Mathematical Programming 73 (1996) 111-127

Speeding up Karmarkar's algorithm for multicommodity flows

Sanjiv Kapoor [1], Pravin M. Vaidya *

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Received 29 February 1988; revised manuscript received 25 September 1990

Abstract

We show how to speed up Karmarkar's linear programming algorithm for the case of multicommodity flows. The special structure of the constraint matrix is exploited to obtain an algorithm for the multicommodity flow problem which requires $O(s^{3.5}v^{2.5}eL)$ arithmetic operations, each operation being performed to a precision of $O(L)$ bits. Here v is the number of vertices and e is the number of edges in the given network, s is the number of commodities, and L is bounded by the number of bits in the input. We obtain a speed up of the order of $(e^{0.5}/v^{0.5}) + (e^{2.5}/v^{2.5}s^2)$ over Karmarkar's modified algorithm which is substantial for dense networks. The techniques in the paper can also be used to speed up any interior point algorithm for any linear programming problem whose constraint matrix is structurally similar to the one in the multicommodity flow problem.

Keywords: Multicommodity flows; Interior point methods; Linear programming

1. Introduction

In this paper we study the problem of finding a multicommodity flow in a directed network (V, E) [4]. The network has a set of sources S and a set of sinks T and it is required that source S_i send f_i units of commodity i to sink T_i through the network. (Note that $S \subseteq V$, $T \subseteq V$.) Moreover, for the ith edge there is a capacity c_i which upper bounds the total of all the commodities that may pass through that edge. For each of the sets V, E, S and T, we shall use the corresponding lower case letter to denote the size of the set. (The symbol e will be used to denote the number of edges as well as the vector of all 1's; the intended usage will be clear from the context.)

* Corresponding author. Research supported by a fellowship from the Shell Foundation.

[1] Research supported by NSF under grant NSF DCR-8404239.

For each source-sink pair (T_i, S_i) we add an extra edge directed from T_i to S_i, with an upper bound of f_i on the flow of commodity i through this edge. The goal is then to find a circulation such that the flow of commodity i in edge (T_i, S_i) equals the capacity f_i. No augmenting path algorithm is known for this problem when the number of commodities exceeds 1. The multicommodity flow problem may be formulated as the following linear program:

$$\begin{aligned}
\text{MF:}\quad \min\quad & p^{\mathrm{T}} w = \sum_{i=1}^{s} y_i,\\
\text{s.t.}\quad & APx = 0 \quad \text{(flow conservation)},\\
& Cx + Iy - zc = 0 \quad \text{(capacity constraints)},\\
& \sum_{i=1}^{n_1} x_i + \sum_{i=1}^{e} y_i + z = \sum_{i=1}^{e} c_i + 1,\\
& x \geqslant 0,\ y \geqslant 0,\ z \geqslant 0,
\end{aligned}$$

where:

(1) $x \in \mathbb{R}^{n_1}$ is the vector of flow variables. For each edge incident to a vertex other than a source or a sink (i.e., for each edge incident to a vertex in $V - (T \cup S)$) there are s variables, each variable corresponding to one of the s commodities; for an edge incident to S_i or T_i there is exactly one variable and this variable corresponds to the flow of commodity i in the edge. Moreover, the first s co-ordinates of x correspond to the flows in the edges (T_i, S_i), $i = 1, \ldots, s$.

(2) $y \in \mathbb{R}^e$ is the vector of slacks.

(3) z is an extra variable that takes on the value 1 for any feasible solution.

(4) $w^{\mathrm{T}} = (x^{\mathrm{T}}, y^{\mathrm{T}}, z)$ and $w \in \mathbb{R}^N$ where $N = n_1 + e + 1$.

(5) $c \in \mathbb{R}^e$ is the capacity vector upper bounding x. The capacity constraints $Cx + Iy - zc = 0$ upper bound the total flow of all commodities that may pass through each edge.

(6) Let n_2 be the total number of edges incident to vertices in $T \cup S$, and $n_3 = e - n_2$.

$$C = \begin{bmatrix} I & 0 \\ 0 & C_1 \end{bmatrix},$$

where $C \in \mathbb{R}^{e \times n_1}$, $I \in \mathbb{R}^{n_2 \times n_2}$ is the identity matrix, $C_1 \in \mathbb{R}^{n_3 \times sn_3}$ and the ith row of C_1 has 1's in the positions $s(i-1)+1$ through si, and 0's in the remaining positions.

(7) A is a block diagonal matrix with s blocks. The ith block of A is the node-edge incidence matrix of the directed graph induced by the vertex set consisting of S_i and the vertices reachable from S_i. P is an appropriate permutation matrix. The flow conservation constraints $APx = 0$ state that for commodity i, $1 \leqslant i \leqslant s$, the flow of commodity i into a vertex equals the flow of commodity i out of the vertex for every vertex.

(8) The objective function $p^{\mathrm{T}} w$ is the sum of the slacks in the edges (T_i, S_i), $i = 1, \ldots, s$.

(9) The normalization constraint $\sum_{i=1}^{n_1} x_i + \sum_{i=1}^{e} y_i + z = \sum_{i=1}^{e} c_i + 1$ is obtained by summing the capacity constraints and combining this with the requirement $z = 1$.

Note that the above formulation of the multicommodity flow problem is not the standard one but rather one which is explicitly in Karmarkar's form [6]. Furthermore, if a required multicommodity flow does exist then the minimum value of $p^{\mathrm{T}}w$ is zero, and an optimal solution to problem MF gives the required flow.

Let $L = \log(\det_{\max}) + \log(\sum_i c_i) + \log N$, where $\det_{\max}$ is the largest absolute value of the determinant of any square submatrix of the constraint matrix in problem MF. We give an algorithm for the multicommodity flow problem (i.e., for problem MF) which requires $O(s^{3.5}v^{2.5}eL)$ arithmetic operations, each operation performed to a precision of $O(L)$ bits. The algorithm is an adaptation of Karmarkar's linear programming algorithm [6]. Note that solving the multicommodity flow problem directly using Karmarkar's modified algorithm (which incorporates rank one updates and exploits the sparsity of the constraint matrix while forming ADA^{T}) takes $O((s^{3.5}v^2e^{1.5} + s^{1.5}e^{3.5})L)$ arithmetic operations. Thus we obtain a speed up of the order of $(e^{0.5}/v^{0.5}) + (e^{2.5}/v^{2.5}s^2)$ over Karmarkar's modified algorithm which is substantial for dense networks. The techniques used in this paper to obtain a speed up for multicommodity flows can also be utilized to obtain speed ups for similarly structured linear programs, for example, those arising in problems with generalized upper bounding and block angular problems [1,2]. For block angular problems, ideas similar in spirit to some in this paper have been used in [1] to reduce the work for projection computation at each iteration, but without any specific analysis.

