EXACT AND APPROXIMATION ALGORITHMS FOR NETWORK FLOW AND DISJOINT-PATH PROBLEMS

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

in

Computer Science

by

Stavros G. Kolliopoulos

DARTMOUTH COLLEGE

Hanover, New Hampshire

August 31, 1998

Examining Committee:

(chair) Clifford Stein

Prasad Jayanti

Fillia Makedon

Aravind Srinivasan

Neal E. Young

Roger D. Sloboda
Dean of Graduate Studies

Abstract

Network flow problems form a core area of Combinatorial Optimization. Their significance arises both from their very large number of applications and their theoretical importance. This thesis focuses on efficient exact algorithms for network flow problems in P and on approximation algorithms for NP-hard variants such as disjoint paths and unsplittable flow.

Given an n-vertex, m-edge directed network G with real costs on the edges we give new algorithms to compute *single-source shortest paths* and the *minimum mean cycle.* Our algorithm is deterministic with $O(n^2 \log n)$ expected running time over a large class of input distributions. This is the first strongly polynomial algorithm in over 35 years to improve upon some aspect of the $O(nm)$ running time of the Bellman-Ford shortest-path algorithm.

In the *single-source unsplittable flow* problem, we are given a network G, a source vertex s and k commodities with sinks t_i and real-valued demands ρ_i, $1 \leq i \leq k$. We seek to route the demand ρ_i of each commodity i along a single s-t_i flow path, so that the total flow routed across any edge e is bounded by the edge capacity u_e. This NP-hard problem combines the difficulty of bin-packing with routing through an arbitrary graph and has many interesting and important variations. We give a generic framework, which yields approximation algorithms that are simpler than

the previously known and achieve significant improvements upon the approximation ratios.

In a *packing integer program,* we seek a vector x of integers, which maximizes $c^T \cdot x$, subject to $Ax \leq b$, $A, b, c \geq 0$. The *edge* and *vertex-disjoint path* problems together with their *multiple-source unsplittable flow* generalization are NP-hard problems with a multitude of applications in areas such as routing, scheduling and bin packing. We explore the topic of approximating disjoint-path problems using polynomial-size packing integer programs. Motivated by the disjoint paths applications, we initiate the study of a class of packing integer programs, called *column-restricted.* We derive approximation algorithms for column-restricted packing integer programs that we believe are of independent interest.

Acknowledgements

Many thanks to my advisor Cliff Stein, for his wit, support and advice on technical and non-technical matters. I learned a lot from Cliff. I have greatly benefited from his perspective on Combinatorial Optimization, which I have come to share, including my long-denied affinity for scheduling problems. Cliff has been more than generous with his time during these last four years and the intellectual climate of dozens of meetings will be fondly remembered.

Thanks to Prasad Jayanti for his out-of-this-world teaching and for being a calm, resolute friend. I am grateful to Fillia Makedon for her support and for accepting to serve on my Thesis Committee in addition to all her other responsibilities. Aravind Srinivasan provided ample technical advice and shared ideas with characteristic generosity. I am grateful also to Aravind for serving on my Committee while based in Singapore. Thanks to Neal Young for reading my thesis and for lengthy, enlightening, discussions on various algorithmic topics. Neal taught me more things on randomized rounding than I ever thought I needed to know. Thanks also to Chandra Chekuri for his late-night remarks on packing integer programs.

I remember Donald Johnson, my first mentor at Dartmouth. He started me working on shortest paths and guided patiently the motivated but not very knowledgable first-year student that I was. I want to believe he would have approved of my later career.

Thanks to Wayne Cripps, our system manager, for his timely and frequent assistance. In my opinion though, he could have given me another Terabyte of disk space. Thanks to Charles Owen for technical assistance with document preparation.

Dimitris Sofotassios and Thanassis Tsakalidis first got me interested in research when I was an undergraduate in the University of Patras. I thank them both.

I have had the good fortune to have a number of excellent teachers over the years, teaching, in addition to subject matter, *ethos,* in the original sense of the word. I thank them all and I would like to mention at least one more name, Dimitri Paraskevopoulos. It is definitely the wrong place to make such a statement, but I feel I have not learned much more Pure Math since 1988.

My friends, from all around the globe, have always been close when I needed them. Their support has been indispensable. I think they know who they are.

My parents, Georgios and Ioustini Kolliopoulos, instilled to me the love of learning and, among other things, the amount of determination needed to make it through ups and downs. *To pronounce the word sea clearly that all the dolphins within it might shine And the desolation so great it might contain all of God And every waterdrop ascending steadfastly toward the sun.* My elder brother, Alexandros, has never failed to provide advice, an example, and help. *And the small cool domes like blue medusae Reached each time higher to the silverwork The wind so delicately worked As a painting For other times more distant For other times more distant.*

I would like to acknowledge partial support by NSF Award CCR-9308701 and NSF Career Award CCR-9624828. Uncredited in-text verse by Odysseas Elytis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

seaweed, slate, seashells,
vast sea glittering, diamonds and
the vast black light.
– *Takis Sinopoulos,* Sophia and Other Things

Network flow problems form a core area of Combinatorial Optimization. The basic *maximum flow* problem is defined as follows. We are given a directed graph $G = (V, E)$ and two distinguished vertices $s, t \in V$; s and t are called respectively the *source* and the *sink.* The graph is capacitated, i.e., a real-valued function u maps edges to *capacities.* A *flow* is a function $f : E \to R$, satisfying the following two constraints:

$$\sum_{(w,v)\in E} f(w,v) = \sum_{(v,w)\in E} f(v,w) \quad \forall v \in V - \{s,t\} \quad \text{(flow conservation)}$$
$$f(v,w) \le u(v,w) \qquad \forall (v,w) \in E \qquad \text{(capacity constraints)}$$

The *value* of a flow f is $\sum_{(s,v)\in E} f(s,v)$. The objective is to find a flow of maximum value. Many important variations of this basic problem exist. Typical problems include:

The minimum-cost flow problem. Define a cost function $c : E \to R$; the objective is

now to find a maximum flow of minimum cost, i.e. a maximum flow minimizing $\sum_{(v,w)\in E} c(v,w)f(v,w)$.

The multicommodity flow problem. In contrast to the maximum flow problem, there are more than one sources and sinks. Flow needs to be routed between specified pairs of sources and sinks while sharing the same underlying network. Formally, *commodities* correspond to a set of pairs (s_i, t_i), $s_i, t_i \in V$, $1 \leq i \leq K$. Every commodity has an associated *demand* $\rho_i \in R$. A flow f_i of value ρ_i is sought for each commodity, such that the following additional constraint is met:

$$\sum_{i=1}^{K} f_i(v,w) \leq u(v,w) \quad \forall (v,w) \in E.$$

The importance of network flow problems is twofold. First, there is a plethora of motivating practical applications in areas ranging from operations research (e.g. scheduling, network design, traffic planning) to telecommunications and VLSI. Second, the considerable theoretical challenges involved have led to the development of a rich network flow theory, which has had a significant impact on the evolution of algorithm design in general. Many reference sources for network flow theory exist; we refer the interested reader to the monographs by Ford and Fulkerson [30], Lawler [70] and Ahuja, Magnanti and Orlin [3].

Different variations of network flow problems are known to lie in different time complexity classes. While the basic maximum and minimum-cost flow problems defined above are in P, i.e. they can be solved in polynomial time, other variations are NP-hard. One example of the latter type of problems is the integral multicommodity flow problem [26]. In this problem demands and capacities are integral and $f_i(u,v)$ is also restricted to lie in Z_+ for every edge (u,v). The special case of integral multicommodity flow, in which all demands and capacities are equal to 1 is

the well-known *edge-disjoint path* problem. When faced with an NP-hard problem, a possible approach is to devise an approximation algorithm. For a minimization (resp. maximization) problem, a *ρ-approximation algorithm,* $\rho > 1$, ($\rho < 1$), outputs in polynomial time a solution with objective value at most (at least) ρ times the optimum. If it is clear from the context that we examine a maximization problem, we will sometimes refer to a ρ-approximation algorithm as achieving a $(1/\rho)$-approximation, $\rho > 1$. Readers unfamiliar with the theory of NP-completeness are referred to the classical text of Garey and Johnson [36].

In this thesis, we study network flow problems with an emphasis on the edge-disjoint path problem and its generalizations. We study both problems that are known to be in P and NP-hard problems. In the former case we are interested in faster exact algorithms, while in the latter we are interested in devising improved approximation algorithms. In what follows, we give a brief overview of our work. Detailed definitions and background are given in individual chapters as appropriate.

In Chapter 2 we give a new algorithm for the *single-source shortest path* problem on a network $G = (V, E)$ with real-valued edge costs. Denote $|V|$ and $|E|$ by n and m for the network of interest. Our algorithm is deterministic with $O(n^2 \log n)$ expected running time over a large class of input distributions. This is the first strongly polynomial algorithm in over 35 years to improve upon some aspect of the $O(nm)$ running time of the Bellman-Ford algorithm. The result extends to an $O(n^2 \log n)$ expected running time algorithm for finding the minimum mean cycle, an improvement over Karp's $O(nm)$ worst-case time bound when the underlying graph is dense. Both of our time bounds are shown to be achieved with high probability.

In Chapter 3 we study the the *single-source unsplittable flow* problem. We are given a network G, a source vertex s and K commodities with sinks t_i and real-

valued demands ρ_i, $1 \leq i \leq K$. We seek to route the demand ρ_i of each commodity i along a single s-t_i flow path, so that the total flow routed across any edge e is bounded by the edge capacity u_e. This NP-hard problem combines the difficulty of bin-packing with routing through an arbitrary graph and has many interesting and important variations. We give a generic framework, which yields approximation algorithms that are simpler than those previously known and significantly improve upon the approximation ratios. Our framework, with appropriate subroutines, applies to all optimization versions previously considered and unlike previous work treats in a unified manner directed and undirected graphs.

In the more general *multiple-source unsplittable flow* problem, different commodities are routed from potentially different sources. Multiple source unsplittable flow contains edge-disjoint paths as a special case. The edge-disjoint path problem addresses a fundamental connectivity question, which is simple to state: given a set of vertex pairs (s_i, t_i), $1 \leq i \leq K$, in a graph $G = (V, E)$, connect a maximum-cardinality subset of the pairs along edge-disjoint paths. Similarly, one can define the *vertex disjoint-path* problem, in which terminal pairs must be connected along vertex-disjoint paths. In addition to their rich combinatorial structure, disjoint-path type problems have been brought to further prominence by emerging applications in high-speed networks [60]. Nevertheless, designing good approximation algorithms that work on general graphs seems currently a very challenging task. Disjoint-path problems are conceptually related with packing integer programs. In a *packing integer program,* we seek a vector x of integers, which maximizes $c^T \cdot x$, subject to $Ax \leq b$, $A, b, c \geq 0$. The connection between these two problem categories, has largely been ignored in approximation algorithms, at least for the maximization metric we focus on in this work. See [84] for results on the congestion metric. In Chapter 4 we explore

the topic of approximating disjoint-path problems using polynomial-size packing integer programs. Our approach gives a unified treatment of different versions of the problems, including multiple-source unsplittable flow, while achieving some of the best known approximation bounds. Motivated by the disjoint-path applications, we initiate the study of a class of packing integer programs, called *column-restricted.* We derive approximation algorithms for packing programs in this class, which we believe are of independent interest. Finally, we use a different approach to provide two alternative approximation algorithms for edge and vertex-disjoint paths: we give two simple greedy algorithms whose approximation ratio matches the currently best, $1/\sqrt{|E|}$, and $1/\sqrt{|V|}$ respectively. Improved guarantees are obtained, when the optimal solution has special structure.

The results in this thesis were obtained jointly with Cliff Stein and preliminary expositions have appeared in [68, 66, 67].

Note on terminology: We use the terms graph and network completely interchangeably throughout the thesis. Given a network $G = (V, E)$ of interest, we use n and m to denote $|V|$ and $|E|$ unless otherwise specified. All logarithms are base 2 unless otherwise noted.

Chapter 2

An Algorithm for Real-Valued Shortest Paths with Fast Average Case[1]

It's all the streets we crossed
not so long ago
— The Velvet Underground, Sunday Morning

2.1 Introduction

Given a directed network $G = (V, E, c)$ where c is a real-valued function mapping edges to costs, the *single-source shortest path* problem (abbreviated SSSP) consists of finding, for each vertex $v \in V$, a simple path of minimum total cost from a designated source vertex s. This is an old and fundamental problem in network optimization with a plethora of applications in operations research (see, for example [3]). It also arises as a subproblem in other optimization problems such as network flows. A survey of

[1]This chapter contains joint work with Cliff Stein [68].

over 200 shortest-path papers through the year 1984 appeared in [20].

The classic Bellman–Ford algorithm solves the SSSP problem in an n-vertex m-edge network in $O(nm)$ time [7, 30]. This simple algorithm has been widely used and studied for over 35 years; however, in all that time, no progress has been made in improving the worst case time bound for arbitrary real-valued shortest path problems.

Our results. We cannot improve on the Bellman–Ford algorithm in the worst case for all networks. However, we provide a new deterministic algorithm for SSSP that, on any network with real edge costs chosen from a large class of input distributions, runs in $O(n^2 \log n)$ expected time. Moreover, the algorithm is shown to be reliable: it achieves the aforementioned time bound *with high probability,* i.e. with probability $1 - O(n^{-c})$ for some constant $c > 0$. Thus we provide a probabilistic guarantee that our algorithm finishes its computation in $O(n^2 \log n)$ time. Our input model is that of a complete graph with random costs, therefore the running time of our algorithm outperforms the $O(n^3)$ bound of the Bellman–Ford algorithm. The recent scaling algorithm of Goldberg [38], runs in $O(\sqrt{n}m \log C)$ time, where C denotes the absolute value of the largest edge cost. Under the similarity assumption [3], i.e. $\log C = O(\log n)$, Goldberg's algorithm runs in $O(n^{2.5} \log n)$ time on dense graphs, therefore our algorithm is faster. To our knowledge, ours is the first strongly polynomial algorithm that solves SSSP in $o(n^3)$ time for dense networks with arbitrary real-valued costs. In the same time bound, our algorithm will detect the existence of a negative-cost cycle. A conceptual contribution of our method is that one can solve real-weighted SSSP by solving $O(n)$ SSSP problems with nonnegative weights. When these problems can be solved in sublinear time, we get a faster algorithm.

Our model of random edge costs is Bloniarz's *endpoint-independent* model [8].

This model can be roughly defined as follows: the distribution according to which the cost of directed edge (u, v) is chosen does not depend on the head v. The endpoint-independent model includes the common case of all edge costs being drawn independently from the same distribution. A more general example from the endpoint-independent model is a cost assignment in which the only restriction is that the costs of all edges emanating from a particular vertex v are chosen from the same distribution. Bloniarz's model extended previous work and is arguably the most general model studied in the shortest path literature (see, for example, [8, 33, 79, 99, 93]). To our knowledge the only random cost model studied, which is incomparable to Bloniarz's is a recent one of Walley and Tan [101]. Following the first publication of this work [65], Cooper et al. [17] studied the vertex-potential model which disallows negative-cost cycles. The vertex-potential model had been previously used by Cherkassky et al. [14] in an experimental evaluation of shortest path algorithms.

Our method uses ideas from Bellman–Ford and from an algorithm of Moffat and Takaoka [79] originally intended for nonnegative-cost assignments, and turns out to be quite simple, drawing on the simplicity of these two algorithms. In addition, we show how ideas from our shortest path algorithm can be used to obtain an $O(n^2 \log n)$ expected time implementation of Karp's algorithm [54] for the *minimum mean cycle* problem when the input satisfies the requirements of the endpoint-independent model. In this problem, given a directed network, a cycle is sought with the smallest *mean cost,* i.e., the ratio of its cost to the number of edges in the cycle. The worst-case time complexity of Karp's minimum mean cycle algorithm is $O(nm)$. Again we show that our algorithm executes within the $O(n^2 \log n)$ time bound with high probability.

Interestingly, our result yields the same bound for the average-case complexity of the SSSP and the minimum mean cycle problems as the one that is known for the

assignment problem. Karp [55] gave an $O(n^2 \log n)$ expected time algorithm for the assignment problem on an input model very similar to the endpoint-independent one. The worst-case complexity of the assignment problem is also $O(n^3)$, the same as that of SSSP and minimum mean cycle.

Finally we show that in a restricted model of computation, the Bellman–Ford algorithm is the best possible. We consider an *oblivious* model of computation, in which the decisions about which edges to relax are made in advance, before the input is seen. We show that in this model, any algorithm whose basic operation is edge relaxation has to perform $\Omega(nm)$ edge relaxations in the worst case to correctly compute shortest paths. The Bellman–Ford algorithm fits into this model.

Previous and Related Work. For arbitrary real costs the existence of negative-cost cycles, i.e. paths of negative cost in which every vertex has in-degree and out-degree 1, makes the SSSP problem NP-hard [36]. In the absence of negative-cost cycles, the fastest strongly polynomial SSSP algorithm, as mentioned above, is attributed to Bellman and Ford [7, 30] and can be implemented to run in $O(nm)$ time, worst case. This is $O(n^3)$ for dense graphs. Until recently, all alternative implementations of Bellman–Ford first solved an assignment problem to find vertex potentials, which allows reweighting of edges so that all edge costs become nonnegative. Then Dijkstra's algorithm [21] is applied to the reweighted network. The bottleneck in this approach is the solution of the assignment problem. The first and fastest strongly polynomial-time algorithm for the assignment problem is Kuhn's Hungarian algorithm [69]. Implemented with Fibonacci heaps [32], this algorithm runs in $O(nm + n^2 \log n)$ time. Gabow and Tarjan [35] gave a scaling algorithm for the assignment problem that runs in $O(\sqrt{n}m \log(nC))$ time, where C is the largest absolute value of an edge cost.

Goldberg [38] proposed a scaling algorithm that finds shortest paths without solving an assignment problem first; this algorithm has a running time of $O(\sqrt{n}m \log C)$. All of these algorithms detect the existence of a negative-cost cycle. We also note that if the costs are nonnegative, faster algorithms are possible, as Dijkstra's algorithm [21] implemented with Fibonacci heaps [32] runs in $O(n \log n + m)$ time.

We are not aware of any work on the average-case complexity of the SSSP problem for real-valued edge costs. However, the *all pairs shortest path* problem with nonnegative edge costs is well studied and the relevant literature spans two decades. In the all pairs shortest path problem, (APSP), we are interested in computing the shortest paths between all $n(n-1)$ pairs of vertices. All previous work on the analysis of algorithms for networks with random nonnegative edge costs has been for the APSP problem. In most of these papers an SSSP routine is run n times, using each vertex as a source in turn. Because these algorithms require an $O(n^2 \log n)$ time preprocessing sorting phase, the running times are only reported for the APSP problem, but these algorithms do give ideas for the SSSP problem. A first APSP algorithm with an expected running time of $O(n^2 \log^2 n)$ on networks with independently and identically distributed edge costs was presented in a classical paper by Spira [93] (see [12] for minor corrections). This result was later refined [9] to take into account nonunique edge costs and improved in [99], where an $O(n^2 \log n \log \log n)$ expected time algorithm was given. Bloniarz [8] achieved an expected running time of $O(n^2 \log n \log^* n)$ and relaxed Spira's initial assumption that edge costs are drawn independently from any single but arbitrary distribution. He introduced the more general endpoint-independent randomness model. Hassin and Zemel [43] considered the case in which the edge costs are uniformly distributed independently and gave an $O(n^2 \log n)$ expected time algorithm. Their result extends under some assumptions to independently and iden-

tically distributed edge costs. Frieze and Grimmett [33] gave an $O(n^2 \log n)$ expected time algorithm for the case in which edge costs are identically and independently distributed with distribution function F, where $F(0) = 0$ and F is differentiable at 0. The fastest algorithm so far under the endpoint-independent model is due to Moffat and Takaoka [79] and runs in $O(n^2 \log n)$ expected time. Mehlhorn and Priebe [78] corrected an oversight in the analysis given by Moffat and Takaoka and provided a slightly modified version of the algorithm that runs in $O(n^2 \log n)$ time with high probability. They also showed that under weak assumptions $\Omega(n \log n)$ time is required with high probability for solving SSSP on networks with the endpoint-independent distribution. Recently some research has been done on randomized algorithms that use ideas from matrix multiplication [4, 89], but, for arbitrary cost assignments, only pseudopolynomial algorithms exist. The extent to which randomization can be used for faster algorithms, as was the case with e.g. minimum cut [48] and minimum spanning tree [49], is an open question.

The outline of the chapter is as follows. In Section 2.2 we start with a high-level description of the new algorithm. Subsequently we present the randomness model, and give an implementation with fast average case. In Section 2.3 we present the lower bound for oblivious algorithms. In Section 2.4 we explain how the shortest path algorithmic ideas extend to the minimum mean cycle problem. In Section 2.5 we conclude with a discussion of our results. A preliminary version of the material in this chapter appeared in [65].

2.2 The Shortest-Path Algorithm

We use n to denote $|V|$ and m to denote $|E|$ for the network $G = (V, E, c)$ of interest. In this section we give an algorithm for SSSP with average-case running time $O(n^2 \log n)$ on a broad class of networks with random edge costs. We will give the algorithm in two parts. In Section 2.2.1 we give a modified version of the Bellman–Ford [7, 30] algorithm that reduces solving a shortest path problem to a sequence of n shortest path problems in a simpler network. Then in Section 2.2.2 we show how to solve this simpler shortest path problem in $O(n \log n)$ time, on average.

The *length* of path $p = (v_1, v_2, \ldots, v_k)$ is $k - 1$ while the *cost* is defined as $c(p) = \sum_{i=1}^{k-1} c(v_i, v_{i+1})$. The *shortest path* between a pair of vertices u, v is a minimum-cost simple path joining the two vertices. This minimum cost is called the *distance*, denoted by $d(u, v)$. Our algorithms will concentrate on finding the distance from the source s to each vertex, and we will use $d(v)$ to denote $d(s, v)$. Our method can easily be modified to find the actual paths without any asymptotic overhead.

2.2.1 Skeleton of the new algorithm

We begin by reviewing and modifying the Bellman–Ford algorithm. On a network in which negative costs are present, Bellman–Ford performs $n - 1$ passes. Let $d^i(v)$ denote the distance estimate from s to v that has been found during the first i passes. During pass i, all edges (u, v) are relaxed in an arbitrary order. More precisely, pass i is implemented by first assigning $d^i(v) := d^{i-1}(v)$ for all v and then, for all edges (u, v), if

$$d^i(u) + c(u, v) < d^i(v), \tag{2.1}$$

```
ALGORITHM FAST_SSSP(G = (V, E), source s)
    for all v ∈ V do
        sort the list of edges out of v;
        d^0(v):= ∞;
    d^0(s):= 0;
    for i = 1 to n − 1 do
        d^i(.):=PASS(G, d^{i−1});
    for all edges (u, v) ∈ E do
        if d^{n−1}(u) + c(u, v) < d^{n−1}(v) then
            report a negative-cost cycle and break;
```

Figure 2.1: Algorithm FAST_SSSP

then $d^i(v)$ is updated to $d^i(u)+c(u,v)$. This step is called *relaxing* edge (u,v). An extra n-th pass will reveal the existence of a negative-cost cycle, since any possible vertex cost decrease will be due to a non-simple path. Adding a simple preprocessing phase to Bellman–Ford, in which we sort the edge lists, we get the algorithm FAST_SSSP, which appears in Figure 2.2.1.

In what follows, we state a number of invariants about the distance estimates computed by the Bellman-Ford and FAST_SSSP algorithms. To avoid cumbersome phrasing, we assume in stating these invariants that the network contains no negative-cost cycle. If this is not the case, the existence of a negative cycle is detected by both algorithms without asymptotic overhead. It is easy to show by induction that after the i-th pass, all shortest paths of length *at most* i have certainly been discovered, and possibly others. More precisely, let $\hat{d}^i(v)$ be the cost of a shortest path from s to v of length at most i. Then for all i and v, Bellman–Ford maintains the invariant that at the end of pass i, $d(v) \leq d^i(v) \leq \hat{d}^i(v)$.

In our statement of FAST_SSSP in Figure 2.2.1, we treat as a black box procedure PASS, which updates the distance labels. The standard Bellman–Ford algorithm implements PASS by relaxing (applying Eq. (2.1) to) all m edges, yielding a total

running time of $O(nm)$.

Implementing Pass**.** Our improvements will come from carefully implementing procedure Pass to run in $O(n \log n)$ expected time. We begin by observing that the correctness of Bellman–Ford is not harmed if we only compute the $\hat{d}^i(v)$'s during iteration i; after $n-1$ iterations, we will then have computed $\hat{d}^{n-1}(v)$, which is exactly what we want.

This allows us to make the following modifications:

Modification 1. *We can relax all edges in "parallel".* Typically, a sequential implementation of Bellman–Ford does not do this, as it only slows the algorithm down. However, we can accomplish this by replacing condition (2.1) with the following condition:

$$d^{i-1}(u) + c(u, v) < d^i(v). \tag{2.2}$$

If we apply this rule instead, it will clearly be the case that after pass i, $d^i(v) = \min\{\min_{u \in V}\{d^{i-1}(u) + c(u, v)\}, d^{i-1}(v)\}$. However, since we are only allowing the length of a path to grow by at most one edge in each pass, we can show inductively that we are actually computing $\hat{d}^i(v)$ in iteration i.

Modification 2. *During pass i only vertices already at length exactly $i-1$ from the source need to have their outgoing edges relaxed.* This is an idea that has been used before, see e.g. [42]. Consider a vertex v and let $j \leq i-1$ be the minimum value such that $d^{i-1}(v) = d^{i-2}(v) = \ldots = d^{j-1}(v)$, i.e. a vertex that is at length exactly $j-1$ from the source. Then according to our rule, v had its outgoing edges relaxed for the last time during pass j, and hence for all neighbors w,

$$d^{i-1}(w) \leq d^{j-1}(v) + c(v, w) = d^{i-1}(v) + c(v, w).$$

Thus, applying (2.2) to v's outgoing edges will not cause any vertex labels to be

updated.

In light of the two modifications above, we observe that during pass i, relaxing all edges in the set I_v of incoming edges to vertex v may be too much. Of all the edges in I_v, only one is crucial, namely the one that will give the minimum value for $d^i(v)$. Of course, we do not know which edge this is *a priori*, but we will capture a set of edges that will contain it. To do this, we recast the implementation of pass i as a modified SSSP problem in an auxiliary network. Let a *single-source, two-level network* be a directed network $G_\varepsilon = (V_\varepsilon, E_\varepsilon, c_\varepsilon)$, where $V_\varepsilon = s \cup V_1 \cup V_2$, and edges connect only the source s to vertices in V_1 and the vertices in V_1 to vertices in V_2. The cost function c_ε ranges over the reals. We show how an auxiliary single-source, two-level network can be used in the implementation of PASS.

We define now a particular G_ε in terms of the input to PASS, namely $G = (V, E, c)$ and $d^{i-1}(.)$. Let V' be the set of vertices v in G whose current shortest path estimate derives from a path of length exactly $i-1$, i.e. $d^{i-1}(v) < d^{i-2}(v)$. Then V_1 contains a copy of every vertex in V'. V_2 contains a copy of every vertex $v \in V - s$. We introduce two types of edges in E_ε. First, for each pair $(x, y) \in V_1 \times V_2$, where x is a copy of u, y is a copy of v and $(u, v) \in E$, we add an edge (x, y) with $c_\varepsilon(x, y) = c(u, v)$. Second, for all $x \in V_1$, x a copy of v, we add an edge (s, x) with $c_\varepsilon(s, x) = d^{i-1}(v)$.

We use these definitions in the upcoming lemma.

Lemma 2.2.1 *Let $T = T(n, m)$ be the time needed to solve a* SSSP *problem in a single-source, two-level network with $\Theta(n)$ vertices and $O(m)$ edges. Then, routine* PASS(G, d^{i-1}) *can be implemented to compute $d^i(.)$ in $O(T + n)$ time. Hence,* FAST_SSSP *is correct and has time complexity $O(n^2 + nT + m \log n)$.*

Proof. The implementation of pass i reduces to solving SSSP in the auxiliary network G_ε, followed by a finishing step. Let SSSP_AUX(G_{aux}, S) be any routine that solves

```
procedure PASS(G, d^{i-1})
    d_ε = SSSP_AUX(G_ε, s ∪ V_1);
    d^i(v) = min{d_ε(v), d^{i-1}(v)};
```

Figure 2.2: Procedure FAST_SSSP

SSSP on a network G_{aux} in time T, where S is the set of vertices in G_{aux} with known distances. We implement PASS as shown in Figure 2.2.1.

By the construction of G_ε, we know that the value of $d_\varepsilon(v)$ for $v \in V_1$ is equal to $d^{i-1}(v)$. Thus computing distances $d_\varepsilon(v)$ in G_ε for the vertices in V_2 is equivalent to computing

$$\min_u \{d^{i-1}(u) + c(u,v)\}, \ \ \forall v \tag{2.3}$$

in G. Note that this is the same as calculating the left hand side of (2.2) for all edges in G. Thus in the last step of PASS, when $d^i(v)$ is calculated as shown, it is equivalent to the process described above, where d^i is first initialized to d^{i-1} and then the test in (2.2) is applied to every edge. Inductively, if the given $d^{i-1}(.)$ is correct, $d^i(.)$ as output by PASS captures the shortest paths of length at most i in G, and thus routine PASS is correct.

