CORC REPORT 2000-02

Asymptotic Analysis of the Flow Deviation Method for the Maximum Concurrent Flow Problem*

Daniel Bienstock and Olga Raskina
Columbia University
New York, NY 10027

December, 2000

Abstract

We analyze the asymptotic behavior of the Flow Deviation Method, first presented in 1971 by Fratta, Gerla and Kleinrock, and show that when applied to packing linear programs such as the maximum concurrent flow problem, it yields a fully polynomial-time approximation scheme.

1 Introduction

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

The input to the problem is a graph (directed or undirected) with capacities on its edges and a set of multicommodity demands to be routed. The objective is to find a largest value $\Gamma^* > 0$ such that a fraction Γ^* of every demand can be simultaneously routed without exceeding the available capacities (possibly $\Gamma^* > 1$). In other words, we want to find a feasible flow with maximum throughput. A precise definition is given in Section 2. An equivalent problem is the *minimum congestion problem*: here we must simultaneously route 100% of every demand so that the maximum load of any edge – the ratio of total flow to capacity – is minimized. In the literature both problems are sometimes regarded as one and the same and are jointly referred to as the "maximum concurrent flow problem", but in this paper we will work with the maximum throughput version.

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

What makes this problem especially noteworthy is that it gives rise to extremely difficult linear programs – instances of a size and type relevant to applications often prove beyond the reach of state-of-the-art linear programming codes.

These two facts have spurred a long line of research on alternative algorithms for the maximum concurrent flow problem, in particular, approximation algorithms. Given the nature of the problem, in a sense an ideal approximation algorithm would be one whose precision can be controlled. Given $\epsilon > 0$, a *fully polynomial polynomial-time approximation scheme* is one that estimates Γ^* within relative error ϵ, and whose worst-case complexity grows polynomially in ϵ^{-1} and in the size of the graph.

The first such algorithm was given by Shahrokhi and Matula [SM91] in the late 80's, and it applied to the minimum congestion problem. [SM91] introduced a "potential" function that exponentially penalizes the load on any edge, so that the approximate minimization of the potential function yields a flow whose maximum load is provably near-optimal. To carry out the approximate optimization of the potential function

*This research was partially funded by NSF awards NCR-9706029, CDA-9726385 and 29132-5538 (CORC)

[SM91] used what amounts to a Frank-Wolfe scheme (see [DW61], [FM68], [FW56]). The overall algorithm was proved to have a worst-case complexity bound that grows proportional to ϵ^{-7} (as well as polynomially on the size of the graph). The Shahrokhi-Matula algorithm can also be viewed as a Lagrangian relaxation approach, where the capacity constraints are relaxed, and the violation of these constraints is penalized using very special multipliers. These are exponential multipliers, which roughly stated arise in the Frank-Wolfe scheme as the gradient of the potential function.

The Shahrokhi-Matula results catalyzed an intense and still active research effort. See [LMPSTT91], [GK94], [GK95], [LR98], [PST91], [R95], [KP95]. This research primarily showed how to better use an exponential potential function, still in the Frank-Wolfe Lagrangian relaxation framework, to obtain faster algorithms for the minimum congestion problem, and generalizations. A logarithmic "sliding barrier" function, applied to the minimum congestion problem, is discussed in [VG97]. The analysis in [Y95] shows how the exponential potential function naturally arises in the derandomization of probabilistic algorithms; the algorithms for the maximum concurrent flow problem described in [GK98] and [F00] are related to the ideas in [Y95].

The fastest of these algorithms have running time bounded bounded by ϵ^{-2} times a low-order polynomial on the size of the graph and number of commodities. Altogether, the algorithms in [R95], [GK98] and [F00] yield the best bounds for the maximum concurrent flow problem, but precisely which is "best" depends in a complex manner on the number of vertices, edges and commodities. Experimental testing of these algorithms has yielded implementations that are substantially more effective than commercial linear programming codes, see [B99], [GOPS].

