on $\tilde{\Phi}_1$, given by (5.1). ∎

Lemma 5.2.4 *Let Ψ_0 and Ψ_s be the value of the potential function Ψ at the beginning of a phase and at the end of this phase (after scaling of the demands and capacities), respectively. If $\Psi_0 \geq \gamma(1+\delta)$ then $\Psi_s \leq (1-\delta/4)\Psi_0$.*

Proof: Observe that scaling down of demands and flows at the end of a phase can not increase Ψ. Hence, the claim holds for the case when the phase was terminated because Ψ fell below $\Psi_0/2$. Otherwise, demands and flows are halved at the end of the iteration, causing halving of the excesses. This corresponds to taking a square root of each one out of $2mk$ terms in Φ_1, which causes $\log[\Phi_1/(2mk)]$ to go down by at least a factor of 2. Since Φ_2 is linear in the value of the flow, we have:

$$\begin{aligned}
\Psi_s &\leq \frac{1}{2}\left(\Psi_0 + (4\sigma\alpha^2\frac{n}{m} + \gamma)\right) \\
&= \Psi_0 - \frac{1}{2}\left(\Psi_0 - (4\sigma\alpha^2\frac{n}{m} + \gamma))\right) \\
&\leq \Psi_0 - \frac{1}{2}\left(\delta\Psi_0 - 4\sigma\alpha^2\frac{n}{m}\right)
\end{aligned}$$

where the last equality is implied by the assumption that $\Psi_0 > (1+\delta)\gamma$. The desired result can be obtained by choosing

$$\sigma = \frac{\delta}{16\alpha^2\frac{n}{m}}\Psi_0.$$

∎

Initial value of Ψ is bounded by $O^*(\alpha)$; by Lemma 5.2.1 the algorithm terminates when it had succeeded to reduce Ψ to $O(\gamma) = O(\alpha\epsilon/m)$. Hence we have the following theorem:

Theorem 5.2.5 *The algorithm terminates in $O(\delta^{-1}\log(m\epsilon^{-1}))$ phases, where each phase consists of $O(\sigma^{-1}) = O^*(\delta^{-1}\epsilon^{-2}mn)$ iterations.*

Proof of Lemma 5.2.2: Consider a min-cost multi-commodity flow f^* that satisfies the original demands d_i^*. Note that $\sigma f_i^*(vw)$ is one of the possibilities for $\Delta f_i(vw)$ computed in the second step of each iteration. Hence, the total increase in Φ in a single iteration is not worse than the increase in Φ if we set

$$\forall i : \Delta f_i(vw) = \sigma f_i^*(vw).$$

In order to bound increase in Φ, we will use the fact that $e^{x+t} \leq e^x + te^x + t^2e^x$ for $|t| \leq 1/2$. Observe that our choice of α and σ implies that $\alpha\sigma d_i^* \leq d_i^*/2$ for all i, and hence we can use the above second order approximation. Consider the contribution to Φ of the change in excesses due to increase in demand of every commodity i by σd_i^* and due to flow change $\Delta f = \sigma f^*$. Observe that for every i, this operation does not change excesses of commodity i at its source s_i and its sink t_i. Recall that we have assumed that there exists a solution of cost $B/(1+\delta)$. Thus, the cost of f^* is bounded by $B/(1+\delta)$. To simplify notation, denote

$$\phi_i(f,v,w) = \exp\left(\alpha \frac{\mathrm{Ex}_i(f,vw)}{d_i^*}\right).$$

Hence the increase in Φ during a single iteration can be bounded by:

$$\sum_{\substack{i,\\ v\notin\{s_i,t_i\}}} \alpha\sigma \sum_{w:vw\in E} \left(\phi_i(f,w,v) - \phi_i(f,v,w)\right) \frac{f_i^*(vw)}{d_i^*} + \sum_{\substack{i,\\ vw\in E}} \alpha^2\sigma^2 \left(\phi_i(f,w,v) + \phi_i(f,v,w)\right) \left(\frac{f_i^*(vw)}{d_i^*}\right)^2 + \sigma \frac{\gamma}{1+\delta}\tilde{\Phi}_1. \tag{5.2}$$

We will bound each term in equation (5.2) separately. In order to bound the first term, decompose f^* into a collection of flow paths from sources to their respective sinks. Consider flow path P of commodity i from s_i to t_i and denote its value by $\sigma f_i^*(P)$. The contribution of this flow path to the first term is bounded by:

$$\sum_{\substack{wv\in P\\ vw'\in P}} \alpha\sigma \left(\phi_i(f,w,v) - \phi_i(f,v,w')\right) \frac{f_i^*(P)}{d_i^*} \tag{5.3}$$

Since the excesses associated by node v with all its incident edges are the same (by Step 3 of ALGORITHM-1), the above contribution is zero. The fact that $f_i^*(vw) \leq d_i^*$, implies that the second term can be bounded by:

$$\sum_{i,vw\in E} \alpha^2\sigma^2 \left(\phi_i(f,w,v) + \phi_i(f,v,w)\right) \frac{f_i^*(vw)}{d_i^*}$$

Since we have assumed that the minimum degree is $\frac{m}{2n}$, the contribution to this term of a flow path P of value $f_i^*(P)$ is bounded by:

$$(5.4) \qquad \sum_{\substack{wv \in P \\ vw' \in P}} \alpha^2\sigma^2 \left(\phi_i(f,w,v) + \phi_i(f,v,w')\right) \frac{f_i^*(P)}{d_i^*}$$
$$\leq 4\alpha^2\sigma^2 \frac{n}{m} \frac{f_i^*(P)}{d_i^*} \Phi_1.$$

The claim of the lemma follows from the fact that the sum of flow values $f_i(P)$ over all flow paths of commodity i in the flow f^* is exactly d_i^*. ▮

Theorem 5.2.5 implies that for constant ϵ and δ, we can find an (ϵ, δ)-approximate preflow in $O^*(mn)$ iterations, where each iteration consists of minimizing convex function (5.2) for each edge. In fact, it is easy to see that the proof of Lemma 5.2.2 remains unchanged if we will optimize over a second-order approximation to (5.2). Thus, each iteration can be implemented in $O(mk \log k)$ time, which implies the following theorem.

Theorem 5.2.6 Algorithm-1 *can be implemented to run in* $O^*(\delta^{-2}\epsilon^{-2}km^2n)$ *time.*

Observe that since each iteration can be implemented in $O(\log k)$ time using km processors on PRAM, the algorithm can be implemented to run in $O^*(\delta^{-2}\epsilon^{-2}mn)$ time in parallel, as mentioned in the introduction.

5.3 Improving The Running Time

Each iteration of Algorithm-1, involves optimizing a quadratic function for each edge, which can take up to $O(mk \log k)$ time per iteration. Intuitively, the problem lies in the fact that large fraction of the flow updates made by the continuous algorithm do not lead to a sufficiently large progress toward the solution. In this section, we will present a variation of Algorithm-1 which not only simplifies the algorithm but ensures that each flow update leads to a substantial progress. The improvement is based on the "packetizing" technique introduced in [9].

In Section 5.4 we will prove that this algorithm converges to a solution in asymptotically the same number of phases as Algorithm-1.

The first modification is to restrict Step 2 of the algorithm so that on any edge it considers for optimization only those commodity/edge pairs $(i, vw \in E)$ that satisfy:

$$(5.5) \qquad \begin{aligned} Ex_i(f, vw) &\geq Ex_i(f, wv) + 4\sigma d_i^* \\ Ex_i(f, vw) &\geq -\beta d_i \end{aligned}$$

where $\beta = \log(mk\delta^{-1}\epsilon^{-1})/\alpha$. All the flow and excess vectors considered henceforth will be restricted to the subspace determined by this restriction. Also, the algorithm uses a new set of excesses $\tilde{E}x$, defined as follows:

$$\tilde{E}x_i(f, vw) = \begin{cases} Ex_i(f, vw) - \sigma d_i & vw \in E \\ Ex_i(f, vw) + \sigma d_i & wv \in E \end{cases}.$$

Let $\Phi'_{(i,vw)}(Ex)$ be the negated gradient of the potential function w.r.t. the flow vector for commodity i in edge $vw \in E$, defined as:

$$\Phi'_{(i,vw)}(Ex) = -\frac{\gamma}{B}c(vw)\tilde{\Phi}_1 + \\ \frac{\alpha}{d_i}\left(\exp(\alpha\frac{Ex_i(f, vw)}{d_i^*}) - \exp(\alpha\frac{Ex_i(f, wv)}{d_i^*})\right).$$

Note that since the edges and commodities obey condition (5.5) the gradient vector $\Phi'(\tilde{E}x)$ is non-negative. The updated algorithm is as follows:

ALGORITHM-2 (single iteration)

1. Add capacity $\sigma u(vw)$ to each edge $vw \in E$. Change demand of each commodity i: $d_i = d_i + \sigma d_i^*$, where d_i^* is the original demand of commodity i. Update the value of excesses accordingly.

2. Find an unsaturated edge and a commodity i satisfying condition (5.5) with the largest gradient $\Phi'_{(i,vw)}\left(\tilde{E}x\right)$ and move a unit σd_i^* or an amount that is sufficient to saturate that edge (whichever is less). Update the flow and excesses. Observe that this is equivalent to finding the change Δf in the flow vector f that maximizes $\sum \Phi'_{(i,vw)}\left(\tilde{E}x\right)\Delta f_i(vw)$ under the constraints that $0 \leq \Delta f_i(vw) \leq \sigma d_i^*$ and $\sum_i \Delta f_i(vw)$ does not exceed available capacity.

3. Recompute new excesses $Ex_i(f, vw)$ at each node v. Rebalance excesses inside each node to within an additive error of σd_i^* by moving excess in increments of at least σd_i^*.

4. Update the current value of $\tilde{\Phi}_1$, if necessary.

As in ALGORITHM-1, a phase is terminated either after $1/\sigma$ iterations or when the value of Ψ falls below $\Psi_0/2$. In the end of a phase, we scale down all the flows, the demands, and the capacities (and hence all the excesses) by a factor of $1 + r\sigma$, where r is the number of iterations in this phase.

Using a heap on every edge-node pair, it is straightforward to implement ALGORITHM-2 to run in $O(\log k)$ time per excess update. Excess updates can be divided into two types depending on whether or not the the update was *large* (exceeded σd_i^*) or *small*. Observe that if flow of commodity i on edge vw was updated by less than σd_i^*, the only reason this flow might not be updated again in the next iteration is if for some commodity j, the excess $Ex_j(f, vw)$ or $Ex_j(f, wv)$ was changed by a multiple of σd_j^* during rebalancing in Step 3 of the current iteration. If, on the other hand, the excess updates add-up to σd_i over several iterations, then we can precompute the number of iterations and do one single excess update of σd_i at the end. Hence, the work associated with each update can be ammortized over the large updates, that require $O(\log k)$ work each. Thus, it is sufficient to bound the number of large excess updates.

Theorem 5.3.1 *The total number of large updates to excesses in a single phase is bounded by* $O^*(\epsilon^{-3}\delta^{-2}kmn^2)$.

Proof: Let β be defined as in condition (5.5), and consider a potential function

$$\Gamma = \sum_{i, vw \in E} \max\left\{-\beta - 2\sigma, \frac{Ex_i(f, vw)}{d_i}\right\}^2$$

Let Ψ_0 denote the value of Ψ at the start of the phase. The fact that at the start of the phase, $\log[\Phi_1/(2mk)] \leq \Psi_0$ together with the fact that $\Psi_0/\alpha \geq 2\sigma$, implies that the initial value of Γ is bounded by $O(mk\Psi_0^2/\alpha^2 + mk\beta^2)$. Increase in Γ due to increase in demands by σd_i^* during an iteration is bounded by

$$O(k\sigma(\Psi_0/\alpha + \beta + \sigma)) = O(k\sigma(\Psi_0/\alpha + \beta)).$$

Thus, the total increase in Γ throughout a phase is bounded by $O(k(\Psi_0/\alpha + \beta))$. Note that each time we move σd_i^* amount of excess across an edge or within a node, the decrease in Γ is $\Omega(\sigma^2)$, and hence the total number of updates can be bounded by

$$\frac{1}{\sigma^2} O(mk\Psi_0^2/\alpha^2 + mk\beta^2 + k(\Psi_0/\alpha + \beta))$$

and the claim follows from the observation that $\Omega(\frac{\epsilon}{m}\alpha) \leq \Psi_0 \leq O(\alpha)$. ▮

Thus, we have the following Theorem:

Theorem 5.3.2 *The total number of operations in a single phase of* ALGORITHM-2 *can be bounded by* $O^*(\epsilon^{-3}\delta^{-2}kmn^2)$.

5.4 Bounding the number of phases of Algorithm-2

In this section, we analyze the behavior of ALGORITHM-2 during a single phase and show a bound on the number of phases needed for obtaining an (ϵ, δ)-approximate preflow. There are several important differences between ALGORITHM-1 and ALGORITHM-2. First, instead of exact rebalancing of excesses inside each node, we execute an approximate rebalancing. Second, we ignore commodities on edges with small potential differences. Third, we do not compute flow increment that leads to the largest reduction in the potential function.

First, we will show that the approximate rebalancing and ignoring commodity/edge pairs with small potential difference does not increase the number of phases by more than a constant factor.