For the running time, we notice that G_ε does not need to be explicitly constructed. All that is required is to identify the set of vertices whose current shortest path length is exactly $i-1$, and hence the modified SSSP computation can easily be set up in $O(n)$ time. The last step requires an additional $O(n)$ time, so the total time required for PASS is $O(T+n)$, and the lemma follows. ■

2.2.2 An implementation with fast average case

Our goal now is to implement the SSSP_Aux routine to run in $o(n^2)$ time on the particular network G_ε. We are unaware of a method achieving such a worst-case bound but we show how to do it in $O(n \log n)$ expected time on networks with random edge costs. In this section we define the class of probability measures for which our analysis holds and then present an algorithm by Moffat and Takaoka [79] and show that it can be used to efficiently find shortest paths in G_ε.

We define first the random cost model used for the analysis. The definition follows [8] except that we allow negative costs as well. Let $\mathcal{G}_n$ be the set of all n-vertex directed networks and suppose P is a probability measure on $\mathcal{G}_n$. We may identify $\mathcal{G}_n$ with the set of all $n \times n$ matrices with entries in $(-\infty, +\infty)$. P is uniquely characterized by its distribution function, $F_P : \mathcal{G}_n \to [0, 1]$, defined by

$$F_P(G) = P\{G' \in \mathcal{G}_n | c_{G'}(i,j) \leq c_G(i,j) \text{ for } 1 \leq i, j \leq n\}.$$

We say that P is *endpoint independent* if, for $1 \leq i, j, k \leq n$ and $G \in \mathcal{G}_n$, we have that

$$F_P(G) = F_P(G'),$$

where G' is obtained from G by interchanging the costs of edges (i, j) and (i, k). Intuitively exchanging endpoints of edges emanating from any fixed vertex does not affect the probability.

Several natural edge-cost assignments are endpoint independent, including the one used by Spira [93] in which edge costs are independently, identically distributed random variables. Another probability measure that meets the endpoint independence criterion is that in which each source vertex i has a probability measure P_i associated with it and the entries $c_G(i, j)$ of the matrix are drawn independently according to

distribution P_i. The reader is referred to [8] for further examples.

We proceed with a high-level description of the Moffat-Takaoka method. Moffat and Takaoka give a SSSP algorithm with an expected running time $O(n \log n)$ under the above input model for a nonnegative-cost assignment, assuming that all of the edge lists are sorted. Their algorithm is similar to Dijkstra's [21] and exploits the fact that vertices extracted in increasing cost order from a priority queue cannot have their cost decreased later on; once a vertex v is removed from the priority queue, its distance estimate is the cost of the actual shortest path from the source to v. Of course this is not true, in general, when edges with negative costs are present.

We now review the algorithm of Moffat and Takaoka. For concreteness, we will refer to the algorithm as MT and assume that is takes, as input, a network G, and a set S of vertices of G whose shortest path distances have already been computed. Recall that it relies on all edges having nonnegative costs. The set S of *labeled* vertices is maintained throughout. These are the vertices for which the shortest paths are known. For every element v in S, a *candidate* edge (v, t) is maintained, which is known to be the least cost unexamined edge out of v. Every candidate edge gives rise to a candidate path; there are at most $|S|$ of them at any one time. The candidate paths are maintained in a priority queue. At every step we seek to expand S by picking the least cost candidate path p. If p with final edge (v, t) leads to an unlabeled vertex t, it is deemed to be *useful*, and t is added to S. Otherwise, the path is ignored, but in both cases the candidate path (v, t) is replaced by a different path ending in (v, t') with $c(v, t') \geq c(v, t)$. There are two extremes in the policy for picking this next candidate (v, t'): either simply select the next largest edge out of v or scan the sorted adjacency list until a useful edge (v, t') is found. Based on the policy selection, we obtain Spira's [93] and Dantzig's [19] methods respectively. The

Moffat-Takaoka routine (MT) uses Dantzig's algorithm up to a critical point with respect to the size of S and then switches to Spira's. Using standard binary heaps, it can be shown that the original analysis of the algorithm by Moffat and Takaoka together with a correction by Mehlhorn and Priebe [78] yields the following lemma.

Lemma 2.2.2 ([79],[78]) *Let G be a network of n vertices with nonnegative edge costs drawn from an endpoint-independent distribution and let S be a set of vertices of G whose shortest path distances have been computed. Then* $\mathrm{MT}(G, S)$ *solves* SSSP *on G in $O(n \log n)$ expected time, given that the edge lists are presorted by cost.*

Proof. See Theorem 1 in [79] and [78]. ■

We note that, in general, the nonnegativity condition is necessary for correctness, as it is in Dijkstra's algorithm. However, we will now show that for a single-source, two-level network with real costs, such as G_ε, MT computes correctly shortest paths.

Lemma 2.2.3 *Let $G_\varepsilon = (s \cup V_1 \cup V_2, E_\varepsilon, c_\varepsilon)$ be the single-source, two-level network defined above. Then* $\mathrm{MT}(G_\varepsilon, s \cup V_1)$ *solves* SSSP *on G_ε in $O(n \log n)$ expected time, given that the edge lists of vertices in V_1 are presorted by cost, and the edge costs are drawn from an endpoint-independent distribution.*

Proof. Given a set S of already labeled vertices, MT chooses to label next out of the vertices adjacent to S the one corresponding to the least cost candidate path. In the case of G_ε the only paths to vertices in V_2 are the ones passing through V_1, and MT will process each of them in sorted order until all the vertices in V_2 are labeled. Thus it will compute correctly shortest paths. For the running time, it suffices to notice that the analysis in Theorem 1 of [79] relies on the following fact. In an endpoint-independent distribution when $|S| = j$ each candidate leads to each of the $n - j$

unlabeled vertices with equal probability. This fact is not harmed by the negativity of a candidate edge. Thus by Lemma 2.2.2, MT on G_ε has $O(n \log n)$ expected time.

An alternative self-contained proof is the following. A sufficiently large constant C may be added to all edge costs in G_ε to make them nonnegative. This reweighting increases the costs of all paths to $v \in V_2$ by $2C$ and thus does not change the relative ordering among the paths; MT can be used to compute the $d_\varepsilon(v)$'s correctly in $O(n \log n)$ expected time. Therefore MT also takes $O(n \log n)$ expected time on the original, nonreweighted network. We note that in the context of our FAST_SSSP algorithm, the reweighting of all the auxiliary networks can be done in $O(n^2)$ worst case total time; however we do not include it in the algorithm to avoid unnecessary complication. ■

By Lemmata 2.2.1 and 2.2.3 we obtain the following result.

Theorem 2.2.1 *Let G be a network of n vertices with real edge costs drawn from an endpoint-independent distribution. Algorithm* FAST_SSSP *solves* SSSP *on G in $O(n^2 \log n)$ expected time if no negative-cost cycle exists. Otherwise it reports the existence of a negative-cost cycle in the same time bound.*

Modifying the Moffat-Takaoka routine to use Fibonacci heaps [32], Mehlhorn and Priebe also obtained a high probability result. In our setting it can be phrased as follows.

Lemma 2.2.4 ([78],[79]) *Let G be a network of n vertices with nonnegative edge costs drawn from an endpoint-independent distribution, and let S be a set of vertices of G whose shortest path distances have been computed. If the edge lists are presorted by cost,* MT(G, S) *solves* SSSP *on G, and its running time is $O(n \log n)$ with probability $1 - O(n^{-\beta})$, for some constant $\beta > 0$.*

Obtaining the high probability equivalent of Lemma 2.2.3 is straightforward. The constant β in Lemma 2.2.4 can be made greater than 1, in which case we obtain the following.

Theorem 2.2.2 *Let G be a network of n vertices with real edge costs drawn from an endpoint-independent distribution. Algorithm* FAST_SSSP *solves* SSSP *on G if no negative-cost cycle exists and its running time is $O(n^2 \log n)$ with probability $1 - O(n^{-\gamma})$, for some constant $\gamma > 0$. Otherwise it reports the existence of a negative-cost cycle in the same time bound.*

The worst-case complexity of the Moffat-Takaoka subroutine is $O(n^2)$, therefore the worst-case running time of our algorithm is $O(n^3)$.

2.3 A Lower Bound

For over 30 years Bellman–Ford has been the fastest algorithm for the general SSSP problem with an $O(nm)$ worst-case running time. It is thus natural to wonder whether one can show a lower bound on the running time of SSSP algorithms in networks with real-valued edge costs. We do not know how to do this, but we can show a lower bound in a restricted model of computation that captures Bellman–Ford-like algorithms.

Time lower bounds in natural computational models are known for very few problems. For shortest paths, we are aware of two previous lower bound results. Karger, Koller and Phillips [50] showed an $\Omega(nm^*)$ lower bound for APSP in a path-comparison model, where m^* denotes the number of edges in a shortest path solution. In the path-comparison model, edge costs are accessed only through pairwise comparisons of path costs. Mehlhorn and Priebe [78] defined a model in which the algorithm can make two types of queries: (i) what is the cost of a given edge e? (ii) given a vertex $v \in V$

and an integer $k \in \{1, \ldots, n\}$, what is the head of the edge with cost k and tail v? Mehlhorn and Priebe show that any SSSP algorithm in this model takes $\Omega(n \log n)$ time with high probability on a complete digraph.

Observe that the Bellman–Ford algorithm disregards any information about the network except the number of edges, and performs a fixed sequence of edge relaxations until no vertex label can be decreased. Motivated by this, we define an *oblivious* algorithm for SSSP to be one that, given as input a network G with m edges numbered $1, \ldots, m$, decides on a sequence $S_m = e_{i_1} e_{i_2} \ldots e_{i_k}$ of k edge relaxations, $1 \leq e_{i_j} \leq m$, on the basis of m alone. An oblivious algorithm performs only the chosen relaxations. Obviously, Bellman–Ford falls into the oblivious class, as do generalizations that do not restrict the algorithm to operate in phases. In the following theorem we prove that the Bellman–Ford algorithm performs an asymptotically optimal number of relaxations within this model. We call an edge *optimal* if it belongs to a shortest path.

Theorem 2.3.1 *Let $\mathcal{A}$ be any oblivious* SSSP *algorithm. There exists a network G with n vertices and m edges on which $\mathcal{A}$ must perform $\Omega(nm)$ edge relaxations worst case to correctly output the shortest paths.*

Proof. We restrict our attention to networks G, with $m > 2n$, in which the maximum length of a shortest path is exactly $n - 1$, i.e., the shortest path tree is a single path. Let k be the length of the relaxation sequence $S_m = e_{i_1} e_{i_2} \ldots e_{i_k}$ that $\mathcal{A}$ performs on input G. A *subsequence* σ of S_m is any sequence of the form $\sigma = e_{i_{j_1}} e_{i_{j_2}} \ldots e_{i_{j_l}}$, with $1 \leq j_1 < j_2 < \ldots < j_l \leq k$. The correctness of any relaxation-based algorithm comes by guaranteeing that the edges in the shortest path tree are relaxed in an order that is a topological sort of the edges of the tree. In our case, in which the shortest

path tree is a path, this means that the edges of the path must be relaxed in the linear order specified by the path. Before seeing the input, it is possible that any sequence s_n of $n-1$ out of the m edges can be the chain of optimal edges, and there are $\frac{m!}{(m-(n-1))!}$ such chains. $\mathcal{A}$ will correctly compute the shortest path distances only if s_n is a subsequence of S_m. The relaxation sequence of length k has exactly $\binom{k}{n-1}$ subsequences of length $n-1$. For the algorithm $\mathcal{A}$ to be correct on all networks, it must be that

$$\binom{k}{n-1} \geq \frac{m!}{(m-(n-1))!} \tag{2.4}$$

For simplicity of presentation we replace $n-1$ with r in the following. We want to compute k_0, the smallest possible value of k, for which (2.4) holds. Increasing the left hand side and decreasing the right hand side can only make this lower bound, k_0, smaller. Let $K(\rho(k))$ denote the minimum k such that a relation $\rho(k)$ holds. We assume, wlog, that $0 \leq n \leq k$. We can upper bound the left-hand side of (2.4) by using the identity [18]

$$\frac{k^k}{r^r(k-r)^{k-r}} \geq \binom{k}{r}. \tag{2.5}$$

We can lower bound the right-hand side of (2.4) by

$$\frac{m!}{(m-r)!} = m(m-1)\ldots(m-r+1) \geq (m-r)^r. \tag{2.6}$$

By (2.5) and (2.6), we get

$$k_0 = K\left(\binom{k}{r} \geq \frac{m!}{(m-r)!}\right) \geq K\left(\frac{k^k}{r^r(k-r)^{k-r}} \geq (m-r)^r\right) = k_1.$$

But $k^{k-r}/(k-r)^{k-r} \leq e^r$ thus we have that

$$k_1 \quad \geq \quad K\left(\frac{k^r e^r}{r^r} \geq (m-r)^r\right)$$

$$
\begin{aligned}
&= K\left(k^r \geq \left(\frac{r(m-r)}{e}\right)^r\right) \\
&= K\left(k \geq \frac{r(m-r)}{e}\right).
\end{aligned}
$$

Substituting $n-1$ for r gives $k_0 = \Omega(nm)$. ■

2.4 A Fast Algorithm for the Minimum Mean Cycle Problem

Given a directed network $G = (V, E, c)$, let the *length* $|W|$ of a cycle W be defined as the number of vertices in W. The minimum mean cycle problem consists of finding a cycle W with the minimum *mean cost,* i.e. a cycle W that minimizes the ratio $\sum_{(i,j)\in W} c(i,j)/|W|$. The currently fastest algorithm is due to Karp [54] and finds a minimum mean cycle in $O(nm)$ worst-case time. We are not aware of any average-case results for this problem. In this section we show how to implement Karp's algorithm to run in $O(n^2 \log n)$ expected time on graphs with edge costs chosen from endpoint-independent distributions. The time bound also holds with high probability.

An *edge progression* of length l from $v_{j_1} \in V$ to $v_{j_{l+1}} \in V$, is a sequence of edges $(v_{j_1}, v_{j_2}), (v_{j_2}, v_{j_3}), \ldots, (v_{j_k}, v_{j_{k+1}}), (v_{j_{k+1}}, v_{j_{k+2}}), \ldots, (v_{j_l}, v_{j_{l+1}})$. Observe that an edge progression may contain cycles. Let λ^* denote the minimum mean cost of a cycle in G and $D^i(v)$ be the minimum cost of an edge progression of length i from a chosen source $s \in V$ to v. As shown in [54] $D^i(v)$ can computed with the following recurrence:

$$
D^i(v) = \min_u \{D^{i-1}(u) + c(u,v)\} \ \forall v \in V, 1 \leq i \leq n \tag{2.7}
$$

Karp's algorithm is based on the following theorem.

Theorem 2.4.1 *[54]*

$$\lambda^* = \min_{v \in V} \max_{0 \le k \le n-1} \left\{ \frac{D^n(v) - D^k(v)}{n-k} \right\}.$$

Therefore, once the quantities $D^i(v)$, $\forall v \in V$, $1 \le i \le n$, have been computed, λ^* can be determined in $O(n^2)$ time. An actual cycle of mean cost λ^* can be found without further asymptotic overhead with some additional bookkeeping operations.

Theorem 2.4.2 *Let G be a network of n vertices with real edge costs drawn from an endpoint-independent distribution. The minimum mean cycle problem on G can be solved in $O(n^2 \log n)$ expected time.*

Proof. By the description above, one deduces that the time bottleneck of Karp's algorithm is the computation of the quantities $D^i(v)$ by the recurrence (2.7). Comparing recurrences (2.3) and (2.7) for a fixed i we see that they are of the same form except for the different meaning of the values computed by each. As a result we can cast the implementation of recurrence (2.7) as a single source problem on a two-level auxiliary network as the one described in Section 2.2.1 for recurrence (2.3). By Lemma 2.2.3 the auxiliary shortest path problem and by consequence the i-th phase of Karp's algorithm can be implemented to take $O(n \log n)$ expected time. ■

By Lemma 2.2.4 we can easily derive a high probability result.

Theorem 2.4.3 *Let G be a network of n vertices with real edge costs drawn from an endpoint-independent distribution. There is an algorithm that solves the minimum mean cycle problem on G, whose running time is $O(n^2 \log n)$ with probability $1 - O(n^{-\gamma})$, for some constant $\gamma > 0$.*

	worst case	average case
assignment	$O(n^3)$ [69, 32]	$O(n^2 \log n)$ [55]
SSSP	$O(n^3)$ [7, 30]	$O(n^2 \log n)$ [this thesis]
minimum mean cycle	$O(n^3)$ [54]	$O(n^2 \log n)$ [this thesis]

Table 2.1: Best strongly polynomial time bounds for three related problems. The input model in [55] is defined as a special case of the endpoint-independent one. Worst-case time bounds are reported for dense graphs.

2.5 Discussion

In this chapter we provided deterministic algorithms that improve on the average-case time complexity of two fundamental network problems: SSSP and minimum mean cycle. The algorithms are relatively simple and in both cases provide the first known strongly polynomial improvement over the worst-case cubic time bounds. In addition, our algorithms achieve the $O(n^2 \log n)$ time bound with high probability. Putting our work in perspective, we may contrast our results with the already known $O(n^2 \log n)$ result for the assignment problem [55]. Table 2.1 provides the best known running times (on dense graphs) for the assignment, SSSP and minimum mean cycle problems, both worst case and average case. These three problems are known to be conceptually related. Table 2.1 indicates that as far as we currently know, these three problems share the same worst-case and average-case time bounds.

It is interesting to note that a linear time (ignoring log factors) reduction is known to exist from SSSP to the assignment problem [34]. We have been unable to use this reduction in conjunction with Karp's $O(n^2 \log n)$ assignment algorithm [55] to obtain a SSSP algorithm as fast as ours. The reduction is based on using an assignment algorithm to compute vertex potentials $\pi(.)$ that dominate the edge costs. In other words it must be

$$-c(u,v) \leq \pi(u) - \pi(v), \ \ \forall u, v.$$

Karp's algorithm in [55] is of the shortest augmenting path type. It starts by performing the standard transformation of adding a big positive constant to the real-valued costs. Thus the potentials it computes do not satisfy the relation above. On the other hand, if the first step of adding the constant is not taken the average-case analysis in [55] does not hold.

Chapter 3

Approximation Algorithms for Single-Source Unsplittable Flow[1]

3.1 Introduction

In the *multiple-source unsplittable flow* problem $(G, \mathcal{T})$, we are given a capacitated network $G = (V, E, u)$, and a set $\mathcal{T}$ of connection requests, also called *commodities.* Each commodity i in $\mathcal{T}$ is a triple (s_i, t_i, ρ_i), where $s_i, t_i \in V$, are the *source* and *sink* vertices and $\rho_i \in (0, 1]$, is a real-valued demand. The minimum edge capacity is assumed to have value at least $\max_i \rho_i$. We seek to route the demand ρ_i of each commodity i, along a single s_i-t_i flow path, so that the total flow routed across any edge e is bounded by the edge capacity u_e. The requirement of routing each commodity on a single path bears resemblance to integral multicommodity flow and in particular to the multiple-source edge-disjoint path problem (see Section 4.1 for further background on disjoint-path problems). For the latter problem, a large amount

[1]This chapter contains joint work with Cliff Stein [66].

of work exists either for solving exactly interesting special cases (e.g. [31], [88], [87]) or for approximating, with limited success, various objective functions (e.g. [86],[37], [63], [64],[60]). We examine the general multiple-source unsplittable flow problem in Chapter 4. In this chapter we focus on the important special case of the *single-source unsplittable flow* problem, denoted UFP. In UFP all commodities in $\mathcal{T}$ share a common source vertex s. See Figure 3.1 for an example instance. Shedding light on single-source unsplittable flow may lead to a better understanding of multiple-source disjoint-path problems. From a different perspective, UFP can be approached as a variant of standard, single-commodity, maximum flow since in both problems there is one source for the flow. From this point of view, UFP generalizes single-source edge-disjoint paths. UFP is also important as a unifying framework for a variety of scheduling and load balancing problems [61]. It can be used to model bin-packing [61] and certain scheduling problems on parallel machines [74]. Another possible application for UFP is in virtual-circuit routing, where the source would represent a node wishing to multicast data to selected sinks. The conceptual difficulty in UFP arises from combining packing constraints due to the existence of capacities with path selection through an arbitrary graph.

The feasibility question for UFP, i.e., can all commodities be routed unsplittably, is strongly NP-complete [60]. We focus thus on efficient algorithms to obtain approximate solutions. Three main optimization versions of unsplittable flow can be defined.

minimum congestion. Find an unsplittable flow f satisfying all demands, which minimizes *congestion,* i.e. the quantity $\max\{\max_e\{f_e/u_e\}, 1\}$. In words, minimum congestion is the smallest number $\alpha \geq 1$, such that if we multiplied all capacities by α, f would satisfy capacity constraints. The congestion metric

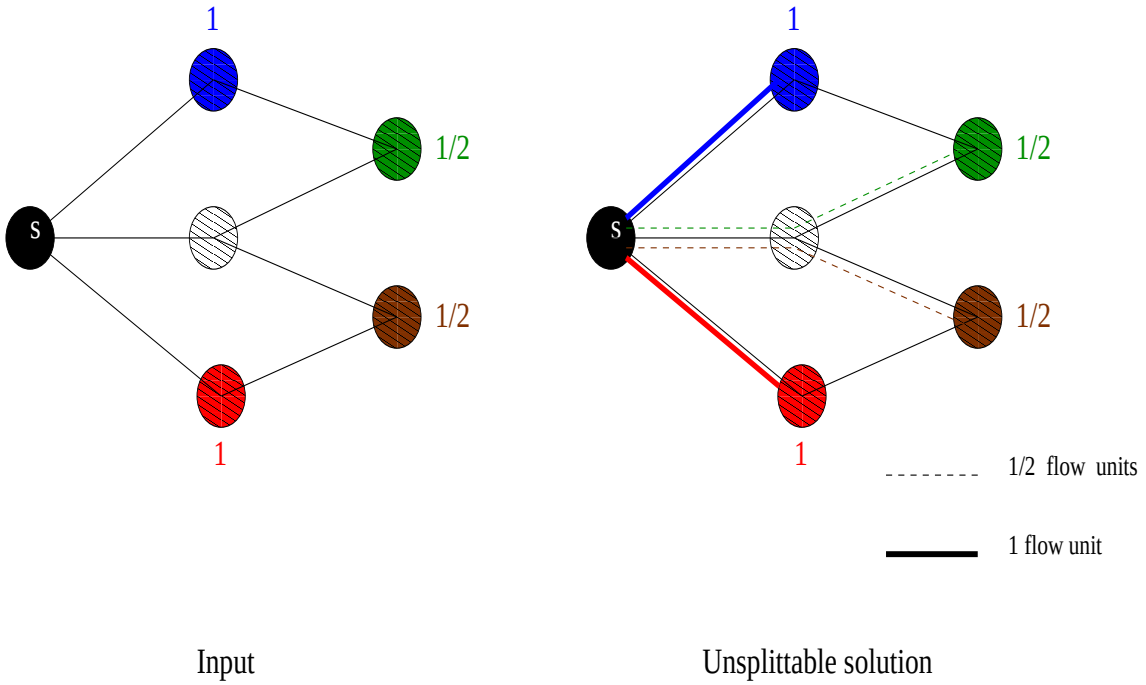


Figure 3.1: Example UFP instance. Numbers on the vertices denote demands; all edge capacities are equal to 1.

was a primary testbed for the randomized rounding technique of Raghavan and Thompson [86] and has been studied extensively for its connections to multi-commodity flow and cuts (e.g. [73, 58, 72, 59]) [2]

maximum routable demand. Route unsplittably a subset of the commodities of maximum total demand, while respecting capacity constraints.

minimum number of rounds. Partition the commodities into the a minimum number of sets, called *rounds,* so that commodities assigned to the same round can be routed unsplittably without violating the capacity constraints.

The randomized rounding technique applies to the more general multiple-source

[2] In [72] congestion is defined as $\max_e\{f_e/u_e\}$. In this setting a congestion of $\lambda \in (0,1)$, implies that it is possible to multiply all demands by $1/\lambda$ and still satisfy them by a flow which respects the capacity constraints. Algorithmically the two definitions are equivalent: with a polynomial number of invocations of an algorithm which minimizes congestion according to our definition, one can minimize congestion according to the definition in [72].

unsplittable flow problem and provides an $O(\log |E| / \log \log |E|)$ approximation for congestion; for the maximum demand and minimum number of rounds versions, it can give meaningful approximations only when the minimum capacity value is $\Omega(1/\log |E|)$ times the value of the maximum demand [86, 85]. Kleinberg [60, 61] was the first to consider UFP as a distinct problem and gave constant-factor approximation algorithms for all three optimization versions presented above, on both directed and undirected graphs.

Our results. We give a new approach for single-source unsplittable flow that has three main advantages.

- The algorithms are simple and do not need the machinery developed in [61].
- Our approach treats in a unified manner directed and undirected graphs.
- We obtain significant improvements upon the approximation ratios for all three optimization versions.

All our algorithms follow the same *grouping-and-scaling* technique. We first find a maximum flow. This is a "splittable" solution, i.e., it allows the demand for a commodity to be split along more than one path. As in the previous work [61] our work uses the maximum flow solution as a guide for the discovery of an unsplittable routing. However we use this solution in a much stronger way: we employ information from maximum flow to *allocate capacity* to a set of subproblems, where each subproblem G^i contains commodities with demands in a fixed range. For each subproblem G^i, a near-optimal routing g^i is separately computed by exploiting the fact that demands are close in range. These solutions are then combined by superimposing the g^i's on the original graph. Since we are interested in constant-factor approximations,

when combining the solutions it is important to avoid an additive deterioration of the approximation guarantees obtained for each individual G^i. One of our conceptual contributions lies into showing that this is possible, given an appropriate initial capacity allocation. Combining solutions presents different challenges for the maximum demand and minimum number of rounds versions of our algorithms as opposed to minimizing congestion. In the latter problem, increasing the original edge capacities as the routings g^i are superimposed on G, only affects the approximation ratio. For the other two versions, it would be infeasible to do so. We note that a grouping scheme on the commodities is also used in [61], though in a more complicated way; ours is based solely on demand values and does not require any topological information from the graph. We now elaborate on the approximation ratios we obtain.

Minimum congestion. We give a 3-approximation algorithm for both directed and undirected graphs. The previously known bounds were 16 for the directed and 8.25 for the undirected case. Our approach gives the first constant-factor approximation for minimum-cost unsplittable flow on directed graphs; it also improves considerably upon the constants known for the minimum cost version on undirected graphs. In particular, existing results for undirected graphs [61] give a simultaneous $(7.473, 10.473)$ approximation for cost and congestion. We provide a $(2, 3)$ simultaneous approximation on both directed and undirected graphs. Moreover, we show how to obtain a $1 + \delta$ approximation for cost, for any $\delta > 0$, at the expense of a larger constant factor for congestion. Interestingly, the minimum cost version of UFP on directed graphs contains as a special case the generalized assignment problem. In the latter problem, the vertices adjacent to the source together with the sinks form a bipartite graph. Shmoys and Tardos [91] give an algorithm for the generalized assignment problem

that simultaneously achieves optimum cost and a 2-approximation for congestion.

Maximum routable demand. We show how to route at least .075, i.e., 7.5% of the optimum. In [61] the constant is not given explicitly but it can be as low as .031 for undirected graphs and of the order of 10^{-9} for directed graphs.

Minimum number of rounds. We show how to route all the demands in at most 13 times the optimum number of rounds, an improvement upon the 32 upper bound given in [61].

We emphasize that our algorithms for all three different versions are simple and make use of the same generic framework. Although they are presented separately for ease of exposition, they could all be stated in terms of one algorithm, with different subroutines invoked for subproblems. The dominant computational steps are maximum flow and flow decomposition; these are tools that work well on both directed and undirected graphs. In [61] it is noted that "the disjoint paths problem is much less well understood for directed than it is for undirected graphs ... in keeping with the general principle we will find that the algorithms we obtain for directed graphs will be more complicated".