In 1971, Fratta, Gerla and Kleinrock [FGK71] proposed a general algorithmic scheme towards the solution of various optimization problems arising in telecommunications, among them that of finding a feasible multicommodity flow, which is tantamount to solving a maximum concurrent flow problem. The approach in [FGK71], which is quite different from those found in the references cited above, involves a simple idea for increasing the throughput of a feasible flow while using a rational barrier function to prevent flows from exceeding capacities, together with a Frank-Wolfe procedure that reduces the barrier function. A partial convergence proof was presented in [FGK71]. Although the flow deviation method is well-known and frequently used in the telecommunications community, it is poorly known in the algorithms community.

In this paper we show that the flow deviation method, properly implemented, yields an algorithm that solves the maximum concurrent flow problem to relative error ϵ by solving $O(\epsilon^{-2}m^3k+m^3k\log m)$ minimum-cost flow computations, where m and k are, respectively, the number of edges and commodities. The algorithm we describe contains the critical algorithmic ideas in [FGK71] together with some refinements designed to achieve the ϵ^{-2} performance. Without these refinements, the algorithm in [FGK71] can be shown to converge in $O(\rho^2 m^5 \epsilon^{-3} k)$ shortest path computations, where ρ is the *width* [PST91]. In contrast, we note that the algorithm in [F00] requires $O^*(\epsilon^{-2}m(k+m))$ *time.*

2 Definitions

Consider a graph G with n vertices, and a capacity $u_e > 0$ for each edge e. Suppose we are given a set of k *commodities*, where for $1 \le j \le k$, commodity j consists of a pair s_j, t_j of vertices and a *demand amount* $d_j > 0$. For $1 \le j \le k$, let P_j denote the set of paths between s_j and t_j, and for any edge e let $P_{e,j}$ denote the subset of P_j consisting of those paths that contain e. The *maximum concurrent flow problem* is the linear program

$$\textbf{(TF)} \quad \text{s.t.} \qquad \Gamma^* = \max \quad \gamma$$

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

$$\sum_{p \in P_j} x_p = \gamma d_j, \quad 1 \le j \le k, \tag{2}$$

$$0 \le x_p \qquad \forall p. \tag{3}$$

This definition and formulation apply whether the graph is directed or not. From the point of view of linear programming, this path-based formulation is not the most efficient – a flow formulation is more compact. Note that for any commodity j a linear program over the constraints (2), (3) reduces to a shortest path problem. Given $\epsilon > 0$, an ϵ-approximate solution to **TF** is a flow $\hat{x}$ with throughput $\hat{\gamma}$, feasible for **TF**, such that $\hat{\gamma} \geq (1-\epsilon)\Gamma^*$, i.e. $\hat{\gamma}$ approximates Γ^* within relative error at most ϵ.

We will say that a flow vector x is *feasible* if there exists a nonnegative value $\gamma = \gamma(x)$ such that (x, γ) is satisfies (1 - 3), in which case we will also say that x has *throughput* γ. Given a flow vector x, we will say x has *load*

$$\lambda(x) \quad \doteq \quad \max_e \lambda_e(x) \tag{4}$$

where for an edge e,

$$\lambda_e(x) \quad \doteq \quad \frac{\sum_{j=1}^k \sum_{p \in P_{e,j}} x_p}{u_e} \tag{5}$$

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

Consider a problem of the form

$$\min \{F(x) \,:\, x \in Q\} \tag{6}$$

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

$$\min \left\{ [\nabla F(x^t)]^T v \,:\, v \in Q \right\} \tag{7}$$

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

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

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

s.t.

$$\sum_{j=1}^{k} \sum_{p \in P_{e,j}} x_p - u_e \lambda \;\leq\; 0 \quad \forall\, e, \tag{8}$$

$$\sum_{p \in P_j} x_p \;=\; d_j \quad 1 \leq j \leq k, \tag{9}$$

$$0 \leq x_p \qquad \forall p. \tag{10}$$

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

3 The central idea

In this section we describe the main idea in [FGK71].

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

OUTLINE

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

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

C1. Find a feasible z with throughput $\frac{1}{\lambda(\hat{x})}\hat{\gamma}$ and $\lambda(z)$ substantially smaller than $\lambda(y)$. Reset $\hat{x} \leftarrow z$ and $\hat{\gamma} \leftarrow \frac{1}{\lambda(\hat{x})}\hat{\gamma}$, and go to **B1**.