In Section 2 we give a brief overview of the two key techniques that are used to speed up Karmarkar's linear programming algorithm for the case of multicommodity flows. In Sections 3 and 4 we give a description of these techniques. In Section 5 we show how to exploit the underlying structure of the multicommodity flow problem to obtain a procedure for quickly finding an optimal solution once an approximate optimum is available. In Section 6 we give some concluding remarks. Finally, the techniques in Sections 3 and 4 can be also applied to the linear programming algorithm in [11] (instead of Karmarkar's algorithm), and this gives a slightly faster algorithm for the multicommodity flow problem which requires $O(s^3v^{2.5}e^{0.5}L + s^3v^2e)$ arithmetic operations. The details of this slightly faster algorithm will be left to the reader.

We shall conclude this section with a short discussion on obtaining a starting point and measuring convergence. An initial strictly interior feasible point can be obtained by assigning a small positive flow of each commodity to each edge such that flow conservation constraints and capacity constraints are satisfied, and the total flow of all the commodities in an edge is strictly less than the capacity of the edge. As in [6] convergence is measured by means of the potential function $\sum_{i=1}^{N} \ln(p^{\mathrm{T}}w/w_i)$. We shall briefly describe how the initialization may be done so that the potential at the initial point is $O(NL)$. First, assign a flow value of $1/2se$ of each commodity to each edge. This results in an incoming surplus at some nodes and an incoming deficit at some other nodes; the total surplus of a commodity i at all nodes (except source S_i and sink T_i) is at most $1/2s$. The surpluses and deficits of commodity i may be eliminated by repeatedly finding: (i) a directed path from source S_i to a node with incoming deficit of commodity i, or (ii) a directed path from a node with incoming surplus of commodity i to sink T_i, or (iii) a directed path from a node with surplus of commodity i to a node with deficit of commodity i. By pushing flow along such a path the number of nodes that have deficits/surpluses of commodity i may be reduced by one. Once the surpluses and deficits of all the commodities have been eliminated, the flow of each commodity in each edge is at least $1/2se$ and at most $\frac{1}{2}$. Moreover, since all capacities are assumed to be integers, each slack y_i is at least $\frac{1}{2}$. It is then easily seen that the initial value of the potential $\sum_{i=1}^{N} \ln(p^{\mathrm{T}}w/w_i)$ is $O(NL)$. It is also straightforward to implement this initialization procedure in $O(sve)$ time.

Note that if the minimum value of $p^{\mathrm{T}}w$ over the feasible region is zero then at each iteration the value of the potential function is reduced by a fixed constant, whereas if during some iteration the potential cannot be reduced by a constant then the minimum value of $p^{\mathrm{T}}w$ is guaranteed to be greater than zero and the given multicommodity flow problem is infeasible. For details the reader may refer to Section 3.1 in reference [6].

At this point we also note that a preliminary version of this paper appeared in [5].

2. An overview

Applying Karmarkar's algorithm [6] to problem MF reduces the global optimization problem to a sequence of $O(NL)$ local optimizations over ellipsoids. The algorithm for the solution of problem MF is an iterative algorithm with $O(NL)$ iterations, and the local optimization during each iteration consists of minimizing a linear function over an ellipsoid. Let

$$w^k = \begin{pmatrix} x^k \\ y^k \\ 1 \end{pmatrix}$$

be the point at the beginning of the kth iteration, where $(x^k)^{\mathrm{T}} = (x_1^k, x_2^k, \ldots, x_{n_1}^k)$ and $(y^k)^{\mathrm{T}} = (y_1^k, y_2^k, \ldots, y_e^k)$. Let α be a constant less than $\frac{1}{4}$. During the kth iteration w^{k+1} is obtained from w^k as follows.

Step 1. Find a direction w^d by solving the local optimization problem

$$\begin{aligned} \min \quad & p^{\mathrm{T}}Dw = \sum_{i=1}^{s} (D_y)_{ii}\, y_i \\ \text{s.t.} \quad & APD_x x = 0 \quad \text{(flow conservation)}, \\ & CD_x x + D_y y - cz = 0 \quad \text{(capacity constraints)}, \\ & e^{\mathrm{T}}w = 0, \\ & w^{\mathrm{T}}w \leqslant 1, \end{aligned}$$

where $e^T = (1, \dots, 1)$ and D, D_x, and D_y are diagonal matrices defined by $D = \mathrm{diag}(D_x, D_y, 1)$, $D_x = \mathrm{diag}(x_1^k, x_2^k, \dots, x_{n_1}^k)$, $D_y = \mathrm{diag}(y_1^k, y_2^k, \dots, y_e^k)$.

Step 2. Compute w' as $w' := (1/N)e + (\alpha/N)w^d$.

Step 3. $w^{k+1} := (\sum_i c_i + 1)(Dw'/e^T Dw')$.

The speed up for multicommodity flows is obtained by the application of two key ideas (techniques). The first idea is to utilize the special nature of the capacity constraints to reduce the dimension of the problem that is solved at each iteration. This technique is similar to the one used in problems with upper (lower) bounds. A direct solution of the local optimization problem in Step 1 above involves inverting an $(sv + e) \times (sv + e)$ matrix. However, the special structure of the matrices CD_x and D_y can be exploited to reduce the above local optimization problem to the problem of inverting a matrix A_1 of dimension sv. (Note that for a dense graph v is much smaller than e.) This reduction is described in Section 3. Specifically, $A_1 = APD_xD_{11}^{-1}D_xP^TA^T$ where $D_{11} = I + D_xC^TD_y^{-2}CD_x$. It is shown in Section 3 that A_1 can be computed from D in $O(s^2e)$ arithmetic operations, and that once $(A_1)^{-1}$ is available, w^d and w^{k+1} can be computed in $O(s^2v^2)$ operations.

The second idea is to balance the work required to directly invert A_1 and to update $(A_1)^{-1}$ via rank one changes over the entire course of the algorithm. As in [6] we work with an approximation D_Δ to D such that $(D_\Delta)_{ii} \in [(D)_{ii}/\sqrt{2}, \sqrt{2}(D)_{ii}]$; this corresponds to replacing D by D_Δ in the expression for A_1. In Section 3 we show that an update of an entry in D_Δ leads to at most a rank five change in A_1 and thereby to at most a rank five change in $(A_1)^{-1}$. This constant rank change in $(A_1)^{-1}$ can be computed in $O(s^2v^2)$ operations using the Sherman–Morrison–Woodbury formula [3,10]

$$(G + UV^T)^{-1} = G^{-1} - G^{-1}U(I + V^TG^{-1}U)^{-1}V^TG^{-1}.$$

Thus if m of the entries in D_Δ are updated during an iteration then suitably updating A_1^{-1} requires $O(\min\{ms^2v^2, s^3v^3\})$ operations during the iteration. In other words, if $m \gg sv$ then during the iteration it is more efficient to compute A_1^{-1} by directly inverting A_1 rather than performing m rank one changes. This discrepancy allows us to improve the time complexity by balancing of the number of operations as follows. Let δ be a parameter.

(a) Whenever the iteration number k is a multiple of $\lceil N^\delta \rceil$, we compute $(A_1)^{-1}$ by directly inverting A_1 in $O(s^3v^3)$ operations. Since the recomputation is performed $O(N^{1-\delta}L)$ times, the total number of operations for recomputation is $O(N^{1-\delta}s^3v^3L)$.