Lemma 5.4.1 *If redistribution of excesses in Step 3 of* ALGORITHM-1 *is done within an additive error of $2\sigma d_i^*$ and if we limit Step 2 to include only those edges and commodities that satisfy condition (5.5) then the increase in Φ during a single iteration is bounded from above by:*

$$O\left(\sigma^2\alpha^2\frac{n}{m}\right)\Phi_1 + \frac{\gamma}{1+\delta}\sigma\tilde{\Phi}_1.$$

Proof: The main ideas in the proof are that relatively small imperfections during rebalancing in Step 3 add-up to only a minor increase in the potential function and that we do not lose much by ignoring edge-commodity pairs that do not give large reduction in the potential function.

Step 3 of ALGORITHM-2 redistributes excesses inside nodes within an additive error of σd_i^*. That is:

$$\begin{aligned}
&\forall i, v \in V, w, w' \text{ adjacent to } v \text{ either :}\\
&\qquad |Ex_i(f, vw) - Ex_i(f, vw')| \leq 2\sigma d_i^*\\
&\text{or } \; -\beta d_i^* \geq \max\{Ex_i(f, vw), Ex_i(f, vw')\}.
\end{aligned}$$

Since for $x \leq 1$ we have $e^x - 1 \leq 3x$, the contribution to the first term in (5.2) of a flow path P of value $f_i^*(P)$ can be bounded by:

$$O\left(\sum_{\substack{wv\in P\\ vw'\in P}} \alpha^2\sigma^2 \min\left(\phi_i(f,w,v), \phi_i(f,v,w')\right)\frac{f_i^*(P)}{d_i^*}\right) + O\left(\sum_{\substack{wv\in P\\ vw'\in P}} \alpha\sigma\exp(-\alpha\beta)\frac{f_i^*(P)}{d_i^*}\right).$$

If we consider the error due to disregarding commodity/edge pairs that have very small excesses i.e.

$$Ex_i(f, vw) \leq -\beta d_i$$

then the associated error can be bound by

$$O\left(\sum_{\substack{wv \in P \\ vw' \in P}} \alpha\sigma \exp(-\alpha\beta) \frac{f_i^*(P)}{d_i^*}\right).$$

Consider the error introduced by disregarding commodity/edge pairs that do not satisfy

$$Ex_i(f, vw) \geq Ex_i(f, wv) + 4\sigma d_i^*. \tag{5.6}$$

Again using the fact that for $x \leq 1$, we have $e^x - 1 \leq 3x$, the error associated with edge vw, commodity i and path P can not exceed:

$$\alpha\sigma\left(\exp\left(\alpha\frac{Ex_i(f, vw)}{d_i^*}\right) - \exp\left(\alpha\frac{Ex_i(f, wv)}{d_i^*}\right)\right)\frac{f_i^*(P)}{d_i^*} \tag{5.7}$$

$$\leq \alpha\sigma \exp\left(\alpha\frac{Ex_i(f, wv)}{d_i^*}\right)(\exp(4\alpha\sigma) - 1)\frac{f_i^*(P)}{d_i^*}$$

$$\leq O\left(\alpha^2\sigma^2 \exp\left(\alpha\frac{Ex_i(f, wv)}{d_i^*}\right)\frac{f_i^*(P)}{d_i^*}\right). \tag{5.8}$$

By summing up over all the flow paths and over all the commodities, substituting the value of β, and using the fact that $\Phi_1 \geq 2mk$, we see that each of the three different types of error can be bounded by

$$O(\alpha^2\sigma^2\frac{n}{m}\Phi_1).$$

Thus, increase in Φ during a single iteration remains bounded by

$$O\left(\sigma^2\alpha^2\frac{n}{m}\right)\Phi_1 + \frac{\gamma}{1+\delta}\sigma\tilde{\Phi}_1.$$

∎

We shall use the notation $Ex + \Delta f$ to indicate the excesses that result after updating the flow by amount Δf. The increase in potential function due to incrementing the demand depends only on the excesses at the beginning of the iteration and is independent of the technique used for incrementing the flow. Hence we need to bound only the change in potential function due to the flow updates. We first claim that using $\tilde{Ex}$ instead of Ex does not adversely affect the potential function reduction. The proof of the following Lemma is essentially identical to the proof of Lemma 5.4.1.

Lemma 5.4.2 *If f^* is the optimal solution then*

$$\Phi(\tilde{Ex} + \sigma f^*) - \Phi(\tilde{Ex}) \leq O\left(\sigma^2 \alpha^2 \frac{n}{m}\right) \Phi_1 + \frac{\gamma}{1+\delta} \sigma \tilde{\Phi}_1.$$

Lemma 5.4.3 *The change in potential function due to the flow increment Δf used by* Algorithm-2 *can be bounded by*

$$\Phi(Ex + \Delta f) - \Phi(Ex) \leq \Phi(\tilde{Ex} + \sigma f^*) - \Phi(\tilde{Ex}).$$

Proof: Since for all i and $vw \in E$ the flow Δf satisfies $0 \leq \Delta f_i(vw) \leq \sigma d_i$ and the excesses satisfy condition (5.5), we know that $\Phi'(Ex + \Delta f) \geq \Phi'(\tilde{Ex})$. Using first-order Taylor series expansion we get

$$\Phi(Ex + \Delta f) - \Phi(Ex) \quad \leq \quad -\Delta f^T \Phi'(Ex + \Delta f) \leq -\Delta f^T \Phi'(\tilde{Ex}).$$

Furthermore, since Δf is the optimal solution in the space (that includes σf^*) that maximizes the linear objective $\Delta f^T \Phi'(\tilde{Ex})$ we have

$$-\Delta f^T \Phi'(\tilde{Ex}) \quad \leq \quad -\sigma f^{*T} \Phi'(\tilde{Ex}) \leq \Phi(\tilde{Ex} + \sigma f^*) - \Phi(\tilde{Ex}).$$

Hence the result follows. ∎

Combining Lemmas 5.4.2 and 5.4.3 we conclude that the increase in the potential function for the modified algorithm is a constant multiple of the bound obtained for Algorithm-1. Hence we have the following theorem.

Theorem 5.4.4 Algorithm-2 *terminates in $O(\delta^{-1} \log(m\epsilon^{-1}))$ phases.*

Using the above theorem together with Theorem 5.3.2, we get the following claim:

Theorem 5.4.5 *An (ϵ, δ)-approximation to the minimum-cost multi-commodity flow problem can be obtained in $O^*(\epsilon^{-3}\delta^{-3}kmn^2)$ time.*

Chapter 6

Routing and Admission Control of Virtual Circuits

6.1 Introduction

The wide spectrum of new consumer services, like video-on-demand, video teleconferencing, etc. expected to be offered on emerging high speed Broadband Integrated Services Digital Networks (B-ISDN), will tax network resources despite rapid technology advances. As a result, the allocation of network resources is of critical importance. The industry consensus is that B-ISDN will be based on the Asynchronous Transfer Mode (ATM) protocol. For ATM, each communication stream between a source and a destination uses a virtual circuit, which is a fixed path through the network. Most applications like voice and video need quality-of-service (QOS) guarantees. As a result, such applications must reserve resources along their virtual circuit paths. Resource reservation, in turn, implies the need for admission control to ensure that resources are not reserved beyond those that are physically available. Thus B-ISDN requires fast and efficient on-line *routing* and *admission control* algorithms.

Generally, the objective of a routing algorithm is to maximize the throughput of the network without violating resource constraints. Many factors complicate routing and admission control decisions. First, the decisions must be made *on-line* without knowledge of future requests. Second, the current state of the network may not be available, thus the routing decision may be based on static or inaccurate state information. Third, the decision must be made in real time i.e., it has to adhere to call setup time requirements.

The problem of bandwidth management in the context of circuit-switched networks has been studied extensively. Until recently, most of the analysis and design effort concentrated on achieving provably good performance under the assumption that the arrival pattern of virtual circuit requests can be described by a simple probabilistic model with known parameters. Recently, a new framework was proposed that leads to routing strategies that can achieve provably good performance without relying on any assumptions about the probabilistic behavior of the traffic [19, 5, 18, 6, 7]. The first step in developing such strategies was to develop an appropriate way to compare these strategies one to another. The idea, proposed in [19, 5], is to use the concept of the *competitive ratio*. Informally, the competitive ratio measures how much the online algorithm loses in performance as a result of not making optimum decisions. In particular, it measures how much is lost due to lack of knowledge of future requests.

In this chapter[1], we see how the idea of using continuous techniques for combinatorial optimization can be extended to the on-line setting to design algorithms with good competitive ratio against an omniscient off-line adversary. We consider the problem of on-line routing of virtual circuits in circuit-switched networks and see how some of the ideas from continuous techniques for multi-commodity flow can be used to design competitive algorithms for this problem. The intuition for our routing algorithm comes from viewing the problem as an on-line variant of the multi-commodity flow problem. In the routing context, commodities correspond to virtual circuits, and demands to the requested bandwidth, and capacity to the link bandwidth. There are several algorithms for approximate multi-commodity flow problems that are based on repeated rerouting of flow on shortest paths based on an exponential length function [47, 49, 67]. These ideas have been used to construct dynamic online routing strategies.

One such competitive strategy for *throughput-maximization* in general network topologies that uses this intuition of multi-commodity flow was developed by Awerbuch, Azar, and Plotkin [6]. In this chapter, we propose a distributed routing and admission control algorithm (EXP) for Switched Virtual Circuits (SVCs), which is an adaptation of the centralized worst-case algorithm in [6] to the more realistic distributed average-case situation. The algorithm works by choosing an appropriate state-dependent metric to determine the "cost" of each link, based on the *local knowledge* about the global state of the network. Admission control is done by rejecting virtual circuits for which there is no path with sufficiently low cost and sufficient available capacity (bandwidth). If an appropriate path exists,

[1]This chapter represents joint work with Rainer Gawlick, Serge Plotkin, and K.G. Ramakrishnan [21].

it is used to route the circuit. Otherwise, the request is rejected. The EXP algorithm is described for the case of SVCs with known holding times. The issue of using statistical information about the holding times instead is addressed in Chapter 7.

We have conducted extensive simulation experiments on realistic topologies to compare the EXP algorithm to several other techniques. For sake of comparison, we consider a simple greedy minimum hop strategy in which we route on a path between the source and destination which has the required capacity and uses as few links as possible. The simulations indicate the following.

- A direct application of the algorithm in [6], using the parameters suggested in that paper gives results which are worse than those based on greedy minimum hop.

- Our EXP algorithm performs significantly better than techniques like greedy minimum hop and minimum hop with trunk reservation (e.g. the algorithm of Sibal and DeSimone [72]).

- Unlike minimum hop based techniques, the performance of our EXP algorithm is not dramatically affected by outdated and inaccurate state information. Thus, its relative performance advantage is even greater in the distributed setting.

6.1.1 Previous work

The problem of bandwidth reservation and management in the context of circuit switched networks has been studied extensively. The proposed admission control and routing strategies can be classified according to the optimization goal and according to the traffic model. Majority of the literature considered either the worst case, i.e. arbitrary sequences of requests generated by an adversarial process, or the following probabilistic model: For each pair of nodes u and v, the arrival of requests for circuits from u to v is represented by a Poisson process; processes associated with different node pairs are independent. The arrival rates are described by a fixed matrix Q, where q_{ij} is the rate of arrival of circuits from node i to node j. The circuits depart (terminate) according to an exponential distribution.

Traditionally, routing strategies were analyzed for the case where the traffic matrix is constant or changing sufficiently slowly so that it can be estimated. A class of *cost-based* routing algorithms have been studied for general networks in theoretical framework by Kelly [44] in the context of state-independent routing, and by Ott and Krishnan [65] in the context of state-dependent routing. Roughly speaking, these algorithms are based on the concept of costs that reflect the effect of a routing decision on the system performance. These costs are used to assign probabilities to different alternate routes. Since the costs are computed using an a priori known traffic matrix, these algorithms are in general not dynamic in the sense that they do not seek to exploit fluctuations in the traffic rates.

Ott and Krishnan [65] considered the routing and admission control as a Markov decision process with exponential number of states, and proposed several heuristics based on this representation. Substantial work was done in analyzing *symmetric networks* case, where the underlying network is a complete graph, and the traffic matrix is uniform. The main result of this work is development of dynamic state-dependent routing strategies based on the concept of *reservation* [43, 60, 58, 59, 57, 56, 23]. Roughly speaking, the idea is to assign each node pair a *primal* path and a set of *alternate* paths. Primal path is used as long as it has available capacity. When there is no capacity on the primal path, an alternate path is used if its current utilization is below some predefined reservation threshold. This is in contrast to the greedy admission control policy that would admit any virtual circuit as long as there is sufficient capacity to accommodate it. Intuitively, the reservation is beneficial since (when properly tuned) it will reject a virtual circuit request when the only available path for the virtual circuit uses resources that might be used more efficiently by future virtual circuits. Reservation-based strategies were shown to be very effective for existing telephone networks [3, 24].

An example of an existing admission control scheme based on trunk reservation is the Real Time Network Routing (RTNR) [4] used in the AT&T long distance network. Logically, this network is a complete graph. All calls are initially routed along the direct link and the overflow traffic uses the least busy 2-hop alternate path. The trunk reservation in RTNR is implemented by forbidding the use of a 2-hop alternate path if one of the links is loaded close to capacity. This least busy alternate routing with trunk reservation has been found to work well in practice in fully connected networks. In fact, Hunt and Laws [33] prove asymptotic optimality of essentially this scheme for direct and 2-link paths in fully connected networks.