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

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

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

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

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

$$\Psi(x) \quad = \quad \sum_e \psi(\lambda_e(x)) \tag{11}$$

and to choose z as the solution to

$$\min \left\{ \Psi(x) \; : \; x \text{ a feasible flow with } \gamma(x) = \frac{\hat{\gamma}}{\lambda(\hat{x})} \right\} \tag{12}$$

In fact, by property (ii) of ψ the optimal solution to the optimization problem (12) "should" have maximum load smaller than $\lambda(\hat{x})$. Note, however, that in view of our construction of y in step **B1**, $\Psi(y)$ is undefined (because $\lambda(y) = 1$), and in general, it may be the case that $\Psi(x)$ is undefined for any $x \in \frac{1}{\lambda(\hat{x})}\hat{\gamma}P$. Thus, we must amend Step **B1** slightly: to obtain y, we scale up $\hat{x}$ by a factor slightly smaller than $\frac{1}{\lambda(\hat{x})}$. Further, it may not be strictly necessary to minimize Ψ – rather, we should decrease it from the value $\Psi(y)$. Our algorithmic scheme is now:

Basic Algorithm

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

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

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

The algorithm implicit in Steps **A2 - C2** is the main contribution in [FGK71]; found in pp. 112(bott.)–114. It significantly differs from algorithms found in the references cited above. Other ingredients in [FGK71] are:

1. The barrier function $\psi(x) = \frac{x}{1-x}$,
2. A Frank-Wolfe procedure to carry out the barrier reduction step **C2**,
3. A choice for the parameter μ in **B2** that provides a sufficiently rapid increase in throughput, and
4. A provably good choice for the starting point $\hat{x}$ in **A2**.

The particular implementation we describe below differs from that in [FGK71] in two aspects:

- Our Frank-Wolfe iterations are minimum-cost flow problems, as opposed to shortest path computations, and

- In **C2** we may carry out several Frank-Wolfe iterations (i.e. several gradient steps) whereas [FGK71] uses a single iteration.

If **A2 - C2** is implemented using shortest path computations only and single Frank-Wolfe iterations in **C2**, the resulting algorithm can be shown to converge to an ϵ-approximate solution in $O(\rho^2 m^5 \epsilon^{-3} k)$ shortest path computations [R01], where ρ is the width parameter [PST91]. Finally, we note that in [FGK71] the term "flow deviation method" appears to refer to the Frank-Wolfe procedure itself; however it has since become synonymous with the entire scheme **A2 - C2**.

4 The algorithm

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

Algorithm FD

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

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

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

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

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

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

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

End

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

End.

4.1 Analysis of Algorithm FD

Given $\gamma > 0$ denote

$$\Psi^*(\gamma) \quad = \quad \min \{ \Psi(x) \,:\, x \text{ feasible with } \gamma(x) = \gamma \}. \tag{13}$$

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

Theorem 4.1 *Consider an execution of the* **For** *loop in* **Step 2***, with input y^t. Suppose the loop exits at iteration h. Then*

$$\Psi(v^h) \quad \leq \quad 2\Psi^*(\gamma(y^t)). \tag{14}$$

Pending the proof of Theorem 4.1, the following sequence of lemmas establish the correctness and workload of the algorithm.

Lemma 4.2 *Let r be a nonnegative integer. (i) Suppose $\gamma < (1-2^{-(r+1)})\Gamma^*$. Then $\Psi^*(\gamma) < 2^{r+1}m$. (ii) Suppose $(1-2^{-r})\Gamma^* \leq \gamma$. Then $2^r - 1 \leq \Psi^*(\gamma)$.*

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

Lemma 4.3 *Let $0 < \tau < 1$, and suppose that an iterate x^t in* **Step 3** *satisfies $\lambda(x^t) \geq 1 - \frac{\tau}{2m}$. Then*

$$\gamma(x^t) \quad \geq \quad (1-\tau)\Gamma^* \tag{15}$$

Proof. Assume by contradiction that (15) does not hold. Let x^* be feasible with throughput Γ^*. Then $u = \frac{\gamma(x^t)}{\Gamma^*}x^*$ has throughput $\gamma(x^t)$, and satisfies

$$\lambda(u) \quad < \quad 1-\tau. \tag{16}$$

As a result

$$\Psi(u) \quad < \quad m(\tau^{-1}-1). \tag{17}$$

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

$$\Psi(x^t) \quad \geq \quad \frac{2m}{\tau} - 1, \tag{18}$$

which together with (17) contradicts Theorem 4.1. ■

Lemmas 4.2(i) and 4.3 imply:

Corollary 4.4 *Upon termination of Algorithm FD, we have a feasible flow with ϵ-optimal throughput.*

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

Lemma 4.5 *The vector x^0 has throughput at least $\frac{\Gamma^*}{2m}$.*

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

$$m\lambda(x) \quad \geq \quad L^*. \tag{19}$$

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

$$\frac{1}{\lambda(z)} \quad \geq \quad \frac{1}{m\lambda(x)}. \tag{20}$$

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

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

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

Proof. By Lemma 4.3, for $r \geq 0$ if $\gamma(x^t) < (1 - 2^{-(r+1)})\Gamma^*$ we have $\lambda(x^t) < 1 - \frac{2^{-r}}{4m}$. Consequently,

$$\gamma(x^{t+1}) = \gamma(y^t) = \frac{1 - \frac{1}{2}(1 - \lambda(x^t))}{\lambda(x^t)}\gamma(x^t) \tag{21}$$

$$= \frac{1}{2}(1 + \frac{1}{\lambda(x^t)})\gamma(x^t) \tag{22}$$

$$> (1 + \frac{2^{-r}}{8m})\gamma(x^t). \tag{23}$$

(i) Suppose $0 < r$. By definition we started Phase r with throughput at least $(1 - 2^{-r})\Gamma^*$. Thus, (23) implies that this Phase will perform $O(m)$ iterations, as desired.

(ii)This follows as (i), using Lemma 4.5.

(iii) Consider an iteration t during Phase r with $r > q$. Since the algorithm has not yet terminated, we can replace (23) with the stronger condition $\gamma(y^t) > (1 + \frac{2^{-q}}{8m})\gamma(x^t)$, and again we obtain that there are altogether at most $O(m)$ iterations in Phases $q+1, q+2, \cdots$. ■

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

Lemma 4.7 *Consider an iteration t of* **Step 1**, *and let e be any edge. Then $1 - \lambda_e(y^t) \geq \frac{1-\lambda_e(x^t)}{2}$.*

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

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

Proof. Suppose first that $r > 0$. Then by Theorem 4.1 and respectively, Lemma 4.2 (for the case $r \leq q$) and the second termination criterion in **Step 3** (for the case $r > q$) we have that $\Psi(x^t) \leq O(m2^{\min\{r,q\}})$. To obtain the desired result, we will show $\Psi(y^t) = O(\Psi(x^t))$. To see this, consider the contributions of an edge e to $\Psi(x^t)$ and to $\Psi(y^t)$. These are, respectively, $\frac{\lambda_e(x^t)}{1-\lambda_e(x^t)}$ and $\frac{\lambda_e(y^t)}{1-\lambda_e(y^t)}$. By Lemma 4.7, the denominator in the second expression is at least half of that in the first. To compare the numerators, note that since this is an iteration during Phase $r > 0$, $\lambda(x^t) \geq 1 - 2^{-r} \geq 1/2$ and consequently $\lambda_e(y^t) = \frac{1+\lambda(x^t)}{2\lambda(x^t)}\lambda_e(x^t) \leq \frac{3}{2}\lambda_e(x^t)$. Hence $\Psi(y^t) = O(\Psi(x^t))$, as desired.

Suppose instead that $r = 0$. We always have (again by Theorem 4.1 and by the choice of x^0) that $\Psi(x^t) = O(m)$. If $\lambda(x^t) \geq 1/2$ the Lemma follows as in the previous paragraph. If instead $\lambda(x^t) < 1/2$, then each edge e will satisfy $\lambda_e(y^t) \leq 3/4$ and thus $\Psi(y^t) = O(m)$. ■

Lemma 4.9 *Let $0 \leq r$. The number of iterations of* **A.h-C.h** *in an execution of* **Step 2** *during Phase r is $O(2^{2\min\{r,q\}}m^2)$.*

