DIMACS Series in Discrete Mathematics
and Theoretical Computer Science
Volume 00, 0000

A Fast Implementation of a Path-Following Algorithm for Maximizing a Linear Function Over A Network Polytope

ANIL JOSHI, ARTHUR S. GOLDSTEIN, AND PRAVIN M. VAIDYA

December 31, 1992

1. Introduction

Many network optimization problems - Assignment, Maximum Flow, and Minimum Cost Flow - can be cast as Linear Programming (LP) problems. Recently several algorithms have been proposed for LP which belong to the general class of algorithms known as Interior Point methods (See [**Vai90, Vai89, Ren88, Kar84**]). The objective of our implementation is to show that path-following interior point algorithms can work well if the problem peculiarities are taken into consideration and appropriate data-structures are chosen. Computational experiments comparing the performance of Network Simplex and Interior Point methods have been carried out by several researchers. The results can be found in [**AM91, KY91, RV90, Raj89, A$^+$85**].

We use the notation e for the number of edges and v for the number of vertices in the given network. A network flow problem is defined as the following LP [**FF62**].

$$\max c^T x$$

subject to the lower and upper bound (*i.e.* capacity) constraints

$$l_i \leq x_i \leq u_i$$

and the flow conservation constraints

$$Ax = b$$

1991 *Mathematics Subject Classification.* Primary 90B10, 90C05, 90-04; Secondary 49M35, 65K05.

where $x \in \Re^e$ is the flow vector whose ith element is the flow in edge i. l_i and u_i are the lower and upper bounds respectively on the flow in edge i. In addition, these capacities are assumed to be integral. Matrix $A \in \Re^{(v-1)\times e}$ describes the network under consideration. A is obtained from the node-arc incidence matrix of the network by dropping the row corresponding to a pre-specified ground node. Each column of A corresponds to an edge of the network. If the edge k is directed from node i to node j then there is a -1 in the ith row and a $+1$ in the jth row in column k of A (If edge k is incident on the ground node there is a $+1$ or a -1 in column k of A). All other entries in that column are zero. Vector $b \in \Re^{v-1}$ corresponds to the node flow values (with appropriate sign for a source or a sink). Vector $c \in \Re^e$ is the cost vector.

Any algorithm for LP can be employed to solve this problem. But since we know that this is a network flow problem, it is possible to speed up the LP algorithm by taking advantage of the special structure of matrix A and the form of the upper and lower bound constraints. This formulation is suitable for Assignment, Maximum Flow and Minimum-Cost Flow problems after some minor transformations are applied to the network, the cost vector c and the node-flow vector b. In Section 2 the algorithm used to solve this problem is described (See [**Vai90**] for a detailed description and rate-of-convergence). One of the time consuming computations in the algorithm is to solve a sparse linear system at each step. We use a Pre-conditioned Conjugate Gradient (PCCG) method to solve the linear system (See [**GL89**]). In Section 3, the construction of the pre-conditioner is described. Section 4 describes additional heuristics such as reduction of problem size by fixing flow values in certain edges, approximate centering, etc. that may be employed to speed up the algorithm in practice. The experimental results are presented in Section 5 and analyzed.

2. An Overview of the Path-Following Algorithm

The network polytope is defined by the upper and lower bound constraints which are the bounding hyperplanes of the polytope. It is denoted as

$$P = \{x : Ax = b,\ l_i \leq x_i \leq u_i\}.$$

The path-following algorithm employed in this implementation follows the trajectory of analytic centers of a family of polytopes defined by the intersection of the network polytope and the objective function hyperplane, *i.e.* the centers of the family of polytopes $P(\beta)$ defined as:

$$P(\beta) = P\ \cap\ \{x : c^T x \geq \beta\}.$$

The analytic center of $P(\beta)$ is the minimizer of the strictly convex logarithmic barrier function $F(x, \beta)$ where

$$F(x,\beta) = -(\sum_{i=1}^{e}(\ln(x_i - l_i) + \ln(u_i - x_i)) + w\ln(c^T x - \beta)).$$

We put a large weight w on the objective function hyperplane so that the analytic center is pushed away from the objective function hyperplane $c^T x = \beta$ towards the optimal facet of the network polytope. Specifically the weight w equals the number of edges e times a large multiplier (which is chosen to be $\min\{\sqrt{e}, \sqrt{100,000}\}$).