Recently, Sibal and DeSimone [72] have proposed a simple and practical alternate routing and trunk reservation strategy for general topologies. In their approach, each node pair is pre-assigned a primary route and a set of alternate routes. If feasible, a call is routed on its primary path. Otherwise, an alternate route is chosen subject to pre-calculated trunk reservation. The trunk reservation parameters are calculated based on the traffic matrix and statistical characteristics of the traffic such as arrival rate and holding times.

Extending reservation based routing to more general case (see e.g. [73]) faces two main problems. First, it is not clear how to choose primary vs. alternate paths in a general topology network. This is in contrast to the fully connected networks, where the notion of direct vs. alternative path is defined in a natural way. Second, all the analysis of these strategies, and in particular heuristics for finding the reservation thresholds, is based on the assumption that the traffic on different links is independent. This assumption seems reasonable in highly connected networks, where overwhelming majority of the traffic is using single-hop paths. This assumption is not applicable in networks where a significant fraction of the traffic is using two or more hops per path.

The disadvantage of both shadow-cost and the reservation-based algorithms mentioned above is that they require knowledge of the traffic matrix in advance; this matrix is used to compute the reservation parameters or the shadow costs. Thus, it is not clear how to make these algorithms dynamically exploit the fluctuations in the traffic rate.

Strategies with provable bound on performance for the *worst-case* request sequences were considered in [5, 6, 7]. Clearly, without restricting the request sequences, one can not put a bound on the acceptance rate. Thus, they proposed to use *competitive ratio* [74, 55] to measure the performance. A competitive ratio is defined with respect to two strategies; both strategies have to deal with the incoming requests, which in our case are requests for establishing virtual circuits. One is the *online* strategy whose performance we want to measure. The online strategy has to deal with the requests one-by-one and can not use any knowledge of the future requests. The other is the best possible (in terms of performance) *off-line* strategy; this algorithm is omniscient in the sense that it is assumed to have a *complete a priori knowledge* of the entire request sequence, including future requests. Informally, the competitive ratio measures how much the online algorithm loses in performance as a result of not making optimum decisions. In particular, it measures how much is lost due to lack of knowledge of future requests. Formally, the competitive ratio of a given online algorithm is defined as the supremum over all input sequences of the performance achieved by the optimum off-line algorithm and the performance achieved by the online algorithm. Roughly speaking, the statement that a particular strategy has competitive ratio α means that its performance is at most a factor of $1/\alpha$ of the performance of the best possible off-line algorithm.

Clearly, the "best" strategy with respect to the competitive ratio depends on the parameter chosen to measure the performance. Two natural measures are *throughput* and *congestion*. In the first, we measure the proportion of the routed (accepted) circuits. A variation on this measure is the total routed bandwidth-duration product, i.e. total throughput. In the second, we measure the maximum link congestion, i.e. maximum ratio of the allocated bandwidth on a link to its total bandwidth. Clearly, to use this measure we have to disallow rejections.

A competitive strategy for the *congestion-minimization* model was developed in [5]. This strategy achieves a competitive ratio of $O(\log n)$ for permanent (i.e. infinite holding time) virtual circuits (PVCs), where n is the number of nodes in the network. It was extended to the case of finite holding time circuits (SVCs) where the holding time of a circuit becomes known upon its arrival in [11]. For this case, the strategy achieves $O(\log nT)$ competitive ratio, where T is the maximum holding time. Competitive strategies for the *throughput-maximization* model for the case where the network consists of a single link were developed in [19] line network was considered in [18], and general topology network was considered in [6]. In particular, the strategy in [6] achieves a competitive ratio of $O(\log L)$ for PVC routing and an $O(\log LT)$ ratio for SVC routing in case the duration of a circuit becomes known upon its arrival, where L is the maximum number of hops taken by a virtual circuit. Practical routing and admission control strategies derived from the competitive strategies were studied in [20] for the PVC case. (See [66] for the survey of competitive routing strategies.)

The main disadvantage of the above competitive strategies is that they provide very weak guarantees on performance. For example, for a sequence of requests that can be completely satisfied by an optimum off-line strategy, the strategy in [6] guarantees rejection ratio of at most $1 - \Omega(1/\log(LT))$. Although one can argue that the value of L is relatively small, the value of T can easily be in the hundreds or thousands, implying that the only guarantee that we get is that the rejection rate is slightly less than 100%, which is not very useful from the practical point of view.

The main advantage of the algorithms in [6, 5] is that they achieve provably good performance without relying on probabilistic assumptions on the offered traffic. Unfortunately, the fact that algorithms in [5, 6] do not rely on any probabilistic assumptions has the effect that they are tuned to deal with worst case situations, thus making these algorithms impractical. The EXP algorithm described in this chapter, can be viewed as a practical adaptation of the theoretical framework developed in [6, 5]. Our EXP algorithm has the advantages of both trunk reservation and shadow cost based algorithms. The EXP algorithm routes along the shortest paths, where the costs are *dynamically* assigned based on the *instantaneous* state of the system. The reservation is implemented by rejecting circuits for which sufficiently inexpensive paths were not found. Note that the EXP algorithm does not restrict routing to a prespecified set of paths. Instead, it provides a dynamic decision policy that chooses between all the available paths according to the currently known information on the load, while giving preference to the shorter ones. Two of its main advantages are that it neither requires a priori knowledge of the traffic matrix, nor it requires computation of "direct" vs. "alternative" paths. As we have mentioned above, our simulations indicate that it outperforms the reservation scheme of [72].

In Section 2, we describe the algorithm in [6] and discuss its theoretical performance guarantees. We show how the algorithm can be adapted to work well in practice and provide simulation results for the adapted algorithm. Section 3 shows how to modify our algorithm for a distributed setting and provides simulations results for that setting. Section 4 discusses several implementation issues.

6.2 Centralized On-line Algorithm for SVC Routing

6.2.1 Exponential-cost based routing

In this section, we describe our routing algorithm, which we refer to as the "EXP" algorithm, and analyze its performance in the centralized setting. The EXP algorithm can be viewed as a modified version of the exponential cost based algorithm of [6], which we will refer to as "AAP", and thus we start by describing the AAP algorithm and then show that the EXP algorithm significantly outperforms it under realistic assumptions on the offered traffic.

The network is represented by a capacitated (directed or undirected) graph $G(V, E, u)$. Let n denote the number of nodes in the graph. The capacity $u(e)$ assigned to each edge $e \in E$ represents bandwidth available on this edge.

The exponential-cost based algorithm of [6] is shown in Figure 6.1. Upon receiving request i, represented by $(s_i, t_i, r_i, T_i^s, T_i^f)$, the AAP algorithm tries to allocate a route of capacity r_i from originating node s_i to the destination node t_i starting at time T_i^s and ending at time T_i^f. For simplicity, we assume that the routing is done at time T_i^s. The algorithm either accepts the request, allocating an appropriate route, or rejects the request. The goal

ROUTE($s_j, t_j, T_j^s, T_j^f, r_j$):
  $\forall \tau, e \in E:\ c_e(\tau, j) \leftarrow u(e)(\mu^{\lambda_e(\tau,j)} - 1)$; (*)
  if $\exists$ path P in $G(V,E)$ from s_j to r_j s.t. the cost of P
    $\sum_{e \in P} \sum_{\tau} \frac{r_j}{u(e)} c_e(\tau, j) \leq LrT$
  then route the requested circuit on P with the minimum cost, and set:
    $\forall e \in P, T_j^s \leq \tau \leq T_j^f$,
      $\lambda_e(\tau, j+1) \leftarrow \lambda_e(\tau, j) + \frac{r_j}{u(e)}$
  else reject the connection

Figure 6.1: The original AAP routing algorithm

of the algorithm is to maximize the total number of routed requests.[2] Let $T_i = T_i^f - T_i^s$ denote the "holding time" of the circuit, T denote the maximum possible T_i, and r denote the maximum request bandwidth (rate) r_i.

The routing decision is based on the current information about the current and future *load* on the edges of the network. The load is measured relative to the edge capacity $u(e)$. Let P_i denote the route used to satisfy the ith request. The load on edge e at time τ as seen by the routing algorithm when routing the kth circuit is defined as follows:

$$\lambda_e(\tau, k) = \sum_{\substack{e \in P_i, i<k \\ \tau \in [T_i^s, T_i^f]}} \frac{r_i}{u(e)}.$$

After computing the load, the next step is to compute the *exponential cost*. The cost of edge e at time τ as seen by the routing algorithm when routing the kth circuit is defined by

$$c_e(\tau, j) = u(e)(\mu^{\lambda_e(\tau,j)} - 1),$$

where μ is a parameter.

The main result in [6] is that choosing $\mu = 2nTr/r_{\min} + 1$ guarantees that for any sequence of requests where for all i we have $r_i \leq \min_e u(e)/\log \mu$, the total number of requests accepted by the AAP algorithm is within $O(\log \mu) = O(\log nTr/r_{\min})$ of the maximum total requests that can be routed by the best *off-line* algorithm, i.e. an algorithm that knows all of the requests in advance. In fact, it is trivial to see that if the maximum allowed hop count is L, then the performance bound is improved to $O(\log LTr/r_{\min})$

[2] It is easy to modify the algorithm to optimize a general "profit" measure, where each routed request bring a predefined profit. For example, such profit can be given by the sum of $\rho_i = r_i T_i$ over all the accepted circuits.

6.2.2 Adding probabilistic assumptions

In [6], it is shown that the performance achieved by the AAP algorithm is within a constant factor of the best possible performance achievable by any online algorithm, i.e. any online algorithm can be presented with a network and a request sequence such that the number of requests accepted by this algorithm will be below $O(1/\log(LTr/r_{\min})$ fraction of the total requests accepted by the best off-line algorithm. In this section, we show that under realistic assumptions about the input sequence (and without any assumptions about the network) it is possible to get a much better performance.

Intuitively, the performance of the AAP algorithm is low due to the fact that it uses high value of μ. Unfortunately, simply reducing μ causes the AAP algorithm to accept circuits in excess of the available capacity, which is not allowed. In other words, the AAP algorithm is correct only for sufficiently large μ.

A natural approach is to reduce μ and to modify the AAP algorithm by explicitly forbidding the use of saturated edges. More precisely, after the line marked (*) in Figure 6.1, we compute a *modified cost function* c'

$$(6.1)\quad \forall \tau, e \in E : \textbf{if } (\lambda_e(\tau, j) + r_j/u(e)) \geq 1 \qquad \begin{array}{l} \textbf{then } c'_e(\tau, j) \leftarrow \infty \\ \textbf{else } c'_e(\tau, j) \leftarrow c_e(\tau, j) \end{array}$$

and then use c' instead of c in the rest of the algorithm.

We will refer to the modified algorithm as the EXP algorithm. Observe that the EXP algorithm produces legal routing decisions for any value of μ. Moreover, it is easy to see that by keeping μ the same as for AAP algorithm, the two algorithms behave identically.

Unfortunately, reducing μ invalidates the proof of the AAP algorithm. In particular, the corner-stone assumption that all the rejected circuits connect distant (in terms of the cost c_e) nodes is no longer valid. In what follows, we will show a modified proof that is correct under certain assumptions on the offered traffic.

In order to analyze the performance of the EXP algorithm, consider some set of requests and divide them into sets according to the way they were treated by the algorithm. Let P_i^* denote the path chosen by the optimum off-line algorithm for routing the ith request. The requests R that were rejected by EXP but accepted by the optimum off-line algorithm fall into two types: R_1 consists of requests i where the cost (when routing i) of the path P_i^* with respect to c exceeds LrT and $R_2 = R - R_1$. Observe that $i \in R_2$ implies that the cost of P_i^* does not exceed LrT but P_i^* passes through at least one saturated edge.

Intuitively, one can view the EXP algorithm as if it consists of two parts: one part proposes a low-cost path, and the other part checks whether there is available capacity on this path. Using this description, R_1 corresponds to the requests for which sufficiently low-cost path was not found, and R_2 corresponds to the requests that were rejected because every low-cost path has a saturated edge.
The proof in [6] can be easily adapted to show a bound on $|R_1|$.

Lemma 6.2.1 ([6]) *Let A be the set of circuits accepted by the EXP algorithm then $|R_1| \leq O(\log \mu)|A|$.*

The main difficulty lies in bounding the value of $|R_2|$. Since the cost is exponential in the load, paths through saturated (or close to saturated) edges will, in general, have high cost. Observe, that unless μ is chosen to be at least $\Omega(LT)$, R_2 is not empty, since the cost of a saturated edge is not high enough to prevent us from trying to use it.

Consider a circuit that was rejected because there was no sufficiently short path with respect to costs c', while there was a short path with respect to costs c. This can happen only if there is an edge on this path that was close to saturation. Let $\{C_e(\delta), 0 \leq \delta \leq T\}$ be a random vector that denotes the costs as computed by the algorithm when trying to route a circuit, where δ corresponds to the number of time steps into the future with respect to the arrival time. We will use $\Lambda_e(\delta)$ to represent the corresponding load on edge e. Observe that, by construction, $\delta > \delta'$ implies that $C_e(\delta) \leq C_e(\delta')$. Let $\tilde{C}_e(\delta)$ be the random vector $C_e(\delta)$ conditioned on the event that $C_e(0) = u(e)(\mu - 1)$, i.e. the event that the arriving circuit sees that e is saturated.

Consider the requests in R_2. As we have mentioned above, request i is in R_2 if and only if the cost of the corresponding P_i^* computed when routing i does not exceed LrT and at the same time, P_i^* contains a saturated edge. Thus, the probability of a request in $R_2 \cup R_1$ ending up in R_2 is bounded by:

$$\mathcal{P} = \max_e \Pr[\sum_{0 \leq \delta < T_i} \tilde{C}_e(\delta) \leq LrT].$$

Let OPT denote the set of requests satisfied by the optimum off-line algorithm.