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

$$\Psi(v^{h+1}) - \Psi(v^h) < -\frac{\Psi(v^h)^2}{128(\Psi(v^h)^3 + m)}, \tag{24}$$

then we will reach a value of Ψ smaller than Ψ^*, a contradiction. Thus, consider an iteration h where (24) holds. Suppose first that

$$\Psi(v^h)^3 \geq m. \tag{25}$$

In this case, the recursion (24) can be abstracted as

$$z_{h+1} - z_h \leq -c\frac{1}{z_h},$$

where $c > 0$ is a constant. This recursion has the property that it reduces z_h by a factor of 2 in $O(z_h^2)$ iterations. Thus, using Lemma 4.8, we obtain that there are at most $O(2^{2\min\{r,q\}}m^2)$ iterations of **A.h-C.h** where (25) holds.

In the remainder of the proof we handle the iterations with $\Psi(v^h)^3 < m$. If $r > 0$ then using Lemma 4.2(ii) we conclude that each iteration satisfying (24) decreases Ψ by $\Omega(1/m)$, and consequently there are at most $O(m^{5/3})$ such iterations (a tighter analysis is possible but not needed). This concludes the proof if $r > 0$. Finally, if $r = 0$ then just as in the previous line we conclude that in at most $O(m^{5/3})$ iterations we obtain $\Psi(v^h) \leq 1$. By Lemma 4.5, $\gamma(x^0) \geq \frac{\Gamma^*}{2m}$, and a variation of the analysis in Lemma 4.2 shows that any time during Phase 0, $\Psi^* \geq \frac{1}{2m}$. We conclude that there are at most $O(\frac{1}{1/2m}) = O(m)$ iterations h with $\Psi(v^h) < 1$. ■

Corollary 4.10 *The total number of Frank-Wolfe iterations* **A.h** *over the course of Algorithm FD is* $O(m^3\epsilon^{-2} + m^3 \log m)$.

In the next section we prove Theorem 4.1. In Section 4.3 we show how to replace the line-search in step **B.h** with a provably good step-size rule that requires $O(1)$ time.

4.2 Proof of Theorem 4.1

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

$$g(\sigma) \doteq \Psi((1-\sigma)v^h + \sigma w^h) = \Psi(v^h + \sigma(w^h - v^h)). \tag{26}$$

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

$$g'(0) = [\nabla\Psi(v^h)]^T \cdot (w^h - v^h), \text{ and} \tag{27}$$

$$g(\sigma) = g(0) + g'(0)\sigma + \frac{1}{2}g''(\alpha)\sigma^2, \tag{28}$$

where $0 < \alpha < \sigma$ (2nd order Taylor expansion). By choice of w^h in **A.h**, the right-hand side of (27) cannot decrease if we replace w^h with any other feasible flow with throughput γ. In particular, if we use a flow w^* with throughput γ and such that $\Psi(w^*) = \Psi^*(\gamma)$, we obtain

$$g(\sigma) - g(0) \leq [\nabla\Psi(v^h)]^T \cdot (w^* - v^h)\sigma + \tfrac{1}{2}g''(\alpha)\sigma^2. \tag{29}$$

which, since Ψ is convex, implies:

$$g(\sigma) - g(0) \leq (\Psi^*(\gamma) - \Psi(v^h))\sigma + \frac{1}{2}g''(\alpha)\sigma^2. \tag{30}$$

Next we will bound the quadratic term in (30). We have

$$g(\alpha) = \sum_e \psi(\lambda_e(v^h) + \sigma[\lambda_e(w^h) - \lambda_e(v^h)]), \tag{31}$$

and consequently

$$g''(\alpha) = \sum_e \frac{(\lambda_e(w^h) - \lambda_e(v^h))^2}{(1 - \lambda_e(v^h) - \alpha[\lambda_e(w^h) - \lambda_e(v^h)])^3} \tag{32}$$