Clearly, when the flow in any edge equals either the capacity, the lower bound or is such that $c^T x = \beta$ then this function goes to ∞. At any point which is strictly in the interior of the polytope $P(\beta)$ the above function is finite. The path-following algorithm proceeds in several phases. Initially we start with a strictly interior point $x^{(0)}$ which is a good approximation to the analytic center of $P(-\infty)$ and $\beta^{(0)} = -\infty$. The path-following algorithm generates a sequence of parameters $\beta^{(0)}, \beta^{(1)}, \ldots, \beta^{(k-1)}, \beta^{(k)}, \ldots$ whose limit is the maximum value of $c^T x$ over the network polytope P, and also a sequence of points $x^{(0)}, x^{(1)}, \ldots, x^{(k-1)}, x^{(k)}, \ldots$ such that $x^{(k)}$ is a good approximation to the analytic center of $P(\beta^{(k)})$. The final β is very close to the optimal value of $c^T x$ which enables us to isolate the optimal facet of the network polytope together with an exact optimum. As β is varied from $-\infty$ to its maximum possible value in a continuous manner, a continuous trajectory is generated that would start from the analytic center of the network polytope P and terminate at a point that maximizes the objective function $c^T x$ over P. This trajectory is characterized by

$$\Delta\tau + tc = A^T \lambda$$

where $t \in \Re$, $t \geq 0$. $\Delta \in \Re^{e \times e}$ is a diagonal matrix whose ith diagonal entry equals $1/(l_i - x_i) + 1/(u_i - x_i)$ (See [**Vai90, Vai89, Ren88**] for more details). $\tau \in \Re^e$ is a vector of all 1's and $\lambda \in \Re^{v-1}$ is a vector of node potentials. Note that here $\Delta\tau$ can be interpreted as a vector of "reduced costs". In each phase, we find $x^{(k)}$ from $x^{(k-1)}$ by applying Newton's method to the logarithmic barrier function $F(x, \beta^{(k)})$ starting from $x^{(k-1)}$.

A Step of Newton's Method. A step of Newton's method applied to minimizing $F(x, \beta)$ proceeds as follows. Let $\tilde{x}$ be the current point. Let η be the gradient and Q be the Hessian of $F(x, \beta)$ at the current point . We first compute a direction ξ by solving the local optimization problem

$$\max \; -\eta^T \xi$$

subject to

$$A\xi = 0$$

and

$$\xi^T Q \xi \leq 1 \; ,$$

and then (approximately) minimize $F(x, \beta)$ on the line $\tilde{x} + t\xi$. For details see [**Vai89**]. During the line minimization, we express the logarithmic barrier function as a function of a single parameter t and then do a bisection search.

Isolating an Exact Optimum. Since the capacities are integral, the analytic center has the property that any edge flow that is not fixed to either the upper or the lower bound on the optimal facet must be at least $1/(e+1)$ away from both the bounds (See [**Vai90, Vai89, Ren88**] for details). Using this property we adaptively try to isolate an optimal point in the following manner. Suppose that we have a good approximation to the center of the polytope $P(\beta)$ for some β (for a measure of goodness of the approximation see section 4). Then any edge flow that is less than $1/(2e)$ from either the upper (resp. lower) bound is guaranteed to be fixed to the upper (resp. lower) bound at the optimum (See [**Vai90, Vai89, Ren88**] for details). We fix each such edge flow to the closer of the two bounds and obtain a flow that may not satisfy the conservation constraints $Ax = b$.