Generalizations. The algorithmic techniques we introduce are quite general and can be applied to related problems. We show how to use them to obtain a constant-factor approximation for congestion on a network, where the minimum edge capacity can be arbitrarily low with respect to the maximum demand. This is the first result of this type that we are aware of. In Chapter 4, we show how to apply the algorithmic techniques developed here for UFP to obtain improved approximations for classes of packing integer programs and multiple-source unsplittable flow. On the negative side, we show in this chapter, that for the unsplittable flow problem with two sources on

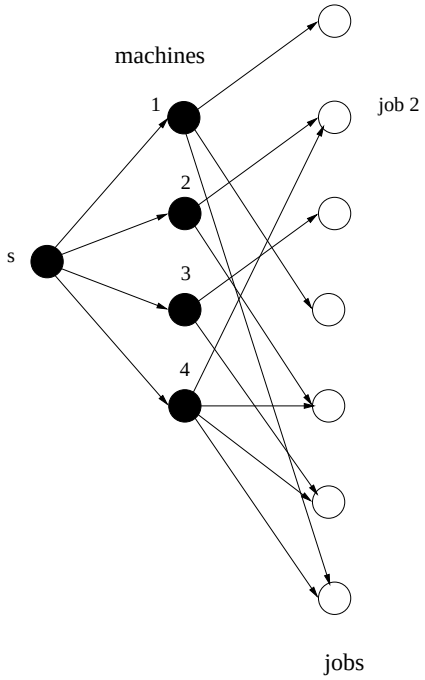


Figure 3.2: Reduction of the scheduling problem to UFP . Job 2 can be processed only on machines 2 and 4. The capacity of the edges connecting s to the machine vertices is equal to a makespan estimate.

a directed network, no ρ-approximation with $\rho < 2$ can be achieved for congestion, unless P=NP, even when all the demands and capacities have value 1. This gives an idea of how hard it might be to formulate an approximate version of the fundamental Theorem 3.2.1 that we use in our single-source algorithms.

Applications to scheduling. We also examine applications of unsplittable flow to scheduling problems. A lower bound of 3/2 exists for the approximability of minimum makespan on parallel machines when a job j can be processed only on a subset $M(j)$ of the available machines [74]. Let $\mathcal{S}$ denote this scheduling problem. The 2-approximation algorithm of Lenstra, Shmoys and Tardos [74] or Shmoys and Tardos [91] for the more general problem of scheduling on unrelated machines is the best known for $\mathcal{S}$. On the other hand an approximation-preserving reduction is known to exist from $\mathcal{S}$ to single-source unsplittable flow (see Fig. 3.2) [61], so the 3/2 lower bound for $\mathcal{S}$ [74] applies to single-source unsplittable flow as well. The lower bound holds when processing times (or demands) take the values p or $2p$ for some $p > 0$. We provide an additive p approximation for this restricted unsplittable flow problem, which corresponds to a ratio of at most 3/2. This is the best possible approximation ratio for the problem unless P=NP. Improved approximation factors are also obtained for the more general case, in which all demand values are powers of 1/2.

The outline of this chapter is as follows. In Section 3.2 we present definitions and a basic theorem from flow theory. In Section 3.3 we provide a $(4+o(1))$-approximation algorithm for congestion, which introduces our basic techniques. In Section 3.4 we provide the improved 3-approximation. In Section 3.5 we give the approximation algorithm for minimum congestion unsplittable flow with arbitrarily small capacities. In Section 3.6 we present the simultaneous cost-congestion approximation. In Sections 3.7 and 3.8 the approximation algorithms for maximum total demand and minimum

number of rounds are presented. In Section 3.9 we present the hardness result for 2-source unsplittable flow. Finally, in Section 3.10 we examine the connection between UFP and scheduling problems. A preliminary version of the material in this chapter appeared in [66].

3.2 Preliminaries

A *single-source unsplittable flow problem* (UFP) $(G, s, \mathcal{T})$ is defined on a capacitated network $G = (V, E, u)$ where a subset $\mathcal{T} = \{t_1, \ldots, t_k\}$ of V is the set of *sinks*; every edge e has a capacity u_e; each sink t_i has a demand ρ_i. The pair (t_i, ρ_i) defines the *commodity* i. To avoid cumbersome notation, we use occasionally the set $\mathcal{T}$ to denote the set of commodities. We assume in our definition that each vertex in V hosts at most one sink; the input network can always be transformed to fulfill this requirement. A *source* vertex $s \in V$ is specified. For any $I \subseteq \mathcal{T}$, a *partial routing* is a set of s-t_i paths P_i, used to route ρ_i amount of flow from s to t_i for each $t_i \in I$. A partial routing that contains a path to each sink in $\mathcal{T}$ is simply called a *routing*. Given a routing g, the flow g_e through edge e is equal to $\sum_{P_i \in g | P_i \ni e} \rho_i$. A (partial) routing g for which $g_e \leq u_e$ for every edge e, is a *(partial) unsplittable flow*. A *fractional routing* is a set of flow paths, satisfying capacity constraints, in which the flow to each sink is split along potentially many paths. A fractional routing can be computed by standard maximum flow. Thus we shall refer to a fractional routing satisfying all demands of the underlying UFP as a *feasible fractional flow solution*. For clarity we will also refer to a standard maximum flow as *fractional maximum flow*. We make two assumptions in our treatment of UFP:

C1. there is a feasible fractional flow solution.

C2. all demands are at most 1 and the minimum edge capacity is 1.

The first assumption is introduced only for simplicity. For each of the optimization metrics studied, an optimal fractional solution, potentially violating the unsplittability requirement, can be efficiently found. Since our analysis uses the fractional optima as lower bounds, our approximation results hold without C1. For the second assumption, we note that the actual numerical value of 1 is not crucial; demands and capacities can be scaled without affecting solutions. The important requirement is that the minimum capacity is greater than or equal to the maximum demand. We will explicitly state it when we deal with a UFP in which the minimum capacity can be arbitrarily small. Finally, we use (k-UFP) to denote a multiple-source unsplittable flow problem with k distinct sources.

The following theorem is an easy consequence of the well-known augmenting path algorithm ([29], [24]) and will be of use. It was also used as part of the approximation techniques in [61].

Theorem 3.2.1 *Let $(G = (V, E, u), s, \mathcal{T})$ be a* UFP *on an arbitrary network G with all demands ρ_i, $1 \leq i \leq k$, equal to the same value σ and edge capacities u_e equal to integral multiples of σ for all $e \in E$. If there is a fractional flow of value f, then there is an unsplittable flow of value at least f. This unsplittable flow can be found in time $O(km)$ where k is the number of sinks in $\mathcal{T}$.*

The above theorem is a consequence of the integrality property for maximum flow after all demands and capacities have been scaled by σ. In the same way a routing corresponds to an edge flow, an edge flow can be represented as a path flow. Our algorithms will make use of the well-known *flow decomposition theorem* [3]. Given problem $(G, s, \mathcal{T})$ let a flow solution f be represented as an edge flow, i.e. an

assignment of flow values to edges. Then one can find in $O(nm)$ time a representation of f as a path and cycle flow, where the paths join s to sinks in $\mathcal{T}$. In the remainder of the chapter, we will assume that the cycles are deleted from the output of a flow decomposition. The flow decomposition theorem trivially applies also to transforming an unsplittable flow from edge representation to a path representation. We will point out cases, in which outputting the unsplittable flow in an edge representation results in faster algorithms.

3.3 The approximation algorithm for minimum congestion

In this section we give a $(4 + o(1))$-approximation algorithm for minimizing congestion. In Section 3.4 we show how to improve the approximation guarantee to 3. All our results can be seen to hold for minimizing *absolute congestion* as well, defined as $\max\{\max_e\{f_e\}, 1\}$. The same algorithm works on both directed and undirected graphs with the same performance guarantee. The algorithm works by partitioning the original UFP into subproblems, each containing only demands in a fixed range. We use an initial maximum-flow solution to allocate capacity to different subproblems. Capacities in the subproblems are arbitrary and may violate Condition C2. The subproblems are separately solved by introducing bounded congestion; the partial solutions are then combined by superimposing on the original network G the flow paths obtained in the respective routings. The combination step will have a subadditive effect on the total congestion and thus near-optimality of the global solution is achieved.

We present first two simple lemmata for cases, in which demands are bounded.

They will be used to provide subroutines to the final algorithm.

Lemma 3.3.1 *For an* UFP *with demands in the interval* $(0, \alpha]$, *there is an algorithm,* α-ROUTING, *which finds, in time* $O(n + m)$, *an unsplittable routing where the flow through each edge is at most* $n\alpha$.

Proof. Algorithm α-ROUTING works as follows. A Depth First Search from the source s gives a path from s to each of the at most n sinks. For each sink t_i route, on the corresponding path, flow equal to the demand $\rho_i \leq \alpha$. ■

Given a real number interval with endpoints a and b, $0 < a < b$, the *ratio* $r(a, b)$ of the interval is b/a. The following fact is also used in [61]; we include a proof for the sake of completeness.

Lemma 3.3.2 *[61] Given an* UFP $\Pi = (G, s, \mathcal{T})$, *with demands in the interval* $(a, b]$ *and arbitrary capacities, there is an algorithm,* INTERVAL_ROUTING, *which finds in polynomial time an unsplittable routing* g, *such that for all edges* e, $g_e \leq r(a, b)u_e + b$.

Remark: The statement of the lemma holds also when demands lie in the interval $[a, b]$. **Proof.** Let f be a feasible fractional solution to Π. Round all demands up to b and call Π' the resulting UFP instance. To obtain a feasible fractional solution for Π', it suffices to multiply the flow f_e and the capacity u_e on each edge e by a factor of at most $r(a, b)$. Further round all capacities up to the closest multiple of b by adding at most b. The new edge capacities are now at most $r(a, b)u_e + b$. By Theorem 3.2.1 an unsplittable routing for Π' can now be found in polynomial time. Algorithm INTERVAL_ROUTING finds first this unsplittable routing g' for Π'. To obtain an unsplittable routing g for Π, from every path in g', the flow in excess of the demands in $\mathcal{T}$ is removed. ■

ALGORITHM PARTITION$(G = (V, E, u), s, \mathcal{T}, \xi, \alpha_1, \ldots, \alpha_\xi)$

Step 1. Find a feasible fractional solution f.

Step 2. Define a partition of the $(0, 1]$ interval into ξ consecutive subintervals $(0, \alpha_1], (\alpha_1, \alpha_2], \ldots, (\alpha_{\xi-1}, \alpha_\xi]$, $\alpha_\xi = 1$. Construct ξ copies of G where the set of sinks in G_i, $1 \leq i \leq \xi$, is the subset $\mathcal{T}_i$ of $\mathcal{T}$ with demands in the interval $(\alpha_{i-1}, \alpha_i]$. Using flow decomposition determine for every edge e the amount u_e^i of u_e used by f to route flow to sinks in $\mathcal{T}_i$. Set the capacity of edge e in G_i to u_e^i.

Step 3. Invoke INTERVAL_ROUTING on each G_i, $2 \leq i \leq \xi$, to obtain an unsplittable routing g^i. Invoke α-ROUTING on G_1 to obtain an unsplittable routing g^1.

Step 4. Output routing g, the union of the path sets g^i, $1 \leq i \leq \xi$.

Figure 3.3: Algorithm PARTITION.

We are now ready to present Algorithm PARTITION in Fig. 3.3. Algorithm PARTITION takes as input a UFP $(G, s, \mathcal{T})$, together with a set of parameters $\xi, \alpha_1, \ldots, \alpha_\xi$ that will determine the exact partitioning scheme. Subsequently we show how to choose these parameters so as to optimize the approximation ratio. The algorithm outputs an unsplittable routing g. The two previous lemmata will be used to provide subroutines. Recall that in an UFP demands lie in the interval $(0, 1]$ and capacities are at least 1.

We need to examine the effect of the combination on the total congestion.

Lemma 3.3.3 *Algorithm* PARTITION *outputs an unsplittable routing g with congestion at most*

$$n\alpha_1 + \max_{2 \leq i \leq \xi} \{r(\alpha_{i-1}, \alpha_i)\} + \sum_{i=2}^{\xi} \alpha_i.$$

The running time of PARTITION *is $O(T_1(n, m) + nm + m\xi)$ where $T_1(n, m)$ is the time*

to solve a fractional maximum flow problem.

Proof. In order to determine in Step 2 the capacity u_e^i of edge e in the i-th copy G_i, $1 \leq i \leq \xi$, of G we use the flow decomposition theorem [30]. Any flow along cycles given by the decomposition theorem is discarded since it does not contribute to the routing of demands. We focus on the congestion of a particular edge e. By Lemma 3.3.2 algorithm INTERVAL_ROUTING finds an unsplittable routing in G_i, $2 \leq i \leq \xi$, at Step 3, by pushing through edge e at most $r(\alpha_{i-1}, \alpha_i)u_e^i + \alpha_i$ units of flow. By Lemma 3.3.1 algorithm α-ROUTING pushes at most $n\alpha_1$ units of flow through edge e to find a routing on G_1. At Step 4 the superposition of the routings g^i gives an approximate unsplittable solution g where g_e is at most

$$\begin{aligned} n\alpha_1 + \sum_{i=2}^{\xi} u_e^i r(\alpha_{i-1}, \alpha_i) + \sum_{i=2}^{\xi} \alpha_i &\leq \\ n\alpha_1 + \max_{2 \leq i \leq \xi} \{r(\alpha_{i-1}, \alpha_i)\} \sum_{i=2}^{\xi} u_e^i + \sum_{i=2}^{\xi} \alpha_i &\leq \\ n\alpha_1 + \max_{2 \leq i \leq \xi} \{r(\alpha_{i-1}, \alpha_i)\} u_e + \sum_{i=2}^{\xi} \alpha_i u_e. & \end{aligned}$$

The last step follows because $\sum_{i=2}^{\xi} u_e^i \leq u_e$ from the feasibility of f, and $u_e \geq 1$.

As for the running time, finding the fractional flow in step 1 requires $T_1(n, m)$ time. Flow decomposition can be done in $O(nm)$ time [3], thus the running time of step 2 is $O(nm + m\xi)$. Implementing INTERVAL_ROUTING with the augmenting path algorithm gives total running time $O(km)$ for all the $\xi - 1$ invocations at Step 3. This time includes the time required by flow decomposition to output each partial routing in a path representation. The reason is that by Theorem 3.2.1 the algorithm takes $O(m)$ time per sink. The total number of sinks in the ξ subproblems is $k \leq n$. Hence the total running time of PARTITION is $O(T_1(n, m) + nm + m\xi)$. ■

In order to substantiate the approximation factor we need to come up with a

partitioning scheme to be used at Step 2 of algorithm PARTITION. The following theorem is the main result of this section.

Theorem 3.3.1 *Given an* UFP Π*, Algorithm* PARTITION *finds a* $(4+\varepsilon)$*-approximation for relative congestion, for any* $\varepsilon > 0$*. The running time of the algorithm is* $O(T_1(n,m)+nm+m\log(n/\varepsilon))$ *where* $T_1(n,m)$ *is the time to solve a fractional maximum flow problem.*

Proof. At step 2 of PARTITION, partition the interval $(0,1]$ of demand values into ξ geometrically increasing subintervals

$$(0, 1/2^{\xi-1}], \ldots, (1/2^{i+1}, 1/2^i], \ldots, (1/2^2, 1/2], (1/2, 1]$$

such that $1/2^{\xi-1} \leq \varepsilon/n$. Thus it suffices for ξ to be $\Theta(\log(n/\varepsilon))$. By Lemma 3.3.3, the congestion of the routing g returned at step 4 of the algorithm is at most $n\frac{\varepsilon}{n} + 2 + \sum_{i=0}^{\xi-2} \frac{1}{2^i} < 4 + \varepsilon$. ■

In order to achieve $\varepsilon = o(1)$, it suffices to set $1/2^{\xi-1} \leq 1/n^c$, $c > 1$. Obtaining a better $o(1)$ factor is straightforward by increasing the number of intervals. A fractional maximum flow can be found by the push-relabel method of Goldberg and Tarjan [39] whose currently fastest implementation has running time $T_1(n,m) = O(nm\log_{\frac{m}{n\log n}} n)$ [57]. In that case even when our algorithm is used to obtain a $4 + \frac{1}{2^n}$ approximation, the running time is dominated by a single maximum flow computation. Alternatively, the new maximum flow algorithm of Goldberg and Rao [41] with $T_1(n,m) = O(\min(n^{2/3}, m^{1/2})m\log(\frac{n^2}{m})\log U)$ may be used, if edge capacities can be expressed as integers in a range $[1,\ldots,U]$. We point out that if an edge representation of the output routing suffices, the Goldberg-Rao algorithm can be used to surpass the $O(nm)$ bottleneck. We sketch the idea. The flow decomposition computation in Step 2 of PARTITION can be replaced by $\xi = O(\log(n/\varepsilon))$ maximum-flow computations;

in each only the demands in the corresponding subinterval are included. The first maximum-flow computation has the entire capacity available on every edge. The flow found at the i-th computation is subtracted from the capacity allotted to the $(i+1)$-th computation. Also the INTERVAL_ROUTING subroutine is implemented by the Goldberg-Rao algorithm. The total running time of the new implementation would be $O(\log(n/\varepsilon)\min(n^{2/3}, m^{1/2})m\log(\frac{n^2}{m})\log(UD))$, assuming in the original network we have integral capacities in the range $[1, \ldots, U]$ and the minimum demand is $1/D$ for $D \in Z_{>0}$.

Although we can improve upon the 4-approximation, algorithm PARTITION introduces some of the basic ideas behind all of our algorithms. A demonstration of the algorithm execution on an example instance follows in Figures 3.4, 3.5, 3.6, 3.7. In these figures, numbers on vertices denote demands while numbers on edges denote capacities. The capacities on the original input are equal to 1. Flow paths are depicted as lines parallel to the original edges. Solid and dashed flow paths carry 1 and 1/2 units of flow respectively.

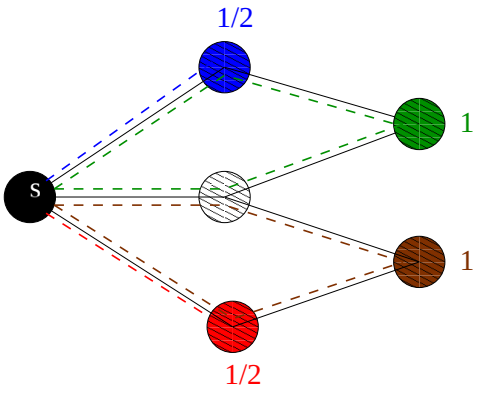
3.4 A 3-approximation algorithm for congestion

An improved approximation algorithm can be obtained by combining the partitioning idea above with a more careful treatment of the subproblems. At the first publication of our results [66] we claimed a $(3.23 + o(1))$-approximation. Subsequently we improved our scheme to obtain a 3 ratio; prior to this improvement, new results on UFP were independently obtained by Dinitz, Garg and Goemans [23].

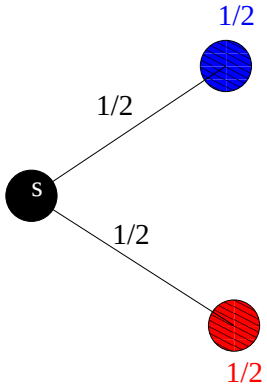
We give first a specialized version of Lemma 3.3.2 to be used in the new algorithm.

Lemma 3.4.1 *Let $\Pi = (G, s, \mathcal{T})$, be a* UFP *in which all demands have value $1/2^x$*

Fractional Solution:



Subproblem G^1 for demands of $1/2$:



Subproblem G^2 for demands of 1:

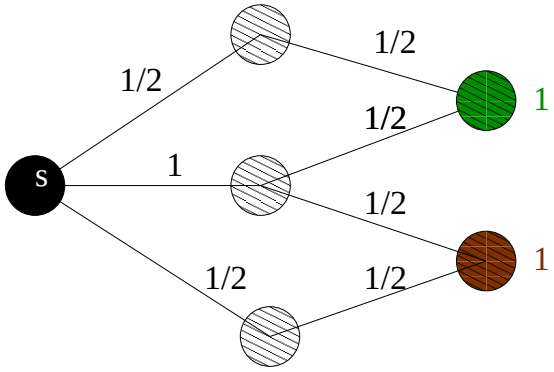


Figure 3.4: Algorithm PARTITION Demonstration I. Numbers on the edges and the vertices denote capacities and demands respectively. Original input capacities are 1.

Scale up capacities for G^1 to $2u_e^1 + (1/2)$. Unsplittable solution:

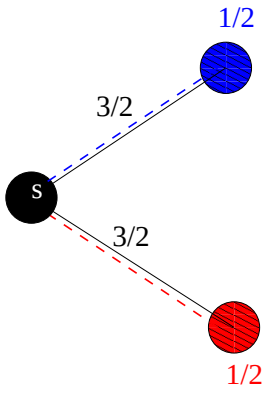


Figure 3.5: Algorithm PARTITION Demonstration II.

Scale up capacities for G^2 to $2u_e^2 + 1$. Unsplittable solution:

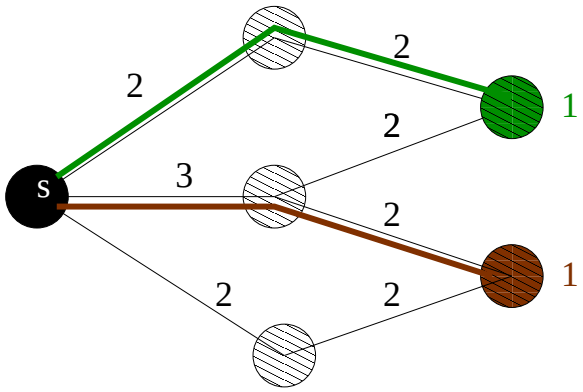


Figure 3.6: Algorithm PARTITION Demonstration III.

Superimpose the two solutions to obtain final solution:

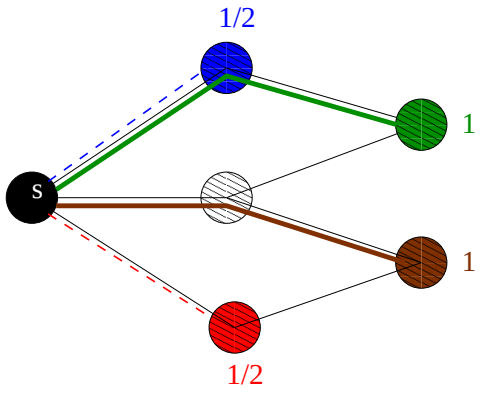


Figure 3.7: Algorithm PARTITION Demonstration IV. Final congestion is 3/2.

for $x \in N$, and all capacities are multiples of $1/2^{x+1}$, and let f be a fractional flow solution such that the flow f_e through any edge is a multiple of $1/2^{x+1}$. We can find in polynomial time an unsplittable routing g where the flow g_e through an edge e is at most $f_e + 1/2^{x+1}$.

Algorithm PARTITION finds a routing for each subproblem by scaling up subproblem capacities to ensure they conform to the requirements of Lemma 3.3.2. Call *granularity* the minimum amount of flow routed along a single path to any commodity, which has not been routed yet unsplittably. Algorithm PARTITION starts with a fractional solution of arbitrary granularity and essentially in one step per commodity, rounds this fractional solution to an unsplittable one. In the new algorithm, we delay the rounding process in the sense that we keep computing a fractional solution of successively increasing granularity. The granularity will always be a power of 1/2 and this unit will be doubled after each iteration. Once the granularity surpasses $1/2^x$, for some x, we guarantee that all commodities with demands $1/2^x$ or less have already been routed unsplittably. Every iteration improves the granularity at the expense of increasing the congestion. The method has the advantage that by Lemma 3.4.1, a

fractional solution of granularity $1/2^x$ requires only a $1/2^x$ additive offset to current capacities to find a new flow of granularity $1/2^{x-1}$. Therefore, if the algorithm starts with a fractional solution of granularity $1/2^\lambda$, for some potentially large λ, the total increase to an edge capacity from then on would be at most $\sum_{j=1}^{j=\lambda} 1/2^j < 1$. Expressing the granularity in powers of $1/2$ requires an initial rounding of the demand values; this rounding will force us to scale capacities by at most a factor of 2. We first provide an algorithm for the case, in which demand values are powers of $1/2$. The algorithm for the general case will then follow easily.

Theorem 3.4.1 *Let $\Pi = (G = (V, E, u), s, \mathcal{T})$ be a* UFP, *in which all demand values are of the form $2^{-\nu}$ with $\nu \in N$, the maximum demand value is denoted by $\rho_{\max}$, and the minimum demand value is denoted by $\rho_{\min}$. There is an algorithm,* 2H_PARTITION, *which obtains in polynomial time an unsplittable routing such that the flow through any edge e is at most $u_e + \rho_{\max} - \rho_{\min}$).*

Remark: We first obtained this theorem for the case with demand values in $\{1/2, 1\}$ in [66]. Dinitz, Garg and Goemans [23] first extended the theorem to arbitrary powers of $1/2$. Their derivation uses a completely different algorithm.

Proof. We describe the algorithm 2H_PARTITION. We introduce an assumption, which we will eliminate at the end of the proof.

Assumption 3.4.1 *There is an unsplittable flow for Π with congestion* 1.

Let $\rho_{\min} = 1/2^\lambda$. We show how to route the commodities in $\mathcal{T}$ with demands $1/2^\lambda, 1/2^{\lambda-1}, \ldots, \rho_{\max} \leq 1$ in $\lambda - \log(\rho_{\max}^{-1}) + 1$ iterations.

Claim 3.4.1 *Let Π' be the modified version of Π in which every edge capacity u_e has been rounded down to the closest multiple u'_e of $1/2^\lambda$. The optimum congestion is the same for Π' and Π.*

Proof of claim. Let G be the network with capacity function u and G' be the network with rounded capacities u'. The amount $u_e - u'_e$ of capacity will be unused by any optimal unsplittable flow in G. The reason is that the sum $\sum_{i \in S} 1/2^i$, for any finite multiset $S \subset N$ is equal to a multiple of $1/2^{i_0}$, $i_0 = \min_{i \in S}\{i\}$. ∎

From now on we will work with the rounded capacity function u'. During the first iteration, split every commodity in $\mathcal{T}$, called an *original commodity*, with demand $1/2^x$, $0 \leq x \leq \lambda$, into $2^{\lambda - x}$ *virtual commodities* each with demand $1/2^\lambda$. Call $\mathcal{T}_1$ the resulting set of commodities. By Theorem 3.2.1 we can find a fractional flow solution $f^0 \equiv g^1$, for $\mathcal{T}_1$, in which the flow through any edge is a multiple of $1/2^\lambda$. Observe that f^0 is a fractional solution for the set of original commodities as well, a solution with granularity $1/2^\lambda$. The flow u^1_e through an edge is trivially $f^0_e \leq u'_e \leq u_e$. At iteration $i > 1$, we extract from g_{i-1}, using flow decomposition, the set of flow paths f^{i-1} used to route the set S_{i-1} of virtual commodities in $\mathcal{T}_{i-1}$ which correspond to original commodities in $\mathcal{T}$ with demand $1/2^{\lambda - i + 1}$ or more. Pair up the virtual commodities in S_{i-1} corresponding to the same original commodity; this is possible since the demand of an original commodity is inductively an even multiple of the demands of the virtual commodities. Call the resulting set of virtual commodities $\mathcal{T}_i$. $\mathcal{T}_i$ contains virtual commodities with demand $1/2^{\lambda - i + 1}$. The set of flow paths f^{i-1} is a fractional solution for the commodity set $\mathcal{T}_i$. By Lemma 3.4.1, we can find an unsplittable routing g^i for $\mathcal{T}_i$ such that the flow g^i_e through an edge e is at most

$$f^{i-1}_e + 1/2^{\lambda - i + 2}.$$

Now the granularity has been doubled to $1/2^{\lambda - i + 1}$. A crucial observation is that from this quantity, g^i_e, only an amount of at most $1/2^{\lambda - i + 2}$ corresponds to new flow, added to e during iteration i. It is easy to see that the algorithm maintains the following two invariants.

INV1: After iteration i, all original commodities with demands $1/2^{\lambda-i+1}$ or less have been routed unsplittably.

INV2: After iteration $i > 1$, the total flow u_e^i though edge e, which is due to all i first iterations, satisfies the relation

$$u_e^i \leq u_e^{i-1} + 1/2^{\lambda-i+2}.$$

Given that u_e^1 is at most u_e the total flow though e used to route unsplittably all commodities in $\mathcal{T}$ is at most

$$u_e + \sum_{i=2}^{\lambda-\log(\rho_{\max}^{-1})+1} 1/2^{\lambda-i+2} \quad = u_e + \rho_{\max} - 1/2^{\lambda} \quad = u_e + \rho_{\max} - \rho_{\min}.$$

For the running time, we observe that in each of the $l = \lambda - \log(\rho_{\max}^{-1}) + 1$ iterations, the number of virtual commodities appears to be $O(k2^l)$, where k is the number of the original commodities. There are two approaches to ensure polynomiality of our method. One is to consider only demands of value $1/2^{\lambda} > 1/n^d$ for some $d > 1$. By Lemma 3.3.1 we can route the very small demands while affecting the approximation ratio by an $o(1)$ factor. The other approach relies on the observation that at iteration i, the set S_{i-1} of flow paths yielded by the flow decomposition has size $O(km)$ since it corresponds to a fractional solution for a subset of the original commodities. In other words, virtual commodities in S_{i-1} corresponding to the same original commodity have potentially been routed along the same path in g^i. Hence, by appropriately merging virtual commodities corresponding to the same original commodity, and increasing the granularity at different rates for distinct original commodities, we can always run an iteration on a polynomial-size set of virtual commodities. We opted to present a "pseudopolynomial" version of the algorithm for clarity.