Lemma 6.2.2

$$\frac{E[|OPT|]}{E[|A|]} \leq O(\frac{\log \mu}{1 - \mathcal{P}}).$$

Proof: From the definition of $\mathcal{P}$, we have:

$$E[|R_2|] \leq \mathcal{P} \cdot (E[|R_1| + |R_2|])$$

This implies:

$$E[|R_2|] \leq \frac{\mathcal{P}}{1-\mathcal{P}} E[|R_1|].$$

The claim follows from the fact that $|R_1| \leq O(\log \mu)|A|$ and the fact that $|OPT| \leq |A| + |R_1| + |R_2|$. ∎

When is $\mathcal{P}$ small ? One family of input traffic sequences for which $\mathcal{P}$ is small is when the expected cost of an edge over the duration of T_i of the ith circuit is sufficiently larger than LrT. More precisely, using Chebyshev inequality, we get the following lemma:

Lemma 6.2.3 *If for all i, we have $E[\sum_{\delta \leq T_i} \tilde{C}_e(\delta) - LrT] \geq 2\sqrt{VAR[\sum_{\delta \leq T_i} \tilde{C}_e(\delta)]}$ then $\mathcal{P}$ can be bounded away from 1 by a constant.*

There is a large general class of cases for which Lemma 6.2.3 gives $O(\log \mu)$ competitive performance for significantly smaller values of μ. For example, consider the following case:

- All circuits have uniform traffic rate $r \leq \min_e \frac{u(e)}{16 \log^2 L}$.
- The conditional probability of a circuit that is alive at time $\delta = 0$ finishing at time $\delta \in [0, T-1]$ is at most $\frac{\alpha}{T}$, where $\alpha > 0$ is a constant, independent of all the other circuits that were alive at $\delta = 0$.

Lemma 6.2.4 *Under conditions outlined above, there exists μ, where $\log \mu = O(\log L)$ is independent of T, such that the number of requests accepted by the EXP algorithm $E[|A|]$ is within $O(\log \mu)$ factor of the expected number of requests $E[|OPT|]$ accepted by the optimum off-line algorithm.*

We show analysis of a simple special case, that corresponds to the simulations discussed in the next section. Consider the following case: arrival times of routed circuits are Poisson-distributed, r_i and T_i are the same for all i. We will refer to this as the "simplified case" and will show that a significantly smaller $\mu = O(L)$ (independent of T) is sufficient for this case.

Since T_i and r_i are the same for all i, we can simplify notation by assuming that $r_i = 1, T_i = T$, and denoting $\tilde{C}_e = \sum_{0 \leq \delta \leq T} \tilde{C}_e(\delta)$. We also assume $\log^2 L \leq \min_e u(e)/2$.

Lemma 6.2.5 *The mean and variance of C_e are bounded as follows:*

$$E[C_e] \geq T u(e) \left(\frac{\mu^{\frac{u(e)+1}{u(e)}} - 1}{(\mu^{\frac{1}{u(e)}} - 1)(u(e)+1)} - 1 \right),$$

$$\sigma^2(C_e) \leq T^2 u(e)^2 \left(\frac{\mu^{\frac{2(u(e)+1)}{u(e)}} - 1}{(\mu^{\frac{2}{u(e)}} - 1)(u(e)+1))} - \frac{\mu^{\frac{2u(e)+1}{u(e)}} - 1}{(\mu^{\frac{1}{u(e)}} - 1)(2u(e)+1))} \right).$$

Proof: Since we assume Poisson arrival of circuits, the number of circuits still alive at time $\delta \in [0, T-1]$ given that they were alive at $\delta = 0$ is binomially distributed between 0 and $u(e)$ with probability $(1 - \delta/T)$ for each $\delta \in [0, T-1]$ i.e. for $k = 0, \ldots, u(e)$ and $t \in [0, T-1]$

$$\Pr\left(\Lambda_e(t) = \frac{k}{u(e)}\right) = \binom{u(e)}{k} (1 - \frac{t}{T})^k (\frac{t}{T})^{u(e)-k}.$$

The results for mean and variance stated in the lemma follow from this observation. ∎

Now we can prove the lemma that states that the smaller value of μ is sufficient.

Lemma 6.2.6 *There exists μ where $\log \mu = O(\log L)$, such that in the "simplified case" defined above, the number of requests accepted by the EXP algorithm $E[|A|]$ is within $O(\log \mu)$ factor of the expected number of requests $E[|OPT|]$ accepted by the optimum off-line algorithm.*

Proof: Immediate from Lemmas 6.2.5, 6.2.3, and 6.2.2. ∎

6.2.3 Simulation Results

In the previous section, we presented the theoretical background underlying on-line routing and admission control algorithms based on exponential costs in general, and the EXP algorithm in particular. Since it is impossible to analyze the performance of such algorithms analytically for realistic networks and traffic patterns, we conducted extensive simulation experiments. The main goal of our simulations was to compare the EXP algorithm to several algorithms which are based on minimum hop ideas. We have used two different realistic network topologies - An existing commercial topology, and the NSFNet backbone T3 topology (as used in [72]).

An Existing Commercial Topology

Our simulations use an existing commercial network that consists of 25 nodes and 61 links. The topology is pictured in Figure 6.2. The capacities of the links are all chosen to be 155 Mbps, which corresponds to SONET OC-3 service. The virtual circuits all require 1 Mbps in both directions and have a holding time of 30 minutes. Virtual circuit requests arrive as a Poisson process. The proportion of the aggregate requests that goes between every source destination pair is determined by a traffic matrix that corresponds to the actual current traffic on the network whose topology we are using.

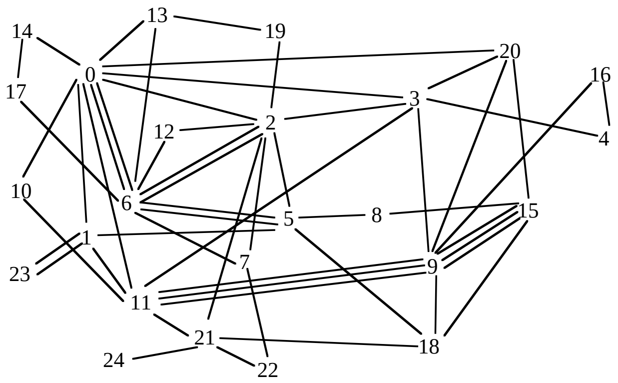

Figure 6.2: Topology for an existing commercial network.

In order to put the performance advantage of the EXP algorithm over minimum hop based algorithms into perspective, we wish to compute the performance of the optimum off-line algorithm. Unfortunately, this computation does not seem to be tractable. Instead, we compute a lower bound on the optimum rejection rate by solving a multi-commodity flow problem in which the objective function is to satisfy the maximum demand between node pairs without violating the capacity constraints, where the demand between node pairs is determined by the traffic matrix. In particular, the demand between nodes i and j is set to the average number of bits per second that are expected to be requested with i and j as the source and destination. It is easy to see that the solution of this optimization problem is indeed a lower bound on the rejection rate. However, this lower bound may be far off from the true optimum since it does not take into account variations in arrival rate due to statistical behavior. Furthermore, the lower bound can split the traffic from a virtual circuit over multiple paths. Unfortunately, it is not clear how far off our lower bound is nor how to compute a better approximation to the optimum rejection rate.

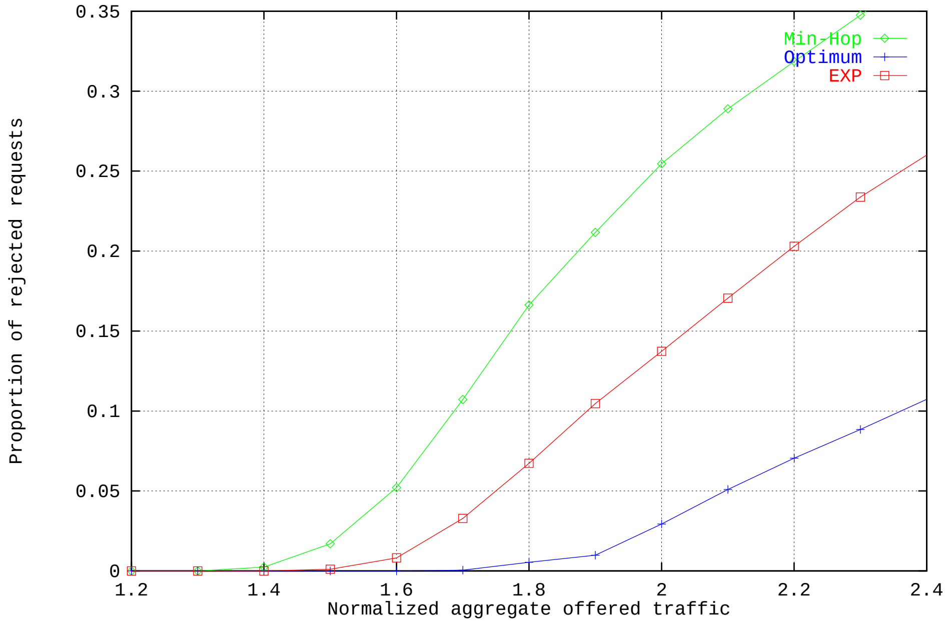

Figure 6.3: Simulation results for the centralized algorithms

Figure 6.3 compares the performance of the EXP algorithm with $\mu = 2.4$ to the performance of a simple minimum-hop algorithm and our lower bound on the performance of the optimum algorithm[3]. The X-axis gives the aggregate arrival rate of virtual circuits per second and the Y-axis gives the percentage virtual circuits rejected for that aggregate arrival rate. It can be seen that the EXP algorithm has a significant performance advantage over the minimum-hop algorithm for a wide range of arrival rates. The EXP algorithm can maintain a much higher arrival rate given a target rejection (loss) rate. For a target maximum rejection rate of 2%, the EXP algorithm can sustain an arrival rate that is 9.2% higher than the arrival rate that can be sustained by the minimum-hop algorithm. The relative performance advantage of the EXP algorithm increases with arrival rate. For example, the improvement for target maximum rejection rate of 20% is 17%.

The main parameter in the EXP algorithm is the base μ used to compute the exponential costs. Figure 6.4 explores the effect of various values of μ on the performance. Essentially, a larger value of μ implies a more conservative admission control policy in the sense that a

[3]The graphs do not include confidence intervals. Our results are very consistent over different seeds so we expect the intervals to be tight.

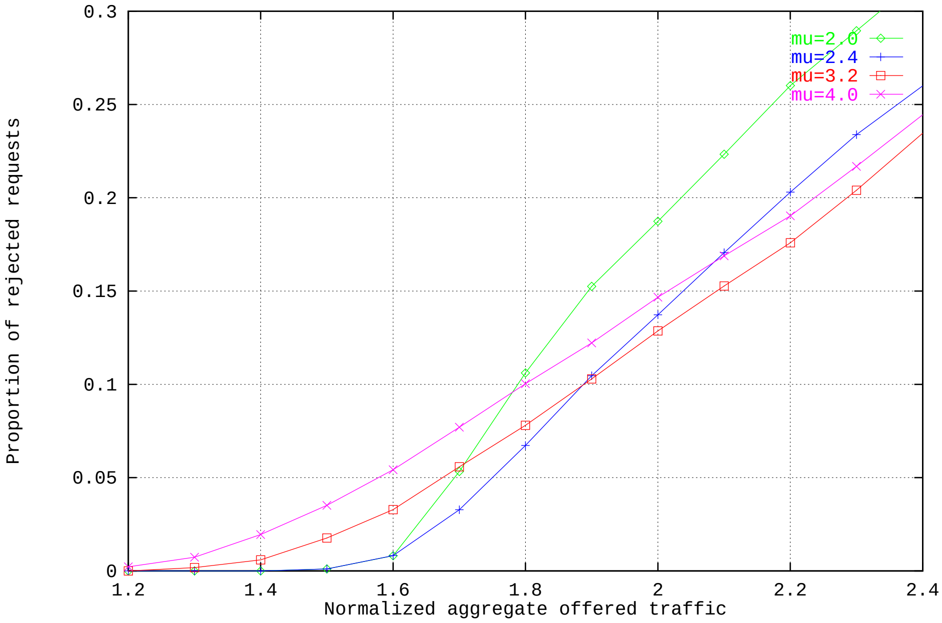

Figure 6.4: Determining μ

virtual circuit requiring a long path is less likely to be accepted when EXP uses a higher μ. Not surprisingly, EXP with a higher μ value performs better than EXP with a lower μ value at high arrival rates and worse at low arrival rates. In our simulations, the average path length is approximately two. Thus, the value for μ calculated by the analysis for "simplified case" in Section 6.2 is ≈ 4.5. Based on the results in Figure 6.4, this μ value is clearly too high. The reason that the analysis for the "simplified case" in Section 6.2, chooses a large μ, is that it attempts to be competitive with the optimal algorithm for *all* arrival rates. However, in practice we are only interested in arrival rates that lead to relatively low rejection rates. Therefore, the value of $\mu = 2.4$, which works well for arrival rates that lead to low rejection rates, is not directly predicted by the "simplified case" in Section 6.2. It is interesting to note that for the network considered in our simulations, the original AAP algorithm suggests a μ value which is in the thousands. It is clear that this would not lead to acceptable performance. (In fact, we have simulated the algorithm with this value of μ and found that it works best for arrival rates that cause rejection rates that exceed 70%.)