Since both v^h and w^h are feasible, $(\lambda_e(w^h) - \lambda_e(v^h))^2 \leq 1$. Further, $\frac{1}{(1-x)^3} < 8(\frac{x}{1-x})^3 + 8$ for all $0 \leq x < 1$. Thus, from (32) we get that

$$g''(\alpha) \leq 8([\Psi(v^h + \alpha(w^h - v^h)]^3 + m). \tag{33}$$

If σ is chosen so that

$$\Psi(v^h + \sigma(w^h - v^h)) \leq \Psi(v^h), \tag{34}$$

then (33) and the convexity of Ψ imply $g''(\alpha) \leq 8(\Psi(v^h)^3 + m)$, and substituting in (30) we obtain:

$$\begin{aligned} g(\sigma) - g(0) &= \\ \Psi(v^h + \sigma(w^h - v^h)) - \Psi(v^h) &\leq (\Psi^*(\gamma) - \Psi(v^h))\sigma + 8(\Psi^3(v^h) + m)\sigma^2. \end{aligned} \tag{35}$$

Inequality (35) holds for any $0 \leq \sigma \leq 1$ that satisfies (34). In Step **B.h** we choose $\sigma = \sigma_h$ so that the left-hand side of (35) is minimized. Instead, let $0 \leq \mu < 1$ be the argument that minimizes the quadratic Q on the right-hand side of (35) (note: a simple check verifies the stated bounds on μ). We would like to argue that $\Psi(v^h + \sigma_h(w^h - v^h)) - \Psi(v^h) \leq Q(\mu)$. This will follow if we can argue that μ is such that (35) holds at $\sigma = \mu$; or, in other words, that $\sigma = \mu$ satisfies (34).

But note that for $0 \leq \sigma$ small enough (34) holds by choice of w^h ($w^h - v^h$ is a descent direction). If (35) holds for all $0 \leq \sigma$ we are done. Otherwise (34) holds for some closed interval $[0, \tilde{\sigma}]$ and does not hold outside the interval. Since (35) holds with equality at 0, and $Q'(0) < 0$, clearly $\mu < \tilde{\sigma}$, as desired.

Thus, from (35) we get (after a calculation to minimize the quadratic)

$$\begin{aligned} \Psi(v^{h+1}) - \Psi(v^h) &= \\ \Psi(v^h + \sigma_h(w^h - v^h)) - \Psi(v^h) &\leq -\frac{(\Psi^*(\gamma) - \Psi(v^h))^2}{32(\Psi^3(v^h) + m)}. \end{aligned} \tag{36}$$

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

$$\Psi(v^{h+1}) - \Psi(v^h) \leq -\frac{\Psi^2(v^h)}{128(\Psi^3(v^h) + m)}. \tag{37}$$

The theorem is proved. ■

4.3 Choosing the step-size

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

One way to choose σ so as to minimize the quadratic $Q(\sigma)$ introduced in the proof of Theorem 4.1, while assuming $\Psi(v^h) > 2\Psi^*(\gamma)$, or in other words, to minimize

$$\hat{Q}(\sigma) \doteq -\frac{\Psi(v^h)}{2}\sigma + 8(\Psi^3(v^h) + m)\sigma^2 \tag{38}$$

whose decrease was used in the proof of Theorem 4.1 to bound the decrease in Ψ itself. We have to show that if we alter Algorithm FD to use this step-size ($= \frac{\Psi(v^h)}{32(\Psi^3(v^h)+m)}$) we still have a correct algorithm with the same complexity bound.

In fact, we proved that as long as $\Psi(v^h) > 2\Psi^*(\gamma)$, choosing σ so as to minimize $\hat{Q}(\sigma)$ guarantees a decrease in Ψ of at least $\frac{\Psi^2(v^h)}{128(\Psi^3(v^h)+m)}$. Thus the validity of the algorithm is preserved. Further, Lemma 4.9 still applies (consider the first line in its proof). This shows that the complexity bound for Algorithm FD still applies.

4.4 Comments on Algorithm FD

In [FGK71], the procedures and analysis are not pitched towards provably good approximation – this field existed only in embryonic form in 1971. This is the primary cause for the differences between Algorithm FD and that in [FGK71]. Nevertheless, the critical idea in [FGK71] is the key ingredient in Algorithm FD.