It remains to eliminate Assumption 3.4.1. In case the optimum congestion is not 1, one can use the algorithm we propose as a $(1 + \rho_{\max} - \rho_{\min})$-relaxed decision

procedure in conjunction with a binary search for the optimum congestion to obtain the claimed approximation. See [45] for details on relaxed decision procedures. ■

Theorem 3.4.2 *Let* $\Pi = (G = (V, E, u), s, \mathcal{T})$ *be a* UFP *with maximum and minimum demand values* $\rho_{\max}$, *and* $\rho_{\min}$ *respectively. There is an algorithm,* H_PARTITION, *which obtains in polynomial time a*

$$\min\{3 - \rho_{\min}, 2 + 2\rho_{\max} - \rho_{\min}\}$$

approximation for congestion.

Proof. We describe the algorithm H_PARTITION. Round up every demand to its closest power of 1/2. Call the resulting UFP Π' with sink set $\mathcal{T}'$. Multiply all capacities by at most 2. Then there is a fractional feasible solution f' to Π'. Let u' be the capacity function in Π'. The purpose of the rounding is to be able to express later on the granularity as a power of 1/2. The remainder of the proof consists of finding an unsplittable solution to Π'; this solution can be efficiently converted to an unsplittable solution to Π without further capacity increase. Such a solution to Π' can be found by the algorithm 2H_PARTITION from Theorem 3.4.1. 2H_PARTITION will route through edge e flow that is at most

$$u'_e + \rho'_{\max} - \rho'_{\min} \leq 2u_e + \rho'_{\max} - \rho'_{\min},$$

where $\rho'_{\max}$ and $\rho'_{\min}$ denote the maximum and minimum demand values in Π'. But $\rho'_{\max} < 2\rho_{\max}$, $\rho'_{\max} \leq 1$, and $\rho'_{\min} \geq \rho_{\min}$ from the construction of Π'. ■

3.5 Minimizing congestion with arbitrary capacities

We examine here UFP in its full generality, when demands lie in $(0, 1]$ and capacities in $(0, \infty)$. In particular, let $1/D$ be the minimum demand value for a real $D \geq 1$. We assume without loss of generality that $u_e \geq 1/D$, for all e, since the optimal solution will not use any edge violating this condition. Our approximation algorithm will have time complexity polynomial in $\lfloor \log D \rfloor$. In subsequent sections we return to the original assumptions on the capacity values as stated in Section 3.2.

The essential modification required in algorithm PARTITION is at Step 1. We compute a feasible fractional solution such that the flow of commodity i through edge e is set to 0 if $\rho_i > u_e$. These constraints can be easily enforced, for example, in a linear programming formulation. The resulting problem formulation is a valid relaxation of the original problem and the unsplittable optimum is not affected by the additional constraints. A second modification is that no subproblem is constructed for a range of the form $(0, \alpha_1]$. The first subinterval of the decomposition is $[a_1, a_2]$ with $a_1 = 1/D$. The analysis of the performance guarantee is largely the same. Let L_PARTITION be the modified algorithm as outlined above. We maintain the notation of PARTITION for the decomposition parameters.

Lemma 3.5.1 *Algorithm* L_PARTITION *runs in polynomial time and outputs an unsplittable routing g with total flow through edge e at most* $\max_{2 \leq i \leq \nu}\{r(\alpha_{i-1}, \alpha_i)\}u_e + \sum_{i=2}^{\nu} \alpha_i$, *where* $u_e \in (\alpha_{\nu-1}, \alpha_\nu]$, $\nu \leq \xi$.

Proof. The proof is similar to that of Lemma 3.3.3. The additional observation is that because of the new constraints in the fractional relaxation, during Step 2 of the algorithm edge e is not included in any graph G_i with $i \geq \nu$. ■

Theorem 3.5.1 *Given* UFP Π *with arbitrary capacities and minimum demand value* $\rho_{\min} = 1/D$, $D \in R_{\leq 1}$, *Algorithm* L_PARTITION *finds a* $(5.8285 - \rho_{\min})$*-approximation for congestion. The algorithm runs in polynomial time.*

Proof. We instantiate the partitioning scheme as follows. The parameter $r > 1$, to be chosen later, is the ratio of the geometrically decreasing intervals. (In the proof of Theorem 3.3.1, $r = 2$.) We partition the interval $[1/D, 1]$ of demands into $O(\log_r D)$ geometrically increasing subintervals:

$$[1/D, 1/r^{\lfloor \log_r D \rfloor}], \ldots, (1/r^{i+1}, 1/r^i], \ldots, (1/r, 1].$$

By Lemma 3.5.1 the total flow through an edge e such that $1/r^{i+1} < u_e \leq 1/r^i$ is at most

$$ru_e + \sum_{j=i}^{\lfloor \log_r D \rfloor} 1/r^j = ru_e + \frac{r}{r^i(r-1)} - \frac{1}{r^{\lfloor \log_r D \rfloor}} \leq ru_e + \frac{r^2 u_e}{r-1} - \frac{1}{D}.$$

Thus we obtain a congestion of $\frac{2r^2-r}{r-1} - 1/D$. Selecting $r = 1.707$, yields $\frac{2r^2-r}{r-1} \leq 5.8285$.

■

3.6 Minimum-cost unsplittable flow

In this section we examine the problem of finding a minimum-cost unsplittable flow for $(G, s, \mathcal{T})$ when a cost $c_e \geq 0$ is associated with every edge e. The cost $c(P)$ of a path P is $\sum_{e \in P} c_e$. Our objective is to find an unsplittable flow solution which minimizes the total cost $\sum_{t_i \in \mathcal{T}} \rho_i c(P_i)$. Since UFP without costs is already NP-hard, it is unavoidable to introduce congestion in our solution. We show how to modify algorithm PARTITION to invoke a minimum-cost flow subroutine instead of maximum-flow routines. The new algorithm will simultaneously achieve a constant factor approximation for both

cost and congestion. Since PARTITION works on both directed and undirected graphs the same holds for the new algorithm. We need the analogue of Theorem 3.2.1.

Theorem 3.6.1 *Let $(G = (V, E, u), s, \mathcal{T})$ be a* UFP *with costs on an arbitrary network G with all demands ρ_i, $1 \leq i \leq k$, equal to the same value σ and edge capacities u_e equal to integral multiples of σ for all $e \in E$. There is a maximum fractional flow of minimum cost that is an unsplittable flow. Moreover, this unsplittable flow can be found in polynomial time.*

This theorem is an easy consequence of the well known successive shortest path algorithm for minimum-cost flow (developed independently by [47], [46], [11]); it is a corollary of the integrality property of minimum-cost flow, with integral units scaled by σ.

Two lemmata for the analysis follow. They generalize Lemmata 3.3.1 and 3.3.2 to accommodate costs.

Lemma 3.6.1 *Given a minimum-cost* UFP *with demands in the interval $(0, \alpha]$ and a fractional solution f with cost $c(f)$, there is an algorithm, α-*C_ROUTING*, which finds an unsplittable routing of cost at most $n\alpha$, such that the flow through an edge is at most $c(f)$. The running time of the algorithm is $O(n \log n + m)$.*

Proof. Algorithm α-C_ROUTING works as follows. Find shortest paths from s to all the sinks in G using Dijkstra's algorithm [21, 32]. For each sink t_i route on the shortest path from s flow equal to the demand ρ_i. ■

Lemma 3.6.2 *Let a minimum-cost* UFP Π *have demands in the interval $(a, b]$, arbitrary capacities, and a fractional solution f of cost $c(f)$. There is an algorithm,* C_INTERVAL_ROUTING, *which finds in polynomial time an unsplittable routing of cost at most $r(a, b)c(f)$ such that the flow through an edge e is at most $r(a, b)u_e + b$.*

Proof. The proof is similar to that of Lemma 3.3.2, but we use Theorem 3.6.1 instead of Theorem 3.2.1. Round up all the demands to b and call Π' the resulting problem. By multiplying the flow f_e and the capacity u_e on every edge e by at most $r(a,b)$ we obtain a feasible fractional solution f' for Π'; the cost of f' is at most $r(a,b)c(f)$. Now add at most b to every edge capacity so that it becomes a multiple of b. Find an unsplittable routing by using Theorem 3.6.1. The cost of the unsplittable routing is at most equal to the cost of the fractional solution f'. ■

Algorithm C_Partition takes the same steps as Partition, with two differences in Steps 1 and 3. At Step 1 of C_Partition a fractional minimum-cost solution f is found. At Step 3 we invoke C_Interval_Routing instead of Interval_Routing. C_Interval_Routing is implemented with the successive shortest paths algorithm for minimum-cost flow. Finally algorithm α-C_Routing is used to compute a routing for graph G_1 of the decomposition. We keep the same notation for the partitioning scheme. The proof is very similar to that of Lemma 3.3.3 and we omit it.

Lemma 3.6.3 *Algorithm* C_Partition *outputs an unsplittable routing g with congestion at most*

$$n\alpha_1 + \max_{2\le i\le \xi}\{r(\alpha_{i-1},\alpha_i)\} + \sum_{i=2}^{\xi}\alpha_i$$

and cost at most $\max_{2\le i\le\xi}\{r(\alpha_{i-1},\alpha_i)\}c(f)$*, where $c(f)$ is the minimum cost of a fractional solution. The running time of* C_Partition *is $O(T_2(n,m)+nm+m\xi)$ where $T_2(n,m)$ is the time to solve a fractional minimum-cost flow problem.*

Using the same partitioning scheme as in Theorem 3.3.1 we obtain:

Theorem 3.6.2 *Given a minimum-cost* Ufp Π*, we can obtain a simultaneous $(2, 4+\varepsilon)$ approximation for cost and congestion, for any $\varepsilon > 0$. The running time of the*

algorithm is $O(T_2(n,m) + nm + m\log(n/\varepsilon))$ *where* $T_2(n,m)$ *is the time to solve a fractional minimum-cost flow problem.*

We observe that the approximation factor for the cost in the above algorithm is the ratio of the subintervals used in the partitioning scheme. By increasing the constant for congestion we can improve upon the cost approximation.

Theorem 3.6.3 *Given a minimum-cost* UFP Π *we can obtain a simultaneous* $(1+\delta, 2+\frac{\delta^2+1}{\delta}+\varepsilon)$ *approximation for cost and relative congestion for any* $\delta, \varepsilon > 0$. *The running time of the algorithm is* $O(T_2(n,m)+m\log(n/\varepsilon))$ *where* $T_2(n,m)$ *is the time to solve a fractional minimum-cost flow problem.*

Proof. The new algorithm is the same as C_PARTITION except for the partitioning scheme. The interval $(0,1]$ of demands is partitioned into ξ geometrically increasing subintervals

$$(0, 1/(\delta+1)^{\xi-1}], \ldots, (1/(\delta+1)^{i+1}, 1/(\delta+1)^i], \ldots, (1/(\delta+1)^2, 1/(\delta+1)], (1/(\delta+1), 1]$$

such that $1/(\delta+1)^{\xi-1} \leq \varepsilon/n$. According to this scheme, by Lemma 3.6.3 the approximation ratio for the cost will be $1+\delta$ and the approximation ratio for the congestion will be at most

$$\begin{aligned} 1+\delta+\sum_{i=0}\frac{1}{(\delta+1)^i}+\varepsilon &\leq \\ 1+\delta+\frac{1+\delta}{\delta}+\varepsilon &= \\ 2+\frac{\delta^2+1}{\delta}+\varepsilon. & \end{aligned}$$

■

Currently the best time bound $T_2(n,m)$ for fractional minimum-cost flow is

$$O(\min\{nm\log(n^2/m)\log(nC), nm(\log\log U)\log(nC), (m\log n)(m+n\log n)\})$$

due to [40, 2, 80]. Finally, by modifying the congestion algorithm of Theorem 3.4.2, it easy to see how to obtain a $(2, 3)$ simultaneous approximation for cost and congestion. The essential modification to H_PARTITION lies in the use of the minimum-cost analogue to Lemma 3.4.1.

Lemma 3.6.4 *Let $\Pi = (G, s, \mathcal{T})$, be a minimum-cost* UFP, *in which all demands have value $1/2^x$ for $x \in N$, and all capacities are multiples of $1/2^{x+1}$, and let f be a fractional flow solution of cost $c(f)$ such that the flow f_e through any edge is a multiple of $1/2^{x+1}$. We can find in polynomial time an unsplittable routing g of cost at most $c(f)$ such that the flow g_e through an edge e is at most $f_e + 1/2^{x+1}$.*

Theorem 3.6.4 *Let Π be a minimum-cost* UFP, *with maximum and minimum demand values $\rho_{\max}$, and $\rho_{\min}$ respectively. We can obtain in polynomial time a simultaneous*

$$(2, \min\{3 - \rho_{\min}, 2 + 2\rho_{\max} - \rho_{\min}\})$$

approximation for cost and congestion.

3.7 Maximizing the routable demand

In this section we turn to the maximization problem of finding a routable subset of sinks with maximum total demand. Let the *value* of a (partial) routing be the sum of the demands of the commodities being routed. We seek a routing of maximum value such that the capacity constraints are satisfied. Note that if the value definition was relaxed to take into account paths carrying flow to commodities without fully satisfying the respective demands, a c-approximation, $c > 1$, for congestion would imply a $1/c$-approximation for this relaxed maximization problem.

Let $(G, s, \mathcal{T})$ be the UFP we want to solve. Consider splitting the interval $(0, 1]$ of demands into two intervals $(0, \alpha]$, and $(\alpha, 1]$, for an appropriate *breakpoint* α. We will give separate constant-factor approximation algorithms for each of the two resulting problems. At least one of the two intervals contains at least half of the total demand in $\mathcal{T}$, so we obtain a constant-factor approximation for the entire problem. This was also the high-level approach used in [61]. In fact, in order to route the demands in the subinterval $(\alpha, 1]$ we will use a simple and intuitive algorithm developed by Kleinberg [61]. For the demands in $(0, \alpha]$, we show how a partitioning scheme, similar to the one used for minimizing congestion, can give a simple algorithm with a constant performance guarantee. For our partitioning scheme to succeed in the maximization setting, it is essential to have the maximum demand bounded away from 1. Hence the different treatment for $(0, \alpha]$, and $(\alpha, 1]$.

Let $\alpha^f(G, s, \mathcal{T})$ denote the maximum total demand routed by a fractional solution f. We will use $\alpha^f(G, s, \mathcal{T})$, with f a fractional solution of maximum value, as an upper bound to measure the quality of our approximation.

The following lemma, shown in [61], will be used to attack the demands in $(\alpha, 1]$. The spirit of the proof is similar to the one of Lemma 3.3.2.

Lemma 3.7.1 *[61] Given an* UFP $\Pi = (G, s, \mathcal{T})$, *with demands in the interval* $(a, b]$, *there is an algorithm,* K_ROUTING, *which finds a partial unsplittable flow* g *of value at least* $\frac{1}{2}[r(a, b)]^{-1}\alpha^f(G, s, \mathcal{T})$. *The algorithm runs in polynomial time.*

Adapting the proof of Lemma 3.7.1 we can show a slightly different one, which will be also of use in our algorithm. We include the proof for the sake of completeness.

Lemma 3.7.2 *Given an* UFP $\Pi = (G, s, \mathcal{T})$, *with demands in the interval* $(a, b]$, *and capacities multiples of* b, *there is an algorithm,* M_ROUTING, *which finds, in polynomial time, a partial unsplittable flow* g *of value at least* $[r(a, b)]^{-1}\alpha^f(G, s, \mathcal{T})$.

Proof. We outline how algorithm M_ROUTING works. Let f be a fractional solution to Π. Round down all demands to a. Multiply all capacities in G by $[r(a,b)]^{-1} < 1$. Let $\Pi' = (G', s, \mathcal{T}')$, be the resulting problem. We show that Π' has a fractional solution f' of value $\alpha^{f'}(G', s, \mathcal{T}') \geq [r(a,b)]^{-1}\alpha^{f}(G, s, \mathcal{T})$: multiply the flow pushed by f on every edge by $[r(a,b)]^{-1}$ and let f' be the resulting fractional flow. Clearly f' does not route more than a units of flow to any sink in $\mathcal{T}'$ and respects the capacities in G', so it is feasible for Π'.

We now show how to obtain a partial unsplittable solution for Π of value $\alpha^{f'}(G', s, \mathcal{T}')$. In G all capacities are multiples of b so in G' they are multiples of a. But all demands in G' are a so by Theorem 3.2.1 there is an unsplittable flow g' of value at least $\alpha^{f'}(G', s, \mathcal{T}')$. Multiplying the flow on every path in g' by a factor of at most $r(a,b)$ gives a partial unsplittable flow g for Π, respecting the capacities in G and of value at least $\alpha^{f'}(G', s, \mathcal{T}')$. ■

We now outline our approach for the second half of the problem, namely the demands in $(0, \alpha]$. Our aim is to use a partitioning scheme as in the congestion case: partition the interval $(0, \alpha]$ of demand values into subintervals and generate an appropriate subproblem for each subinterval. We would like to find a near-optimal solution to each subproblem, by exploiting Lemma 3.7.2. Ideally, we could then combine these near-optimal solutions to the subproblems to obtain a close-to-maximum unsplittable flow for the original problem. However, in the congestion version, only a fraction of the capacity of an edge is assigned initially to each subproblem, a fraction determined by flow decomposition. On the other hand, in order to make use of Lemma 3.7.2 on each subproblem, we require capacities to be multiples of the maximum demand. In the congestion algorithm, adding the necessary amount to each capacity only increased the approximation ratio. In the current setting we are not allowed to

increase the original input capacities, as this would violate feasibility. To circumvent this difficulty we scale down the fractional solution to the original problem. On each edge a constant fraction of the edge capacity is then left unused and can provide the required extra capacity for the subproblems. The scaling of the fractional solution by a $1 - x$ factor, for appropriately chosen x, will incur a scaling by the same factor to the approximation ratio, as the value of the final routing is given in terms of the value $\alpha^f(G, s, \mathcal{T})$ of the maximum flow with which we begin. We give the general algorithm in Figure 3.8. It takes as input a UFP $(G, s, \mathcal{T})$ together with a set of parameters $x, \xi, \alpha_1, \ldots, \alpha_\xi$ that will determine the exact partitioning scheme. The breakpoint α will be $\alpha_{\xi-1}$. Subsequently we show how to choose these parameters so as to optimize the approximation ratio. The algorithm outputs a partial routing.

By Lemma 3.7.1, partial routing g^ξ computed in Step 4 is a partial unsplittable flow, which respects the capacities. We need to establish this fact for partial routing g computed in Step 5 as well, and also get an estimate of its value. Note that algorithm M_PARTITION does not route any demands in $(0, \alpha_1]$. Let $a^f(G, s, \mathcal{T}_i)$ denote the flow routed by the fractional solution to demands in the interval $(\alpha_{i-1}, \alpha_i]$.

Lemma 3.7.3 *Partial routing g found by algorithm* M_PARTITION *is a partial unsplittable flow of value at least*

$$\min_{2 \le i \le \xi - 1} \{[r(\alpha_{i-1}, \alpha_i)]^{-1}\}(1 - x)\alpha^f(G, s, \bigcup_{2 \le i \le \xi - 1} \mathcal{T}_i).$$

Moreover g respects the capacity constraints in G. The running time of M_PARTITION *is $O(T_1(n, m) + nm + m\xi)$ where $T_1(n, m)$ is the time to solve a fractional maximum flow problem.*

Proof. We examine first the value of g. Let f_i^x be the fractional optimal flow on G_i. By Lemma 3.7.2 the value of the partial routing g_i, $2 \le i \le \xi - 1$, found at

ALGORITHM M_PARTITION$(G = (V, E, c), s, \mathcal{T}, \xi, \alpha_1, \ldots, \alpha_\xi, x)$

Step 1. Find a feasible fractional solution f.

Step 2. Let x be a constant, $0 < x < 1$. Define a partition of the $(0, 1]$ interval into ξ consecutive subintervals $(0, \alpha_1], (\alpha_1, \alpha_2], \ldots, (\alpha_{\xi-1}, \alpha_\xi]$, $\alpha_\xi = 1$, such that $\sum_{i=1}^{\xi-1} \alpha_i \leq x$. Construct $\xi - 1$ copies of G where the set of sinks in G_i, $2 \leq i \leq \xi$, is the subset $\mathcal{T}_i$ of $\mathcal{T}$ with demands in the interval $(\alpha_{i-1}, \alpha_i]$. Using flow decomposition determine for every edge e the amount c_e^i of u_e used by f to route flow to sinks in $\mathcal{T}_i$.

Step 3. Set the capacity of edge e in G_i, $2 \leq i \leq \xi - 1$, equal to the smallest multiple of α_i greater than or equal to $(1 - x)c_e^i$. Set the capacity of edge e in G_ξ to u_e.

Step 4. Invoke M_ROUTING on each G_i, $2 \leq i \leq \xi - 1$, to obtain a partial unsplittable flow g^i. Invoke K_ROUTING on G_ξ to obtain a partial unsplittable flow g^ξ.

Step 5. Set partial routing g to be the union of the path sets g^i, $2 \leq i \leq \xi - 1$. Of the two partial routings g and g^ξ output the one of greater value.

Figure 3.8: Algorithm M_PARTITION.

Step 5, is at least $[r(\alpha_{i-1}, \alpha_i)]^{-1} f_i^x$. By the capacity assignment in G_i, $\alpha^{f_i^x}(G, s, \mathcal{T}_i) \geq (1-x)\alpha^f(G, s, \mathcal{T}_i)$. Hence the value of the partial routing g^i is at least

$$(1-x)[r(\alpha_{i-1}, \alpha_i)]^{-1}\alpha^f(G, s, \mathcal{T}_i).$$

Since $\alpha^f(G, s, \bigcup_{2 \leq i \leq \xi-1} \mathcal{T}_i) = \sum_{i=2}^{i=\xi-1} \alpha^f(G, s, \mathcal{T}_i)$ the claim on the value follows.

For the capacity constraints, the aggregate capacity used in all partial routings g^i, $2 \leq i \leq \xi - 1$, on any edge e, is by Step 4 at most $(1-x)\sum_{i=2}^{i=\xi-1} c_e^i + \sum_{i=2}^{i=\xi-1} \alpha_i$. By Step 2, $\sum_{i=2}^{\xi-1} \alpha_i \leq x$ and $\sum_{i=2}^{i=\xi-1} c_e^i \leq u_e$. Thus, the aggregate capacity used by g is at most $(1-x)u_e + x \leq u_e$. The running time follows in the same manner as in Lemma 3.3.3. ■

It remains to choose the parameters x, ξ and the α_i of the partitioning, and to account for the "missing" flow $a^f(G, s, \mathcal{T}_1)$.

Theorem 3.7.1 *Let x_1, x_2, x_3 be constants in $(0,1)$ such that $x_3 = x_1(1-x_2)$. Given an* UFP *$\Pi = (G, s, \mathcal{T})$, algorithm* M_PARTITION *finds a $\beta(1-\varepsilon)$-approximation for maximum routable demand, for any $0 < \varepsilon < \beta$ and $\beta = \min\{(1-x_1)x_2/2, x_3/4\}$. The running time of the algorithm is $O(T_1(n,m) + nm + m\log(n/\varepsilon))$ where $T_1(n,m)$ is the time to solve a fractional maximum flow problem.*

Proof. Without loss of generality we assume that there is one demand in $\mathcal{T}$ of value 1. Otherwise we can rescale the interval of demands. Thus $a^f(G, s, \mathcal{T})$ is at least 1. At step 3 of M_PARTITION, partition the interval $(0, 1]$ of demand values into ξ geometrically increasing subintervals

$$(0, x_3(x_2)^{\xi-2}], \ldots, (x_3(x_2)^{i+1}, x_3(x_2)^i], \ldots, (x_3x_2, x_3], (x_3, 1]$$

such that $x_3(x_2)^{\xi-2} \leq \varepsilon/n$. Thus it suffices for ξ to be $\Theta(\log n/\varepsilon)$. The flow $a^f(G, s, \mathcal{T}_1)$

is at most ε so

$$a^f(G, s, \bigcup_{2 \le i \le \xi} \mathcal{T}_i) \ge a^f(G, s, \mathcal{T}) - \varepsilon \ge a^f(G, s, \mathcal{T}) \ge (1 - \varepsilon) a^f(G, s, \mathcal{T}).$$

Achieving therefore a β-approximation to $a^f(G, s, \bigcup_{2 \le i \le \xi} \mathcal{T}_i)$ will yield the claimed $\beta(1 - \varepsilon)$ ratio.

Set parameter x in the algorithm to x_1. Note that by the choice of x_3 the sum $\sum_{i=0}^{i=\xi-2} x_3(x_2)^i$ is at most x_1 as required by the algorithm. By Lemma 3.7.3, the partial unsplittable flow g found by M_PARTITION is a $(1 - x_1)x_2$-approximation to $a^f(G, s, \bigcup_{2 \le i \le \xi-1} \mathcal{T}_i)$. By Lemma 3.7.1, the partial unsplittable flow g^ξ is a $x_3/2$-approximation to $a^f(G, s, \mathcal{T}_\xi)$. Choosing of the two the one of greatest value at Step 5 yields a β-approximation to $a^f(G, s, \bigcup_{2 \le i \le \xi} \mathcal{T}_i)$. ■

By choosing x_1, x_2, x_3 to be $4/10, 1/4$ and $3/10$ respectively we obtain $(1-x_1)x_2 = x_3/2 = 0.15$. Accordingly, $\beta = .075$.

3.8 Minimizing the number of rounds

In this section we examine the problem of routing in rounds. A partitioning of the set of sinks into a minimum number of communication rounds is sought so that the set of terminals assigned to each round is routable under the capacity constraints. The requirement to satisfy the capacity constraints leads to a high-level treatment similar to the one for the maximum demand metric. We split the interval of demand values into $(0, \alpha]$ and $(\alpha, 1]$, for an appropriate breakpoint α, and give a constant-factor approximation for each of the two resulting problems. Demands in $(0, \alpha]$ will be processed under a partitioning scheme, that will generate subproblems. Scaling of the fractional solution by $1 - x$ will be employed to free the necessary capacity to fulfill the requirements of the subproblems. This scaling, will be reflected in an

increase on the approximation ratio by $1/(1-x)$. Intuitively, since we are using paths from the fractional solution in the unsplittable routings, when the fractional solution carries $(1-x)$ times less flow for each commodity, we need more rounds.

Recall that the input UFP comes with the assumption that a fractional solution with relative congestion 1 exists. In this section we will generate subproblems on which this condition does not hold. Therefore we introduce notation $\zeta^f(G, s, \mathcal{T})$ for the congestion of a *fractional* solution f to an UFP $(G, s, \mathcal{T})$. Note that if f is a fractional solution of minimum congestion for a problem $(G, s, \mathcal{T})$, the quantity $\lceil \zeta^f(G, s, \mathcal{T}) \rceil$ is a lower bound on the minimum number of rounds, which we denote by $\chi(G, s, \mathcal{T})$. We employ a subroutine that we call KR_ROUTING (see Lemma 5.3 in [61]) and a variant R_ROUTING to deal with subproblems having demands in a bounded range. The subroutines are similar in spirit to the subroutines K_ROUTING and M_ROUTING used for the maximum demand metric; however their basic ingredient is not Theorem 3.2.1. Maximum flow integrality is not useful in the rounds setting and instead a result from [61] is used, given as Theorem 3.8.1 below. The proof of the latter theorem uses results from matroid theory.

Theorem 3.8.1 *[61] Given an* UFP $(G, s, \mathcal{T})$ *with all demands equal to* σ *and all capacities multiples of* σ, $\lceil \zeta^f(G, s, \mathcal{T}) \rceil = \chi(G, s, \mathcal{T})$. *Moreover, a routing in* $\chi(G, s, \mathcal{T})$ *rounds can be found in polynomial time.*

Lemma 3.8.1 *[61] Let* $\Pi = (G, s, \mathcal{T})$, *be a* UFP *with demands in the interval* $(a, b]$, *and* f *a corresponding fractional solution. There is a polynomial-time algorithm,* KR_ROUTING, *which routes* Π *in at most* $\lceil 2r(a, b)\zeta^f(G, s, \mathcal{T}) \rceil$ *rounds.*

Again we can adapt the proof of Lemma 3.8.1 to show a slightly different result.

Lemma 3.8.2 *Let* $\Pi = (G, s, \mathcal{T})$ *be a* UFP *with demands in the interval* $(a, b]$, *and capacities multiples of* b, *and* f *a corresponding fractional solution. There is a polynomial-time algorithm,* R_ROUTING, *which routes* Π *in at most* $\lceil r(a,b)\zeta^f(G, s, \mathcal{T})\rceil$ *rounds.*

Proof. We outline algorithm R_ROUTING. Round all the demands up to b. Call Π' the resulting problem and f' a corresponding fractional solution. In Π' all demands are equal to b and all capacities are multiples of b; therefore by Theorem 3.8.1 we can route in $\chi(\Pi') = \lceil \zeta^{f'}(\Pi')\rceil$ rounds and use the paths from this routing to route Π. Clearly, the paths used in each round respect the capacity constraints in $(G, s, \mathcal{T})$. This completes the description of the algorithm.