A natural variation of the minimum-hop algorithm is the *randomized* minimum-hop algorithm, where we choose one of the available minimum-hop paths uniformly at random.

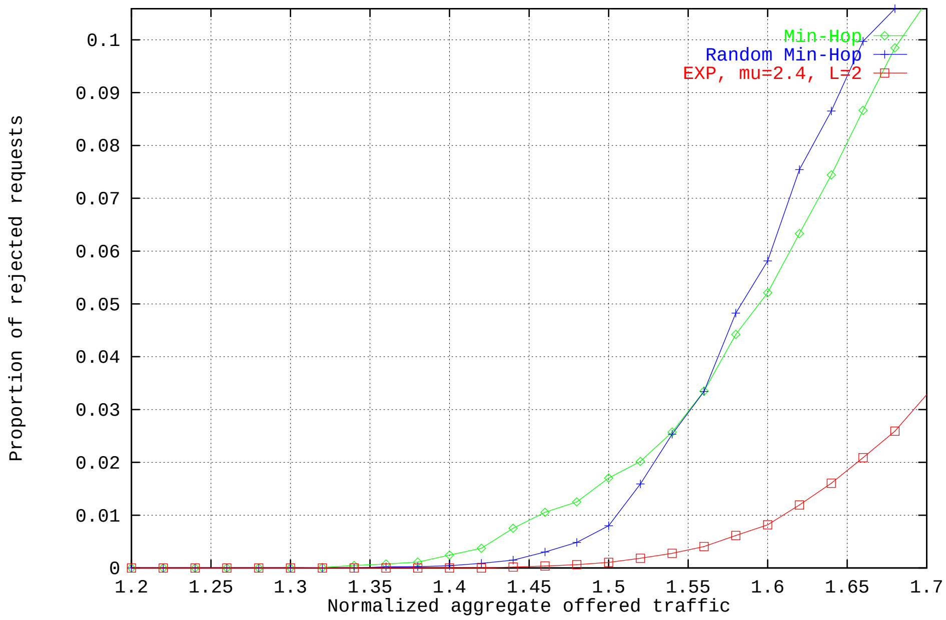

Figure 6.5: Comparing EXP with randomized minimum-hop

Not surprisingly, this algorithm works better than the simple minimum-hop algorithm for small loads. However, the randomized minimum-hop algorithm is still significantly outperformed by the EXP algorithm. This can be observed in Figure 6.5. Note that this figure covers only a small portion of the range of the aggregate offered traffic shown in Figure 6.3.

NSFNet T3 Topology

In this section, we describe results of simulating our algorithms on the NSFNet T3 backbone network. The definition of the network and of the traffic matrix was taken from the paper by Sibal and DeSimone [72]. For completeness, we give the topology and the traffic profile here as well.

The topology of NSFNet is depicted in Figure 6.6. This NSFNet configuration was extant in 1992. Each edge in the topology represents a 100Mbits bidirectional link. All arrivals were assumed to be medium picture quality video calls requiring 1Mb/sec.

The traffic matrix is shown in Table 6.1. Each entry denotes the traffic in Erlang. For our experiments, we generated exponentially distributed holding times with a mean of 100 seconds. This, together with the traffic profile in 6.1 defines the call arrival rate.

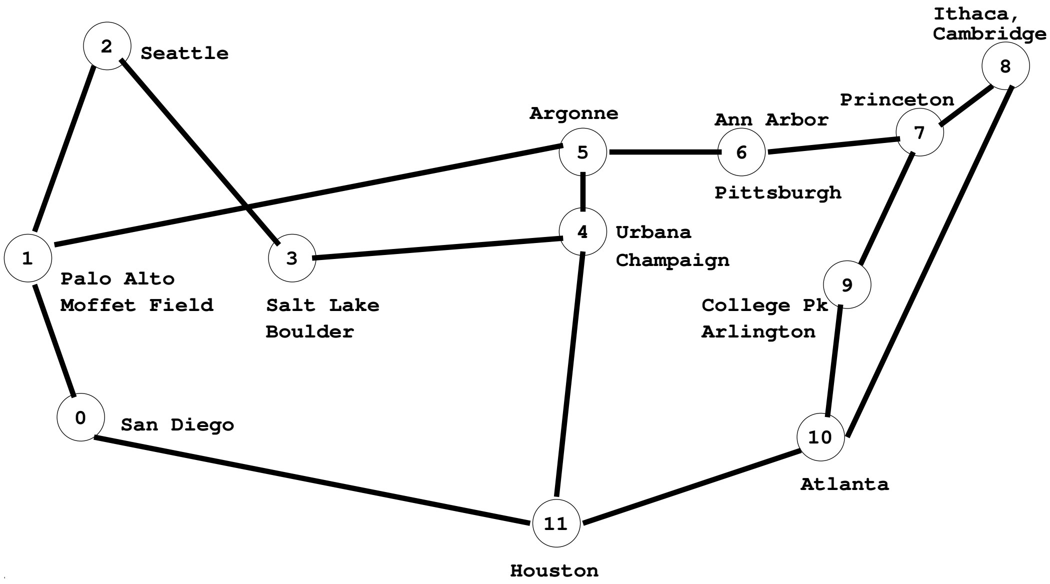

Figure 6.6: Topology for NSFNet T3 Backbone.

$$\begin{pmatrix}
0 & 4 & 2 & 3 & 6 & 0 & 7 & 1 & 9 & 5 & 2 & 3 \\
4 & 0 & 3 & 6 & 13 & 0 & 15 & 2 & 19 & 9 & 4 & 6 \\
2 & 3 & 0 & 2 & 5 & 0 & 6 & 1 & 7 & 3 & 1 & 2 \\
3 & 6 & 2 & 0 & 8 & 0 & 10 & 1 & 12 & 6 & 2 & 4 \\
7 & 14 & 5 & 9 & 0 & 0 & 24 & 3 & 31 & 15 & 6 & 9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 17 & 6 & 11 & 25 & 0 & 0 & 4 & 37 & 18 & 7 & 11 \\
1 & 2 & 1 & 1 & 3 & 0 & 3 & 0 & 4 & 2 & 1 & 1 \\
11 & 22 & 9 & 15 & 33 & 0 & 39 & 5 & 0 & 24 & 9 & 15 \\
5 & 9 & 4 & 6 & 14 & 0 & 16 & 2 & 21 & 0 & 4 & 6 \\
2 & 4 & 1 & 2 & 5 & 0 & 6 & 1 & 8 & 4 & 0 & 2 \\
3 & 6 & 2 & 4 & 8 & 0 & 10 & 1 & 12 & 6 & 2 & 0
\end{pmatrix}$$

Table 6.1: NSFNet Traffic Profile

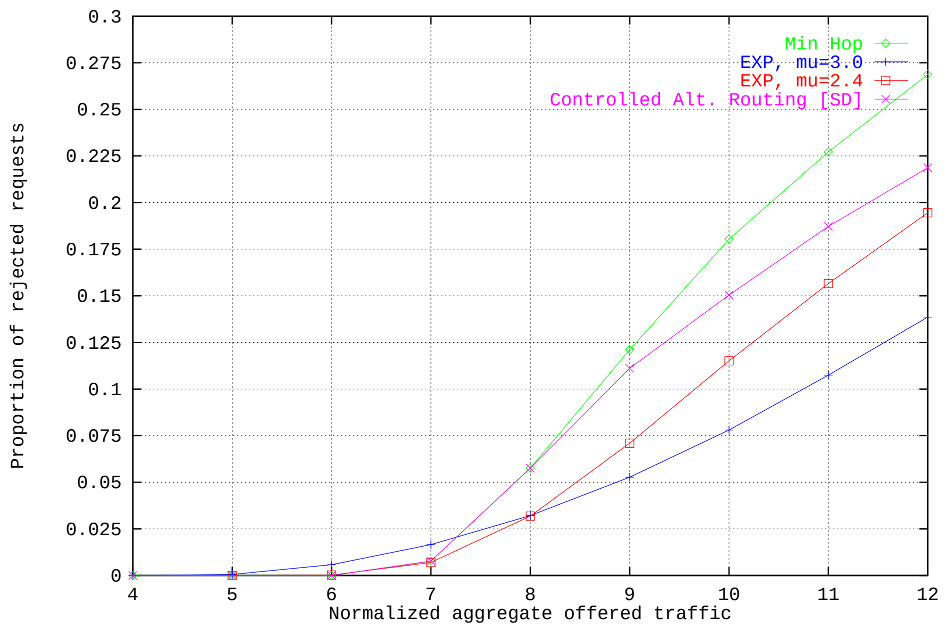

Figure 6.7: Comparative performance for Internet topology

The simulation results are presented in Figure 6.7. The figure shows comparative performance of the EXP, min-hop, and the reservation-based scheme used by Sibal and DeSimone (marked [SD] in the figure) on the internet topology. The X-axis corresponds to aggregate normalized traffic and Y-axis corresponds to the fraction of rejected circuits. The results for "controlled alternate routing" were copied from [72]. The figure shows that the "controlled alternate routing scheme" proposed in [72] significantly outperforms the min-hop algorithm, while the EXP algorithm outperforms this scheme for rejection rates above 2%. For example, if the cutoff point is taken as 5% rejection rate, the EXP algorithm with $\mu = 3.0$ can sustain an aggregate offered load of about 11% higher than the SD algorithm.

As for the commercial topology discussed in the previous section, the EXP algorithm behaves differently for different values of parameter μ. In general, higher μ produce better results for heavy traffic conditions, while low values of μ are best for lightly loaded networks. Figure 6.7 shows behavior of the EXP algorithm for two values of $\mu = 2.4$ and $\mu = 3.0$. It can be seen that $\mu = 3$ becomes a better alternative starting from about 3% rejection rate. An interesting open problem is whether it is possible to change μ dynamically depending on the traffic conditions.

6.3 Distributed Algorithm

Large scale networks cannot expect to have accurate state information available at every source node. As a consequence, such a large scale network must either make all routing decisions at a centralized site, or must make routing decisions locally based on static or inaccurate state information. In this section, we describe how the EXP routing scheme of Section 6.2 can be effectively adapted to a setting where only inaccurate state information is available.

Our focus on local routing decisions rather than on routing decisions at a centralized site is motivated by several systems considerations. In particular, a centralized solution is more likely to suffer from reliability problems since it has a single point of failure and cannot function properly in the presence of network partitions. Furthermore, fast virtual circuit setup is not possible since round trip communication with a potentially distant central node must precede the establishment of every circuit.

A key parameter of a distributed algorithms is the degree to which the local state information is inaccurate. This is largely a function of the system constraints. In order to ensure the applicability of our work to a broad range of system environments, we have adopted a very conservative system model that, for example, does not include any type of periodic state information exchange. Specifically, we permit a node to collect state information only for the links along the path on which it attempts to establish a virtual circuit. Thus, the state information gathering task can be carried by the control messages used during a virtual circuit setup. The overhead of this scheme is clearly very low.

The motivation for our focus on local routing decisions mentions the need for fast virtual circuit setup. In discussions with several switch manufacturers, we have learned that setup time requirements of the order of several milliseconds is not uncommon. In a national network, setup times of that order of magnitude can only be achieved by pipelining the data packets immediately behind the setup packet. In such environments, multiple setup attempts may not be possible. Fortunately, our algorithm, in contrast to a minimum hop based algorithm, achieves good performance even if it is restricted to a single setup attempt.

To convert the centralized EXP algorithm described in Section 6.2 to a distributed setting, we proceed as follows. Each source maintains a local view of the network state. That view is constructed from the state of the incident links and from information about other links gathered during circuit setup. When a virtual circuit request arrives, the EXP algorithm is invoked. It either rejects the request outright, or it returns a path, in which case we attempt to use this path to route the circuit.

If the path has sufficient capacity to accommodate the virtual circuit, the circuit is routed[4]. The acknowledgement packet that informs the source node whether or not the circuit was indeed routed collects the current state information of each link along the circuit's path. That state information is used to update the local view of the network at the source node. If the acknowledgement packet indicated that the circuit was not routed due to insufficient available capacity, the algorithm rejects the request or if it is allowed to do multiple attempts it tries to route the circuit again as if it were a new request.

We distinguish the distributed algorithms according to the number of allowed tries. The next section presents results for a single try and 3-tries variants. Note that the state information returned by the acknowledgement packet(s) may involve many bits since it includes information about the future state of the links. In Section 6.4, we show how to reduce the number of required bits to encode the state information in the acknowledgement packet(s).

6.3.1 Simulation Results

The results of our simulations for the existing commercial topology are summarized in Figure 6.8. This figure compares the performance of several variants of the EXP algorithm (centralized, distributed with 1 attempt, distributed with 3 attempts) with several variants of min-hop algorithm (centralized and distributed with 1 or 3 attempts). The conversion of centralized min-hop into a distributed one was done exactly as for the EXP algorithm. The reason we have chosen to present the "3 attempts" case is that our simulations indicated that increasing the number of attempts beyond three does not increase the performance.

The simulations show that the performance advantage of the EXP algorithm increases in the distributed setting. When comparing the distributed exponential strategies to the distributed minimum hop strategies we observe a significant performance advantage for the exponential strategies over a wide range of arrival rates. It seems that the performance penalty suffered due to the inaccurate state information is much less pronounced for the variants of the EXP algorithm. The EXP-based algorithms can sustain a much higher arrival rate given a target rejection (loss) rate. For a target rejection rate of 2% and a limit

[4]Issues arising from multiple circuits concurrently sending setup packets are not modeled by our simulations.