(b) Between successive recomputations of $(A_1)^{-1}$ we maintain $(A_1)^{-1}$ by performing rank one changes. By the Update Lemma in Section 4, there are at most $O(N^{2\delta})$ updates to the entries in D_Δ between successive recomputations of $(A_1)^{-1}$. Consequently, there are at most $O(N^{2\delta})$ rank one changes to $(A_1)^{-1}$ between successive recomputations. Since there are $O(N^{1-\delta}L)$ recomputations, the total number of rank one changes to $(A_1)^{-1}$ during the entire algorithm is $O(N^{1+\delta}L)$, and this leads to a total of $O(N^{1+\delta}s^2v^2L)$ operations for updating $(A_1)^{-1}$ via rank one changes.

Choosing $N^\delta = (sv)^{0.5}$ balances the operations in (a) and (b) above, and leads to a total of

$$\mathrm{O}(N^{1-\delta}s^3v^3L) = \mathrm{O}(N^{1+\delta}s^2v^2L) = \mathrm{O}(s^{3.5}v^{2.5}eL)$$

arithmetic operations for maintaining $(A_1)^{-1}$, since $N = \mathrm{O}(se)$.

The balancing techniques described here are quite general and are applicable whenever the constraint matrix may be expresed as $\binom{B_1}{B_2}$ where B_1 has much fewer rows than B_2 and B_2 has a special structure. Moreover, since the idea is to trade-off reinversion versus rank one updates it can be applied in the context of any interior point algorithm for linear programming.

We can account for the time complexity of the multicommodity flow algorithm as follows. In $\mathrm{O}(NL)$ iterations we either obtain a point $\hat{w}$ such that $p^\mathrm{T}\hat{w} \leqslant 2^{-\mathrm{O}(L)}$ or obtain a proof that the minimum value of the objective function in problem MF is strictly greater than zero which means that the given multicommodity flow problem is infeasible. Once $\hat{w}$ is available, an optimal solution to problem MF and a feasible multicommodity flow may be found in $\mathrm{O}(s^3v^2e)$ operations as described in Section 5. The total number of operations for maintaining $(A_1)^{-1}$ for $\mathrm{O}(NL)$ iterations is $\mathrm{O}(s^{3.5}v^{2.5}eL)$, and $\mathrm{O}(s^2v^2)$ operations are spent in computing w^{k+1} from w^k once $(A_1)^{-1}$ is available. Thus the total number of arithmetic operations performed by the multicommodity flow algorithm is $\mathrm{O}(s^{3.5}v^{2.5}eL)$.

3. Reducing cost of local optimizations

In this section we shall show how the special structure of the capacity constraints may be exploited to reduce the local optimization problem at each iteration to the problem of inverting a matrix A_1 of dimension sv. Throughout this section I will denote the identity matrix of the appropriate dimension. (The dimension will be clear from the context.) During the kth iteration w^{k+1} is obtained from w^k as follows.

Step 1. Find a direction w^d by solving the local optimization problem

$$\begin{aligned} \min \quad & p^\mathrm{T}Dw = \sum_{i=1}^{s}(D_y)_{ii}\,y_i \\ \text{s.t.} \quad & APD_xx = 0 \quad \text{(flow conservation)}, \\ & CD_xx + D_yy - cz = 0 \quad \text{(capacity constraints)}, \\ & e^\mathrm{T}w = 0, \\ & w^\mathrm{T}w \leqslant 1, \end{aligned}$$

where $D = \mathrm{diag}(D_x, D_y, 1)$, $D_x = \mathrm{diag}(x_1^k, x_2^k, \ldots, x_{n_1}^k)$, $D_y = \mathrm{diag}(y_1^k, y_2^k, \ldots, y_e^k)$, $e^\mathrm{T} = (1, \ldots, 1)$, and $e^\mathrm{T} = (e_x^\mathrm{T}, e_y^\mathrm{T}, 1)$.

Step 2. Compute $w' = (1/N)e + (\alpha/N)w^d$, where $w \in \mathbb{R}^N$ and $\alpha < \frac{1}{4}$.
Step 3. $w^{k+1} = (\sum_i c_i + 1)(Dw'/e^{\mathrm{T}}Dw')$.

Eliminating the capacity constraints by substituting $y = D_y^{-1}(-CD_x x + cz)$ reduces the local optimization problem in Step 1 above to

$$\begin{aligned} \min \quad & p_y^{\mathrm{T}}(-CD_x x + cz) \\ \text{s.t.} \quad & B\begin{pmatrix} x \\ z \end{pmatrix} = 0, \\ & (x^{\mathrm{T}}, z)Q\begin{pmatrix} x \\ z \end{pmatrix} \leqslant 1, \end{aligned}$$

where

$$Q = \begin{pmatrix} I + D_x C^{\mathrm{T}} D_y^{-2} C D_x & -D_x C^{\mathrm{T}} D_y^{-2} c \\ -c^{\mathrm{T}} D_y^{-2} C D_x & 1 + c^{\mathrm{T}} D_y^{-2} c \end{pmatrix}, \qquad B = \begin{pmatrix} APD_x & 0 \\ b_1^{\mathrm{T}} & d_2 \end{pmatrix},$$

$b_1 = e_x - D_x C^{\mathrm{T}} D_y^{-1} e_y$, $d_2 = 1 + e_y^{\mathrm{T}} D_y^{-1} c$, $e^{\mathrm{T}} = (e_x^{\mathrm{T}}, e_y^{\mathrm{T}}, 1)$, and $p^{\mathrm{T}} = (p_x^{\mathrm{T}}, p_y^{\mathrm{T}}, p_z)$. Eliminating the slacks y transforms the sphere $w^{\mathrm{T}}w \leqslant 1$ into the ellipsoid

$$(x^{\mathrm{T}}, z)Q\begin{pmatrix} x \\ z \end{pmatrix} \leqslant 1,$$

and the dimension of the constraint matrix is reduced from $sv + e$ to sv at the cost of slightly increasing the complexity of the quadratic constraint. (Q is expressible as the sum of a block diagonal matrix plus a rank two matrix.) Let

$$w^d = \begin{pmatrix} x^d \\ y^d \\ z^d \end{pmatrix}.$$

By the theory of convex functions and Lagrange multipliers [13], the solution $\binom{x^d}{z^d}$ to the reduced problem is given up to a scale factor λ by

$$\lambda\begin{pmatrix} x^d \\ z^d \end{pmatrix} = (I - Q^{-1}B^{\mathrm{T}}(BQ^{-1}B^{\mathrm{T}})^{-1}B)Q^{-1}\begin{pmatrix} -D_x C^{\mathrm{T}} p_y \\ p_y^{\mathrm{T}} c \end{pmatrix}, \qquad \text{(i)}$$

and y^d is given by

$$y^d = D_y^{-1}(-CD_x x^d + z^d c). \qquad \text{(ii)}$$

Next we shall show how to efficiently obtain expressions for Q^{-1} and $(BQ^{-1}B^{\mathrm{T}})^{-1}$ which involve the addition, subtraction, and multiplication of a constant number of matrices. The direction vector w^d may then ber obtained using the above equations. In the following computations we shall repeatedly use the Sherman–Morrison–Woodbury formula [3],

$$(G + UV^{\mathrm{T}})^{-1} = G^{-1} - G^{-1}U(I + V^{\mathrm{T}}G^{-1}U)^{-1}V^{\mathrm{T}}G^{-1}.$$