It remains to demonstrate a suitable f', which will help us to upper bound $\zeta^{f'}(\Pi')$ in terms of $\zeta^f(G, s, \mathcal{T})$. In the fractional solution f for Π, consider the paths routing demand to a particular sink t_i. Increase the flow uniformly on each path until b units of flow are routed to t_i. Do this for all sinks. Let f' be the resulting fractional solution. f' satisfies all the demands for Π' and has congestion $\zeta^{f'}(\Pi')$ at most $r(a,b)\zeta^f(G, s, \mathcal{T})$.

■

Our algorithm R_PARTITION and its analysis are very similar to M_PARTITION up to the subroutine level. When dealing with $(\alpha_1, \alpha_{\xi-1}]$, the fractional solution is scaled down to free capacity on the edges. Steps 1 through 3 are exactly the same for both algorithms. In Step 4 routines R_ROUTING and KR_ROUTING are invoked on G_i, $2 \le i \le \xi - 1$, and G_ξ respectively. Algorithm R_ROUTING outputs for each G_i, $2 \le i \le \xi - 1$, a set of paths $\mathcal{P}_{ij}$ to be used on round j. Let j^* be the maximum number of rounds output from R_ROUTING for any G_i, $2 \le i \le \xi - 1$. During round j, $1 \le j \le j^*$, we route on all paths in $\bigcup_i \mathcal{P}_{ij}$. Subsequently we route the demands in $(0, \alpha_1]$ and $(\alpha_{\xi-1}, 1]$ in two separate sets of rounds. Recall that $\zeta^f(G, s, \mathcal{T}_i)$ is the

minimum congestion for routing demands fractionally in $\mathcal{T}_i$ on the original unscaled capacities and thus is equal to 1. The following lemma accounts for the rounds needed to route all demands except for the ones in the $(0, \alpha_1]$.

Lemma 3.8.3 *Algorithm* R_PARTITION *runs in polynomial time and routes the demands in* $(\alpha_1, 1]$ *in at most*

$$\max_{2 \leq i \leq \xi - 1} \{\lceil \frac{1}{1-x} r(\alpha_{i-1}, \alpha_i) \zeta^f(G, s, \mathcal{T}_i) \rceil\} + \lceil 2r(a_{\xi-1}, 1) \zeta^f(G, s, \mathcal{T}_\xi) \rceil$$

rounds.

Proof. For the number of rounds to route demands in $(\alpha_i, \alpha_{i+1}]$, $1 \leq i \leq \xi - 2$, we note that Lemma 3.8.2 applies in the corresponding subproblems. A fractional solution satisfying all demands in subproblem G_i would have to potentially introduce congestion given that at Step 3, we assign capacity to edge e in G_i which is as low as $(1-x)c_e^i$. There is a fractional solution f_i to G_i with congestion at most $\frac{1}{1-x}\zeta^f(G, s, \mathcal{T}_i)$, obtained by setting $f_e^i = c_e^i$. Therefore, the number of rounds for a subproblem is at most $\lceil \frac{1}{1-x} r(\alpha_{i-1}, \alpha_i) \zeta^f(G, s, \mathcal{T}_i) \rceil$. By exactly the same argument as in Lemma 3.7.3, the aggregate capacity on any edge e used on all these subproblems does not exceed u_e. Thus during the same round j, $1 \leq j \leq j^*$, we can route all paths in $\bigcup_{2 \leq i \leq \xi - 1} \mathcal{P}_{ij}$.

To route the demands in $(\alpha_{\xi-1}, 1]$ we need by Lemma 3.8.1 at most an additional $\lceil 2r(a_{\xi-1}, 1) \zeta^f(G, s, \mathcal{T}_\xi) \rceil$ number of rounds. ■

By choosing a partitioning scheme as in Theorem 3.7.1, the total demand in $(0, \alpha_1]$ is small enough to be routed in one round.

Theorem 3.8.2 *Let* x_1, x_2, x_3 *be constants in* $(0, 1)$ *so that* $x_3 = x_1(1-x_2)$. *Given an* UFP $\Pi = (G, s, \mathcal{T})$, *we can obtain in polynomial time a* β*-approximation for minimum number of rounds where* $\beta = \lceil 1/(1-x_1)x_2 \rceil + \lceil 2/x_3 \rceil + 1$.

Proof. The partitioning scheme is the same as in the proof of Theorem 3.7.1. The demands in $(0, x_3(x_2)^{\xi-2}]$ are small enough to be routed in one round using the algorithm α-ROUTING from Lemma 3.3.1. Substituting x_1, x_2, x_3 in the number of rounds given by Lemma 3.8.3 and adding one extra round for the demands in $(0, x_3(x_2)^{\xi-2}]$ completes the proof. ■

By choosing x_1, x_2, x_3 to be $1/2, 1/2$ and $1/4$ respectively we obtain $\beta = 13$.

3.9 A hardness result for unsplittable flow

In this section we consider the hardness of approximation for minimum congestion unsplittable flow on a directed network with 2 sources (2-UFP). We give a gap-preserving reduction from the NP-complete problem 3-D MATCHING [52]. Our reduction establishes that it is NP-hard to achieve an approximation ratio better than 2 for 2-UFP. In the 3-D MATCHING problem, we are given a set $M \subseteq A \times B \times C$, where A, B, and C are disjoint sets each of cardinality n. The objective is to find a *perfect matching,* i.e. a subset $M' \subseteq M$ such that $|M'| = n$ and no two elements of M' agree in any coordinate. In our reduction we use ideas from Theorem 5 in [74].

Lemma 3.9.1 *Given an instance I of* 3-D MATCHING, *we can obtain in polynomial time an instance I' of* 2-UFP *on a directed network such that:*

$$I \in \text{3-D MATCHING} \Rightarrow OPT(I') = 1,$$

$$I \notin \text{3-D MATCHING} \Rightarrow OPT(I') \geq 2.$$

Proof. Let the m triples in I be of the form (a_i, b_j, c_k) and n the size of a perfect 3-D matching. We show how to construct a directed network G for the corresponding instance I' of unsplittable flow. G contains two source vertices s_d and s_c. For each a_i

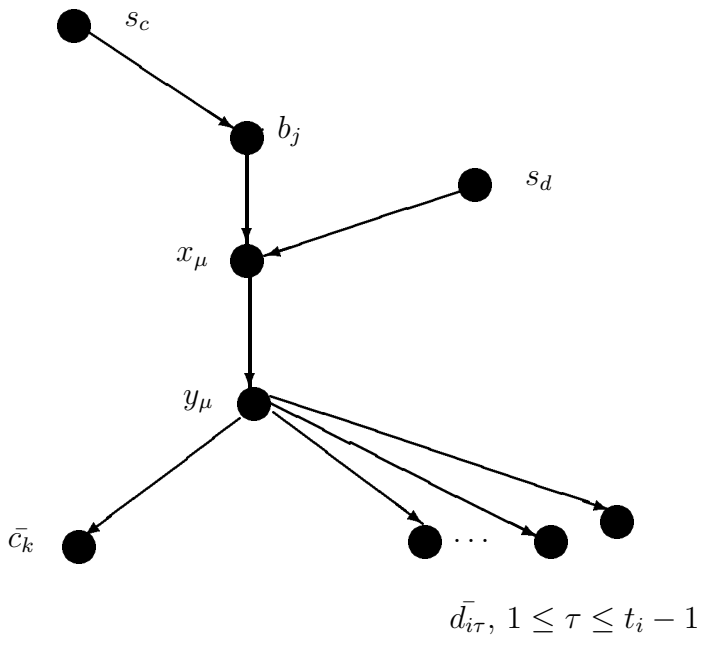


Figure 3.9: Gadget corresponding to the μ-th triple (a_i, b_j, c_k) together with the edges connecting it to the two sources.

occurring in t_i triples in I, G contains $t_i - 1$ vertices called the *dummies*. We denote as $\bar{d_{i\tau}}$, $1 \leq \tau \leq t_i - 1$, the dummy vertices corresponding to a_i. For each element b_j and c_k in I, we have respectively a vertex b_j and $\bar{c_k}$ in I'. We refer collectively to the $2n$ vertices b_j, $\bar{c_k}$, $1 \leq j, k \leq n$, as *b-* and *c-type vertices* respectively. Commodities corresponding to terminal pairs $(s_c, \bar{c_k})$, $1 \leq k \leq n$, and $(s_d, \bar{d_{i\tau}})$, $1 \leq i \leq n, 1 \leq \tau \leq t_i - 1$, have demand 1 each. All the edges in G have capacity 1.

For each of the m triples in I there will be a *triple gadget* (see Fig. 3.9) in G. Let the μ-th triple be (a_i, b_j, c_k). The gadget for μ contains two dedicated vertices: x_μ and y_μ. The edges of the gadget are (x_μ, y_μ), (b_j, x_μ), $(y_\mu, \bar{c_k})$ and an edge from y_μ to each $\bar{d_{i\tau}}$, $1 \leq \tau \leq t_i - 1$. This completes the description of a gadget. Finally, an edge is directed from s_d to each x_μ, $1 \leq \mu \leq m$, and from s_c to each b_j, $1 \leq j \leq n$.

We say that *vertex v covers* vertex w if the unique flow path satisfying the demand of w goes through v. Similarly we say that a *path covers w* or if it is clear from context to which gadget a path belongs that a *gadget μ covers w*. Note that every unit of flow towards a c-type or a dummy vertex must go through some gadget μ_i and in particular through the edge (x_{μ_i}, y_{μ_i}) of the gadget. The ensuing facts follow by the construction of G.

Fact 3.9.1 *If capacities are to be respected, at most one commodity can be routed unsplittably through the μ-th gadget corresponding to triple (a_i, b_j, c_k). The routed commodity will be either $(s_c, \bar{c_k})$ or $(s_d, \bar{d_{i\tau}})$, for some $1 \leq \tau \leq t_i - 1$.*

Fact 3.9.2 *In an unsplittable flow h, the flow routed to sink $\bar{c_k}$ through the gadget μ_k corresponding to triple (a_{i_k}, b_{j_k}, c_k), must saturate the edge (b_{j_k}, x_{μ_k}).*

Fact 3.9.3 *If capacities are to be respected, at most one unit of flow can be routed from source s_c through vertex b_j, for any j.*

If I contains a 3-D perfect matching M', it is straightforward to construct an unsplittable flow g in G with maximum congestion 1. Flow g uses the n gadgets corresponding to triples in M' to satisfy the demands of vertices $\bar{c_k}$, $1 \leq k \leq n$. The remaining gadgets are used to route 1 unit of flow each (via vertex y_μ for the μ-th gadget) to a dummy vertex. Since M' is a matching, only one of the t_i gadgets associated with a_i, for any i, is used to route flow to a c-type vertex. As a result, for any i, there are $t_i - 1$ gadgets available to route the dummy vertices associated with a_i.

We show now that if I' has an unsplittable flow g', which satisfies the capacity constraints, then I contains a 3-D perfect matching. Let Γ be the set of gadgets used

in g' to route flow to c-type vertices. By Fact 3.9.1, $|\Gamma|$ is equal to n. Since the size of Γ is n and g' routes all commodities, the triples corresponding to the gadgets in Γ do not agree in any c coordinate. By Fact 3.9.2, in each gadget in Γ, flow is routed along the edge of the gadget going out of a b-type vertex. Hence, by Fact 3.9.3, the triples corresponding to the gadgets in Γ do not agree in any b coordinate either. Again by Fact 3.9.1, no gadget in Γ is used to route positive flow to a dummy vertex. However flow g' satisfies also the demands of all the dummy vertices: by Fact 3.9.1 for each i there must be $t_i - 1$ distinct gadgets, none of them in Γ as we just showed, used to route flow to the dummies associated with a_i. Therefore, at most one gadget in Γ corresponds to a triple containing a_i for any given i. But Γ contains exactly n gadgets, therefore exactly one gadget in Γ corresponds to a triple containing a_i for any given i. Since we have already established that the gadgets in Γ do not agree in any b or c coordinate, the triples corresponding to Γ must form a perfect 3-D matching in I. ■

The ensuing theorem is an immediate consequence of Lemma 3.9.1.

Theorem 3.9.1 *No ρ-approximation algorithm, $\rho < 2$, exists for* 2-UFP *on directed graphs unless $P = NP$. The result holds even when all capacities and demands are equal to* 1.

3.10 Restricted sets of demands and applications to scheduling

In this section we examine connections between unsplittable flow and scheduling problems. Consider the scheduling problem $\mathcal{S}$ defined as follows. A set J of jobs is to be scheduled on a set M of parallel machines. A job j can be scheduled to run with processing time p_j on a set of machines $M(j)$ and has processing time

∞ on all machines in $M - M(j)$. In other words, it is technologically infeasible for job j to run on machines in $M - M(j)$. The objective is to find a schedule, which minimizes the *makespan,* i.e. the maximum completion time of a job. $\mathcal{S}$ is a special case of minimizing makespan on unrelated machines. In the unrelated machine setting, job j has a machine-dependent processing time p_{ij} on $i \in M$. The best approximation algorithm known for $\mathcal{S}$ is the 2-approximation for unrelated machine scheduling [74, 91]. From a straightforward modification to Theorem 5 in [74], the following hardness result is obtained.

Theorem 3.10.1 *[74] Unless P=NP, no approximation better than* $3/2$ *exists for* $\mathcal{S}$ *with processing times from the set* $\{1, 2, \infty\}$.

Kleinberg [61] gave an approximation-preserving reduction from $\mathcal{S}$ to minimum congestion unsplittable flow on a three-level directed graph G. A source vertex has edges directed to vertices representing the machines. The vertex set of G contains also one vertex for each job. Machine vertex i has an edge directed to job vertex j if and only if $i \in M(j)$. The edges out of the source have capacity T and the edges into the job vertices have infinite capacity. Finally every job vertex has demand equal to p_j. An unsplittable flow routing in G where at most ρT amount of flow is pushed through an edge corresponds to a schedule with makespan ρT for $\mathcal{S}$. See Figure 3.2 for an example network.

Consider a UFP on an arbitrary network with demands p and $2p$, for some $p > 0$. This includes the family of networks obtained for the scheduling problem with processing times $\{1, 2, \infty\}$. As a corollary to Theorem 3.4.2 we obtain a tight approximation ratio of $3/2$. The lower bound comes from Theorem 3.10.1.

Corollary 3.10.1 *Given an* UFP $\Pi = (G, s, \mathcal{T})$, *with demands from the set* $\{p, 2p\}$, $p > 0$, *there is an algorithm to find in polynomial time an unsplittable routing where the flow through an edge* e *is at most* $u_e + p$. *Thus the approximation ratio for congestion is at most* $3/2$. *This is best possible, unless P=NP.*

Corollary 3.10.2 *For problem* $\mathcal{S}$ *with processing times from* $\{p, 2p, \infty\}$ *there is a polynomial-time algorithm, which outputs a schedule within an additive* p *of the optimum makespan. The approximation ratio is at most* $3/2$ *and this is best possible, unless P=NP.*

It is instructive to consider the action taken by algorithm 2H_PARTITION on the UFP resulting from scheduling problem $\mathcal{S}$. The algorithm splits every job of processing time $2p$, into two virtual jobs each of processing time p. Then a schedule g_1 with optimum makespan is computed for the resulting instance. In the scheduling context, when all the jobs have the same processing time, an optimum schedule can be found by solving an assignment problem. Schedule g_1 corresponds to a half-integral superoptimal solution for $\mathcal{S}$. In the second phase of 2H_PARTITION this half-integral solution is rounded to an integral one.

The results of Theorem 3.10.1 can be generalized to the following.

Corollary 3.10.3 *Given an* UFP $\Pi = (G, s, \mathcal{T})$, *with demands from the set* $\{p, Cp\}$, $C > 1, p > 0$ *there is an algorithm to find in polynomial time an unsplittable routing where the flow through an edge* e *is at most* $u_e + (C-1)p$. *Thus the approximation ratio for congestion is at most* $2 - \frac{1}{C}$.

We now proceed to examine the case, in which the demands lie in the interval $[p, Cp]$, $p > 0$. Lemma 3.3.2 would give in this setting a $(C+1)$-approximation for

congestion, since the minimum capacity is now Cp. We show how to achieve a C-approximation, the same as the ratio of the interval.

Theorem 3.10.2 *Given an* UFP $\Pi = (G, s, \mathcal{T})$, *with demands from the interval* $[p, Cp]$, $C > 1, p > 0$, *there is an algorithm to find in polynomial time an unsplittable routing where the flow through an edge* e *is at most* Cu_e.

Proof. As in the proof of Theorem 3.4.1 we assume without loss of generality that Π has an unsplittable routing with congestion 1. Otherwise one can use the algorithm we propose as a C-relaxed decision procedure [45] in conjunction with a binary search for the optimum congestion to obtain the claimed approximation. The algorithm is as follows. Round all demands down to p. Call the resulting problem Π'. Since an unsplittable flow solution exists for Π, one exists for Π' as well. Rounding down the capacities of edges to the closest multiple of p does not affect the existence of this solution. But for Π' all demands are p and all capacities are multiples of p therefore by Theorem 3.2.1 we can find an unsplittable flow solution g' in polynomial time. To obtain from g' an unsplittable routing for Π it suffices to increase the flow along paths in g' that lead to sinks in $\mathcal{T}$ with demands more than p. The increase will be at most by a multiplicative factor of C on each path, hence the approximation theorem. ■

Interestingly, the approach in Theorem 3.10.2 does not seem to yield an improvement on the result of Lemma 3.3.3. For problem Π' in the proof above, the capacities are already multiples of p, an assumption we cannot make in the setting of Lemma 3.3.2.

Chapter 4

Approximating Disjoint-Path Problems Using Packing Integer Programs[1]

You know, space is unending,
you know, you don't have to fly
— *Paul Celan,* The No-One's-Rose

4.1 Introduction

In a *packing integer program,* (PIP), we seek a vector x of integers, which maximizes $c^T \cdot x$, subject to $Ax \leq b$, $A, b, c \geq 0$. The term "packing" arises from the following consideration. Assume for simplicity that all entries of c and b are equal to 1. The objective is then to select a maximum number of the column vectors of the $m \times n$ matrix A so that their vector sum "fits" in the m-cube. We are therefore interested in packing a maximum number of the column vectors of A into the m-cube. Packing in-

[1]This chapter contains joint work with Cliff Stein [67].

teger programs are a well-studied class of integer programs. They can model several NP-complete problems, including independent set, hypergraph k-matching [76, 1], job-shop scheduling [84, 90, 98, 77]. Many of these problems seem to be difficult to approximate, and not much is known about their worst-case approximation ratios. A large category of problems, which involve path selection in a graph under capacity constraints, such as edge-disjoint paths, and more generally integral multicommodity flow, have natural packing formulations. In this setting, each entry of the x vector corresponds to a path between a source-sink pair of the underlying flow network, while rows correspond to the capacity constraints on the edges. Relaxing the integrality constraint on x yields a fractional solution to the underlying flow problem. The given formulation has however a variable for each of an exponential number of paths. (See [84] for an alternative packing formulation and how despite the exponential size a fractional solution can be efficiently computed). The close connection between packing and disjoint paths has found only limited use in the design of approximation algorithms. To our knowledge the connection has been previously used only for the congestion metric [84]. In the latter case, which we will not consider here, congestion corresponds to an amount by which a solution can violate capacity constraints. This allows solutions to be infeasible and the objective is to bound the amount of infeasibility introduced. See Chapter 3 for the study of the congestion metric for single-source unsplittable flow.

In this chapter we explore the topic of approximating disjoint-path problems using polynomial-size packing integer programs. Motivated by disjoint-paths applications we provide improved approximation algorithms for classes of packing integer programs, a result that we believe is of independent interest. An underlying theme of this work is the efficient rounding of a solution to a fractional relaxation of a problem,

such as PIP or disjoint paths, to an integral feasible solution while incurring bounded degradation to the objective value.

Disjoint-Path Problems. In the *edge(vertex)-disjoint path* problem, we are given a graph $G = (V, E)$ and a set $\mathcal{T}$ of *connection requests,* also called *commodities.* Every connection request in $\mathcal{T}$ is a vertex pair (s_i, t_i), $1 \leq i \leq K$. The objective is to connect a maximum number of the pairs via edge(vertex)-disjoint paths. For the vertex-disjoint paths problem, the connection requests are assumed to be disjoint. We call the set of connected pairs *realizable.* This is an *admission control* optimization problem, in which a set of requests will be rejected. A generalization of the edge-disjoint paths problem is *multiple-source unsplittable flow* defined in Section 3.1. In this problem every commodity k in the set $\mathcal{T}$ has an associated demand ρ_k, and every edge e has a capacity u_e. The demand ρ_k must be routed on a single path from s_k to t_k. The objective is to maximize the sum of the demands that can be fully routed while respecting the capacity constraints. Without loss of generality, we assume that $\max_k \rho_k = 1$, and following the standard definition of the problem in the literature, $u_e \geq 1$, $\forall e \in E$. Unsplittable flow models the general situation of heterogeneous requirements for different commodities, which have to be routed using the same underlying network. When all demands and capacities are 1 in the multiple-source unsplittable flow problem we obtain the edge-disjoint path problem. See [60] and Chapter 3 for further applications and motivation for unsplittable flow. In all the above problems one can assign a weight $w_i \leq 1$ to each connection request and seek to find a realizable set of maximum total weight. In this chapter we will state explicitly when we deal with the weighted version of a problem.

Both the edge- and vertex-disjoint path problems are fundamental, NP-hard [53, 26, 82],extensively studied problems (see e.g. [87, 31, 88, 81, 60, 64, 10]), which

combine two basic combinatorial notions: packing and routing. They have applications in diverse areas such as telecommunications, VLSI and scheduling. Disjoint-path problems have been brought to further prominence due to the emergence of high-speed, large-bandwidth networks. It is hoped that a better theoretical understanding will lead to good heuristics for practical admission control and virtual-circuit routing problems. See Kleinberg's thesis for relevant background [60]. In dealing with these computationally hard problems, we focus on ρ-approximation algorithms; we will also give and refer to algorithms, which output solutions whose size is a non-linear function of the optimum value OPT, such as $OPT^2/|E|$. Despite the attention they have received, disjoint-path problems on general graphs remain notoriously hard in terms of approximation; even for edge-disjoint paths, no algorithm is known which can find even an $\omega(1/\sqrt{|E|})$ fraction of the realizable paths.

Overview of previous work. Two main approaches have been followed for approximation.

(i) The first approach, which we call the rounding approach, consists of solving a fractional relaxation and then use rounding techniques to obtain an integral solution. The fractional relaxation is typically multicommodity flow and the rounding techniques used to date involved sophisticated and use of randomized rounding [96]. The objective value of the resulting solution is compared to the *fractional optimum* y_*, which is an upper bound on the *integral optimum,* OPT. This approach has been the more successful one and recently yielded the first approximation algorithm for *uniform unsplittable flow* [96] which is the special case of unsplittable flow where all the capacities have the same value. Let d denote the *dilation* of the fractional solution, i.e. the maximum length of a flow

path in the fractional relaxation. Bounds that rely on the dilation are particularly appealing for expander graphs, where it is known that $d = O(\text{polylog}(n))$ [71, 62]. The rounding approach yields, for unweighted uniform unsplittable flow (and thus for unweighted edge-disjoint paths as well) a realizable set of size $\Omega(\max\{(y_*)^2/|E|, y_*/\sqrt{|E|}, y_*/d\})$ and an $\Omega(\max\{(y_*)^2/|E|, y_*/d\})$ bound for the weighted version [96]. This approach is known to have limitations, e.g. it is known that a gap of $\Omega(\sqrt{|V|})$ exists between the fractional and integral optima for both the edge- and vertex-disjoint path problems on a graph with $|E| = \Theta(|V|)$ [37].

(ii) Under the second approach, which we call the routing approach, a commodity is never split, i.e. routed fractionally along more than one path during the course of the algorithm. In the analysis, the objective value of the solution is compared to an estimated upper bound on the OPT. This approach has found very limited applicability so far, one reason being the perceived hardness of deriving upper bounds on OPT without resorting to a fractional relaxation. The only example of this method we are aware of is the on-line Bounded Greedy Algorithm in [60] whose approximation guarantee depends also on the diameter of the graph. The algorithm can be easily modified into an off-line procedure, which outputs realizable sets of size $\Omega(OPT/\sqrt{|E|})$ ($\Omega(OPT/\sqrt{|V|})$) for edge(vertex)-disjoint paths. The $\Omega(OPT/\sqrt{|V|})$ bound is the best known bound to date for vertex-disjoint paths.

Our contribution. In this chapter we provide techniques for approximating disjoint-path problems that bear on both of the above approaches. Tables 4.1 and 4.2 summarize previous and new bounds for edge-, vertex-disjoint path and unsplittable flow

	routing approach	rounding approach
unweighted EDP	$\frac{OPT}{\sqrt{\lvert E\rvert}}$ [60], $\frac{\mathbf{OPT}}{\sqrt{\lvert\mathbf{E_o}\rvert}}$, $\frac{\mathbf{OPT^2}}{\lvert\mathbf{E_o}\rvert}$, $\frac{\mathbf{OPT}}{\mathbf{d_o}}$	$\frac{y_*}{\sqrt{\lvert E\rvert}}$ [96], $\frac{(y_*)^2}{\lvert E\rvert}$ [96], $\frac{y_*}{d}$ [96]
weighted EDP	—	$\frac{\mathbf{y_*}}{\sqrt{\lvert\mathbf{E}\rvert}}$, $\frac{(y_*)^2}{\lvert E\rvert}$ [96], $\frac{y_*}{d}$ [96]
weighted UCUFP	—	$\frac{(y_*)^2}{\lvert E\rvert}$ [96], $\frac{y_*}{d}$ [96]
weighted UFP	—	$\frac{\mathbf{y_*}}{\log\log\lvert\mathbf{E}\rvert\sqrt{\lvert\mathbf{E}\rvert}}$, $\frac{\mathbf{y_*}}{\mathbf{d}}$ $\frac{(\mathbf{y_*})^\mathbf{2}}{\lvert\mathbf{E}\rvert\log^\mathbf{2}\lvert\mathbf{E}\rvert\log\log\lvert\mathbf{E}\rvert}$

Table 4.1: Known approximation bounds for edge-disjoint paths (EDP), uniform capacity unsplittable flow (UCUFP), and general unsplittable flow (UFP), Ω-notation omitted. E_o denotes the set of edges used by some path in an integral optimal solution and d_o the average length of the paths in the same solution. Results with no citation come from the the thesis at hand. Our $y^*/\sqrt{|E|}$ bound for the weighted EDP problem holds under the assumption that the number of connection requests K is $O(|E|)$.

problems. Under the routing approach we provide a greedy algorithm for edge and vertex disjoint paths. See section 4.1.3 for further details.

We turn to the rounding approach (approach **(i)**) to handle the weighted disjoint path and unsplittable flow problems. We propose the use of *packing integer programs* as a unifying framework that abstracts away the need for customized and complex randomized rounding schemes. We develop, as part of our tools, improved approximation algorithms for a class of packing integer programs, called *column restricted,* that are relevant to unsplittable flow problems. Armed with both this new algorithm and existing algorithms for general packing integer programs, we show how packing formulations both provide a unified and simplified derivation of many results from [96] and also how they lead to new ones. In particular, we obtain the first approximation algorithm for weighted multiple-source unsplittable flow on networks with arbitrary demands and capacities and the first approximation algorithm for weighted

vertex-disjoint paths. Further, we believe that both our new algorithm for column-restricted packing integer programs and the technique associated with its derivation are of independent interest.

We now elaborate on our results under the rounding approach, providing further background as necessary.

4.1.1 Packing Integer Programs

Following [95] a packing integer program (PIP) is defined as follows.

Definition 1 *Given* $A \in [0,1]^{m \times n}$, $b \in [1,\infty)^m$ *and* $c \in [0,1]^n$ *with* $\max_j c_j = 1$, *a PIP* $\mathcal{P} = (A, b, c)$ *seeks to maximize* $c^T \cdot x$ *subject to* $x \in Z_+^n$ *and* $Ax \leq b$. *Constraints of the form* $0 \leq x_j \leq d_j$ *are also allowed. If* $A \in \{0,1\}^{m \times n}$, *each entry of* b *is assumed integral. Let* $B = \min_i b_i$, *and* α *be the maximum number of non-zero entries in any column of* A.