Let

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad l = \begin{bmatrix} l_1 \\ l_2 \end{bmatrix} \quad u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad A = \begin{bmatrix} A_1 & A_2 \end{bmatrix}$$

where subscript 1 indicates those edges fixed to appropriate bounds. Let $\tilde{x} = \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{bmatrix}$ denote the solution before any edge flows are fixed to upper or lower bounds. Let $\hat{x}_1$ be the flow vector obtained from $\tilde{x}_1$ after fixing edge flows to the closer of the two bounds. Since the flow vector $\begin{bmatrix} \hat{x}_1 \\ \tilde{x}_2 \end{bmatrix}$ may not satisfy conservation constraints we need to obtain an $\hat{x}_2$ satisfying the conservation constraints

$$A_2\hat{x}_2 = b - A_1\hat{x}_1 = \hat{b}_2.$$

We obtain $\hat{x}_2$ from $\tilde{x}_2$ by applying a correction to edge flows in a spanning forest (Though any spanning forest will do, we find a maximum spanning forest with edge weights as defined in section 3, subsection **Reducing to a Full-Dimensional Form**, since these edges have large slacks). Next we check if $\hat{x} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix}$ satisfies the capacity and lower bound constraints. If it does then we have a feasible solution $\hat{x}_2$ to the reduced problem

$$\min c_2^T x_2$$

subject to

$$A_2 x_2 = \hat{b}_2$$

$$l_2 \leq x_2 \leq u_2.$$

The correction to $\tilde{x}_2$ may be computed as follows. Let

$$x_2 = \begin{bmatrix} x_{2_f} \\ x_{2_o} \end{bmatrix}$$

where x_{2_f} corresponds to edges in a spanning forest and x_{2_o} corresponds to edges outside the forest (Here the spanning forest is in the network defined by the edges corresponding to x_2). We want to find a correction $\varepsilon = \begin{bmatrix} \varepsilon_f \\ 0 \end{bmatrix}$ such that

$$\hat{x}_2 = \tilde{x}_2 + \varepsilon$$

satisfies conservation constraints. Let

$$c_2 = \begin{bmatrix} c_{2_f} \\ c_{2_o} \end{bmatrix} \quad l_2 = \begin{bmatrix} l_{2_f} \\ l_{2_o} \end{bmatrix} \quad u_2 = \begin{bmatrix} u_{2_f} \\ u_{2_o} \end{bmatrix} \quad A_2 = \begin{bmatrix} A_{2_f} & A_{2_o} \end{bmatrix}.$$

The correction ε may be obtained by solving the linear system

$$A_{2_f}\varepsilon_f = \hat{b}_2 - A_2\tilde{x}_2.$$

This may be done in linear time since A_{2_f} corresponds to a spanning forest.

By further eliminating the variables x_{2_f} (i.e. those in a spanning forest) we can reduce the problem to full-dimensional form:

$$\min\ (c_{2_o} - A_{2_o}^T (A_{2_f}^{-1})^T c_{2_f})^T x_{2_o}$$

subject to

$$l_{2_f} \le A_{2_f}^{-1}(\hat{b}_2 - A_{2_o}x_{2_o}) \le u_{2_f}$$

$$l_{2_o} \le x_{2_o} \le u_{2_o}.$$

If the reduced cost $c_{2_o} - A_{2_o}^T (A_{2_f}^{-1})^T c_{2_f}$ is zero (we test for numerical zero) then the flow $\hat{x}$ is guaranteed to be optimal.

To summarize, we carry out the following procedure after every centering step in the final phases of the algorithm. Fix all the flows that are within $1/(2e)$ from the upper (resp. lower) bound to the upper (resp. lower) bound. Find a maximum spanning forest in the network with the edges corresponding to these fixed variables removed. Apply a correction to the variables in this spanning forest (as described above) and check if the resulting flow is feasible. If this flow is feasible, we have a facet, namely, the one where the variables in x_1 are fixed, which is a candidate for being the optimal facet. If the reduced cost function $c_{2_o} - A_{2_o}^T (A_{2_f}^{-1})^T c_{2_f}$ is zero then this facet is declared to be the optimal and the algorithm stops. Otherwise the algorithm continues. Note that the algorithm always halts with an exact optimum. Specifically, this procedure allows us to isolate the entire optimal facet along with a good approximation to the center of the optimal facet.