1. Q can be expressed as $Q = D_1 + U_1 V_1^{\mathrm{T}}$, where D_1 is a block diagonal matrix with each block of dimension at most s, and $U_1 V_1^{\mathrm{T}}$ is a matrix of rank 2. Specifically,

$$D_1 = \begin{pmatrix} D_{11} & 0 \\ 0 & d_1 \end{pmatrix},$$

where $D_{11} = I + D_x C^{\mathrm{T}} D_y^{-2} C D_x$, $d_1 = 1 + c^{\mathrm{T}} D_y^{-2} c$, and

$$U_1^{\mathrm{T}} = \begin{pmatrix} 0 \cdots 0 & 1 \\ -c^{\mathrm{T}} D_y^{-2} C D_x & 0 \end{pmatrix}, \qquad V_1^{\mathrm{T}} = \begin{pmatrix} -c^{\mathrm{T}} D_y^{-2} C D_x & 0 \\ 0 \cdots 0 & 1 \end{pmatrix}.$$

2. Let D_w be the diagonal matrix given by $D_w^2 = (I + D_y^{-1} C D_x^2 C^{\mathrm{T}} D_y^{-1})^{-1}$. Note that D_w^2 is computable in $O(se)$ operations. Then D_{11}^{-1} can be expressed as

$$D_{11}^{-1} = I - D_x C^{\mathrm{T}} D_y^{-1} D_w^2 D_y^{-1} C D_x$$

in $O(se)$ operations. Note that D_{11}^{-1} is a block diagonal matrix, with each block of size at most s, and explicitly computing all the entries in D_{11}^{-1} would require $O(s^2 e)$ operations. However, we do not explicitly compute all the entries in D_{11}^{-1}. Each block in D_{11}^{-1} may be represented as an identity matrix minus an outer product of vectors, namely as $I_\mu - \pi\pi^{\mathrm{T}}$ where I_μ is the $\mu \times \mu$ identity, π is a μ-dimensional vector and $\mu \leqslant s$. For each block we compute only the corresponding vector π rather than explicitly computing the outer product $\pi\pi^{\mathrm{T}}$; the block is implicitly represented by the vector π. Such a representation of D_{11}^{-1} has $O(se)$ non-zero entries and is computable in $O(se)$ operations. Since the only computation in which D_{11}^{-1} is involved consists of multiplication by some other matrix this representation is adequate. Next, computing $d_1 = 1 + c^{\mathrm{T}} D_y^{-2} c$ requires $O(e)$ operations, and thus D_1^{-1} can be expressed as

$$D_1^{-1} = \begin{pmatrix} D_{11}^{-1} & 0 \\ 0 & d_1^{-1} \end{pmatrix} = \begin{pmatrix} I - D_x C^{\mathrm{T}} D_y^{-1} D_w^2 D_y^{-1} C D_x & 0 \\ 0 & d_1^{-1} \end{pmatrix}$$

in $O(se)$ operations.

3. Q^{-1} can then be expressed as

$$Q^{-1} = D_1^{-1} - U_2 U_3 U_4^{\mathrm{T}} = \begin{pmatrix} I - D_x C^{\mathrm{T}} D_y^{-1} D_w^2 D_y^{-1} C D_x & 0 \\ 0 & d_1^{-1} \end{pmatrix} - U_2 U_3 U_4^{\mathrm{T}},$$

where $U_2 = D_1^{-1} U_1$, $U_3 = (I + V_1^{\mathrm{T}} D_1^{-1} U_1)^{-1}$, and $U_4 = D_1^{-1} V_1$, in $O(se)$ operations. This is because U_1 and V_1 are computable in $O(se)$ operations, and using the expression for D_1^{-1} in 2 above, U_2, $V_1^{\mathrm{T}} D_1^{-1} U_1$, U_3, and U_4 are also computable in $O(se)$ operations. The above expression for Q^{-1} contains $O(se)$ non-zeros.

4. We next note that

$$BQ^{-1}B^{\mathrm{T}} = BD_1^{-1}B^{\mathrm{T}} + (BU_2U_3)(BU_4)^{\mathrm{T}} = A_2 + U_5 V_5^{\mathrm{T}},$$

where

$$A_2 = \begin{pmatrix} A_1 & 0 \\ 0 & d_3 \end{pmatrix}, \qquad A_1 = APD_x D_{11}^{-1} D_x P^{\mathrm{T}} A^{\mathrm{T}}, \qquad d_3 = b_1^{\mathrm{T}} D_{11}^{-1} b_1 + d_2^2 d_1^{-1},$$

$$U_5^{\mathrm{T}} = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ & b_2^{\mathrm{T}} & & \\ & (BU_2U_3)^{\mathrm{T}} & & \end{pmatrix}, \quad V_5^{\mathrm{T}} = \begin{pmatrix} & b_2^{\mathrm{T}} & & \\ 0 & \cdots & 0 & 1 \\ & (BU_4)^{\mathrm{T}} & & \end{pmatrix}, \quad b_2^{\mathrm{T}} = ((APD_xD_{11}^{-1}b_1)^{\mathrm{T}}, 0).$$

We observe that once U_2, U_3, U_4 are available, U_5 and V_5 can both be computed in $\mathrm{O}(se)$ operations since both B and the expression of D_{11}^{-1} in 2 above contain $\mathrm{O}(se)$ non-zero entries. Also, d_3 is computable in $\mathrm{O}(se)$ operations.

5. $(BQ^{-1}B^{\mathrm{T}})^{-1}$ can now be written as

$$(BQ^{-1}B^{\mathrm{T}})^{-1} = A_2^{-1} - A_2^{-1}U_5(I + V_5^{\mathrm{T}}A_2^{-1}U_5)^{-1}V_5^{\mathrm{T}}A_2^{-1}.$$

We shall show four properties of A_1. First, once A_1^{-1} is available, the direction w^d may be obtained in $\mathrm{O}(s^2v^2)$ extra operations. Second, A_1 can be computed from D in $\mathrm{O}(s^2e)$ operations. Third, an update of an entry in D leads to at most a rank five change in A_1 and thereby to at most a rank five change in A_1^{-1}; this change in A_1 and A_1^{-1} is computable in $\mathrm{O}(s^2v^2)$ operations. Four, if each entry in D is multiplied by a scalar θ then each entry in A_1 gets multiplied by θ^2; so if D is changed by a scale factor then A_1 also gets changed by a scale factor. (As mentioned in Section 2, to reduce the time complexity we work with a suitable approximation D_Δ to D which is defined in Section 4; this corresponds to replacing D by D_Δ in all the expressions derived in this section and the four properties of A_1 hold with D replaced by D_Δ.)