When $A \in \{0,1\}^{m \times n}$, we say that we have a $(0,1)$-*PIP.* The parameters m, B and α in the definition above appear in the approximation bounds. For convenience we call the i-th entry b_i of vector b the *capacity* of row i. The crucial constraint on the values of the entries of A, b, c is nonnegativity. The restrictions on the actual values of the entries of A, b are without loss of generality; the values in an arbitrary packing program can be scaled to satisfy the above requirements [94]. Some complications arise with respect to the cost vector c. If the maximum entry C_0 of c is larger than 1, dividing c by C_0 ensures that the entries of the resulting c' lie in $[0,1]$. Let $OPT(\mathcal{P})$, $OPT(\mathcal{P}')$ be the respective optima of the original PIP $\mathcal{P}$ and the resulting $\mathcal{P}'$ after rescaling c. Trivially $OPT(\mathcal{P}) = C_0 OPT(\mathcal{P}')$. We solve approximately $\mathcal{P}'$ to obtain a feasible solution $\bar{x}$. The objective value we output for a solution to $\mathcal{P}$ is $C_0(c^T \cdot \bar{x})$. If

$c^T \cdot \bar{x} \geq \rho OPT(\mathcal{P}')$ we have obtained a ρ-approximation to $OPT(\mathcal{P})$ as well. Existing algorithms also output solutions with guarantees that are nonlinear in the optimum, e.g. $\Omega(OPT(\mathcal{P}')(OPT(\mathcal{P}')/m)^{1/\lfloor B \rfloor})$ when $\mathcal{P}'$ is a $(0,1)$-PIP. (Cf. Theorem 4.1.1 for the known bounds). But then the approximation guarantee achieved for $OPT(\mathcal{P})$ will be scaled down with respect to the guarantee for $OPT(\mathcal{P}')$ by a function of C_0, in the example above by $(1/C_0)^{1/\lfloor B \rfloor}$. In what follows, we assume that the cost vector for the problem of interest always belongs to $[0,1]^n$. The requirement of scaling down c by C_0, arises from the analysis of the algorithms in [94, 95] that we use as subroutines and in particular from the use of the Chernoff bound (cf. Theorem 4.2.1).

We will state explicitly when some packing program in this chapter deviates from the requirements in Definition 1.

Previous Work on Packing Programs. The basic techniques for approximating packing integer programs have been the *randomized rounding* technique of Raghavan and Thompson [85, 86] and the work of Plotkin, Shmoys and Tardos [84]. Randomized rounding schemes in the packing setting can be typically derandomized through the use of suitable pessimistic estimators, resulting in deterministic algorithms. Let y_* denote the optimum value of the linear relaxation of the PIP of interest. Standard randomized rounding yields integral solutions of value $\Omega(y_*/m^{1/B})$ for general PIP's and $\Omega(y_*/m^{1/(\lfloor B \rfloor+1)})$ for $(0,1)$-PIP's [86] (see also [94]). Observe that for a $(0,1)$-PIP, the integral optimum is not affected if B is rounded down to the nearest integer. Therefore it is not unnatural to work with $\lfloor B \rfloor$. Srinivasan [94, 95] improved on the exponent of m in the above bounds and introduced new ones, depending on α. His work examines more closely the correlation between the packing constraints and introduces probabilistic tools of independent interest. For future reference, we codify

previous work in the following theorem.

Theorem 4.1.1 *[86, 94, 95] Let $\mathcal{P}$ be a PIP conforming to Definition 1, and y_* be the optimum of the linear relaxation of $\mathcal{P}$. One can compute in deterministic polynomial time a feasible solution to $\mathcal{P}$ of value*

$$\Omega(\max\{y_*/m^{1/B}, y_*(y_*/m)^{1/(B-1)}), y_*/\alpha^{1/(B-1)}\}).$$

If $\mathcal{P}$ is a $(0,1)$-PIP the guarantee on the solution value is

$$\Omega(\max\{y_*/m^{1/(\lfloor B\rfloor+1)}, y_*(y_*/m)^{1/\lfloor B\rfloor}), y_*/\alpha^{1/\lfloor B\rfloor}\}).$$

New results for column-restricted PIP's. The above results show that for various combinations of values for y_*, m and B, the bounds obtained for a $(0,1)$-PIP are significantly better than those for general PIP's. To facilitate comparison let us consider the case when B is integral. The PIP bounds are always better when $y_* < m$. As another example, the approximation ratio $m^{1/(\lfloor B\rfloor+1)}$ obtained for a $(0,1)$-PIP is *polynomially better* than the approximation ratio of a PIP with the same parameters. Thus it is natural to ask whether we can bridge this gap. We make progress in this direction by defining a *column-restricted* PIP $\mathcal{P}_r$ as one in which all non-zero entries of the j-th column of A have the same value $\rho_j \leq 1$. Column-restricted PIP's arise in applications such as unsplittable flow problems (see next section). We show how to obtain approximation guarantees for column-restricted PIP's that are similar to the ones obtained for $(0,1)$-PIP's. For a column-restricted PIP, we obtain an integral solution of value

$$\Omega(\max\{\frac{y_*}{m^{1/(\lfloor B\rfloor+1)}},\ \frac{y_*}{\alpha^{1/\lfloor B\rfloor}},\ y_*\left(\frac{y_*}{m\log\log m}\right)^{1/\lfloor B\rfloor}\}).$$

We now give an overview of our *grouping-and-scaling* technique. First we find an optimum solution x^* to the linear relaxation of the column-restricted PIP $\mathcal{P}_r$. We

partition the ρ_j's into a fixed number of intervals according to their values and generate a packing subproblem for each range. In a packing subproblem $\mathcal{P}^L$ corresponding to range L, we only include the columns of A with $\rho_j \in L$ and to each component of the b^L-vector we allocate only a fraction of the original b_i value, a fraction that is determined based on information from x^*. Next we find approximate solutions to each subproblem and combine them to obtain a solution to the original problem. Perhaps the key idea is in using the solution x^* to define the capacity allocation to the b^L-vector for subproblem $\mathcal{P}^L$. The other key idea is that each subproblem can be approximated almost as well as a $(0,1)$-PIP. This set of ideas generalizes work in Chapter 3 on single-source unsplittable flow and in particular the ideas used in algorithm M_Partition in Section 3.7. We believe that the grouping-and-scaling technique can find further applications.

4.1.2 Applications of packing to approximation

We introduce an approach for applying packing techniques to disjoint-path problems. First, we formulate an integer program (which is not necessarily a PIP) and solve a linear relaxation of this integer program to obtain a solution x. Typically this is a multicommodity flow problem. We then explicitly use the solution x to guide the formation of a column-restricted or $(0,1)$-PIP. A related usage of a solution to the linear relaxation of integer programs in a different context can be found in [56, 97]. An integral approximate solution to the created PIP will be an approximate solution to the original disjoint path problem (with possibly some small degradation in the approximation factor). This integral solution can be found using existing algorithms for approximating PIP's as a black box; we abstract away the relevant complications arising from sophisticated randomized rounding schemes and their derandomization

[94, 95]. Our algorithms apply to the case when there are weights on the commodities, and thus generalize those of Srinivasan for edge-disjoint paths. This general approach yields four applications which we explain below.

Application 1: weighted unsplittable flow. Let $\alpha^f(\mathcal{T})$ optimum of a fractional relaxation with dilation d. We obtain a realizable set of weight

$$\Omega(\max\{\frac{\alpha^f(\mathcal{T})}{\log|E|\sqrt{|E|}},\ \frac{(\alpha^f(\mathcal{T}))^2}{|E|\log^2|E|\log\log|E|},\ \frac{\alpha^f(\mathcal{T})}{d}\})$$

for unsplittable flow with arbitrary demands and capacities. We provide improved bounds when the maximum demand is bounded away from 1 (cf. Theorem 4.3.4). In the case where the number of commodities $K = O(|E|)$ we also show how to obtain an unsplittable flow of value

$$\Omega(\max\{\frac{\alpha^f(\mathcal{T})}{\sqrt{|E|}},\ \frac{(\alpha^f(\mathcal{T}))^2}{|E|\log\log|E|}\}).$$

For the edge-disjoint path problem this is a natural assumption since at most $|E|$ connection requests can be feasibly routed. We also note that a ρ-approximation for the maximization problem entails an $O(\rho\log|E|)$ approximation for the problem of *routing in rounds* [5, 60] (cf. Chapter 3 for a study of the problem in a single-source setting). We do not pursue the latter problem any further in this thesis.

Application 2: weighted vertex-disjoint paths. We give an algorithm that outputs a solution of value

$$\Omega(\max\{\frac{\alpha^f(\mathcal{T})}{\sqrt{|V|}},\ \frac{(\alpha^f(\mathcal{T}))^2}{|V|},\ \frac{\alpha^f(\mathcal{T})}{d}\}).$$

The algorithm relies on the observation that, after solving a fractional relaxation, the problem of rounding is essentially an instance of hypergraph matching; thus it can be formulated as a packing program with $|V|$ constraints. The algorithm is surprisingly

simple but the performance guarantee matches the integrality gap known for the problem [37].

Application 3: routing with low congestion. A problem that has received a lot of attention in the literature on routing problems (e.g. [86, 73, 84, 83, 60, 66]) is that of minimizing *congestion,* i.e. the factor by which one is allowed to scale up capacities in order to achieve an optimal (or near-optimal) realizable set. See also Chapter 3 for study of the congestion metric in the single-source unsplittable flow setting. In our usage of packing in the rounding algorithms we have assumed that the parameter B of the packing program is equal to 1. Allowing $B > 1$ is equivalent to allowing congestion B in the corresponding disjoint-path problem. Thus another advantage of the packing approach is that tradeoffs with the allowed congestion B can be obtained immediately by plugging in B in the packing algorithms that we use as a black box. For example the approximation for edge-disjoint paths becomes $\Omega(\max\{y^*(y^*/|E|\log\log|E|)^{1/B}, y^*/|E|^{1/(B+1)}, y^*/d^{1/B}\})$, when the number of connection requests is $O(|E|)$. Our congestion tradeoffs generalize previous work by Srinivasan [96] who showed the $\Omega(y^*/d^{1/B})$ tradeoff for uniform capacity unsplittable flow. We do not state the tradeoffs explicitly for the various problems since they can be easily obtained by simple modifications to the given algorithms.

Application 4: Independent Set in powers of a graph. Given a graph $G = (V, E)$ the *k-th power,* $k \in Z_{\geq 1}$, $G^k = (V, E^k)$ of G is a graph where two vertices are adjacent if and only if they are at distance at most k in G. We further demonstrate the power of packing formulations by providing an $O(\sqrt{|V|})$ approximation algorithm for finding a maximum independent set in an even powers of G. We also give results that depend on the maximum vertex degree Δ in G. Our approximation ratio cannot be

polynomially improved in the sense that no $(|V|/(k+2))^{1/2-\varepsilon}$ approximation, for any fixed $\varepsilon > 0$, can be obtained in polynomial time unless $NP = ZPP$. For odd powers of a graph we show how to achieve an $O(\sqrt{|E|})$ approximation; this result is well known for $G^1 = G$. Depending on the actual graph G, it is possible that $|E| = o(|E^k|)$, $k \geq 2$, therefore our result existentially improves upon the naive bound. Studying NP-hard problems in powers of graphs is a topic that has received some attention in the literature [28, 100, 75, 16].

4.1.3 A Greedy Algorithm

Under the routing approach (approach **(ii)**) we give a simple greedy algorithm Greedy_Path for edge-disjoint paths that has performance guarantees comparable to those obtained by the multicommodity flow based algorithms [96]. Greedy algorithms have been extensively studied in combinatorial optimization due to their elegance and simplicity. Our work provides another example of the usefulness of the greedy method.

Greedy_Path outputs a realizable set of size $\Omega(\max\{OPT^2/|E_o|, OPT/\sqrt{|E_o|}\})$ for the edge-disjoint path problem. Here $E_o \subseteq E$ is the set of edges used by the paths in an optimal solution. Note that $OPT^2/|E_o|$ always dominates $OPT/\sqrt{|E_o|}$ in the unweighted case that we consider; we give both bounds to facilitate comparison with existing work and to conform to the traditional notion of a ρ-approximation algorithm. Our approximation existentially improves upon the multicommodity-flow based results when $|E_o| = o(|E|)$, i.e. when the optimal solution uses a small portion of the edges of the graph. Another bound can be obtained by noticing that $OPT^2/|E_o| = OPT/d_o$, where d_o denotes the average length of the paths in an optimal solution.

Essentially the same algorithm, Greedy_VPath, obtains for the vertex-disjoint

	routing approach	rounding approach
unweighted	$\frac{OPT}{\sqrt{\lvert V\rvert}}$ [60], $\frac{\mathbf{OPT}}{\sqrt{\mathbf{\lvert V_o\rvert}}}$, $\frac{\mathbf{OPT^2}}{\mathbf{\lvert V_o\rvert}}$, $\frac{\mathbf{OPT}}{\mathbf{d_o}}$	$\frac{\mathbf{y^*}}{\sqrt{\mathbf{\lvert V\rvert}}}$, $\frac{\mathbf{(y^*)^2}}{\mathbf{\lvert V\rvert}}$, $\frac{\mathbf{y^*}}{\mathbf{d}}$
weighted	—	$\frac{\mathbf{y^*}}{\sqrt{\mathbf{\lvert V\rvert}}}$, $\frac{\mathbf{(y^*)^2}}{\mathbf{\lvert V\rvert}}$, $\frac{\mathbf{y^*}}{\mathbf{d}}$

Table 4.2: Known approximation bounds for vertex-disjoint paths, Ω-notation omitted. V_o denotes the set of vertices used by some path in an integral optimal solution and d_o the average length of the paths in the same solution. Results with no citation come from the thesis at hand.

path problem a realizable set of size $\Omega(\max\{OPT^2/|V_o|, OPT/\sqrt{|V_o|}\})$, where $V_o \subseteq V$ is the set of vertices used by the paths in an optimal solution. Recall that the best known bound to date is $t = \Omega(OPT/\sqrt{|V|})$. The realizable set output by our algorithm has size $\Omega(t^2)$ and potentially better than this when $|V_o| = o(|V|)$. This is a significant improvement when $OPT = \omega(\sqrt{|V|})$. For example, when $OPT = \Omega(|V|)$, we obtain a constant-factor approximation. Again an $\Omega(OPT/d_o)$ guarantee follows immediately.

The outline of the chapter is as follows. In Section 4.2 we present approximation algorithms for packing integer programs. In Section 4.3 we show how unsplittable flow and related problems can be approximated by resorting to packing. In Section 4.4 we conclude with the greedy algorithm. A preliminary version of the material in this chapter appeared in [67]. Independently of our work, Baveja and Srinivasan [6] have obtained results similar to ours for approximating vertex-disjoint paths under the rounding approach, unsplittable flow and column-restricted packing integer programs. Their work follows a different approach and builds on the methods in [96].

4.2 Approximating a column-restricted PIP

4.2.1 Notation and basic facts

Let $\mathcal{P} = (A, b, c)$ be a column-restricted PIP. We call $\rho_j \leq 1$, the value of the non-zero entries of the j-th column, $1 \leq j \leq n$, the *value* of column j. Throughout this section we use y_* to denote the optimum of the linear relaxation of the PIP under consideration. We will also refer to y_* as the *fractional optimum* of $\mathcal{P}$ and to a solution of the linear relaxation of $\mathcal{P}$ as a *fractional solution.* We now introduce some notation.

Given α_i, α_j, let J^{α_i,α_j} be the set of column indices k for which $\alpha_i < \rho_k \leq \alpha_j$. We then define A^{α_i,α_j} to be the $m \times |J^{\alpha_i,\alpha_j}|$ submatrix of A consisting of the columns in J^{α_i,α_j}, and for any vector x, x^{α_i,α_j} to be the $|J^{\alpha_i,\alpha_j}|$-entry subvector x consisting of the entries whose indices are in J^{α_i,α_j}. We will also need to combine back together the various subvectors, and define $x^{\alpha_1,\alpha_2} \cup \cdots \cup x^{\alpha_{k-1},\alpha_k}$ to be the n-entry vector in which the various entries in the various subvectors are mapped back into their original positions. Any positions not in any of the corresponding index sets J^{α_i,α_j} are set to 0. Let x_* be the optimal solution of the linear relaxation of $\mathcal{P}$, i.e., $y_* = c^T x_*$. We use $y_*^{r_1,r_2}$ to denote $c^{r_1,r_2} x_*^{r_1,r_2}$.

When column values are much smaller than B, one can obtain a constant-factor approximation. In particular, the randomized rounding scheme of Raghavan and Thompson [86] together with a straightforward application of the Chernoff bound [15] yields the well-known result we present in Theorem 4.2.2. We give first the relevant Chernoff-type bound. We provide a simplified bound, which is sufficient for our purposes.

Theorem 4.2.1 *[15, 85] If Y is the sum of independent random variables each in*

$[0, U]$, *with* $E(Y) \leq \mu$, *then, for any* $\epsilon > 0$,

$$Pr[Y \geq (1+\epsilon)\mu] \leq e^{-\min\{\epsilon,\epsilon^2\}\mu/3U}.$$

For the sake of completeness we sketch the proof of the following basic result. The value 12 for the constant β is indicative; it can be decreased based on a trade-off with γ. In the remainder of the chapter, β is used to denote a constant greater than or equal to 12.

Theorem 4.2.2 *Let* $\mathcal{P} = (A, b, c)$ *be a column-restricted PIP such that the maximum column value* $\rho_{\max}$ *is at most* $1/(\beta \log m)$ *for some constant* $\beta \geq 12$. *Then one can obtain in deterministic polynomial time a feasible solution to* $\mathcal{P}$ *of value at least* γy_*, *for some constant* $0 < \gamma < 1$.

Proof sketch. For simplicity, we assume that the entries of a feasible solution are restricted to lie in $\{0, 1\}$. Let x_* be the n-vector which solves optimally the linear relaxation of $\mathcal{P}$. Obtain an integral vector $\bar{x}$ by the following random experiment. Set $\bar{x}_j$ to 1 with probability $(1/2)x_{*j}$ and to 0 with probability $1-(1/2)x_{*j}$. Consider now the m packing constraints, and let $Z_i = \sum_{j=1}^{n} A_{ij}\bar{x}_j$, $1 \leq i \leq m$. Z_i is the sum of at most n independent random variables each lying in $[0, (\beta \log m)^{-1}]$; The expectation $E(Z_i)$ is at most $(1/2)b_i$. The event $\{Z_i > b_i\}$ corresponds to the i-th constraint being violated by solution $\bar{x}$. By the Chernoff bound with $\epsilon = 1$,

$$Pr[Z_i > b_i] = Pr[Z_i > 2(1/2)b_i] \leq e^{-(1/2)b_i/3(\beta \log m)^{-1}}.$$

Using the fact that $b_i \geq 1$, and setting $\beta = 12$, we obtain that

$$Pr[Z_i > b_i] < \frac{1}{m^2}.$$

Invoking the union bound, the probability that at least one of the m constraints is violated is at most $1/m$. The expected value of the objective $c^T\bar{x}$ is $(1/2)$ times

the fractional optimum. Let p denote $Pr[c^T\bar{x} \geq (1/4)y_*]$. By the definition of the expectation we deduce that

$$(1/2)y_* < (1/4)y_*(1-p) + (1/2)y_*p \Rightarrow p > 1/3.$$

Therefore, with probability at least $1/3 - 1/m$, the random experiment yields a feasible solution to $\mathcal{P}$ of value at least $(1/4)y_*$. The outlined algorithm can be derandomized by applying the method of conditional probabilities [25, 92, 85]. ■

Theorem 4.2.2 combined with the linearity of the objective function has interesting consequences. If $y_*^{0,(\beta \log n)^{-1}} \geq y_*/2$, a vector $\bar{x}'$ yielding a constant-factor approximation to the fractional optimum of $\mathcal{P}' = (A^{0,(\beta \log m)^{-1}}, b, c)$, can be trivially extended with zeros to a feasible solution to $\mathcal{P}$, yielding a constant-factor approximation to y_*. Similarly if $y_*^{(\beta \log n)^{-1},1} \geq y_*/2$, a vector $\bar{x}''$ yielding a ρ-approximation to the fractional optimum of $\mathcal{P}'' = (A^{(\beta \log n)^{-1},1}, b, c)$, can be extended to a $(\rho/2)$-approximation to y_* for $\mathcal{P}$. The argument extends for approximation guarantees which are non-linear in y_* and we codify it in Fact 4.2.1. Therefore we assume in the remainder of this section that the minimum column value in A is $\Omega(1/\log m)$.

Fact 4.2.1 *Let $\mathcal{P} = (A, b, c)$ be a column-restricted PIP and let γ denote the constant in the statement of Theorem 4.2.2. If there is an algorithm that finds in polynomial time a feasible solution to $\mathcal{P}'' = (A^{(\beta \log n)^{-1},1}, b, c)$ of value at least $f(y_*^{(\beta \log n)^{-1},1})$, for some function f with the property $f(y) \leq \gamma y$, $\forall y \geq 0$, then one can find in polynomial time a feasible solution to $\mathcal{P}$ of value at least $f(y_*/2)$.*

Our goal is to demonstrate that column-restricted PIP's admit approximations of similar quality to those known for $(0,1)$-PIP's. Accordingly, we will give many of our results in terms of a generic bound for a $(0,1)$-PIP. Throughout this section we assume that there is a polynomial-time algorithm $\mathcal{A}$, which given a $(0,1)$-PIP with

fractional optimum y_*, outputs an integral solution of value at least $\sigma(m, \lfloor B \rfloor, \alpha, y_*)$; m, B, α are the parameters of the packing program under consideration. Our primary interest lies in the asymptotic behavior of our approximation guarantees; knowledge of the asymptotic behavior of σ suffices for our purposes. The asymptotic behavior of known functions σ is given in Theorem 4.1.1. For example σ is known to be $\Omega(y_*/m^{1+\lfloor B \rfloor})$.

4.2.2 The approximation algorithm

At a high level our algorithm partitions $\mathcal{P}$ into subproblems, each containing a subset of the columns of matrix A. The column values in each subproblem will be close in range. We start by providing a subroutine for solving a column-restricted PIP when the column values are close in range.

Theorem 4.2.3 *Let $\mathcal{P} = (A, b, c)$ be a column-restricted PIP, in which all column values ρ_j are equal to ρ and each b_i equals $k_i \Gamma \rho$, $\Gamma \geq 1$, k_i positive integer, $1 \leq i \leq m$. Here $\min_i b_i$ is not necessarily greater than 1. Let y_* denote the optimum of the linear relaxation of $\mathcal{P}$ and $k = min_i k_i$. Then we can find in polynomial time a feasible solution for $\mathcal{P}$ of value at least $\sigma(m, \lfloor k\Gamma \rfloor, \alpha, y_*)$.*

Proof. Transform the given program $\mathcal{P}$ to a $(0,1)$-PIP $\mathcal{P}' = (A', b', c)$, in which $b'_i = k_i \Gamma$, and $A'_{ij} = A_{ij}/\rho$. Every feasible solution (either fractional or integral) $\bar{x}$ to $\mathcal{P}'$ is a feasible solution to $\mathcal{P}$ and vice versa. Therefore the fractional optimum y_* is the same for both programs. Also the maximum number of non-zero entries on any column is the same for A and A'. Thus we can unambiguously use α for both. Invoking algorithm A for $\mathcal{P}'$ returns an integral solution of objective value $\sigma(m, \lfloor k\Gamma \rfloor, \alpha, y_*)$.

■

The proof of the following lemma generalizes that of Lemma 4.1 in [61]. See also Lemma 3.7.1 in Chapter 3.

Lemma 4.2.1 *Let $\mathcal{P} = (A, b, c)$, be a column-restricted PIP with column values in the interval $(a_1, a_2]$, and $b_i \geq \Gamma a_2$, $\forall i$ and some number $\Gamma \geq 1$. Here $\min_i b_i$ is not necessarily greater than 1. Let y_* denote the optimum of the linear relaxation of $\mathcal{P}$. There is a polynomial-time algorithm, α_PACKING, which finds a solution g to $\mathcal{P}$ of value at least $\sigma(m, \lfloor \Gamma \rfloor, \alpha, \frac{a_1}{2a_2} y_*)$.*

Remark: The statement of the lemma holds also when the column values of $\mathcal{P}$ lie in $[a_1, a_2]$.

Proof. We sketch the algorithm α_PACKING. Obtain a PIP $\mathcal{P}' = (A', b', c)$ from $\mathcal{P}$ as follows. Round down b_i to the nearest multiple of Γa_2. Since $a_2 > 0$, the effect of this rounding will be to at most halve b_i while still keeping it greater than or equal to Γa_2. Now multiply the rounded b_i with a_1/a_2. Set b'_i equal to the resulting value. Every b'_i is now between $a_1/2a_2$ and a_1/a_2 times the corresponding b_i. Set A'_{ij} to a_1 if $A_{ij} \neq 0$ and to 0 otherwise. If x_* is an optimal fractional solution to $\mathcal{P}$, $(a_1/2a_2)x_*$ is a feasible fractional solution to $\mathcal{P}'$ of value at least $(a_1/2a_2)c^T x_* = (a_1/2a_2)y_*$.

All column values in $\mathcal{P}'$ are equal to a_1 and every b'_i is a multiple of Γa_1. Thus we can invoke Theorem 4.2.3 and find a feasible solution g' to $\mathcal{P}'$ of value at least $v = \sigma(m, \lfloor \Gamma \rfloor, \alpha, (a_1/2a_2)y_*)$. Scaling up every component of g' by a factor of at most a_2/a_1 yields a vector g that is feasible for $\mathcal{P}$ and has value at least v. ■

The following lemma is a specialized version of Lemma 4.2.1.

Lemma 4.2.2 *Let $\mathcal{P} = (A, b, c)$, be a column-restricted PIP with column values in the interval $(a_1, a_2]$, and b_i a multiple of Γa_2, $\forall i$ and some number $\Gamma \geq 1$. Here $\min_i b_i$ is not necessarily greater than 1. Let y_* denote the optimum of the linear relaxation of*

$\mathcal{P}$. *There is a polynomial-time algorithm* INTERVAL_PACKING, *which finds a solution* g *to* $\mathcal{P}$ *of value at least* $\sigma(m, \lfloor \Gamma \rfloor, \alpha, \frac{a_1}{a_2} y_*)$.

Proof. Similar to that of Lemma 4.2.1. Since all b_i's are already multiples of Γa_2, we don't pay the 1/2 factor for the initial rounding. ■

We now outline in some detail the grouping-and-scaling method employed in the full algorithm. The technique generalizes the ideas used for the single-source unsplittable flow problem in Chapter 3. Let x_* denote the optimum solution to the linear relaxation of $\mathcal{P}$. We are going to create packing subproblems $P^\lambda = (A^\lambda, b^\lambda, c^\lambda)$ such that A^λ contains only the columns of A with values in some fixed range $(\alpha_{\lambda-1}, \alpha_\lambda]$. The subproblems will be amenable to good approximations because of the fixed range of the column values. We will obtain our integral solution to $\mathcal{P}$ by combining these approximate solutions to the subproblems. The crucial step is capacity allocation to subproblems. Consider a candidate for the b^λ-vector that we call the λ-th *fractional capacity vector.* In the fractional capacity vector the i-th entry is equal to $\sum_{j|\rho_j \in (\alpha_{\lambda-1}, \alpha_\lambda]} A_{ij} x_{*j}$. In other words, we allocate capacity to the λ-th subproblem by "pulling" out of b the amount of capacity used up in the solution x^* by the columns with values in $(\alpha_{\lambda-1}, \alpha_\lambda]$. The benefit of such a scheme would be that by using the relevant entries of x^* we could obtain a feasible solution to the linear relaxation of P^λ. However, to be able to benefit from Lemma 4.2.1 to find an approximate integral solution to each P^λ, we need all entries of b^λ to be larger than $B\alpha_\lambda$. This increases the required capacity for each subproblem potentially above the fractional capacity. We resort to more sophisticated capacity allocation to ensure that

(i) there is enough total capacity in the b-vector of $\mathcal{P}$ to share among the subproblems

(ii) the subproblems can benefit from the subroutines in Lemmata 4.2.1 and 4.2.2.

ALGORITHM COLUMN_PARTITION($\mathcal{P}$)

Step 1. Find the n-vector x_* that yields the optimal solution to the linear relaxation of $\mathcal{P}$.

Step 2a. Define a partition of the interval $[(\beta \log m)^{-1}, 1]$ into $\xi = O(\log \log m)$ consecutive subintervals $[(\beta \log m)^{-1}, 4^{-\lfloor \log_4(\beta \log m) \rfloor}], \ldots, (4^{-\lambda}, 4^{-\lambda+1}], \ldots, (4^{-2}, 4^{-1}], (4^{-1}, 1]$. For $\lambda = 1 \ldots \xi - 1$ form subproblem $P^\lambda = (A^\lambda, b^\lambda, c^\lambda)$. A^λ and c^λ are the restrictions defined by $A^\lambda = A^{4^{-\lambda}, 4^{-\lambda+1}}$ and $c^\lambda = c^{4^{-\lambda}, 4^{-\lambda+1}}$. Define similarly $P^\xi = (A^\xi, b^\xi, c^\xi)$ such that $A^\xi = A^{(\beta \log m)^{-1}, 4^{-\lfloor \log_4(\beta \log m) \rfloor}}$, $c^\xi = c^{(\beta \log m)^{-1}, 4^{-\lfloor \log_4(\beta \log m) \rfloor}}$.

Step 2b. Let d_i^λ be the ith entry of $(A^\lambda \cdot x_*^\lambda)$. Define $b^\lambda = b$ when λ is 1, and otherwise $b_i^\lambda = B(1/4)^{\lambda-1} + (1/2)d_i^\lambda$ when $\lambda = 2, 3, \ldots, \xi$.