(i) The fact that the Frank-Wolfe iterations are minimum-cost flow problems is only needed in the paragraph immediately following (32) so that we can guarantee $\lambda(w^h) \leq 1$. If we were to replace the minimum-cost flow calls with shortest path calls, then instead we could only claim $\lambda(w^h) \leq \rho$, where ρ is the *width* ([PST91] – in the context of this paper, the width of the problem is upper bounded by the ratio of the largest demand d_j to the smallest capacity u_e). This results in an increase in our complexity estimate by a factor of ρ^2. The "trick" of using minimum-cost flow calls to avoid the dependence on ρ is not new: it is used in [PST91] and [GK94]. Recent algorithms for the maximum concurrent flow problem avoid the dependence on ρ while only resorting to shortest path calls ([GK98], [F00]). We conjecture that a more refined version of our algorithm will achieve the same behavior.

(ii) The algorithmic outline in [FGK71] uses a single Frank-Wolfe iteration in each **Step 2**. The algorithm in this paper can be adapted so as to use single Frank-Wolfe iterations in each **Step 2**; it can be shown [R01] that the dependence on ϵ of the adapted algorithm grows as $O(\epsilon^{-3})$. This adaptation entails a finer tuning of the multiplier in **Step 1**.

(iii) The magnification factor $\frac{1-\frac{1}{2}(1-\lambda(x^t))}{\lambda(x^t)}$ used in **Step 1** is as in [FGK71], except that the choice of the parameter 1/2 is ours ([FGK71] only states that "a proper tolerance" should be used). In **Step 0** our choice for flow z is precisely the same as the choice in the initialization step in [FGK71]. We start with $x^0 = \frac{1}{2\lambda(z)} z$, rather than with $\frac{z}{\lambda(z)}$, so that we can guarantee that $\Psi(x^0) = O(m)$ (as opposed to $O(m^2)$).

(iv) Our proof of Theorem 4.1, up to inequality (30), follows fairly closely a similar analysis in [FGK71](pp. 128 - 130). The latter part of our proof follows [FGK71] somewhat less closely. In particular, [FGK71] does not analyze the quadratic term in (30), which is key to the overall complexity estimate. Finally, our analysis prior to the proof of Theorem 4.1 is new.

(v) Algorithm FD is easily generalized to general packing linear programs [PST91].

(vi) [FGK71] points out that [DS69] contains some ideas similar to those in the Flow Deviation method, although in less developed form. Courant [Cou43] has been credited with one of the earliest uses of "potential" functions in the solution of systems of equations.

(vii) Some pointers for the reader who decides to tackle [FGK71]. Here we use the notation from [FGK71]. (a) The algorithm in pp. 113-114 is geared towards finding a feasible flow with throughput 1. This includes the exit condition in Step 1. (b) Step 2, and the definition of the FD operator in page 110 inductively show the for $n > 0$ we always have $\sigma_n < 1$ and consequently $RE_{n+1} > RE_n$. (c) There is an error in the paper for the case $n = 0$: the exit condition should be $\sigma_0 / RE_0 \leq 1$.

4.5 Computational outlook

Despite the clear theoretical inferiority of the worst-case complexity bound we proved for algorithm FD as compared to e.g. that in [F00] or [GK98], practitioners who employ the flow deviation method swear by its effectiveness.

There are two likely advantages of the rational potential function used in FD over the (implicit) exponential potential in [F00], [GK98], or the Karmarkar-like function in [VG97]: one is better numerical stability, and the other is independence from ϵ. By this we mean that the potential function in FD does not include ϵ in its definition – in fact, the entire algorithm only uses ϵ in the termination conditions. This may lead to "cleaner" implementations that are easier to tune.

Our on-going computational research (see [B99], and the forthcoming [B01]) has focused on explicitly seeking to reduce potential as a means to accelerate convergence (again as opposed to [F00], [GK98] where the potential function is implicit) and we feel that this is an effective paradigm, in that we can leverage the considerable machinery developed by the nonlinear programming community. In this context, exploring the rational potential function from a computational standpoint will be a topic for our future work.

References

[B99] D. Bienstock, Approximately solving large-scale linear programs. I. Strengthening lower bounds and accelerating convergence, CORC Report 1999-1, Columbia University. (Extended abstract: *Proc 11th. Ann. ACM-SIAM Symposium on Discrete Algorithms*, January 2000).