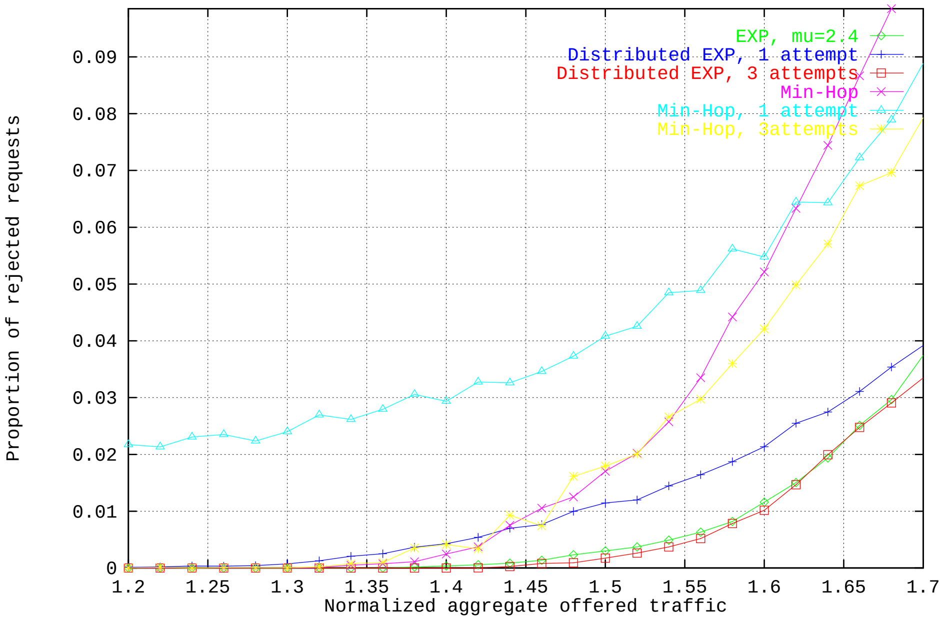

Figure 6.8: Performance of distributed vs centralized schemes.

of three setup attempts, the distributed EXP algorithm can maintain an arrival rate that is 8% higher than the arrival rate that the minimum hop based algorithm can maintain. When we restrict the algorithms to a single attempt, the advantage of the distributed EXP algorithm increases to 38%.

The sensitivity of the minimum hop based algorithms to the number of permitted attempts is particularly dramatic. The simulations show that a minimum hop based strategy would not be appropriate in a network requiring fast setup. In contrast, our exponential function based strategy still achieves good performance when fast setup, and consequently at most one setup attempt, is required. This is due to the fact that the exponential algorithm tends to pick paths that use edges with lower perceived load, thus reducing the probability of picking a saturated edge.

Another interesting phenomenon is the fact that the distributed minimum hop algorithm performs better than the centralized minimum hop algorithm at high arrival rates. The phenomenon provides a good illustration of the importance of trunk reservation. Since the centralized minimum hop algorithm is greedy, it will attempt to route a virtual circuit even if the only available path is very long. However, routing a virtual circuit on a long path will lead to inefficiencies at high arrival rates since it is likely that two or more future virtual circuit requests, each requiring shorter paths, can be routed on the long path. Since the distributed minimum hop algorithms only make a few routing attempts, they are unlikely to try long paths. Thus they effectively are implementing a simple trunk reservation policy. Unfortunately, this simple trunk reservation policy is too aggressive for low arrival rates. In particular, an aggressive trunk reservation policy will lead to inefficiencies at low arrival rates since future, shorter (in terms of hops) virtual circuit requests are less likely to arrive to replace a long virtual circuit request that was rejected due to trunk reservation. Thus, the greedy policy of the centralized minimum hop algorithm is superior to distributed single attempt minimum hop algorithm at low arrival rates.

One can view our distributed EXP algorithm as implementing a trunk reservation policy that is adjusted with the arrival rate. At low arrival rates, the link cost is low and so the admission control policy is greedy. When the arrival rate is very high, the link cost is high, thus implementing an aggressive trunk reservation policy. By dynamically adjusting its trunk reservation policy with the arrival rate, the EXP algorithm is able to outperform both minimum hop based algorithms at high and low arrival rates.

6.4 Discussion

There are several implementation issues that are not completely addressed by our simulations. This section suggests some approaches to these issues.

Both the centralized and the distributed EXP algorithms assume that the duration of the virtual circuit is known in advance. While this is a reasonable assumption for many applications, e.g. movies, there are a large class of applications, e.g. phone calls, for which this is not a reasonable assumption. However, a statistical description of the duration is available for many applications for which the duration is not known in advance. Again, phone calls are a good example. We would like to use the statistical duration information about the virtual circuits to make routing and admission control decisions. In fact, as we shall see in the next chapter, it is possible to modify the EXP algorithm to give theoretically good competitive ratio for the case where statistical information on the duration is known [37]. If the majority of virtual circuits either have holding times that are known in advance or belong to a class of connections for which statistics on the holding times are known, we would expect to see performance similar to that presented in this chapter.

Recall that our distributed EXP algorithm collects the entire state of each link on the path on which a circuit setup is attempted. Since our algorithm makes use of circuit duration information, the link state includes not only the currently allocated load, but also the load allocated by the current virtual circuits for all future points in time. In general, a large number of bits will be needed to transmit this state information. The following mechanism can be used to decrease the amount of transmitted state information to approximately 60 bits per link. For each edge, we transmit the current load and the times at which the load allocated by the current circuits reaches 95%, 85%, 65%, and 25% of the current load. Notice that the sample points are spaced so as to provide the greatest amount of detail at high load levels. Preliminary simulations of this approach suggest that such a decrease in accuracy of state information will not have a dramatic impact on the performance of our algorithm. With this approach, link information for paths with up to 9 links can be accommodated in a single ATM cell.

Chapter 7

Competitive routing under known Holding time Distributions

7.1 Introduction

In the previous chapter, we considered an on-line routing algorithm for virtual circuits in high-speed networks which gave competitive performance, when the holding time or duration of the circuits were known at arrival. For many services like video-on demand, the holding time of the request is known on arrival. But for many other types of services, the duration of the request is not known at arrival. We may, however, be able to determine a distribution on the holding time for each type of request. In this chapter, we consider an extension of the on-line algorithm for routing virtual circuits in high-speed networks, described in [6]. We show how we can get competitive performance in the average case, working with the distribution on the holding time rather than using the exact holding time.

Classical routing and admission control strategies achieve provably good performance by relying on an assumption that the arrival pattern of virtual circuits can be described by some a priori known probabilistic model. Traditionally, routing strategies were analyzed for the case where the traffic matrix is constant or changing sufficiently slowly so that it can be estimated. The framework in this chapter, is geared towards design of strategies that have provably good performance even in the case where there are *no statistical assumptions* on the arrival pattern and parameters of the virtual circuits. In our model, the arrival times of requests are arbitrary. In addition, we also do not assume that the traffic matrix (i.e. the rates of arrival for each node pair) is known. Also, in contrast to the previously proposed strategies in [43, 48, 65], we do not assume that the departure rate of circuits on different links are independent. In that sense, it is closer to reality. It is interesting to note that there can be no strategy (deterministic or randomized) that achieves a polylogarithmic competitive ratio in case there is no a priori information (and hence no probabilistic assumptions) about the holding times [10, 54].

We describe an admission and routing strategy for virtual circuits given their capacity requirements and a probability distribution on the holding time. We also prove that for an n node graph, assuming that the expected duration of any request is at most T, this routing strategy achieves an expected profit which is within an $O(\log nT)$ factor of the expected profit of the optimal off-line algorithm that has complete knowledge of all future requests except their exact duration. We show that for the special case, in which all the virtual circuits have the same bandwidth requirement and have the same exponential distribution on holding time, our routing strategy has a $O(\log L)$ average competitive ratio which is independent of the holding time of the requests, where L is a bound on the number of edges that can be used to route a request.

7.2 Notation and Definitions

The network is represented by a capacitated (directed or undirected) graph $G(V, E, u)$ having nodes V and edges E. The capacity $u(e)$ assigned to each edge $e \in E$ represents the bandwidth available on that edge. The input sequence consists of a collection of connection requests: $\beta_1, \beta_2, \ldots, \beta_k$, where the ith request is represented by the tuple:

$$\beta_i \stackrel{def}{=} (s_i, t_i, r_i(\tau), p_i(\tau), T_i^s).$$

Node s_i is the origin of the connection, node t_i is the destination, $r_i(\tau)$ is the function that defines the traffic at time τ, $p_i(\tau)$ is the probability that the connection exists at time τ and T_i^s is the starting time for the connection. We assume that the traffic rate is constant at r_j for the duration of the life of the circuit. The maximum expected duration of any connection is T. We also assume that a path that is used to route a request can have no more than L edges. We shall use a parameter μ which is defined later. We assume that for all j and τ,

$$r_j(\tau) \leq \frac{\min_e u(e)}{\log \mu}.$$

The profit earned by accepting this connection is assumed to be proportional to the number of bits transmitted by the connection and is defined as

$$\rho_i \stackrel{def}{=} L \sum_{\tau} r_i(t) X_i(t),$$

where $X_i(t)$ is an indicator variable that denotes if the connection i is alive at time t. Each connection request i that has been accepted is routed on a path denoted by P_i.

7.3 Adaptive Adversary

For purpose of our analysis, we shall be comparing our performance to the best possible off-line strategy which is omniscient in the sense that it is assumed to have a priori knowledge of all possible future requests and the probability distribution on their holding times, though it may not be aware of the exact termination times of each possible request. We shall then bound the worst possible expected performance of our algorithm with respect to the expected performance of this optimum off-line strategy. For the purpose of computing such a bound, one can assume the existence of an off-line adversary, who does not know the exact holding time of each circuit but knows the distribution on the holding time. The adversary makes the next request based on what has happened to the previous requests. This is similar to the adaptive off-line adversary described in [14] but unlike in the case described by them, the on-line algorithm is deterministic. The randomization is rather due to the fact that circuits have holding times which are random.

We can view the off-line adversary as a rooted tree where each level of the tree corresponds to a time and each node corresponds to the state of the previous circuits at that time. The children of a node correspond to the possible states at the next time instance given the state of the parent node. The off-line adversary decides its next move depending on the state of the network at each time instance. Different circuit requests are likely to arrive depending on what has happened to the previous requests. We can define a filtration $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \ldots \subset \mathcal{F}$ where $\mathcal{F}_t$ denotes the state of all the previous circuits (circuits that arrived before time t) at time t. The filtration represents how information is revealed through time. If a circuit arrived before time t then $\mathcal{F}_t$ has the information whether it is alive or finished at time t.

For a request β_j, we shall denote $\mathcal{F}_j = \mathcal{F}_{T_j^s}$. If Y is a random variable then we denote

$$E_j[Y] \stackrel{def}{=} E[Y|\mathcal{F}_j].$$

We can define $A(\mathcal{F}_t)$, the set of circuits that have been routed by our algorithm until time t as a random set which is a function of the state of information available at time t. The load on edge e just before considering the kth request is a random variable which is defined by:

$$\lambda_e(\tau, k) \stackrel{def}{=} \sum_{e \in P_i, i<k} \frac{r_i(\tau) X_i(\tau)}{u(e)}.$$

The cost of edge e, for time τ, just before arrival of the jth connection, is a random variable defined as

$$C_e(\tau, j) \stackrel{def}{=} u(e)(\mu^{\lambda_e(\tau,j)} - 1).$$

7.4 Admission Control and Routing Algorithm

The algorithm chooses to route a connection β_j on a path that is short with respect to a weighted average of the expected costs of the edge at the time of the request. The cost is computed using the current state of the previously routed circuits i.e. if $e \in P_j$ then e's contribution to cost of the path is computed as:

$$\sum_\tau \frac{r_j(\tau) p_j(\tau)}{u(e)} E_j[C_e(\tau, j)].$$

If there exists a path with cost less than the expected profit $\hat{\rho}_j = E_j[\rho_j] = L \sum_\tau r_j(\tau) p_j(\tau)$ then this path is used to route the circuit. Otherwise the connection is rejected.

We first choose μ such that we never route a circuit on an edge which does not have sufficient spare capacity.

Lemma 7.4.1 *For $\mu = 2LT+1$, the routing algorithm never violates capacity constraints.*

Proof: Suppose the algorithm routes circuit j on edge e and violates capacity constraint, then we know that $\lambda_e(T_j^s, j) > 1 - r_j/u(e) \geq 1 - 1/\log\mu$. At T_j^s, the cost $C_e(T_j^s, j)$ is not a random variable and

$$E_j\left[\frac{r_j}{u(e)} C_e(T_j^s, j)\right] > L r_j T = \hat{\rho}_j.$$

Hence the circuit can not be routed on that edge. ∎

Lemma 7.4.2 *Let $A(\mathcal{F}_t)$ be the set of indices of connections routed by our on-line algorithm until time t. Then the expected profit of our algorithm is at least within a $\log\mu$ factor of* *the total final cost on all the edges of the graph i.e.*

$$2\log\mu \;\; E\left[\sum_{j\in A(\mathcal{F}_t)} \rho_j\right] \geq \sum_{\tau}\sum_{e} E[C_e(\tau)]$$

where $C_e(\tau)$ is a random variable that indicates the final total cost on edge e at time τ.