First, we show that once A_1^{-1} is available, the direction w^d can be obtained in $\mathrm{O}(s^2v^2)$ operations. As noted in 4 above, computing d_3, U_5, and V_5 requires $\mathrm{O}(se)$ operations. So once A_1^{-1} is available, A_2^{-1}, $V_5^{\mathrm{T}}A_2^{-1}U_5$, and the expression in 5 above for $(BQ^{-1}B^{\mathrm{T}})^{-1}$ can be obtained in $\mathrm{O}(s^2v^2)$ extra operations. Moreover, the expression in 3 above for Q^{-1} is also computable in $\mathrm{O}(se)$ operations. Then since both B and the expression for Q^{-1} in 3 above contain $\mathrm{O}(se)$ non-zeros, from equations (i) and (ii) it follows that w^d can be computed in $\mathrm{O}(s^2v^2)$ additional operations. Thus once A_1^{-1} is available, finding the direction w^d takes $\mathrm{O}(s^2v^2)$ arithmetic operations.

Next, we show that computing A_1 from D requires $\mathrm{O}(s^2e)$ operations. Note that D_{11}^{-1} can be expressed as $D_{11}^{-1} = I - D_xC^{\mathrm{T}}D_y^{-1}D_w^2D_y^{-1}CD_x$ in $\mathrm{O}(se)$ operations. Also, $A_1 = APD_xD_{11}^{-1}D_xP^{\mathrm{T}}A^{\mathrm{T}}$. Thus A_1 may be written as $A_1 = A_{11} - A_{12}$ where $A_{11} = APD_x^2P^{\mathrm{T}}A^{\mathrm{T}}$, and $A_{12} = APD_x^2C^{\mathrm{T}}D_y^{-1}D_w^2D_y^{-1}CD_x^2P^{\mathrm{T}}A^{\mathrm{T}}$. We shall describe how to obtain each of A_{11} and A_{12} in $\mathrm{O}(s^2e)$ operations.

(a) APD_x is a block diagonal matrix with s blocks, the ith block being a weighted node–edge incidence matrix corresponding to commodity i. The dot product of two rows in distinct blocks of APD_x is zero. The dot product of two distinct rows in the same block of APD_x is non-zero only if there is an edge between the two vertices corresponding to the two rows, and this dot product can be evaluated in $\mathrm{O}(1)$ operations. So we may conclude that A_{11} has $\mathrm{O}(se)$ non-zero entries, and can be computed in $\mathrm{O}(se)$ operations.

(b) Each column of $APD_x^2C^{\mathrm{T}}D_y^{-1}D_w$ corresponds to an edge and is a weighted sum of the $\mathrm{O}(s)$ columns of A that correspond to the $\mathrm{O}(s)$ flow variables for the edge. Since each column of A has $\mathrm{O}(1)$ entries, each column of $APD_x^2C^{\mathrm{T}}D_y^{-1}D_w$ has $\mathrm{O}(s)$ entries.

There are s rows in $APD_x^2C^TD_y^{-1}D_w$ corresponding to a vertex in V; the dot product of two rows is non-zero only if there is an edge between the two vertices corresponding to the two rows or the rows correspond to the same vertex. The dot product of two rows corresponding to distinct vertices with an edge between them may be evaluated in O(1) operations; the dot product of two rows corresponding to the same vertex may be evaluated in a number of operations proportional to the degree of the vertex. Thus we may conclude that A_{12} has $O(s^2e)$ non-zero entries and is computable in $O(s^2e)$ operations.

Thus A_1 can be computed from D in $O(s^2e)$ arithmetic operations.

Third, we show that modifying an entry in D leads to at most a rank five change in A_1 and in A_1^{-1}. Now suppose that an entry in D is modified. This can lead to the update of at most a single entry in D_x^2, and hence to at most a rank one change in A_{11} since $A_{11} = APD_x^2P^TA^T$. The modification of an entry in D leads to the modification of at most a single entry in D_w^2, since D_w^2 is diagonal and can be written as $D_w^2 = (I + D_y^{-1}CD_x^2C^TD_y^{-1})^{-1}$. Also, the modification of an entry in D can lead to the modification of either an entry in D_x^2 or D_y^{-1} but not both. Thus an update of an element of D leads to at most a rank four change in A_{12}, since $A_{12} = APD_x^2C^TD_y^{-1}D_w^2D_y^{-1}CD_x^2P^TA^T$. Hence, A_1 can change by a matrix of rank at most five. In other words if A_1' is the new matrix obtained by changing an entry in D then A_1' is expressible as

$$A_1' = A_1 + UV^T,$$

where U and V each have at most five columns, and are both computable in $O(s^2e)$ operations. Furthermore, a rank five change in A_1 can lead to at most a rank five change in A_1^{-1} and this change is computable in $O(s^2v^2)$ arithmetic operations using the Sherman-Morrison-Woodbury formula [3,10].

Finally, suppose each element of D is multiplied by θ. Then D_{11} remains unchanged since $D_{11} = I + D_xC^TD_y^{-2}CD_x$, and each entry in A_1 is multiplied by θ^2 since $A_1 = APD_xD_{11}^{-1}D_xP^TA^T$. Thus multiplying each entry in D by a scalar θ is equivalent to multiplying each entry in A_1 by the scalar θ^2.

4. Balancing the number of arithmetic operations

In this section we show how to balance the number of arithmetic operations required to directly invert A_1 and to update A_1^{-1} via rank one changes over the entire course of the algorithm. As mentioned in Section 2 we use an approximation D_Δ instead of D in the local optimization at each iteration. D_Δ satisfies the condition that for $1 \leqslant i \leqslant N$, $(D_\Delta)_{ii} \in [(D)_{ii}/\sqrt{2}, \sqrt{2}(D)_{ii}]$, where $D = \mathrm{diag}(w_1^k, w_2^k, \ldots, w_N^k)$ and $w = (w_1^k, w_2^k, \ldots, w_N^k)$ is the point at the beginning of the kth iteration. We note that even though D_Δ is used instead of D, the number of iterations in the algorithm is still $O(NL)$ [6]. Initially, $D_\Delta = (w_1^0, w_2^0, \ldots, w_N^0)$. At the start of the kth iteration D_Δ is modified as follows. Let δ be a parameter.

Step 1. $D_\Delta := ((1/N)\sum_{i=1}^{N}(w_i^k/w_i^{k-1}))D_\Delta$.
Step 2. For $i = 1, 2, \ldots, N$, if $(D_\Delta)_{ii} \notin [w_i^k/\sqrt{2}, \sqrt{2}w_i^k]$ then $(D_\Delta)_{ii} = w_i^k$.
Step 3. If k is a multiple of $\lceil N^\delta \rceil$ then $D_\Delta := \operatorname{diag}(w_1^k, w_2^k, \ldots, w_N^k)$.

In Step 1 D_Δ is modified by a scale factor and as a result A_1 and A_1^{-1} are changed by a scale factor as described in Section 3. Modifying an element of D_Δ in Step 2 leads to at most a rank five change in A_1 and in A_1^{-1} as was shown in Section 3, and this change can be computed in $O(s^2v^2)$ operations. D_Δ gets reset in Step 3 whenever the iteration number k is a multiple of $\lceil N^\delta \rceil$, and A_1^{-1} has to be recomputed whenever D_Δ is reset. The parameter δ determines the trade-off between the number of operations for recomputing A_1 and the number of operations for maintaining A_1^{-1} via rank one updates.