Step 3. On each P^λ, $2 \leq \lambda \leq \xi$, invoke α_PACKING to obtain a solution vector x^λ. Combine the solutions to subproblems 2 through ξ to form n-vector $\hat{x} = \cup_{2 \leq \lambda \leq \xi} x^\lambda$.

Step 4. Invoke INTERVAL_PACKING on P^1 to obtain a solution vector x^1. Let $\bar{x} = \cup x^1$.

Step 5. Of the two vectors $\hat{x}$ and $\bar{x}$, output the one, call it $\dot{x}$, which maximizes $c^T \cdot \dot{x}$.

Figure 4.1: Algorithm COLUMN_PARTITION.

(iii) the fractional optimum of the λ-th subproblem, $\forall \lambda$, is within a constant factor of the subproblem's contribution to y_*, i.e., $\sum_{j | \rho_j \in (\alpha_{\lambda-1}, \alpha_\lambda]} c_j x_{*j}$.

In particular, we initially assign to each subproblem λ only 1/2 of the fractional capacity vector; this has the asymptotically negligible effect of scaling down the fractional optimum for each subproblem by at most 2. We exploit the unused $(1/2)b$ vector of capacity to add an extra $B\alpha_\lambda$ units to the entries of b^λ, $\forall \lambda$.

We present the full algorithm in Figure 4.1. Algorithm COLUMN_PARTITION takes as input a column-restricted packing integer program $\mathcal{P}$ and outputs a feasible integral solution $\dot{x}$. Recall from Fact 4.2.1 that the minimum column value is assumed to be $1/(\beta \log m)$ for some constant $\beta \geq 12$.

Two tasks remain. First, we must show that the vector $\dot{x}$ output by the algorithm is a feasible solution to the original packing problem $\mathcal{P}$. Second, we must lower bound $c^T \cdot \dot{x}$ in terms of the optimum $y_* = c^T \cdot x_*$ of the linear relaxation of $\mathcal{P}$. If $r_1 = (1/4)^\lambda$ and $r_2 = (1/4)^{\lambda-1}$, then we abbreviate $y_*^{r_1,r_2}$ as y_*^λ. We examine first the vector $\hat{x}$.

Lemma 4.2.3 *Algorithm* COLUMN_PARTITION *computes an n-vector $\hat{x}$ which is a feasible solution to $\mathcal{P}$ of value at least*

$$\sum_{\lambda=2}^{\lambda=\xi} \sigma(m, \lfloor B \rfloor, \alpha, (1/16)y_*^\lambda).$$

Proof. Let $\hat{y}^\lambda$ be the optimum of the linear relaxation of P^λ. By Lemma 4.2.1 the value of the solution x^λ, $2 \leq \lambda \leq \xi$, found at Step 3 is at least $\sigma(m, \lfloor B \rfloor, \alpha, (1/2)(1/4)\hat{y}^\lambda)$. By the definition of b^λ in P_λ, $(1/2)x_*^\lambda$,i.e. the restriction of x_* scaled by 1/2, is a feasible fractional solution for P_λ. Thus $\hat{y}^\lambda \geq (1/2)y_*^\lambda$. Hence, the value of x^λ is at least $\sigma(m, \lfloor B \rfloor, \alpha, (1/16)y_*^\lambda)$. The claim of the theorem on the value follows.

For the feasibility, we observe that the aggregate capacity used by $\hat{x}$ on row i of A is the sum of the capacities used by x^λ, $2 \leq \lambda \leq \xi$, on each subproblem. This sum is by Step 2b at most

$$(1/2) \sum_{\lambda=2}^{\lambda=\xi} d_i^\lambda + B \sum_{\lambda=2}^{\lambda=\xi} (1/4)^{\lambda-1}.$$

But

$$B \sum_{\lambda=2}^{\xi} (1/4)^{\lambda-1} < (1/2)B \leq (1/2)b_i$$

and

$$\sum_{\lambda=2}^{\lambda=\xi} d_i^\lambda \leq b_i.$$

Thus the aggregate capacity used by $\hat{x}$ is at most

$$(1/2)b_i + (1/2)b_i = b_i.$$

Therefore $\hat{x}$ is a feasible solution to $\mathcal{P}$. ■

It remains to account for $\bar{x}$. The following theorem is the main result of this section.

Theorem 4.2.4 *Let $\mathcal{P} = (A, b, c)$ be a column-restricted PIP and y_* be the optimum of the linear relaxation of $\mathcal{P}$. Algorithm* Column_Partition *finds in polynomial time a solution g to $\mathcal{P}$ of value*

$$\Omega(\max\{\frac{y_*}{m^{1/(\lfloor B\rfloor+1)}},\ \frac{y_*}{\alpha^{1/\lfloor B\rfloor}},\ y_*\left(\frac{y_*}{m\log\log m}\right)^{1/\lfloor B\rfloor}\}).$$

Proof. Each of the two vectors $\hat{x}, \bar{x}$ solves a packing problem with column values lying in $((\beta\log m)^{-1}, 1/4]$ and $(1/4, 1]$ respectively. Let $\hat{\mathcal{P}}, \bar{\mathcal{P}}$ be the two induced packing problems. The optimal solution to the linear relaxation of at least one of them will have value at least $1/2$ of the optimum y_* of the linear relaxation of $\mathcal{P}$. It remains to lower bound the approximation achieved by each of the two vectors on its corresponding domain of column values. Since m, B, and α are the same for all subproblems, note that σ is a function of one variable, y_*. By Lemma 4.2.1, vector $\bar{x}$ outputs a solution to $\bar{\mathcal{P}}$ of value at least $\sigma(m, \lfloor B\rfloor, \alpha, (1/4)y_*^{1/4,1})$.

By Lemma 4.2.3, the value of the solution output for $\hat{\mathcal{P}}$, is lower bounded by the sum $A \equiv \sum_{\lambda=2}^{\xi}\sigma(m, \lfloor B\rfloor, \alpha, (1/16)y_*^{\lambda})$. We distinguish two cases. If σ is a function linear in y_* then

$$A \geq \sigma(m, \lfloor B\rfloor, \alpha, (1/16)y_*^{(\beta\log m)^{-1},1/4}).$$

The sum A has $\Theta(\log\log m)$ terms, hence if σ is a function convex in y_*, A is minimized when all the terms y_*^{λ} are equal to $\Theta(y_*^{(\beta\log m)^{-1},1/4}/\log\log m)$. By Theorem 4.1.1 we

can instantiate $\sigma(y_*)$ in asymptotic form with a function $\Omega(\max\{y_*/m^{1/(\lfloor B\rfloor+1)}, y_*/\alpha^{1/\lfloor B\rfloor}\})$ in the linear case and with $\Omega(y_*(y_*/m)^{1/\lfloor B\rfloor})$ in the convex case. ■

4.2.3 Improved approximations for small column values

The approximation ratios achieved for column-restricted and general $(0,1)$-PIP's are admittedly large, but there is evidence they cannot be polynomially improved in the general case. Finding a maximum independent set in a graph $G = (V, E)$ can be formulated as a $(0,1)$ packing integer program $\mathcal{P}_G$ with $|E|$ constraints and $B = 1$. No ρ-approximation, $\rho = |V|^{1-\varepsilon}$, $\varepsilon > 0$, can be obtained in polynomial time for independent set unless $NP = ZPP$ [44]. Thus no $1/|E|^{1/2-\varepsilon}$ approximation can be achieved for $\mathcal{P}_G$, unless $NP = ZPP$.

Theorem 4.2.5 *There exists a family of column-restricted PIP's, for which no approximation better than* $1/m^{1/(\lfloor B\rfloor+1)-\varepsilon}$, $\varepsilon > 0$, *can be obtained in polynomial time, unless* $NP = ZPP$.

Theorem 4.2.2 demonstrates that when the maximum column value is bounded by $O(1/\log m)$, one can obtain a constant-factor approximation. This is in sharp contrast to the hardness result of Theorem 4.2.5. In this section, we generalize Theorem 4.2.2 by obtaining improved bounds for instances with column values bounded by a generic $\delta < 1$. We would like to point out that the approximation ratios in this section can be also obtained in a more direct fashion by using an observation of Srinivasan: the column values and B can be rescaled so that the $\min_i b_i$ becomes B/δ [94]. Rescaling does not affect feasibility of solutions. We choose to give a modified version of algorithm Column_Partition to demonstrate the versatility of the grouping-and-scaling technique. The approximation ratio we achieve will be a

ALGORITHM NEW_PARTITION ($\mathcal{P}$, ζ)

Step 1. Find the n-vector x_* that yields the optimal solution to the linear relaxation of $\mathcal{P}$.

Step 2a. Pick δ_2 such that $\delta_2 < 1 - \zeta$. Define a partition of the interval $[(\beta \log m)^{-1}, \delta]$
into $\xi = \lfloor \log_{(1/\delta_2)}(\delta\beta \log m) \rfloor + 1$, geometrically increasing subintervals $[(\beta \log m)^{-1}, \delta(\delta_2)^{\xi-1}], \ldots, (\delta(\delta_2)^{i+1}, \delta(\delta_2)^i], \ldots, (\delta\delta_2, \delta]$. For $\lambda = 1 \ldots \xi$ form subproblem $P^\lambda = (A^\lambda, b^\lambda, c^\lambda)$. A^λ and c^λ are the restrictions defined by $A^\lambda = A^{\delta(\delta_2)^\lambda, \delta(\delta_2)^{\lambda-1}}$ and $c^\lambda = c^{\delta(\delta_2)^\lambda, \delta(\delta_2)^{\lambda-1}}$.

Step 2b. Let d_i^λ be the ith entry of $(A^\lambda \cdot x_*^\lambda)$ and ζ be a constant in $(0, 1)$. Define $b_i^\lambda = (\zeta/\delta)B\delta(\delta_2)^{\lambda-1} + (1 - \frac{\zeta}{1-\delta_2})d_i^\lambda$ for $\lambda = 1, 2, \ldots, \xi$.

Step 3. On each P^λ, $1 \leq \lambda \leq \xi$, invoke α_PACKING to obtain a solution vector x^λ. Combine the solutions to subproblems 1 through ξ to form n-vector $\hat{x} = \cup_{1 \leq \lambda \leq \xi} x^\lambda$.

Step 4. Output vector $\hat{x}$.

Figure 4.2: Algorithm NEW_PARTITION.

function of δ and to the limit, i.e., when $\delta = O(1/\log m)$, will be constant. Based on Fact 4.2.1, we assume that the minimum column value is $(\beta \log m)^{-1}$. Therefore the interval of column values is $[(\beta \log m)^{-1}, \delta]$.

The algorithm we propose is a modified version of COLUMN_PARTITION, called NEW_PARTITION. We give the algorithm in Figure 4.2. It uses a generic partitioning scheme into geometrically increasing subintervals of ratio $\delta_2 \in (0, 1)$. (In COLUMN_PARTITION the corresponding ratio was 1/4.) Let ζ be a constant of our choice in $(0, 1)$ that will appear in the approximation ratio. The algorithm picks δ_2 based on ζ.

If $r_1 = \delta(\delta_2)^\lambda$ and $r_2 = \delta(\delta_2)^{\lambda-1}$, then we abbreviate $y_*^{r_1,r_2}$ as y_*^λ. The following lemma is the analogue of Lemma 4.2.3.

Lemma 4.2.4 *Algorithm* NEW_PARTITION *runs in polynomial time and the n-vector* $\hat{x}$ *it outputs is a feasible solution to* $\mathcal{P}$ *of value at least*

$$\sum_{\lambda=1}^{\lambda=\xi} \sigma(m, \lfloor(\zeta/\delta)B\rfloor, \alpha, \frac{\delta_2}{2}(1 - \frac{\zeta}{1-\delta_2})y_*^\lambda).$$

Proof. Note first that by our choice of δ_2 in Step 2a, the quantity $1 - \frac{\zeta}{1-\delta_2}$ is positive. The remainder of the proof is a generalization of the proof of Lemma 4.2.3. We outline the differences. By the definition of b^λ, the optimal solution to the linear relaxation of $\mathcal{P}_\lambda$ has value $\hat{y}^\lambda$ at least $(1 - \frac{\zeta}{1-\delta_2})y_*^\lambda$. The claim on the value follows by using Lemma 4.2.1 to estimate the approximation achieved for each subproblem.

For the feasibility, we observe that the aggregate capacity used by $\hat{x}$ on row i of A is the sum of the capacities used by x^λ, $1 \leq \lambda \leq \xi$, on each subproblem. This sum is by Step 2b at most

$$(1 - \frac{\zeta}{1-\delta_2}) \sum_{\lambda=1}^{\lambda=\xi} d_i^\lambda + (\zeta/\delta)B \sum_{\lambda=1}^{\lambda=\xi} \delta(\delta_2)^{\lambda-1}.$$

But

$$(\zeta/\delta)B \sum_{\lambda=1}^{\lambda=\xi} \delta(\delta_2)^{\lambda-1} < (\zeta/\delta)B\frac{\delta}{1-\delta_2} \leq b_i\frac{\zeta}{1-\delta_2}.$$

On the other hand

$$\sum_{\lambda=1}^{\lambda=\xi} d_i^\lambda \leq b_i.$$

Thus the aggregate capacity used by $\hat{x}$ is at most

$$\frac{\zeta}{1-\delta_2}b_i + (1 - \frac{\zeta}{1-\delta_2})b_i = b_i.$$

Therefore $\hat{x}$ is a feasible solution to $\mathcal{P}$. ■

The following theorem instantiates the achieved approximation.

Theorem 4.2.6 *Let $\mathcal{P} = (A, b, c)$ be a column-restricted PIP, in which the maximum column value is $\delta < 1$, and let y_* be the optimum of the linear relaxation of $\mathcal{P}$. Let ζ be a constant of our choice in $(0, 1)$. Algorithm* NEW_PARTITION *finds in polynomial time a solution $\hat{x}$ to $\mathcal{P}$ of value*

$$\Omega(\max\{\frac{y_*}{m^{1/(\lfloor(\zeta/\delta)B\rfloor+1)}},\ \frac{y_*}{\alpha^{1/\lfloor(\zeta/\delta)B\rfloor}},\ y_*\left(\frac{y_*}{m\log\log m}\right)^{1/\lfloor(\zeta/\delta)B\rfloor}\}).$$

Proof. Instantiate the function σ in the statement of Lemma 4.2.4 with the bounds from Theorem 4.1.1. The rest is similar to the proof of Theorem 4.2.4. ■

It is instructive to evaluate the bounds of Theorem 4.2.6 compared to the general case in Theorem 4.2.4. For example, consider a column-restricted PIP with $B = 1$, and maximum column value 1/10. The approximation ratio from Theorem 4.2.4 is $\Omega(1/(m^{1/2}))$. With appropriate choice of ζ, the approximation ratio from Theorem 4.2.6, is $\Omega(1/(m^{1/10}))$.

4.3 Applications of PIP's to approximation

4.3.1 Weighted multiple-source unsplittable flow

Our approach for weighted multiple-source unsplittable flow consists of finding first the optimum of the fractional relaxation, i.e. weighted multicommodity flow, which can be solved in polynomial time via linear programming. For the unweighted version of the problem, a fast combinatorial approximation algorithm can be used instead [72]. The relaxation allows commodity k to be shipped along more than one path. Call these paths the *fractional paths.* At this point, a simple rounding scheme for the fractional solution would be to pack as many of the fractional paths as possible, each carrying flow equal to some demand ρ_i. The problem with this naive approach is that

we may end up sending for some commodity k, $l\rho_k$ units of flow along $l > 1$ distinct paths. Therefore we round in two stages. In the first stage we select at most one of the fractional paths for each commodity, at the expense of congestion, i.e. some capacities may be violated. In addition, some commodities may not be routed at all. The first stage is implemented by resorting to a well-known result, given in Theorem 4.3.1 below. In the second stage, among the commodities routed during the first stage, we select those that will ultimately be routed while respecting the capacity constraints. It is in this last stage that a column-restricted PIP is used.

A well-known analogue of Theorem 4.2.2, follows using standard randomized rounding [86, 85] and the Chernoff bound of Theorem 4.2.1. The proof is quite similar in spirit to that of Theorem 4.2.2. See for example [60] for a randomized algorithm.

Theorem 4.3.1 *Let $\Pi = (G = (V, E), \mathcal{T})$ be a weighted multiple-source unsplittable flow problem such that the maximum demand value $\rho_{\max}$ is at most $1/(\beta \log |E|)$ times the minimum capacity, for some constant $\beta \geq 12$. Let $\alpha^f(\mathcal{T})$ be the value of a fractional routing f. Then one can obtain in deterministic polynomial time a feasible solution to Π of value at least $\gamma\alpha^f(\mathcal{T})$, for a constant $0 < \gamma < 1$.*

In light of Theorem 4.3.1, we will focus solely on instances with minimum demand value $1/(\beta \log |E|)$; the rationale behind this decision is the same as the one outlined for column-restricted PIP's in Fact 4.2.1. Before giving the algorithm, we introduce some terminology, which generalizes the definitions from Section 3.2 to the multisource setting. We drop the characterization of a routing or flow as partial from the definitions, since in the metric we consider partial unsplittable flows are feasible solutions. A *routing* is a set of s_{k_i}-t_{k_i} paths P_{k_i}, used to route ρ_{k_i} amount of flow

from s_{k_i} to t_{k_i} for each $(s_{k_i}, t_{k_i}) \in I \subseteq \mathcal{T}$. Given a routing g, the flow g_e through edge e is equal to $\sum_{P_i \in g, P_i \ni e} \rho_i$. A routing g for which $g_e \leq u_e$ for every edge e is an *unsplittable flow.* A *fractional routing* is a routing, in which for commodity k (i) the flow is split along potentially many paths (ii) demand $f_k \leq \rho_k$ is routed. A fractional routing corresponds thus to standard multicommodity flow. A *fractional single-path routing* is a routing in which the flow for a commodity is shipped on one path if at all, but only a fraction $f_k \leq \rho_k$ of the demand is shipped for commodity k. The *value* of a routing g (fractional or otherwise) is the weighted sum of the flow routed routed in g for each commodity. Recall that by the definition of the problem, the maximum demand and the minimum capacity values are respectively at most and at least 1. The full algorithm is given in Figure 4.3. Algorithm MAX_ROUTING takes as input a weighted multiple-source unsplittable flow instance $(G, \mathcal{T})$ and outputs an unsplittable flow g''.

Theorem 4.3.2 *Let $(G = (V, E), \mathcal{T})$ be a weighted multiple-source unsplittable flow problem, and $\alpha^f(\mathcal{T})$ be the value of an optimum fractional routing f. Algorithm* MAX_ROUTING *finds in polynomial time an unsplittable flow of value*

$$\Omega(\max\{\frac{\alpha^f(\mathcal{T})}{\log|E|\sqrt{|E|}}, \frac{(\alpha^f(\mathcal{T}))^2}{|E|\log^2|E|\log\log|E|}\}).$$

Proof. First note that since routing g' is feasible for G', after scaling down the flow at Step 3, routing g is feasible for G. The approximate solution to $\mathcal{P}$ maintains feasibility.

We now examine the claim of the theorem on the value of $\hat{g}$. By Theorem 4.3.1, the value of g' is a constant fraction of $\alpha^f(\mathcal{T})$. Set z_{k_j} to be equal to the amount of flow that is routed in g for commodity k_j divided by ρ_{k_j}. The ν-vector z is a feasible fractional solution to $\mathcal{P}$ of value $\Omega(\alpha^f(\mathcal{T})/\log|E|)$. The theorem follows by using

ALGORITHM MAX_ROUTING$(G = (V, E, u), \mathcal{T})$

Step 1. Find an optimum fractional routing f by invoking a weighted multicommodity flow algorithm. Denote by $\alpha^f(\mathcal{T})$ the value of f.

Step 2. Scale up all capacities by a factor of $\beta \log |E|$ to obtain network G' with capacity function u'. Invoke the algorithm from Theorem 4.3.1 on G' to round f to a routing g'.

Step 3. Scale down the flow on every path of g' by a factor of at most $\beta \log |E|$ to obtain a fractional single-path routing g, which is feasible for G.

Step 4. Construct a column-restricted PIP $\mathcal{P} = (A, b, c)$ as follows. Let $k_1, k_2, \ldots, k_\nu$ be the set of commodities shipped in g, $\nu \leq K$. A has ν columns, one for each commodity in g, and $|E|$ rows, one for each edge of G. $A_{ij} = \rho_{k_j}$ if the path P_{k_j} in g for commodity k_j goes through edge e_i and 0 otherwise. The cost vector c has entry c_j set to $w_{k_j}\rho_{k_j}$ for each commodity k_j shipped in g. Finally, b_i is set to u_{e_i}, the capacity of edge e_i, $1 \leq i \leq |E|$.

Step 5. Invoke algorithm COLUMN_PARTITION to find an integral solution $\hat{g}$ to $\mathcal{P}$. Construct an unsplittable flow g'' by routing commodity k_j on path P_{k_j} if and only if $\hat{g}_j = 1$.

Figure 4.3: Algorithm MAX_ROUTING.

Theorem 4.2.4 to lower bound the value of the integral solution $\hat{g}$ to $\mathcal{P}$. ■

We now give a bound that depends on the dilation d of the fractional routing f.

Theorem 4.3.3 *Let $(G = (V, E), \mathcal{T})$ be a weighted multiple-source unsplittable flow problem, and $\alpha^f(\mathcal{T})$ be the value of an optimum fractional routing f with dilation d. There is a polynomial-time algorithm, which finds an unsplittable flow of value $\Omega(\alpha^f(\mathcal{T})/d)$.*

Proof. The algorithm is similar to MAX_ROUTING but omits Steps 2 and 3 that find the single-path fractional routing. Step 4 is modified as follows. Let $1, \ldots, M$ be an arbitrary ordering of the paths in the flow decomposition of f. Matrix A of $\mathcal{P}$ has $M + |\mathcal{T}|$ total columns, one for each path in the decomposition and an additional $|\mathcal{T}|$ special columns. The number of rows is $|E| + |\mathcal{T}|$. The first $|E|$ rows correspond to capacity constraints in G and their entries are filled in the manner described in Algorithm MAX_ROUTING. The remaining rows correspond to constraints that enforce that only one path is chosen per commodity in the final integral solution. In particular row $|E| + i$, $1 \leq i \leq |\mathcal{T}|$, has ρ_i in column j, $1 \leq j \leq M$, if the j-th path in the flow decomposition carries flow for commodity i; otherwise the entry is zero. The entries of column $M + l$, $1 \leq l \leq |\mathcal{T}|$, are all zero except for the $(|E| + l)$-th row where the entry is equal to $1 - \rho_l$. Finally, each entry $b_{|E|+i}$, $1 \leq i \leq |\mathcal{T}|$, of the b-vector is set to 1. We set c_i, $1 \leq i \leq M$, to the weight of the commodity that flows along the i-th path in the decomposition. The entries in the c-vector past the M-th position are set to 0.

It is straightforward to verify that the above construction enforces an integral solution where at most one of the fractional paths is chosen for each commodity. The resulting PIP is column-restricted with fractional optimum y_* and maximum number of non-zero entries in a column equal to $d + 1$. The theorem follows. ■

The construction in the proof of Theorem 4.3.3 can also be used to obtain improved bounds for the case, in which the number of commodities $|\mathcal{T}| = O(|E|)$. More generally, the approximation bounds can be expressed as a function of $|\mathcal{T}|$:

Corollary 4.3.1 *Let $(G = (V, E), \mathcal{T})$ be a weighted multiple-source unsplittable flow problem, and $\alpha^f(\mathcal{T})$ be the value of an optimum fractional routing f. Algorithm* Max_Routing *finds in polynomial time an unsplittable flow of value*

$$\Omega(\max\{\frac{\alpha^f(\mathcal{T})}{\sqrt{|E|+|\mathcal{T}|}},\ \frac{(\alpha^f(\mathcal{T}))^2}{(|E|+|\mathcal{T}|)\log\log|E|}\}).$$

If the maximum demand is bounded by $\delta < 1$, at Step 2 of Max_Routing it is enough to scale by $\delta\beta\log|E|$. Using algorithm New_Partition instead of Column_Partition at Step 5 of either Max_Routing or the algorithm of Theorem 4.3.3, we obtain the generalization of Theorem 4.3.1.

Theorem 4.3.4 *Let $(G = (V, E), \mathcal{T})$ be a weighted multiple-source unsplittable flow problem, and $\alpha^f(\mathcal{T})$ be the value of an optimum fractional routing f of dilation d. Let $\delta < 1$, be the value of the maximum demand and $\zeta \in (0, 1)$ be any constant of our choice. There is a polynomial-time algorithm, which finds an unsplittable flow of value*

$$\Omega(\max\{\frac{\alpha^f(\mathcal{T})}{\delta\log|E|(|E|)^{1/(\lfloor\zeta/\delta\rfloor+1)}},\ \frac{\alpha^f(\mathcal{T})}{\delta\log|E|}\left(\frac{\alpha^f(\mathcal{T})}{(|E|\delta\log|E|\log\log|E|)}\right)^{1/\lfloor(\zeta/\delta)\rfloor},\ \frac{\alpha^f(\mathcal{T})}{d^{1/(\lfloor\zeta/\delta\rfloor)}}\}).$$

4.3.2 Weighted vertex-disjoint paths

A relaxation of the weighted vertex-disjoint path problem is integral multicommodity flow, in which every commodity has an associated demand of 1. We can express vertex-disjoint paths as an integer program $\mathcal{I}$ that has the same constraints as multicommodity flow together with additional "bandwidth" constraints on the vertices.

Let f_{ij}^k denote in $\mathcal{I}$ the flow through edge $(i,j) \in E$ corresponding to the k-th commodity, i.e. to the terminal pair (s_k, t_k).

$$\text{maximize} \sum_{k=1}^{K} w_k \sum_{(i,t_k)\in E} f_{it_k}^k$$

$$\text{subject to} \quad \sum_{(j,i)\in E} f_{ji}^k - \sum_{(i,j)\in E} f_{ij}^k \quad \begin{cases} \leq 1 & \text{if } i = t_k \\ \geq -1 & \text{if } i = s_k \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in V, 1 \leq k \leq K$$

$$\sum_{k=1}^{K} f_{ij}^k \leq 1 \qquad \forall (i,j) \in E$$

$$\sum_{k=1}^{K} \sum_{(j,i)\in E} f_{ji}^k \leq 1 \qquad \forall i \in V$$

$$\sum_{k=1}^{K} \sum_{(i,j)\in E} f_{ij}^k \leq 1 \qquad \forall i \in V$$

Let $\mathcal{LP}$ be the linear relaxation of $\mathcal{I}$. The optimal solution f to $\mathcal{LP}$ consists of a set of *fractional flow paths*. Our algorithm relies on the observation that f gives rise (and at the same time is a fractional solution) to a PIP. The particular PIP models a 1-matching problem on a hypergraph H with vertex set V and a hyperedge (subset of V) for every path in the fractional solution. In other words, the paths in f may be viewed as sets of vertices *without any regard for the flow through the edges of* G. We give the full algorithm in Figure 4.4.

Theorem 4.3.5 *Let $(G = (V,E), \mathcal{T})$ be a weighted vertex-disjoint paths problem, and and $\alpha^f(\mathcal{T})$ be the value of an optimum solution to the linear relaxation $\mathcal{LP}$. Let d be the dilation of the solution. Algorithm* Path_Packing *finds in polynomial time a solution of value*

$$\Omega(\max\{\frac{\alpha^f(\mathcal{T})}{\sqrt{|V|}}, \frac{(\alpha^f(\mathcal{T}))^2}{|V|}, \frac{\alpha^f(\mathcal{T})}{d}\}).$$

ALGORITHM PATH_PACKING($G = (V, E), \mathcal{T}$)

Step 1. Formulate the linear relaxation $\mathcal{LP}$ and find an optimal solution f to it. Using flow decomposition express f as a set of paths $P_1, P_2, \ldots, P_\nu$, each connecting a pair of terminals and carrying $z_i \leq 1$, $1 \leq i \leq \nu$, units of flow.

Step 2. Construct a $(0,1)$-PIP $\mathcal{P} = (A, b, c)$ as follows. A is a $|V| \times \nu$ matrix; A_{ij} is 1 if path P_j includes vertex i and 0 otherwise. b is a vector of ones. c_j is equal to w_k such that path P_j connects terminals s_k and t_k.

Step 3. Find an integral solution g to $\mathcal{P}$ by using the algorithms of Theorem 4.1.1 and output the corresponding set of paths $P(g)$.

Figure 4.4: Algorithm PATH_PACKING.

Proof. We show first that $P(g)$ is a feasible solution to the problem. Clearly the constraints of the packing program $\mathcal{P}$, constructed at Step 2 ensure that the paths in $P(g)$ are vertex disjoint. This excludes the possibility that more than one (s_k, t_k)-path is present in $P(g)$ for some k. The optimal value of the linear relaxation of $\mathcal{P}$ is at least y_* since setting x_j equal to z_j, $1 \leq j \leq \nu$, yields a feasible fractional solution to $\mathcal{P}$. The claim on the value of g follows from the bounds in Theorem 4.1.1. ■

4.3.3 An Application to Independent Set

In this section we show how a packing formulation leads to an $O(\sqrt{|V|})$ approximation for the following problem: find a maximum weight independent set in the square of the graph $G = (V, E)$. More generally the result applies to the even powers of G. We subsequently give an inapproximability result and an $O(\sqrt{|E|})$ approximation for odd powers of G.