Proof: Let $j \in A(\mathcal{F}_t)$, then since $\frac{r_j}{u(e)} \leq \frac{1}{\log\mu}$, we know that the expected increase in the cost at time τ, of edge $e \in P_j$, due to routing circuit j, can be bounded as follows

$$\begin{aligned} (7.1)\quad E_j[\Delta_j C_e(\tau)] &= E_j\left[u(e)\left(\mu^{\lambda_e(\tau,j)+r_j(\tau)X_j(\tau)/u(e)} - \mu^{\lambda_e(\tau,j)}\right)\right] \\ (7.2)\quad &\leq E_j\left[X_j(\tau)\frac{r_j(\tau)}{u(e)}\left(C_e(\tau,j)+u(e)\right)\right]\log\mu. \end{aligned}$$

When circuit j arrives, its duration is unknown and hence whether the circuit j will be alive or not at a future time is independent of the cost of the edge at that future time (as computed on arrival of circuit j). In other words, the random variables $C_e(\tau,j)$ and $X_j(\tau)$ are independent. This observation is crucial to our proof and gives us

$$(7.3)\qquad E_j\left[\frac{r_j(\tau)}{u(e)}X_j(\tau)C_e(\tau,j)\right] = \frac{r_j(\tau)}{u(e)}p_j(\tau)E_j[C_e(\tau,j)].$$

Since $j \in A(\mathcal{F}_t)$, we know that it satisfies the admission criterion given by

$$(7.4)\qquad \hat{\rho}_j \geq \sum_{\tau}\sum_{e}\frac{r_j(\tau)p_j(\tau)}{u(e)}E_j[C_e(\tau,j)].$$

Combining the results in 7.4, 7.3 and 7.2, and using the fact that $\hat{\rho}_j \geq E_j[\sum_\tau \sum_e r_j(\tau)X_j(\tau)]$ we get the following result,

$$\sum_{\tau}\sum_{e} E_j[\Delta_j C_e(\tau)] \leq 2\hat{\rho}_j \log\mu.$$

Summing over all the circuits in $A(\mathcal{F}_t)$ we get,

$$2\log\mu \sum_{j\in A(\mathcal{F}_t)} \hat{\rho}_j \geq \sum_{\tau}\sum_{e}\sum_{j\in A(\mathcal{F}_t)} E_j[\Delta_j C_e(\tau)].$$

Taking expectations w.r.t. $\mathcal{F}_t$ on both sides and observing that the sum of the changes gives the final cost i.e.

$$E\left[\sum_{j\in A(\mathcal{F}_t)} E_j[\Delta_j C_e(\tau)]\right] = E\left[\sum_{j\in A(\mathcal{F}_t)} \Delta_j C_e(\tau)\right] = E[C_e(\tau)]$$

gives us the desired result. ∎

Next we bound the profit earned by the circuits that were not admitted by the on-line algorithm but admitted by the off-line algorithm. The off-line algorithm, though aware of the arrival time and probability distribution in holding time of all the future requests, is not aware of their termination time and admission or rejection of a request by the off-line algorithm is a random variable which is independent of the exact termination time of that request.

Lemma 7.4.3 *Let l denote the last connection and let $Q(\mathcal{F}_l)$ be the set of indices of connections that were not admitted by the on-line algorithm. Then the expected profit that the on-line algorithm lost but the off-line algorithm gained can be bounded as*

$$E[\sum_{j \in Q(\mathcal{F}_l)} \rho_j] \leq \sum_{\tau} \sum_{e} E[C_e(\tau)].$$

Proof: Suppose the off-line algorithm accepts connection j and routes it on path P'_j then since the connection was rejected by our on-line algorithm, we have

$$\hat{\rho}_j \leq \sum_{\tau} \sum_{e \in P'_j} \frac{r_j(\tau) p_j(\tau)}{u(e)} E_j[C_e(\tau, j)].$$

Summing over all the rejected circuits, and using the independence of the random variables $C_e(\tau, j)$ and $X_j(\tau)$ just as in equation 7.3, we have

$$\begin{aligned} \sum_{j \in Q(\mathcal{F}_l)} \hat{\rho}_j &\leq \sum_{j \in Q(\mathcal{F}_l)} \sum_{\tau} \sum_{e \in P'_j} \frac{r_j(\tau) p_j(\tau)}{u(e)} E_j[C_e(\tau, j)] \\ &= \sum_{\tau} \sum_{e} \sum_{j \in Q(\mathcal{F}_l), e \in P'_j} E_j \left[C_e(\tau, j) \frac{r_j(\tau) X_j(\tau)}{u(e)} \right]. \end{aligned}$$

The cost of an edge can only increase with the arrival of new circuits and hence the cost as seen at time j is less than the final cost of an edge i.e. $\forall \tau, C_e(\tau, j) \leq C_e(\tau)$. Taking expectations on both sides of inequality 7.5 and using this observation gives us

$$E\left[\sum_{j \in Q(\mathcal{F}_l)} \hat{\rho}_j\right] \leq \sum_{\tau} \sum_{e} E\left[\sum_{j \in Q(\mathcal{F}_l), e \in P_j} E_j \left[C_e(\tau) \frac{r_j(\tau) X_j(\tau)}{u(e)}\right]\right]. \tag{7.5}$$

Since the adversary obeys capacity constraints, we know that for all edges e and time τ,

$$\sum_{j \in Q(\mathcal{F}_l), e \in P_j} X_j(\tau) \frac{r_j(\tau)}{u(e)} \leq 1. \tag{7.6}$$

Multiplying inequality 7.6 by $C_e(\tau)$ and taking expectations gives us

$$E\left[\sum_{j\in Q(\mathcal{F}_l),e\in P_j} E_j\left[C_e(\tau)\frac{r_j(\tau)X_j(\tau)}{u(e)}\right]\right] \leq E[C_e(\tau)]. \tag{7.7}$$

which when substituted in 7.5, along with the fact that

$$E[\sum_{j\in Q(\mathcal{F}_l)} \hat{\rho}_j] = E[\sum_{j\in Q(\mathcal{F}_l)} \rho_j],$$

gives us the desired result. ∎

Lemma 7.4.4 *The expected profit accrued by our on-line algorithm is at least* $1/(2\log\mu+1)$*-fraction of the expected profit that can be accrued by the optimal off-line algorithm.*

Proof: The profit earned by the off-line algorithm is bounded by

$$\sum_{j\in Q(\mathcal{F}_l)}\sum_{\tau} r_j(\tau)X_j(\tau) + \sum_{j\in A(\mathcal{F}_l)}\sum_{\tau} r_j(\tau)X_j(\tau).$$

Taking expectations and using the results of lemmas 7.4.3 and 7.4.2 we conclude that the expected profit of the off-line algorithm is bounded by

$$(2\log\mu+1)E[\sum_{j\in A(\mathcal{F}_l)} \rho_j]$$

which implies the desired result. ∎

7.5 Exponential Holding Time

In the algorithm described previously, we needed large values of μ to ensure competitive performance and this resulted in very poor competitive ratios. A large factor in the competitive ratio was introduced due to the presence of T, the maximum expected holding time of the circuits. For certain distributions in holding times of circuits, we can reduce the value of μ and hence improve the guarantee on performance. In this section, we extend our previous result to the case where all circuits have the same rate and the same exponential distribution on holding time with mean T. Moreover, we also assume that the profit of the virtual circuit is proportional to the holding time of the circuit. This gives rise to a competitive algorithm, which uses just the current load on the edges to make the routing decisions and is $O(\log L)$-competitive in the expected case. We assume without loss of generality that all the circuits have unit rate i.e. $r_j = 1$. The idea behind our algorithm is simple. We use the same algorithm as described in the previous section, except now that the circuits have an exponential distribution on the holding time, the total cost of adding a new circuit on an edge can be computed directly as a function of the current congestion level. This results in an algorithm that has a simplified cost computation and also a better competitive performance.

Lemma 7.5.1 *If on arrival, a circuit j, finds that the load on an edge e is λ then the expected cost of routing the circuit on this edge over the duration of the circuit is given by*

$$E_\lambda[C_e] = Tu(e)\left(\frac{\mu^{\lambda+1/u(e)} - 1}{(\mu^{1/u(e)} - 1)(\lambda u(e) + 1)} - 1\right). \tag{7.8}$$

Proof: Let S_e be the set of circuits alive on edge e at time $t = 0$. Since $r_j = 1$, we know that $|S_e| = \lambda u(e)$. The circuits terminate independently of each other, and hence the random variables X_j are independent. The expected cost of an edge e at time τ in the future due to the circuits in S_e is given by

$$\begin{aligned} E[C_e(\tau)|\lambda_e(0) = \lambda] &= u(e)\left(E\left[\mu^{\sum_{j\in S_e} X_j(\tau)/u(e)}|\forall j \in S_e, X_j(0) = 1\right] - 1\right) \\ &= u(e)\left(\Pi_{j\in S_e} E\left[\mu^{X_j(\tau)/u(e)}|X_j(0) = 1\right] - 1\right). \end{aligned}$$

Because of the memoryless property, we know that given that a circuit is alive at time $t = 0$, the probability that it is alive at time $t = \tau$ in the future is given by $e^{-\tau/T}$. Therefore we have

$$E\left[\mu^{X_j(\tau)/u(e)}|X_j(0) = 1\right] = e^{-\tau/T}\mu^{1/u(e)} + (1 - e^{-\tau/T}).$$

The total cost of a new circuit over the duration of the call is given by

$$u(e)\int_0^\infty e^{-\frac{t}{T}}\left(\left(e^{-\frac{t}{T}}\mu^{\frac{1}{u(e)}} + (1 - e^{-\frac{t}{T}})\right)^{\lambda u(e)} - 1\right)dt = Tu(e)\left(\frac{(\mu^{\lambda+\frac{1}{u(e)}} - 1)}{(\lambda u(e) + 1)(\mu^{\frac{1}{u(e)}} - 1)} - 1\right).$$

The proof then follows. ∎

It is easy to observe that lemmas 7.4.2 and 7.4.3 apply to the exponential case. Note that the proofs of the lemma work for any value of μ. The only place where the value of μ is crucial, is in the proof of Lemma 7.4.1. If we were to use the lemma suggested in that proof we would get a $O(\log LT)$-competitive algorithm but the following lemma shows that we can reduce the value of μ to get a better competitive ratio.

Lemma 7.5.2 *Choosing the value of* $\log \mu = O(\log L)$ *is sufficient to enforce capacity constraints.*

Proof: When the edge is filled to capacity i.e. $\lambda_e = 1$, then by Equation 7.8, the expected cost of routing a circuit on that edge is given by $T(\mu^{1+1/u(e)} - 1)/((1 + u(e))(\mu^{1/u(e)} - 1))$ and the expected profit is given by LT. Hence to cause rejection of full edges, it is sufficient to set the value of μ such that $\log \mu = O(\log L)$. ∎

The following theorem then follows from lemmas 7.5.2, 7.4.2 and 7.4.3.

Theorem 7.5.3 *An algorithm that routes using the cost function given in (7.8) has a* $O(\log L)$*-competitive ratio.*

7.6 Simulation Results

In the previous section, we presented the theoretical background underlying an on-line routing and admission control strategy that uses knowledge of distribution on holding time. Since it is impossible to analyze the performance of such algorithms analytically for realistic networks and traffic patterns, we conducted extensive simulation experiments. We assumed an exponential memoryless distribution on holding time of circuits. One of the goals of our simulations was to compare the algorithm to a greedy minimum-hop strategy. We have used an existing commercial topology for our simulations. This topology is described in subsection 6.2.3 in the previous chapter. For our experiments, we generated exponentially distributed holding times with a mean of 1800 seconds. This, together with the observed traffic profile for that network, defines the call arrival rate.

The simulation results are presented in Figure 7.1. The figure shows comparative performance of min-hop, the EXP algorithm that uses exact information about holding time and the EXPD which does not use the explicit holding time information but uses the knowledge that it is exponentially distributed.

It is obvious that the EXP algorithm that uses the holding time information has the best performance. However, even EXPD which does not have information about holding time can use its limited knowledge of exponential distribution on holding time to perform significantly better than min-hop based algorithm that does not use any information at all. This suggests that if holding time information is available, it can be used fruitfully,

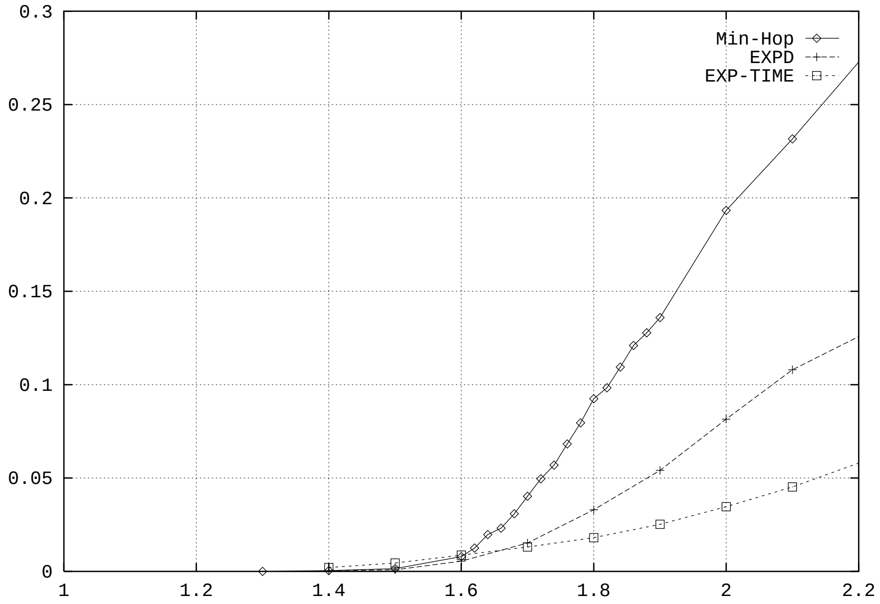

Figure 7.1: Comparative performance for Commercial topology

but in its absence EXPD gives us a routing and admission control mechanism to use the distribution on holding times to improve the performance.