If an optimal vertex is desired, the cost vector may be modified by adding a "small" random perturbation. An optimal solution to this modified problem is also optimal for the original problem, and with high probability the optimal facet of the modified problem is a single vertex (See [**Cha52**]). As this procedure requires too many bits of precision in the perturbed cost vector, it was not implemented. Instead our algorithm isolates a good approximation to the analytic center of the optimal facet.

Getting an Initial Feasible Point. To get an initial interior feasible point, the flow in each edge is set midway between the upper and lower bound. Since such a flow may violate the conservation constraints, we introduce extra edges to a distinguished ground node to carry the excess or deficiency from those nodes where the conservation constraint is not satisfied. We put a high cost on each of these edges to ensure that at an optimum the flows in these edges are zero provided the original problem is feasible. We use a couple of simple heuristics to estimate the cost on these newly introduced edges; for example, letting the cost of a new edge be greater than the cost of a Maximum Spanning Tree (MST) of (absolute) costs suffices.

3. Solving the Linear System at Each Step

During each step of the Newton's Method (while computing the center), we have to find a direction ξ which is the solution to the local optimization problem

$$\max -\eta^T \xi$$

subject to

$$A\xi = 0$$

and

$$\xi^T Q \xi \leq 1 \ ,$$

Matrix Q is the Hessian of the logarithmic barrier function $F(x, \beta)$ and can be expressed as $D + \hat{w}cc^T$ where D is a diagonal matrix whose ith diagonal entry is

$$D_{ii} = \frac{1}{(x_i - l_i)^2} + \frac{1}{(u_i - x_i)^2}$$

and $\hat{w} = w/(c^T x - \beta)^2$ where w is a problem dependent multiplier. The above problem can be reduced to full-dimensional form by eliminating the flow variables in a spanning tree using the conservation constraints $A\xi = 0$.

Reducing to a Full-Dimensional Form. Let $\tilde{x}$ be the current point and $\tilde{D}$ the matrix D evaluated at $\tilde{x}$. Let the weight of edge i with respect to the current point $\tilde{x}$ be defined as $\tilde{D}_{ii}^{-\frac{1}{2}}$. We reduce the above problem to full-dimensional form by eliminating the flow variables corresponding to the edges in a specific spanning tree, namely a Maximum Spanning Tree(MST) with edge weights defined above. Intuitively the MST edges are those edges that have the "largest" slacks for changing the flow. We split the variables of ξ into two blocks - those belonging to the MST, denoted by ξ_z, and those that do not, denoted by ξ_y. Let

$$\begin{aligned} \xi &= \begin{bmatrix} \xi_z \\ \xi_y \end{bmatrix} \\ A &= \begin{bmatrix} A_z & A_y \end{bmatrix} \\ D &= \begin{bmatrix} D_z & 0 \\ 0 & D_y \end{bmatrix} \\ c &= \begin{bmatrix} c_z \\ c_y \end{bmatrix} \end{aligned}$$

be the block representation of the vectors and matrices involved in the above local optimization problem, where subscript z (y) represents the columns/rows corresponding to the MST (non-MST) variables. Let $B = -A_z^{-1} A_y$. Eliminating the MST variables reduces the above local optimization problem to

$$\max \ -\hat{\eta}^T \xi_y$$

subject to

$$\xi_y^T \hat{Q} \xi_y \leq 1 \ , \tag{1}$$

where $\hat{\eta} = B^T \eta_z + \eta_y$, $\hat{Q} = B^T D_z B + D_y + \hat{w}\hat{c}\hat{c}^T$, and $\hat{c} = B^T c_z + c_y$.

The direction ξ_y may be obtained by solving the linear system

$$\hat{Q}\xi_y = -\hat{\eta} \tag{2}$$

and then ξ_z may be computed as $\xi_z = B\xi_y$. We solve the linear system (2) by a Pre-conditioned Conjugate Gradient (PCCG) method (See [**GL89**]) with $M = D_y + \hat{w}\hat{c}\hat{c}^T$ as the pre-conditioner. Note that $\hat{Q}$ may be expressed as

$$\hat{Q} = M + B^T D_z B$$

Since we eliminated variables ξ_z corresponding to an MST, it is guaranteed that the absolute value of any entry in $D_z^{\frac{1}{2}} B D_y^{-\frac{1}{2}}$ is bounded by 1 and thereby the trace of $M^{-\frac{1}{2}} B^T D_z B M^{-\frac{1}{2}}$ is bounded by ve. The bound on any entry in $D_z^{\frac{1}{2}} B D_y^{-\frac{1}{2}} = D_z^{\frac{1}{2}} A_z^{-1} A_y D_y^{-\frac{1}{2}}$ may be obtained as follows:

Consider a column a_y of A_y corresponding to a non-tree edge e. Let Π be the unique path in the MST between the end points of e. Note that a numbering of the columns of A_z naturally induces a numbering of the edges in the MST. The ith entry in $A_z^{-1} a_y$ is ± 1 if the ith edge in the MST lies on the path Π; otherwise the entry is zero. So a non-zero entry in $D_z^{\frac{1}{2}} A_z^{-1} a_y D_y^{-\frac{1}{2}}$ is of the form w_e/w where w_e is the weight of edge e and w is the weight of some edge on path Π. It then follows that the absolute value of any entry in $D_z^{\frac{1}{2}} A_z^{-1} A_y D_y^{-\frac{1}{2}}$ is bounded by 1 (See [**Vai91**] for details).

The above argument ensures that M is a good pre-conditioner for $\hat{Q}$. As we approach a vertex solution, entries in $D_z^{\frac{1}{2}}$ remain bounded because they correspond to edges with non-zero slack whereas entries in $D_y^{-\frac{1}{2}}$ approach zero. Hence, the closer we are to a vertex solution, the better is the pre-conditioner.

In our experiments, it has been observed that the PCCG with this pre-conditioner M halts in about 60-70 iterations on the average (with a solution of desired accuracy) for problems with up to 250,000 edges. Note that solving a linear system $Mu = r$ can be trivially accomplished in $O(e)$ operations, and we never need to form the matrix $\hat{Q}$ (or any other matrix) explicitly. Moreover, multiplication of $\hat{Q}$, A or A^T by a vector may be performed in O(e) operations, and $A_z u = r$ may be solved in O(v) operations, using a standard graph adjacency list data structure.

MST Computation. For the sake of efficiency, we round off the edge weights to multiples of $(1+\delta)$ for a suitable δ and use Bucket Sorting coupled with Kruskal's MST algorithm (See [**TCR90**]). Using such an approximate MST - instead of an exact MST - only slightly degrades the quality of the pre-conditioner M.

Reducing the Constants in PCCG. All the intermediate vectors generated in our implementation of PCCG lie in a Krylov Space of dimension at most $v+1$ rather than e. This may be seen as follows:

Note that $\hat{Q}$ may be written as $\hat{Q} = D_y + UU^T$ where $U = [\hat{w}^{\frac{1}{2}}c \quad B^T D_z^{\frac{1}{2}}]$, and $U \in \Re^{e \times v}$. Let $\tilde{Q} = D_y^{-\frac{1}{2}} \hat{Q} D_y^{-\frac{1}{2}} = I + \tilde{U}\tilde{U}^T$ where $\tilde{U} = D_y^{-\frac{1}{2}} U$. Instead of directly solving $\hat{Q}\xi_y = -\hat{\eta}$ we first compute $\tilde{\xi}$ by solving

$$\tilde{Q}\tilde{\xi} = -D_y^{-\frac{1}{2}} \hat{\eta} = \tilde{\eta}$$

and then compute $\hat{\xi}$ as $\hat{\xi} = D_y^{\frac{1}{2}} \tilde{\xi}$. During the solution of $\tilde{Q}\tilde{\xi} = \tilde{\eta}$ by PCCG successive vectors are generated as linear combinations of previous vectors or by multiplying previous vectors by M^{-1} or $\tilde{Q}$ (See [**GL89**]). The initial vector is $\tilde{\eta}$. Note that if $x \in span\{\tilde{\eta}, \tilde{u}_1, \ldots, \tilde{u}_v\}$ where $\tilde{u}_1, \ldots, \tilde{u}_v$ are the columns of $\tilde{U}$ then both $M^{-1}x$ and $\tilde{Q}x$ lie in the span of $\{\tilde{\eta}, \tilde{u}_1, \ldots, \tilde{u}_v\}$. As a result all the vectors generated during PCCG lie in the Krylov space given by $span\{\tilde{\eta}, \tilde{u}_1, \ldots, \tilde{u}_v\}$ and may be implicitly represented by $v+1$ coefficients.