Theorem 4.3.6 *Given an undirected graph $G = (V, E)$ and $c \in [0,1]^{|V|}$ a weight vector on the vertices, there exists a polynomial-time algorithm, which outputs an independent set in the $G^{2k} = (V, E^{2k})$, $k \in Z_{\geq 1}$, of weight*

$$\Omega(\max\{y_*/\sqrt{|V|}, (y_*)^2/|V|, y_*/\Delta^k\}),$$

where y_* denotes the optimum of a fractional relaxation and Δ is the maximum vertex degree in G.

Proof. Assume an ordering $v_1, \ldots, v_{|V|}$ on the vertices in V. We form a $(0,1)$-PIP $\mathcal{P} = (A, b, c)$ as follows. A is a $|V| \times |V|$ matrix. Row i corresponds to the neighborhood of v_i of radius k. $A_{ij} = 1$ if and only if $j = i$ or v_j is at distance at most k from v_i. All the b-entries are set to 1. Clearly, $\mathcal{P}$ can be constructed in polynomial time.

We now show that any feasible integral solution $\bar{x}$ to $\mathcal{P}$ is an independent set in G^{2k}. Let $i \neq j$ be such that $\bar{x}_i = \bar{x}_j = 1$. Assume for purpose of contradiction that edge (v_i, v_j) belongs to E^{2k} either $(v_i, v_j) \in E$ or there is a vertex v_k such that v_i and v_j are each at distance k or less from v_k in G. In the first case the i-th constraint of $\mathcal{P}$ is violated; in the second case the k-th constraint of $\mathcal{P}$ is violated, thereby yielding a contradiction. Conversely, consider an independent set I of G^{2k}. We will show that its characteristic vector x_I is a feasible solution of $\mathcal{P}$. Assume this is not the case and that the inner product of x_I with the i-th row of A exceeds 1. Then there exist two vertices v_j, v_l (one of them possibly identical to v_i) such that both v_j, v_l belong to I and both are at a distance at most k from v_i in G. But then v_j and v_l are at a distance at most $2k$ in G^{2k} and edge (v_k, v_l) must belong to E^{2k} by the definition of G^{2k}, a contradiction.

Let y_* be the optimum of the linear relaxation of $\mathcal{P}$. Notice that the maximum number of non-zero entries in a column is at most $\Delta^k + 1$. The bounds of the theorem

follow by Theorem 4.1.1. ■

A hardness of approximation result for the problem of finding a maximum independent set in the k-th power of a graph follows. This inapproximability result demonstrates that the approximation achieved in Theorem 4.3.6 is likely to be tight up to subpolynomial improvements.

Theorem 4.3.7 *For the problem of finding a maximum independent set in the k-th power $G^k = (V, E^k)$ of a graph $G = (V, E)$ for any fixed integer $k > 0$, there is no ρ-approximation with $\rho = (\frac{|V|}{k+2})^{1/2-\varepsilon}$, for any fixed $\varepsilon > 0$, unless $NP = ZPP$.*

Proof. Lin and Skiena [75] give an approximation-preserving reduction from finding an independent set in an arbitrary graph $G_0 = (V_0, E_0)$ to finding an independent set in the k-th power $(G')^k$ of a graph $G' = (V', E')$ such that $|V'| \leq (k+2)|E_0|$. By Håstad's result [44] there is no $|V_0|^{1-\varepsilon}$-approximation algorithm, $\varepsilon > 0$, for maximum independent set in G_0 unless $NP = ZPP$. This implies that no $|E_0|^{1/2-\varepsilon/2}$-approximation algorithm exists unless $NP = ZPP$.

Therefore no $(|V'|/(k+2))^{1/2-\varepsilon/2}$ approximation exists for maximum independent set in $(G')^k$ under the same complexity-theoretic assumption. ■

For odd powers of G, we can prove a weaker $O(\sqrt{|E|})$ approximation. By Håstad's result [44], it is unlikely that an $O(|E|^{1/2-\varepsilon})$, $\varepsilon > 0$, approximation can be achieved for $G^1 = G$. From this point of view, the $O(\sqrt{|E|})$ bound is tight up to subpolynomial factors. Note that one can achieve naively an $O(\sqrt{|E^k|})$ approximation for independent set in $G^k = (V, E^k)$ by using the reduction to a PIP described in Section 4.2.3. Depending on the actual graph G it is possible that $|E| = o(|E^k|)$, $k \geq 2$; therefore the following theorem existentially improves upon the naive bound.

Theorem 4.3.8 *Given an undirected graph $G = (V, E)$ and $c \in [0,1]^{|V|}$ a weight vector on the vertices, there exists a polynomial-time algorithm, which outputs an*

independent set in the $G^{2k+1} = (V, E^{2k+1})$, $k \in Z_{\geq 0}$, *of weight*

$$\Omega(\max\{y_*/\sqrt{|E|}, (y_*)^2/|E|, y_*/\Delta^{k+1}\}),$$

where y_* *denotes the optimum of a fractional relaxation and* Δ *is the maximum vertex degree in* G.

Proof. The proof is similar to the one of Theorem 4.3.6 therefore we only sketch the differences. The constructed $(0, 1)$-PIP $\mathcal{P}$ has $|E|$ constraints, one for each edge $(v, w) \in E$. The row for edge (v, w) in the matrix has a 1 in the j-th column if and only if vertex $v_j \in V$ is at a distance k or less from either of the two. Two vertices x and y are at distance 0 if and only if they are the same vertex. Observe that the maximum number of entries in a column j is now equal to the maximum number of edges one of whose endpoints is at distance at most k from v_j. Therefore the maximum number of entries in a column is at most Δ^{k+1}. ■

4.4 Greedy Algorithms for Disjoint Paths

In this section we turn to the routing approach for the unweighted edge- and vertex-disjoint path problems. Our algorithm is motivated by the blocking flow method for the s-t maximum flow problem ([22], see also [27]). The latter method proceeds by iterations on the current residual graph. On every iteration a flow is found, which greedily saturates at least one edge on (i.e. blocks) all shortest paths from the source to the sink. Our initial motivation came from the following question: what amount of the residual flow has survived if one is not allowed to ever reroute the blocking flow found at a given iteration? GREEDY_PATH, given in Figure 4.5 does not use rerouting. The underlying idea is that if one keeps routing commodities along sufficiently short

ALGORITHM GREEDY_PATH$(G, \mathcal{T})$

Step 1. Set $\mathcal{A}$ to $\emptyset$.

Step 2. Let (s_*, t_*) be the commodity in $\mathcal{T}$ such that the shortest path P_* in G from s_* to t_* has minimum length. If no such path exists, halt and output $\mathcal{A}$.

Step 3. Add P_* to $\mathcal{A}$ and remove the edges of P_* from G. Remove (s_*, t_*) from $\mathcal{T}$. Goto Step 2.

Figure 4.5: Algorithm GREEDY_PATH.

paths the final number of commodities routed is lowerbounded with respect to the optimum.

We begin by lowerbounding the approximation ratio of the algorithm. The proof will provide the intuition behind GREEDY_PATH. We say that a path P_1 *hits* a path P_2, if P_1 and P_2 share an edge.

Theorem 4.4.1 *Algorithm* GREEDY_PATH *runs in polynomial time and outputs a solution to an edge-disjoint path problem* $(G = (V, E), \mathcal{T})$ *of size at least* $1/(\sqrt{|E_o|}+1)$ *times the optimum, where* $E_o \subseteq E$ *is the set of edges used by the paths in an optimal solution.*

Proof. Let t be the total number of iterations of GREEDY_PATH and $\mathcal{A}_i$ be the set $\mathcal{A}$, G_i the updated graph and $\mathcal{T}_i$ the current set $\mathcal{T}$ at the end of the i-th iteration. Let $\mathcal{O}$ be an optimal set of paths. We define the set $\mathcal{O} \ominus \mathcal{A}$ as the paths in $\mathcal{O}$ that correspond to commodities not routed in $\mathcal{A}$. We claim that after every iteration i the path P_i in $\mathcal{A}_i - \mathcal{A}_{i-1}$ hits at most $\sqrt{|E_o|}$ paths in $\mathcal{O} \ominus \mathcal{A}_i$ that are not hit by a path in $\mathcal{A}_{i-1}$. Then, since upon termination of GREEDY_PATH all paths in $\mathcal{O} \ominus \mathcal{A}_t$ must

hit some path in $\mathcal{A}_t$ we have that

$$|\mathcal{O} \ominus \mathcal{A}_t| \leq \sqrt{|E_o|}|\mathcal{A}_t|.$$

But $|\mathcal{O}| - |\mathcal{A}_t| \leq |\mathcal{O} \ominus \mathcal{A}_t|$ and the theorem follows.

We prove our claim by induction on the number of iterations. After the first iteration assume that the path P_1 in $\mathcal{A}_1$ hits k paths in $\mathcal{O} \ominus \mathcal{A}_1$. Note that an edge of P_1 can be used to hit only at most one other path in $\mathcal{O}$ so P_1 has length at least k. Since P_1 is the shortest path among all paths between terminal pairs of $\mathcal{T}$ in G, all k paths in $\mathcal{O} \ominus \mathcal{A}_1$ have length at least k. But $k^2 \leq |E_o|$, so $k \leq \sqrt{|E_o|}$.

For the inductive step assume the claim holds for $i \leq j$. Let P_{j+1} hit k paths in $\mathcal{O} \ominus \mathcal{A}_{j+1}$ that are not hit by a path in $\mathcal{A}_j$. The length of P_{j+1} is at least k. Assume that a path P' of these k paths hit by P_{j+1} has length less than k. Then the shortest path between the endpoints of P' in G_j has length less than k since no edge belonging to P' has been deleted so far. Moreover, the endpoints of P' are in $\mathcal{T}_j$ since P' belongs to $\mathcal{O} \ominus \mathcal{A}_{j+1}$. Thus P' would have been selected instead of P_{j+1} during the $(j+1)$-th iteration, a contradiction. We conclude that all k paths in $\mathcal{O} \ominus \mathcal{A}_{j+1}$ hit by P_{j+1} have length at least k and since $k^2 \leq |E_o|$ the inductive step is complete. ■

Algorithm Greedy_Path gives a solution to vertex-disjoint paths with the following modification at Step 3: remove the vertices of P_* from G. Call the resulting algorithm Greedy_VPath. The analysis above can be easily modified to yield the following.

Theorem 4.4.2 *Algorithm* Greedy_VPath *runs in polynomial time and outputs a solution to a vertex-disjoint paths problem* $(G = (V, E), \mathcal{T})$ *of size at least* $1/(\sqrt{|V_o|}+1)$ *times the optimum, where* $V_o \subseteq V$ *is the set of vertices used by the paths in an optimal solution.*

We now show improved bounds on the size of the realizable set output by GREEDY_PATH. Recall that OPT denotes the optimal size of a realizable set.

Theorem 4.4.3 *Algorithm* GREEDY_PATH *outputs a solution to an edge-disjoint path problem* $(G = (V, E), \mathcal{T})$ *of size* $\Omega(OPT^2/|E_o|)$, *where* $E_o \subseteq E$ *is the set of edges used by the paths in an optimal solution.*

Proof. We maintain the notation from the proof of Theorem 4.4.1. Let k_i, $1 \leq i \leq t$ be the number of paths in $\mathcal{O} \ominus \mathcal{A}_i$ that are not hit by a path in $\mathcal{A}_{i-1}$ and are hit by the path P_i in $\mathcal{A}_i - \mathcal{A}_{i-1}$. It was established in the proof of Theorem 4.4.1 that all these k_i paths have length at least k_i. Furthermore all paths in $\mathcal{O}$ are edge-disjoint with total number of edges $|E_o|$. Therefore

$$\sum_{i=1}^{t} k_i^2 \leq |E_o|.$$

Applying the Cauchy-Schwartz inequality on the left-hand side we obtain that

$$(\sum_{i=1}^{t} k_i)^2/t \leq |E_o|.$$

But $\sum_{i=1}^{t} k_i = |\mathcal{O} \ominus \mathcal{A}_t|$, thus we obtain

$$\frac{|\mathcal{O} \ominus \mathcal{A}_t|^2}{t} \leq |E_o|.$$

Wlog we can assume that $|\mathcal{A}_t| = o(|\mathcal{O}|)$, since otherwise GREEDY_PATH obtains a constant-factor approximation. It follows that $t = \Omega(|\mathcal{O}|^2)/|E_o|) = \Omega(OPT^2/|E|)$. ■

Similarly we obtain the analogous result for GREEDY_VPATH.

Theorem 4.4.4 *Algorithm* GREEDY_VPATH *outputs a solution to a vertex-disjoint path problem* $(G = (V, E), \mathcal{T})$ *of size* $\Omega(OPT^2/|V_o|)$, *where* $V_o \subseteq V$ *is the set of vertices used by the paths in an optimal solution.*

It is straightforward to obtain the ensuing corollaries from Theorems 4.4.3 and 4.4.4.

Corollary 4.4.1 *Algorithm* GREEDY_PATH *outputs a solution to an edge-disjoint path problem* $(G = (V, E), \mathcal{T})$ *of size* $\Omega(OPT/d_o)$, *where* d_o *is the average length of the paths in an optimal solution.*

Corollary 4.4.2 *Algorithm* GREEDY_VPATH *outputs a solution to a vertex-disjoint path problem* $(G = (V, E), \mathcal{T})$ *of size* $\Omega(OPT/d_o)$, *where* d_o *is the average length of the paths in an optimal solution.*

Chapter 5

Conclusions

Manacled to the city, manacled to the city
Hit the North
Those big big big wide streets
Those useless MPs
Just savages, just savages
Hit the North
Manacled to the city
All estate agents alive yell down the night
in hysterical breath
And from the back of the third eye psyche
the inducement come forth
Hit the North

– *The Fall,* Hit the North

In this chapter we discuss open problems and possible extensions to the work in this thesis.

In Chapter 2 we exhibited the first strongly polynomial algorithm in over 30 years that achieves $o(n^3)$ running time for SSSP with real-valued costs, although not for the worst case. The complexity gap between $O(nm)$ time for for dense graphs and $O(n^2 \log n)$ time is considerable. This raises the natural question of whether an $o(n^3)$ worst-case time algorithm for SSSP exists. If the answer is no, can one reason about a lower bound in a nonoblivious computational model? Another persistent open

question is whether one can decouple negative cycle detection from shortest path computation and achieve an $o(n^3)$ worst-case time bound for the former problem.

For single-source unsplittable flow, a number of improvements on our results were recently obtained by Dinitz, Garg and Goemans [23]. They obtain a $1 + \rho_{\max} \leq 2$ approximation for congestion. Recall that $\rho_{\max}$ denotes the maximum demand value. This result matches the approximation ratio obtained by Lenstra et al. [74] and Shmoys and Tardos [91] for makespan minimization on unrelated machines. As explained in Section 3.1, a special case, denoted by $\mathcal{S}$, of this scheduling problem reduces to single-source unsplittable flow. The result of Shmoys and Tardos [91] is in fact a simultaneous $(1, 1 + \rho_{\max})$ approximation of cost and congestion for the generalized assignment problem (see Section 3.1). It is interesting to examine whether the ideas of Dinitz, Garg and Goemans can yield the same bound for minimum-cost single-source unsplittable flow.

The approximation ratio of $1 + \rho_{\max}$ matches known integrality gaps both for minimum-congestion unsplittable flow and the scheduling problem $\mathcal{S}$. On the other hand, the known inapproximability result states that it in NP-hard to achieve a ratio less than 3/2 [74]. A straightforward extension of the result in [74] yields actually a $1 + \rho_{\min}$ hardness bound for unsplittable flow (see also [23]). It is an important open question whether 3/2 is possible either for $\mathcal{S}$ or more generally for single-source unsplittable flow. If this is the case, a new more powerful relaxation than maximum flow is possibly required. Dinitz, Garg and Goemans [23] improve on the other metrics as well, and show approximation ratios of at least 0.226 and 5 for maximum demand and minimum number of rounds; they prove corresponding integrality gaps of 0.3845 and 3 respectively.

For $(0, 1)$ packing integer programs with $B = 1$, the inapproximability result for

independent set leaves little hope for substantial improvements to the known approximation ratio (cf. Theorem 4.2.5). It is open whether an approximation ratio $1/m^{1/(\lfloor B \rfloor+1)-\varepsilon}$, $\varepsilon > 0$, can be achieved for general B. In recent work, Chekuri and Khanna showed that no $1/\alpha^{1/(B+1)-\varepsilon}$, $\varepsilon > 0$, can be achieved unless $NP = ZPP$ [13]. It is interesting to investigate further whether the $\Omega(1/m^{1/(\lfloor B \rfloor+1)})$ ratio extends beyond the column-restricted class to general PIP's. The grouping-and-scaling technique does not seem to apply to the general setting. In the multiple-source unsplittable flow domain, it would be interesting to extend the greedy algorithm GREEDY_PATH in any of the following two directions: (i) obtain the approximation with respect to the fractional optimum $\alpha^f(\mathcal{T})$ (ii) modify the greedy algorithm to handle demands and capacities. The work of Young on oblivious rounding [102] may provide a starting point for establishing a connection between the rounding and routing approaches. Finally, an important open question, is whether an $\Omega(1/|V|^{\varepsilon})$, $\varepsilon < 1$, approximation ratio is possible for edge-disjoint paths on general graphs. The work in [96] and in this thesis demonstrates that this is possible if one of the following conditions holds:

(i) the optimal solution uses $O(n^{2-\delta})$, $\delta > 0$, edges.

(ii) a fractional solution with dilation $O(n^{1-\delta})$, $\delta > 0$ can be efficiently computed.

(iii) the average length of a path in an optimal solution is $O(n^{1-\delta})$, $\delta > 0$.

An $\Omega(1/|V|^{\varepsilon})$, $\varepsilon < 1$, approximation would be of great interest, even in the case where the input is a *permutation,* i.e., the terminal pairs are vertex-disjoint. Work in progress by the author shows that in the latter case the following condition is also sufficient:

(iv) the subgraph induced by an optimal solution is acyclic.

The idea behind Condition (iv) is as follows. Let P be the set of paths in an optimal solution. Since the input is a permutation, $|P| < |V|$. Connect the sources and the sinks of the paths in P to an auxiliary source s' and sink t' respectively. Call G_P the induced graph. P can be viewed as a flow solution to a unit-capacity s'-t' maximum flow problem in G_P. By a theorem of Karger and Levine [51], if the graph induced by P is acyclic, the total length of the paths in P is $O(|V|\sqrt{|P|}) = O(|V|^{3/2})$. Therefore Condition (i) is satisfied.

Despite the evidence provided by Conditions (i)-(iv), it is possible that the edge-disjoint path problem proves to be as hard as maximum independent set. Even so, an inapproximability result would be of considerable theoretical interest.

Bibliography

[1] R. Aharoni, P. Erdős, and N. Linial. Optima of dual integer linear programs. *Combinatorica*, 8:13–20, 1988.

[2] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. Finding Minimum-Cost Flows by Double Scaling. *Mathematical Programming*, 53:243–266, 1992.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.

[4] N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 569–575, 1991.

[5] Y. Aumann and Y. Rabani. Improved bounds for all-optical routing. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 567–576, 1995.

[6] A. Baveja and A. Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. Submitted for publication, 1998.

[7] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[8] P. A. Bloniarz. A shortest path algorithm with expected time $O(n^2 \log n \log^* n)$. *SIAM Journal on Computing*, 12:588–600, 1983.

[9] P. A. Bloniarz, A. Meyer, and M. Fischer. Some observations on Spira's shortest path algorithm. Technical Report 79-6, Department of Computer Science, State University of New York at Albany, August 1979.

[10] A. Z. Broder, A. M. Frieze, and E. Upfal. Static and dynamic path selection on expander graphs: a random walk approach. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 531–539, 1997.

[11] R. G. Busaker and P. J. Gowen. A procedure for determining minimal-cost flow network patterns. Technical Report ORO-15, Operational Research Office, Johns Hopkins University, Baltimore, MD, 1961.

[12] J. S. Carson and A. M. Law. A note on Spira's algorithm for the all-pairs shortest path problem. *SIAM Journal on Computing*, 6:696–699, 1977.

[13] C. Chekuri and S. Khanna. On multi-dimensional packing problems. Unpublished manuscript, 1998.

[14] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest path algorithms: theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1996.

[15] H. Chernoff. A measure of the asymptotic efficiency for tests of a hypothesis based on sum of observations. *Ann. Math. Stat.*, 23:493–509, 1952.

[16] C. Cooper. The thresold of Hamilton cycles in the square of a random graph. *Random Structures and Algorithms*, 5:25–31, 1994.

[17] C. Cooper, A. Frieze, K. Mehlhorn, and V. Priebe. Average-case complexity of shortest-paths problems in the vertex-potential model. In *Randomization and Approximation Techniques in Computer Science*, volume 1269 of *Lecture Notes in Computer Science*, pages 15–26. Springer, 1997.

[18] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[19] G. B. Dantzig. On the shortest route through a network. *Management Science*, 6:197–190, 1960.

[20] N. Deo and C. Pang. Shortest-path algorithms: taxonomy and annotation. *Networks*, 14:275–323, 1984.

[21] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:260–271, 1959.

[22] E. A. Dinitz. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

[23] Y. Dinitz, N. Garg, and M. Goemans. On the single-source unsplittable flow problem. To appear in Proceedings of 39th Symposium on Foundations of Computer Science, 1998.

[24] P. Elias, A. Feinstein, and C. E. Shannon. Note on maximum flow through a network. *IRE Transactions on Information Theory IT-2*, pages 117–199, 1956.

[25] P. Erdős and J. L. Selfridge. On a combinatorial game. *Journal of Combinatorial Theory A*, 14:298–301, 1973.

[26] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5:691–703, 1976.

[27] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4:507–518, 1975.

[28] H. Fleischner. The square of every two-connected graph is Hamiltonian. *Journal of Combinatorial Theory B*, 16:29–34, 1974.

[29] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.

[30] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[31] A. Frank. Packing paths, cuts and circuits - a survey. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows and VLSI-Layout*, pages 49–100. Springer-Verlag, Berlin, 1990.

[32] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.

[33] A. M. Frieze and G. R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10:57–77, 1985.

[34] H. Gabow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31:148–168, 1985.

[35] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

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

[37] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997.

[38] A. V. Goldberg. Scaling algorithms for the shortest path problem. *SIAM Journal on Computing*, 24:494–504, 1995.

[39] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

[40] A. V. Goldberg and R. E. Tarjan. Solving minimum-cost flow problems by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[41] A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 2–11, 1997.

[42] D. Goldfarb, J. Hao, and S. R. Kai. A computational comparison of label correcting and simplex algorithms for computing shortest paths. Technical report, IEOR Department, Columbia University, New York, 1986.

[43] R. Hassin and E. Zemel. On shortest paths in graphs with random weights. *Mathematics of Operations Research*, 10(4):557–564, November 1985.

[44] J. Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 627–636, 1996. Updated version appears in Electronic Colloquium on Computational Complexity.

[45] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[46] M. Iri. A new method of solving transportation-network problems. *Journal of the Operations Research Society of Japan*, 3:27–87, 1960.

[47] W. S. Jewell. Optimal flow through networks. Technical Report 8, Operations Research Center, MIT, Cambridge, MA, 1958.

[48] D. Karger and C. Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43:601–640, 1996.

[49] D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42:321–328, 1995.

[50] D. R. Karger, D. Koller, and S. J. Phillips. "Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22:1199–1217, 1993.

[51] D. R. Karger and M. S. Levine. Finding maximum flows in simple undirected graphs seems easier than bipartite matching. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998.

[52] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[53] R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[54] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[55] R. M. Karp. An algorithm to solve the $M \times N$ assignment problem in expected time $O(MN \log N)$. *Networks*, 10:143–152, 1980.

[56] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani. Global wire routing in two-dimensional arrays. *Algorithmica*, 2:113–129, 1987.

[57] V. King, S. Rao, and R. Tarjan. A faster deterministic maximum flow algorithm. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, pages 157–164, January 1992.

[58] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 726–737, 1990.

[59] P. Klein, S. A. Plotkin, C. Stein, and É. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23(3):466–487, June 1994.

[60] J. M. Kleinberg. *Approximation algorithms for disjoint paths problems*. PhD thesis, MIT, Cambridge, MA, May 1996.

[61] J. M. Kleinberg. Single-source unsplittable flow. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 68–77, October 1996.

[62] J. M. Kleinberg and R. Rubinfeld. Short paths in expander graphs. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 86–95, 1996.

[63] J. M. Kleinberg and É. Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 26–35, 1995.

[64] J. M. Kleinberg and É. Tardos. Disjoint paths in densely-embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 52–61, 1995.

[65] S. G. Kolliopoulos and C. Stein. Finding real-valued single-source shortest paths in $o(n^3)$ expected time. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proceedings of the 5th Conference on Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 94–104. Springer-Verlag, 1996.

[66] S. G. Kolliopoulos and C. Stein. Improved approximation algorithms for unsplittable flow problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 426–435, 1997.

[67] S. G. Kolliopoulos and C. Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In R. E. Bixby, E. A. Boyd, and R. Z. Rios-Mercado, editors, *Proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 153–168. Springer-Verlag, 1998.

[68] S. G. Kolliopoulos and C. Stein. Finding real-valued single-source shortest paths in $o(n^3)$ expected time. *Journal of Algorithms*, 28:125–141, 1998.

[69] H. W. Kuhn. The Hungarian method for the assignment problem. In *Naval Research Logistics Quarterly*, volume 2, pages 83–97, 1955.

[70] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[71] F. T. Leighton and S. B. Rao. Circuit switching: a multi-commodity flow approach. In *Workshop on Randomized Parallel Computing*, 1996.

[72] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50:228–243, 1995.

[73] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[74] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[75] Y.-L. Lin and S. E. Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8(1):99–118, February 1995.

[76] L. Lovász. On the ratio of optimal and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[77] P. Martin and D. B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In *Proceedings of the 5th Conference on Integer Programming and Combinatorial Optimization*, pages 389–403, 1996.

[78] K. Mehlhorn and V. Priebe. On the all-pairs shortest path algorithm of Moffat and Takaoka. *Random Structures and Algorithms*, 10:205–220, 1997.

[79] A. Moffat and T. Takaoka. An all pairs shortest path algorithm with expected time $O(n^2 \log n)$. *SIAM Journal on Computing*, 16:1023–1031, 1987.

[80] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 377–387, 1988.

[81] D. Peleg and E. Upfal. Disjoint paths on expander graphs. *Combinatorica*, 9:289–313, 1989.

[82] Y. Perl and Y. Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *Journal of the ACM*, 25:1–9, 1978.

[83] S. Plotkin. Competitive routing of virtual circuits in ATM networks. *IEEE J. Selected Areas in Comm.*, pages 1128–1136, 1995. Special issue on Advances in the Fundamentals of Networking.

[84] S. Plotkin, D. B. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[85] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[86] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[87] N. Robertson and P. D. Seymour. Outline of a disjoint paths algorithm. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows and VLSI-Layout*. Springer-Verlag, Berlin, 1990.

[88] A. Schrijver. Homotopic routing methods. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows and VLSI-Layout*. Springer-Verlag, Berlin, 1990.

[89] R. Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 745–749, 1992.

[90] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, 23(3):617–632, June 1994.

[91] D. B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62:461–474, 1993.

[92] J. Spencer. *Ten Lectures on the Probabilistic Method*. SIAM, Philadelphia, 1987.

[93] P. M. Spira. A new algorithm for finding all shortest paths in a graph of positive arcs in average time $O(n^2 \log^2 n)$. *SIAM Journal on Computing*, 2:28–32, 1973.

[94] A. Srinivasan. Improved approximations of packing and covering problems. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 268–276, 1995.

[95] A. Srinivasan. An extension of the Lovász Local Lemma and its applications to integer programming. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 6–15, 1996.

[96] A. Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow and related routing problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, pages 416–425, 1997.

[97] A. Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 636–643, 1997.

[98] C. Stein. *Approximation algorithms for multicommodity flow and shop scheduling problems*. PhD thesis, MIT, Cambridge, MA, August 1992. Also appears as MIT/LCS/TR-550.

[99] T. Takaoka and A. M. Moffat. An $O(n^2 \log n \log \log n)$ expected time algorithm for the all shortest distance problem. In P. Dembinski, editor, *Mathematical Foundations of Computer Science*, volume 88 of *Lecture Notes in Computer Science*, pages 643–655. Springer-Verlag, 1980.

[100] P. Underground. On graphs with Hamiltonian squares. *Discrete Mathematics*, 21:323, 1978.

[101] S. K. Walley and H. H. Tan. Shortest paths in random weighted graphs. In *Proc. 1st Int. Conf. on Computing and Combinatorics*, pages 213–222, 1995.

[102] Neal Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.