Bibliography

[1] I. Adler, N.K. Karmarkar, M.G.C. Resende, and G. Veiga. An Implementation of Karmarkar Algorithm for Linear Programming. *Mathematical Programming*, 44:297–335, 1989.

[2] Y. Afek, B. Awerbuch, and E. Gafni. Applying Static Network Protocols to Dynamic Networks. In *Proc. 28th IEEE Annual Symposium on Foundations of Computer Science*, pages 358–370, 1987.

[3] G.R. Ash, J.S. Chen, A.E. Frey, and B.D. Huang. Real-time network routing in an integrated network. In *Proc. of ITC-13, Copenhagen, Denmark*, 1991.

[4] G.R. Ash, J.Chen, A.E. Frey, B.D. Huang, C.Lee, and G.L. McDonald. Real-time network routing in the AT&T network-improved service quality at lower cost. In *Proceedings of the IEEE Global Telecommunicationsi Conference, Orlando, Florida*, pages 802–813, Dec. 1992.

[5] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line machine scheduling with applications to load balancing and virtual circuit routing. In *Proc. 25th Annual ACM Symposium on Theory of Computing*, pages 623–631, May 1993.

[6] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput competitive on-line routing. In *Proc. 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 32–40, November 1993.

[7] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proc. 5'th ACM-SIAM Symp. on Discrete Algorithms*, pages 321–327, 1994.

[8] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proc. 34rd IEEE Annual Symposium on Foundations of Computer Science*, pages 459–468, 1993.

[9] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multicommodity flow problem and local competitive routing in dynamic networks. In *Proc. 26th Annual ACM Symposium on Theory of Computing*, pages 487–495, 1994.

[10] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. In *Proc. 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 218–225, 1992.

[11] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. In *Procedings of Workshop on Algorithms and Data Structures*, pages 119–130, August 1993.

[12] E.R. Barnes, S. Chopra, and D.L. Jensen. A Polynomial Time Version of the Affine Scaling Algorithm. Technical Report 88-101, Graduate School of Business Administration, New York University, Washington Square, N.Y. 1988., 1988.

[13] D.A. Bayer and J.C. Lagarias. The Nonlinear Geometry of Linear Programming I, Affine and Projective Scaling Trajectories. *Transactions of American Mathematical Society*, 314:499–526, 1989.

[14] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms . *Algorithmica*, 11:2–14, 1994.

[15] D. P. Bertsekas and E. N. Gafni. Projected Newton Methods and Optimization of Multicommodity Flows. *IEEE Transactions on Automatic Control*, AC-28(12):1000–1006, Dec 1983.

[16] R.C. Carden and C.-K. Cheng. A Global Router using an Efficient Approximate Multicommodity Multiterminal Flow Algorithm. In *28th ACM/IEEE Design Automation Conference*, pages 316–321, June 1991.

[17] CPLEX Optimization Inc. *CPLEX*, 1990.

[18] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proc. of 2nd Annual Israel Conference on Theory of Computing and Systems*, 1993.

[19] J.A. Garay and I.S. Gopal. Call preemption in communication networks. In *Proc. INFOCOM '92*, volume 44, pages 1043–1050, Florence, Italy, 1992.

[20] R. Gawlick, C. Kalmanek, and K. Ramakrishnan. On-line permanent virtual circuit routing. In *Proceedings of IEEE Infocom*, April 1995.

[21] R. Gawlick, A. Kamath, S. Plotkin, and K.G. Ramakrishnan. Routing and admission control of virtual circuits in general topology networks. Technical Report 11216-901004-13TM, ATT, Murray Hill, NJ, Sept 1994.

[22] A. Gersht and A. Shulman. A New Algorithm for the Solution of the Minimum Cost Multicommodity Flow Problem. In *Proceedings of the 26th IEEE Conference on Decision and Control*, pages 748–758, 1987.

[23] R.J. Gibbens and F. P. Kelly. Dynamic routing in fully connected networks. *IMA J. Math. Contr. Inform.*, 7:77–111, 1990.

[24] R.J. Gibbens, F.P. Kelly, and P.B. Key. Dinamic alternative routing - modelling and behaviour. In *Proc. of ITC-12, Torino, Italy*, 1988.

[25] A. V. Goldberg, S. A. Plotkin, and É. Tardos. Combinatorial Algorithms for the Generalized Circulation Problem. In *Proc. 29th IEEE Annual Symposium on Foundations of Computer Science*, pages 174–185, 1988.

[26] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *J. Assoc. Comput. Mach.*, 35:921–940, 1988.

[27] A. V. Goldberg and R. E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Math. of Oper. Res.*, 15:430–466, 1990.

[28] D. Goldfarb and M.D. Grigoriadis. A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow. *Annals of Operations Research*, 13:83–123, 1988.

[29] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Hopkins, second edition, 1989.

[30] C.C. Gonzaga. Polynomial Affine Algorithms for Linear Programming. *Mathematical Programming*, 49:7–21, 1990.

[31] A. Hipolito. A Long-step Linear Programming Algorithm Based on Quasi-Newton Inverse Barrier Centering. Technical Report 91-12, Dept. of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, October 1991.

[32] T. C. Hu. Multi-Commodity Network Flows. *J. ORSA*, 11:344–360, 1963.

[33] P.J. Hunt and C.J. Laws. Asymptotically optimal loss network control. *Mathematics of Operations Research*, 18(4):880–900, 1993.

[34] A. Kamath and O. Palmon. Improved interior-point algorithms for approximate and exact algorithm for multicommodity flow. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[35] A. Kamath, O. Palmon, and S. Plotkin. Simple and fast distributed multicommodity flow algorithm. Unpublished Manuscript.

[36] A. Kamath, O. Palmon, and S. Plotkin. Fast Approximation Algorithm for Minimum Cost Multicommodity Flow. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[37] A. Kamath, S. Plotkin, and K.G. Ramakrishnan. Online routing of ATM circuits with known distribution of holding times. Unpublished manuscript, May 1994.

[38] A.P. Kamath, K.G. Ramakrishnan, and N.K. Karmarkar. Computational and Complexity Results for an Interior Point Algorithm on Multicommodity Flow Problems. In *Netflow*, October 1993.

[39] S. Kapoor and P. M. Vaidya. Fast Algorithms for Convex Quadratic Programming and Multicommodity Flows. In *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 147–159, 1986.

[40] N. Karmarkar. A New Polynomial-Time Algorithm for Linear Programming. *Combinatorica*, 4:373–395, 1984.

[41] N. K. Karmarkar and K.G. Ramakrishnan. Implementation and Computational Results of the Karmarkar Algorithm for Linear Programming, Using an Iterative Method for Computing Projections. Technical Report 11211-891011-10TM, AT&T Bell Laboratories, Murray Hill, NJ, October 1989.

[42] N. K. Karmarkar and K.G. Ramakrishnan. Computational Results of an Interior Point Algorithm for Large Scale Linear Programming. *Mathematical Programming*, 52:555–586, 1991.

[43] F. P. Kelly. Blocking probabilities in large circuit-switched networks. *Adv. Appl. Prob.*, 18:473–505, 1986.

[44] F.P. Kelly. Routing in circuit-switched networks: optimization, shadow prices and decentralization. *Advances in Applied Probability*, 20:112–144, 1988.

[45] J. Kennington. A Survey of Linear Cost Multicommodity Network Flows. *Operations Research*, 26:206–236, 1978.

[46] J. Kennington. A Primal Partitioning Code for Solving Multicommodity Flow Problems (Version 1). Technical Report 79009, Department of Industrial Engg. and OR, Southern Methodist University, 1979.

[47] P. Klein, S. A. Plotkin, C. Stein, and É. Tardos. Faster Approximation Algorithms for the Unit Capacity Concurrent Flow Problem with Applications to Routings and Finding Sparse Cuts. Technical Report 961, School of ORIE, Cornell University, 1991.

[48] K.R. Krishnan and T.J. Ott. Forward-looking routing: A new state-dependent routing scheme. In *Proc. of ITC-12, Torino, Italy*, 1988.

[49] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proc. 23st Annual ACM Symposium on Theory of Computing*, 1991.

[50] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proc. 29th IEEE Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[51] T. Leong, P. Shor, and C. Stein. Implementation of a Combinatorial Multicommodity Flow Algorithm. In D.S. Johnson and C.C. McGeoch, editors, *Network Flows and Matching: first DIMACS Implementation Challenge; Volume 12*, pages 387–406. American Mathematical Society, 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science.

[52] S.Y. Lin. A Textured Decomposition Based Algorithm for Large-scale Multicommodity Network Flow Problems. In *Proceedings of the 1990 American Control Conference, San Diego, CA*, May 1990.

[53] D. G. Luenberger. *Linear and Nonlinear Pragramming.* Addison Wesley Publishing Company, Reading, MA, second edition, 1984.

[54] Y. Ma and S. Plotkin. Improved lower bounds for load balancing of tasks with unknown duration. Unpublished manuscript, 1995.

[55] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for online problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 322–332, 1988.

[56] V. Marbukh. An asymptotic study of a large fully connected communication network with reroutes. *Problemy Peredachi Informatsii*, 3:89–95, 1981.

[57] V. Marbukh. Study of a fully connected communication network with many nodes and circumventing routes. *Automatica i Telemechanica*, 12:86–94, 1983.

[58] D. Mitra, R.J. Gibbens, and B.D. Huang. State-dependent routing on symmetric loss networks with trunk reservations – I. *Annals of Operations Research*, 35:3–30, 1992.

[59] D. Mitra, R.J. Gibbens, and B.D. Huang. State-dependent routing on symmetric loss networks with trunk reservations – II. *IEEE Trans. on Communications*, 41(2), February 1993.

[60] D. Mitra and J.B. Seery. Comparative evaluations of randomized and dynamic routing strategies for circuit switched networks. *IEEE Trans. on Communications*, 39(1):102–116, jan 1991.

[61] R. D.C. Monteiro and I. Adler. Interior Path Following Primal-Dual Algorithms. Part II: Convex Quadratic Programming. *Math. Prog.*, 44:43–66, 1989.

[62] S. M. Murray. *An Interior Point Approach to the Generalized Flow Problem with Costs and Related Problems.* PhD thesis, Department of Operations Research, Stanford University, Stanford, CA., 1992.

[63] H. Nagamochi, M. Fukushima, and T. Ibaraki. Relaxation Methods for the Strictly Convex Multicommodity Flow Problem with Capacity Constraints on Individual Commodities. *Networks*, 20(4):409–426, 1990.

[64] H. Nagamochi and T. Ibaraki. On Max-flow Min-cut and Integral Flow Properties for Multicommodity Flows in Directed Networks. *Information Processing Letters*, 31(6):279–285, 1989.

[65] T.J. Ott and K.R. Krishnan. State-dependent routing of telephone traffic and the use of separable routing schemes. In *Proc. of ITC-11, Kyoto, Japan*, 1985.

[66] S. Plotkin. Competitive routing in ATM Networks. *J. Selected Areas in Comm.*, 1995. To appear.

[67] S. A. Plotkin, D. Shmoys, and É. Tardos. Fast Approximation Algorithms for Fractional Packing and Covering. In *Proc. 32nd IEEE Annual Symposium on Foundations of Computer Science*, 1991.

[68] T. Radzik. Fast Deterministic Approximation for the Multicommodity Flow Problem. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[69] M.H. Rossiter and R.J. Harris. A Multi-class, Multi-commodity Flow Model of a Fast Packet Switched Network. *Australian Telecommunication Research*, 23(2):39–46, 1989.

[70] F. Shahrokhi and D. Matula. The maximum concurrent flow problem. *J. Assoc. Comput. Mach.*, 37:318–334, 1990.

[71] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. Technical Report CSR-183, Department of Computer Science, New Mexico Tech., 1988.

[72] S. Sibal and A. DeSimone. Controlling alternate routing in general-mesh packet flow networks. In *Proceedings of SIGCOMM 94, London, England*, 1994.

[73] S. Sibal and A. DeSimone. Controlling alternate routing in general-mesh packet flow networks. Technical Report BL045370F-940603-01TM, AT&T, June 1994.

[74] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.

[75] C. Stein. *Approximation algorithms for multicommodity flow and scheduling problems.* PhD thesis, MIT, 1992.

[76] M.J. Todd. A Low Complexity Interior Point Algorithm for Linear Programming. *SIAM Journal of Optimization*, 2:198–209, 1992.

[77] P. M. Vaidya. Speeding up Linear Programming Using Fast Matrix Multiplication. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, 1989.

[78] P.M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proc. 30th IEEE Annual Symposium on Foundations of Computer Science*, pages 338–343, 1989.

[79] Y. Ye. Toward Probabilistic Analysis of Interior-Point Algorithms for Linear Programming. *Math. of Oper. Res.*, pages 38–52, 1994.

[80] A.Ya. Zaslavskii. Multicommodity Models of Economic Dynamics with Discrete Innovations. *Cybernetics*, 26(6):829–837, 1990.