[B01] D. Bienstock, Potential Function Methods for Approximately Solving Linear Programs: Theory and Practice, to appear, CORE Lecture Series Monograph, CORE, Belgium (2001).

[Cou43] R. Courant, Variational methods for the solution of problems of equilibrium and vibration, *Bull. Amer. Math. Soc.* **49** (1943), 1 – 43 .

[DS69] S.C. Dafermos and F.T. Sparrow, The traffic assignment problem for a general network, *Journal of Research of the National Bureau of Standards - B*, **73B** (1969).

[DW61] G.B. Danzig and P. Wolfe, The decomposition algorithm for linear programming, *Econometrica* **29** (1961), 767 – 778.

[F00] L. Fleischer, Approximating fractional multicommodity flow independent of the number of commodities, *SIAM J. Disc. Math.* **3** (2000) 505 – 520.

[FM68] A.V. Fiacco and G.P. McCormick, *Nonlinear Programming: Sequential Unconstrained Optimization Techniques*, Wiley, New York (1968).

[FW56] M. Frank and P. Wolfe, An algorithm for quadratic programming, *Naval Res. Logistics Quarterly* **3** (1956), 149 – 154.

[FGK71] L. Fratta, M. Gerla and L. Kleinrock, The flow deviation method: an approach to store-and-forward communication network design, *Networks* **3** (1971), 97 – 133.

[GOPS] A.V. Goldberg, J.D. Oldham, S. Plotkin and C. Stein, An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow, *Proc. IPCO* 1998, 338 - 352.

[GK94] M.D. Grigoriadis and L.G. Khachiyan Fast approximation schemes for convex programs with many blocks and coupling constraints, *SIAM Journal on Optimization* **4** (1994) 86 – 107.

[GK95] M.D. Grigoriadis and L.G. Khachiyan An exponential-function reduction method for block-angular convex programs, *Networks* **26** (1995) 59 – 68.

[GK96] M.D. Grigoriadis and L.G. Khachiyan, Coordination complexity of parallel price-directive decomposition, *Math. Oper. Res.* **21** (1996) 321 – 340.

[GK98] N. Garg and J. Könemann, Faster and simpler algorithms for multicommodity flow and other fractional packing problems, *Proc. 39th Ann. Symp. on Foundations of Comp. Sci.* (1998) 300 – 309.

[KP95] D. Karger and S. Plotkin, Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows, *Proc. 27th Ann. ACM Symp. on Theory of Computing* (1995) 18 – 25.

[KPST90] P. Klein, S. Plotkin, C. Stein and E. Tardos, Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts, *Proc. 22nd Ann. ACM Symp. on Theory of Computing* (1990), 310 – 321.

[KY98] P. Klein and N. Young, On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms. *Proc. IPCO* 1999, 320 – 327.

[LMPSTT91] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos and S. Tragoudas, Fast approximation algorithms for multicommodity flow problems, *Proc. 23nd Ann. ACM Symp. on Theory of Computing* (1991), 101-111.

[LR98] T. Leighton and S. Rao, An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms, *Proc. FOCS 29* (1988), 422 – 431.

[PST91] S. Plotkin, D.B. Shmoys and E. Tardos, Fast approximation algorithms for fractional packing and covering problems, *Math. of Oper. Res.* **20** (1995) 495 – 504. Extended abstract: *Proc. 32nd Annual IEEE Symp. On Foundations of Computer Science*, (1991), 495 – 504.

[R95] T. Radzik, Fast deterministic approximation for the multicommodity flow problem, *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms* (1995), 486 – 492.

[R01] O. Raskina, Ph. D. Dissertation, Dept. of IEOR, Columbia University (2001).

[SM91] F. Shahrokhi and D.W. Matula, The maximum concurrent flow problem, *Journal of the ACM* **37** (1991), 318 – 334.

[VG97] J. Villavicencio and M.D. Grigoriadis, Approximate Lagrangian decomposition using a modified Karmarkar logarithmic potential, in *Network Optimization* (Pardalos and Hager, eds.) Lecture Notes in Economics and Mathematical Systems **450** Springer-Verlag, Berlin (1995), 471 – 485.

[Y95] N. Young, Randomized rounding without solving the linear program, in The maximum concurrent flow problem, *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms* (1995), 170 – 178.