The total number of operations for maintaining A_1^{-1} during the entire course of the algorithm is obtained as follows. Whenever D_Δ is reset in Step 3, we recompute A_1^{-1} by directly inverting A_1 in $O(s^3v^3)$ operations. Between successive resettings of D_Δ in Step 3 (i.e., between successive recomputations of A_1^{-1}), A_1^{-1} is maintained by performing rank one updates at the cost of $O(s^2v^2)$ operations per rank one update. Since the recomputation of A_1^{-1} is performed $O(N^{1-\delta}L)$ times, the total number of operations spent in recomputing A_1^{-1} during the entire algorithm is $O(N^{1-\delta}s^3v^3L)$. From the Update Lemma below it follows that the number of updates to D_Δ in Step 2 between successive resettings in Step 3 is $O(N^{2\delta})$. Hence the total number of updates to entries in D_Δ in Step 2 during the entire algorithm is $O(N^{1+\delta}L)$. Since a modification of an entry in D_Δ in Step 2 leads to at most five rank one changes in A_1^{-1}, the total number of rank one updates to A_1^{-1} during the entire algorithm is $O(N^{1+\delta}L)$. Hence the total number of operations to maintain A_1^{-1} via rank one changes is $O(N^{1+\delta}s^2v^2L)$ over the entire course of the algorithm. Choosing $N^\delta = (sv)^{0.5}$ balances the total number of arithmetic operations needed for recomputing and for performing rank one changes. Then since $N = O(se)$, we get that the total number of arithmetic operations for maintaining A_1^{-1} is

$$O(N^{1-\delta}s^3v^3L) = O(N^{1+\delta}s^2v^2L) = O(s^{3.5}v^{2.5}eL).$$

A natural question to ask is "how much does the balancing save us over just performing rank one corrections to A_1^{-1}". If we were to perform only rank one corrections to A_1^{-1} (without any recomputations), the total number of operations to maintain A_1^{-1} would be $O(s^{3.5}v^2e^{1.5}L)$ whereas with recomputation and balancing the number of operations reduces to $O(s^{3.5}v^{2.5}eL)$. Thus balancing reduces the number of operations by $(e/v)^{0.5}$ which is substantial for dense networks.

Next, we prove the Lemma that bounds the total number of updates to D_Δ in Step 2 between successive resettings in Step 3. The naive bound on the number of updates that would follow from the results in [4] would be $O(N^{0.5+\delta})$. However, one can get a better bound by formalizing the following intuitive argument:

Only those coordinates of w that change significantly can lead to rank one corrections; if the relative change in w_i accumulated over N^δ iterations is small then $(D_\Delta)_{ii}$ cannot be updated in Step 2 during those N^δ iterations.

Update Lemma. *Between successive resettings of D_Δ in Step 3, the total number of modifications to elements of D_Δ in Step 2 is $O(N^{2\delta})$ if $\delta < \frac{1}{2}$.*

Proof. Let n_i be the number of times $(D_\Delta)_{ii}$ is modified in Step 2 between successive resettings in Step 3. For convenience let the iteration numbers between the successive resettings of D_Δ under consideration range from 0 to $\lceil N^\delta \rceil - 1$. Let $d_i^k = ((1/N)\sum_{i=1}^N (w_i^k/w_i^{k-1}))^{-1}(w_i^k/w_i^{k-1})$, $h_i^k = \ln(d_i^k)$, and J^k be the set of those indices i such that $|d_i^k - 1| \geqslant 1/16\lceil N^\delta \rceil$. For $1 \leqslant i \leqslant N$, let $\pi_i = \{k\colon\ 0 \leqslant k \leqslant \lceil N^\delta \rceil - 1,\ i \in J^k\}$, and $\theta_i = \{k\colon\ 0 \leqslant k \leqslant \lceil N^\delta \rceil - 1,\ i \in J^k\}$. Thus if d_i^k is very close to 1 then iteration number k is in the set θ_i, whereas if d_i^k differs from 1 by at least $1/16\lceil N^\delta \rceil$ then iteration number k is in π_i.

Since $(D_\Delta)_{ii}$ is modified whenever the product of successive d_i^k's exceeds $\sqrt{2}$ or falls below $1/\sqrt{2}$, we have that for $1 \leqslant i \leqslant N$,

$$n_i \ln\sqrt{2} \leqslant \sum_{k=0}^{\lceil N^\delta \rceil - 1} |h_i^k|. \tag{i}$$

As $\sum_{k\in\theta_i} |h_i^k| \leqslant \frac{1}{8}$ from (i) we get that

$$n_i \geqslant 1 \ \Rightarrow\ n_i \ln\sqrt{2} \leqslant 2\sum_{k\in\pi_i} |h_i^k|. \tag{ii}$$

In [6, Section 6.3] it is shown that for all k,

$$\sum_{i=1}^N (d_i^k - 1)^2 \leqslant \beta^2, \quad \text{for some constant } \beta < 1, \tag{iii}$$

and that

$$\sum_{i=1}^N |h_i^k - (d_i^k - 1)| \leqslant \frac{\beta^2}{2(1-\beta)}. \tag{iv}$$

For each $i \in J^k$, $|d_i^k - 1| \geqslant 1/16\lceil N^\delta \rceil$, and so from (iii) we get that $|J^k| \leqslant (16\lceil N^\delta \rceil)^2$ and that for all k,

$$\sum_{i\in J^k} |d_i^k - 1| \leqslant 16\lceil N^\delta \rceil \beta. \tag{v}$$

Thus (from (iv) and (v))

$$\sum_{i\in J^k} |h_i^k| \leqslant \sum_{i=1}^N |h_i^k - (d_i^k - 1)| + \sum_{i\in J^k} |d_i^k - 1| \leqslant \frac{\beta^2}{2(1-\beta)} + 16\lceil N^\delta \rceil \beta.$$

So from (ii) we get that

$$\sum_{i=1}^{N} n_i \leqslant 8\sum_{i=1}^{N}\sum_{k\in\pi_i}|h_i^k| = 8\sum_{k=0}^{\lceil N^\delta\rceil -1}\sum_{i\in J^k}|h_i^k| = O(N^{2\delta}\beta).$$

Thus $\sum_{i=1}^{N} n_i = O(N^{2\delta}\beta)$. □

At this point we note that the above analysis depends on the algorithm taking small steps.

5. Finding an exact optimum from an approximate optimum

In this section we shall show how to obtain an optimal solution to problem MF from an approximate optimum $\hat{w}$ in $O(s^3v^2)$ arithmetic operations, each operation being performed to a precision of $O(L)$ bits. Let us assume that we have a feasible point

$$\hat{w} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{pmatrix}$$

such that $p^T\hat{w} \leqslant 2^{-k_1L}$, where k_1 is a suitably large constant. Let C_l^T denote the lth row of C. Note that the objective function $p^Tw = \sum_{l=1}^{s} y_l = \sum_{l=1}^{s}(c_l - C_l^Tx)$. Also, note that for any feasible point $z = 1$. Hence $\hat{x}$ satisfies the conditions

$$\sum_{l=1}^{s}(c_l - C_l^T\hat{x}) \leqslant 2^{-k_1L},$$

$$AP\hat{x} = 0,$$

$$C\hat{x} < c,$$

$$\hat{x} > 0.$$

Let Π be the polytope given by

$$\Pi = \left\{ x\colon \sum_{l=1}^{s}(c_l - C_l^Tx) \leqslant 2^{-k_1L},\ APx = 0,\ Cx \leqslant c,\ x \geqslant 0 \right\}.$$

Let $C(x)$ be a submatrix of C such that C_l^T is a row of $C(x)$ if $|C_l^Tx - c_l| \leqslant 2^{-k_1L}$. Let $c(x)$ be the subvector of c corresponding to $C(x)$. Let $I(x)$ be a submatrix of the identity matrix I such that the jth row of I is a row of $I(x)$if $|x_j| \leqslant 2^{-k_1L}$. Let

$$B = \begin{bmatrix} \sum_{l=1}^{s} C_l^T \\ AP \end{bmatrix}, \quad \text{and} \quad M(x) = \begin{bmatrix} B \\ C(x) \\ I(x) \end{bmatrix}.$$

Computing an optimal vertex

To obtain an optimal vertex from an approximate optimum $\hat{x}$ we follow the procedure in [9, pp. 173–174]. Suppose that $x' \in \Pi$, and that there exists a row r^{T} of $[\,{}^{C}_{I}\,]$ such that r^{T} is linearly independent of the rows of $M(x')$. Since r^{T} does not lie in the row space of $M(x')$, the following system of linear equations has a solution:

$$M(x')x = 0,$$

$$r^{\mathrm{T}}x = 1.$$

The orthogonal projection d of r onto the subspace $\{x\colon\ M(x')x = 0\}$ is a solution to the above system up to a scale factor. We can find a suitable scalar λ such that $x' + \lambda d$ is in the polytope Π and the rank of $M(x' + \lambda d)$ is strictly greater than the rank of $M(x')$. (We essentially move in the direction d till we hit a bounding plane of the polytope Π.) We can thus create a sequence of points $\hat{x} = x^0, x^1, \ldots, x^m$ $(m \leqslant se)$ such that each x^i is in the polytope Π, the rank of $M(x^{i+1})$ is strictly greater than the rank of $M(x^i)$, and the row space of $M(x^i)$ is contained in the row space of $M(x^{i+1})$. (The rank of $M(x^m)$ equals the dimension of x i.e., n_1.) An optimal vertex x^* is obtained as the solution to the following system of linear equations [9, pp. 173–174].

$$\sum_{l=1}^{s}(c_l - C_l^{\mathrm{T}}x^*) = 0,$$

$$APx^* = 0,$$

$$C(x^m)x^* = c(x^m),$$

$$I(x^m)x^* = 0.$$

Let $x^{i+1} = x^i + d^i\lambda^i$, where d^i is the non-zero projection of some row of $[\,{}^{C}_{I}\,]$ onto the subspace $\{x\colon\ M(x^i)x = 0\}$ and λ^i is a scalar. We check if a row of $[\,{}^{C}_{I}\,]$ is linearly independent of the rows of $M(x^i)$ by computing its orthogonal projection onto the subspace $\{x\colon\ M(x^i)x = 0\}$; if the projection is non-zero then it serves as the direction d_i. Once we know that a row of $[\,{}^{C}_{I}\,]$ is linearly dependent on the rows of $M(x^i)$, we do not need to compute its projection again, since the row space of $M(x^{i+1})$ includes the row space of $M(x^i)$. Thus there are $O(se)$ projection computations. We shall show below that the number of operations to compute all the projections is $O(s^3v^2e)$. Once d^i is available, λ^i may be obtained in $O(se)$ operations, since computing the product of $[\,{}^{C}_{I}\,]$ with d^i takes $O(se)$ operations. This leads to a total of $O(s^3v^2e)$ arithmetic operations for generating the sequence $x^0 = \hat{x}, x^1, \ldots, x^m$ and computing x^* from x^m.

Projection computation

To efficiently compute the projections, we maintain a matrix

$$\begin{bmatrix} B \\ C' \\ I' \end{bmatrix}$$

such that the rows of this matrix form a basis for the row space of $M(x^i)$ (at the start of the $(i+1)$st step). Here C' and I' are submatrices of C and I respectively. Let J' be the index set defined as $j \in J'$ if the jth colum of I' contains a 1. Let $\overline{B}$ ($\overline{C}$) be the matrix obtained from B (C') by dropping every column j such that $j \in J'$. ($\overline{B}$ and $\overline{C}$ have $n_1 - |J'|$ columns.) Let $\overline{x}$ ($\overline{r}$) be the $n_1 - |J'|$ dimensional vector obtained from x (r) by dropping each coordinate x_j (r_j) such that $j \in J'$. The problem of projecting r onto the subspace $\{x\colon\ M(x^i)x = 0\}$ is equivalent to the problem of projecting $\overline{r}$ onto the subspace $\{\overline{x}\colon\ \overline{B}\overline{x} = 0,\ \overline{C}\overline{x} = 0\}$.

Let $G = \overline{BB}^{\mathrm{T}} - \overline{BC}^{\mathrm{T}}(\overline{CC}^{\mathrm{T}})^{-1}\overline{CB}^{\mathrm{T}}$. We shall show that once G^{-1} is available, the required projection of $\overline{r}$ may be obtained in $\mathrm{O}(s^2v^2)$ operations. We shall also show that maintaining G^{-1} requires a total of $\mathrm{O}(s^3v^2e)$ arithmetic operations. Since there are $\mathrm{O}(se)$ projection computations, this gives a total of $\mathrm{O}(s^3v^2e)$ operations for all projection computations.

First, we show that once G^{-1} is available, computing the projection of $\overline{r}$ takes $\mathrm{O}(s^2v^2)$ operations. The required projection of $\overline{r}$ is given by

$$\overline{r} - [\ \overline{B}^{\mathrm{T}}\ \ \overline{C}^{\mathrm{T}}\ ] \begin{bmatrix} \overline{BB}^{\mathrm{T}} & \overline{BC}^{\mathrm{T}} \\ \overline{CB}^{\mathrm{T}} & \overline{CC}^{\mathrm{T}} \end{bmatrix}^{-1} \begin{bmatrix} \overline{B} \\ \overline{C} \end{bmatrix} \overline{r}.$$

Since each of $\overline{B}$, $\overline{C}$ contains $\mathrm{O}(se)$ non-zeros, it suffices to show that once G^{-1} is available, solving the system of linear equations

$$\begin{bmatrix} \overline{BB}^{\mathrm{T}} & \overline{BC}^{\mathrm{T}} \\ \overline{CB}^{\mathrm{T}} & \overline{CC}^{\mathrm{T}} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \tag{i}$$

takes $\mathrm{O}(s^2v^2)$ operations. Premultiplying both sides of (i) by

$$\begin{bmatrix} I & -\overline{BC}^{\mathrm{T}}(\overline{CC}^{\mathrm{T}})^{-1} \\ 0 & I \end{bmatrix}$$

we get

$$\begin{bmatrix} \overline{BB}^{\mathrm{T}} - \overline{BC}^{\mathrm{T}}(\overline{CC}^{\mathrm{T}})^{-1}\overline{CB}^{\mathrm{T}} & 0 \\ \overline{CB}^{\mathrm{T}} & \overline{CC}^{\mathrm{T}} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{bmatrix} I & -\overline{BC}^{\mathrm{T}}(\overline{CC}^{\mathrm{T}})^{-1} \\ 0 & I \end{bmatrix} \begin{pmatrix} g_1 \\ g_2 \end{pmatrix}. \tag{ii}$$

Since $\overline{CC}^{\mathrm{T}}$ is diagonal and since each of $\overline{B}$, $\overline{C}$, has $\mathrm{O}(se)$ non-zeros, from (ii) it follows that once $G^{-1} = (\overline{BB}^{\mathrm{T}} - \overline{BC}^{\mathrm{T}}(\overline{CC}^{\mathrm{T}})^{-1}\overline{CB}^{\mathrm{T}})^{-1}$ is available, the solution to (i) above can be obtained in $\mathrm{O}(s^2v^2)$ operations.

Next, we count the number of operations for maintaining G^{-1}. Note that $\overline{CC}^{\mathrm{T}}$ is a diagonal matrix. Adding an extra row to I' corresponds to setting another coordinate of x to zero; this adds an index to J' and leads to a column being dropped from $\overline{B}$ and $\overline{C}$. Adding a row to C' corresponds to adding a row to $\overline{C}$. Thus if a row is added to I' or C' then at most a single entry is affected in $\overline{CC}^{\mathrm{T}}$, and at most a single column is affected in $\overline{B}$ as well as $\overline{BC}^{\mathrm{T}}$. Consequently, adding a row to I' or C' leads to at most a rank three change in $G = \overline{BB}^{\mathrm{T}} - \overline{BC}^{\mathrm{T}}(\overline{CC}^{\mathrm{T}})^{-1}\overline{CB}^{\mathrm{T}}$ and in G^{-1}, and this change can be computed in $O(s^2v^2)$ operations using the Morrison–Woodbury rank one update formula [3,10]. Since at most se rows may be added to I' and C' during the computation of $x^0, x^1, \ldots, x^m$, the number of rank one changes to G is $O(se)$. Hence the total number of operations for maintaining G^{-1} is $O(s^3v^2e)$.

Finally, we briefly discuss the precision requirement. The subspace $\{x\colon\ M(x^i)x = 0\}$ has a basis such that each vector in the basis has rational coordinates with a common denominator of magnitude at most 2^L. Thus if a row r of $[\,^C_I\,]$ has a non-zero projection into this subspace then the 2-norm of this projection is at least 2^{-4L}. Hence it suffices to compute the projection of r to an accuracy of say k_1L bits, and from the computed approximate projection we may correctly determine whether r is or is not linearly dependent on the rows of $M(x^i)$. To guarantee such a bound on the error in the projection, it suffices to perform arithmetic operations to a precision of k_2L bits, and maintain G^{-1} to an accuracy of k_2L bits, for a suitably large constant $k_2 > k_1$. The growth of the error in G^{-1} during rank one updates is controlled as described in [11]. The point x^m thus computed could be slightly infeasible, however, the point x^* will still be an optimal vertex [9, pp. 173–174].

6. Concluding remarks

We have shown how to speed up Karmarkar's algorithm for the case of multicommodity flows. This gives an algorithm for the multicommodity flow problem which requires $O(s^{3.5}v^{2.5}eL)$ arithmetic operations, each operation being performed to a precision of $O(L)$ bits. We conclude with the following remarks.

(1) The proof of the claim that $O(L)$ bits of precision is adequate for arithmetic operations is a detailed but straightforward exercise and is left to the reader. To obtain this bound on the precision one first proves a bound of $2^{O(L)}$ on the norms and the condition numbers of all the intermediate matrices arising in the computation, and one then applies standard theorems in [3,10]. The growth of the error in A_1^{-1} during rank one updates is controlled as described in [11]. For similar but detailed arguments concerning precision of arithmetic operations required for linear programming and convex quadratic programming the reader may refer to [5,11].

(2) The techniques in this paper can also be applied to speed up the linear programming algorithm in [11] for the case of multicommodity flows. This gives a slightly faster algorithm for the multicommodity flow problem which requires $O(s^3v^{2.5}e^{0.5}L + s^3v^2e)$ arithmetic operations. The application of the two ideas discussed in Section 2 to the algorithm in [11] proceeds in a manner identical to the one in this paper.

(3) The techniques in this paper can also be applied to speed up linear programs whose structure is similar to that of the multicommodity flow problem. Speed up is obtained when the constraint matrix B can be expressed as $\binom{B_1}{B_2}$ where B_1 has much fewer rows compared to B_2, and B_2 has some special structure. Examples of such

problems are given in [1,2]. Interestingly enough, the multicommodity flow problem with costs on the flows in each edge can be solved in the same time complexity as that obtained in this paper for the plain multicommodity flow problem without edge costs.

References

[1] J. Birge and L. Qi, "Computing block-angular Karmarkar projections with applications to stochastic programming," Technical Report, Department of Industrial and Operations Engineering, University of Michigan (Ann Arbor, MI, 1986).

[2] V. Chvatal, *Linear Programming* (Freeman, New York, 1983).

[3] G.H. Golub and C.F. Van Loan, *Matrix Computations* (The Johns Hopkins University Press, Baltimore, MA, 1983).

[4] T.C. Hu, *Integer Programming and Network Flows* (Addison-Wesley, Reading, MA, 1969).

[5] S. Kapoor and P.M. Vaidya, "Fast algorithms for convex quadratic programming and multicommodity flows," *Proceedings 18th Annual ACM Symposium Theory of Computing* (1986), pp. 147-159.

[6] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica* 4(4) (1984) 373-395.

[7] L.G. Khachian, "Polynomial algorithms in linear programming," *USSR Computational Mathematics and Mathematical Physics* 20 (1979) 191-194.

[8] M.K. Kozlov, S.P. Tarasov and L.G. Khachian, "Polynomial solvability of convex quadratic programming," *Doklady Akademii Nauk SSSR* 5 (1979) 1051-1053.

[9] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[10] G.W. Stewart, *Introduction to Matrix Computations* (Academic Press, New York, 1973).

[11] P.M. Vaidya, "An algorithm for linear programming which requires $O(((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations," in: *Proceedings 19th Annual ACM Symposium Theory of Computing* (1987) pp. 29-38. [Extended version in: *Mathematical Programming* 47 (1990) 175-201.]

[12] J.H. Wilkinson, *The Algebraic Eigenvalue Problem* (Oxford University Press, Oxford, 1965).

[13] G. Zoutendijk, *Mathematical Programming Methods* (North-Holland, New York, 1976).