Thus most of the intermediate calculations may be implicitly performed using vectors of size $v+1$; this leads to reduced constants in the running time as well as storage and to an improvement in practical performance.

4. Some Techniques to Improve Speed and Stability

In our implementation we use several techniques to gain additional speed and improve the stability of the algorithm. We describe them in this section.

Approximate Centering. In the initial phases of the algorithm, it suffices to compute a very coarse analytic center. A good approximation to the center is required only when the algorithm is getting ready to isolate the optimal facet. So in the initial phases we find a very coarse approximation to the center which is sufficiently far from the objective function hyperplane, and eventually switch to an almost exact centering when the width of the polytope in the direction of the cost vector falls below a certain threshold. Closeness to the center is measured by the quantity $\hat{\eta}^T \hat{Q}^{-1} \hat{\eta}$. Initially a coarse approximation x to the center such that $\hat{\eta}^T \hat{Q}^{-1} \hat{\eta} \leq \epsilon e$ (ϵ a small constant) suffices. Later on when we are close to isolating the optimal facet, we switch to a finer approximation x satisfying the condition $\hat{\eta}^T \hat{Q}^{-1} \hat{\eta} \leq 0.5$. Note that for a point x close to the center ω, $\hat{\eta}^T \hat{Q}^{-1} \hat{\eta} \approx 2(F(x, \beta) - F(\omega, \beta))$ (See [**Vai89, Ren88, Kar84**] for theoretical results). The empirical data suggests that in practice the complexity of the two centerings is much better than the theoretical bounds.

Variable PCCG Stopping Threshold. The farther we are from the analytic center the larger may be the error in the Newton direction ξ without degrading the rate of convergence to the center. As a result, stopping (error) threshold for the PCCG can be coarser in the beginning when we are far from the center and may be progressively tightened as we approach the center. The PCCG as implemented in our algorithm stops when

$$r^T M^{-1} r \leq \mathit{threshold}$$

where r is the current residue and M is the preconditioner as defined earlier. When we are not too close to the center i.e. $\hat{\eta}^T \hat{Q}^{-1} \hat{\eta} \geq 0.1w$ where w is the weight on the objective function plane, the *threshold* is set to $\max\{10^{-4}w, 0.1\}$. When we are close to the center i.e. $\hat{\eta}^T \hat{Q}^{-1} \hat{\eta} < 0.1w$, the *threshold* is set to 0.1.

Reducing the Problem Size. Since the capacities are integral, the center has the property that any edge flow that is not fixed to either the upper or the lower bound on the optimal facet must be at least $1/(e+1)$ away from both the bounds. This may be argued as follows:

At the analytic center ω of the polytope $P(\beta) = P \cap \{x : c^T x \geq \beta\}$ we have that

$$\sum_{i=1}^{e} \left(\frac{u_i - x_i}{u_i - \omega_i} + \frac{x_i - l_i}{\omega_i - l_i} \right) + w \frac{c^T x - \beta}{c^T \omega - \beta} = w + 2e$$

(See [**Vai90, Ren88**] for details). At any optimum point $x^\star$, we have that $c^T x^\star - \beta \geq c^T \omega - \beta$. Thus

$$\sum_{i=1}^{e} \left(\frac{u_i - x_i^\star}{u_i - \omega_i} + \frac{x_i^\star - l_i}{\omega_i - l_i} \right) \leq 2e$$

Also, each term in the summation on the left hand side in the above inequality is at least 1. Thus for each i and each optimal point $x^\star$ we have

$$\frac{u_i - x_i^\star}{u_i - \omega_i} + \frac{x_i^\star - l_i}{\omega_i - l_i} \leq e + 1 \tag{3}$$

Furthermore, for any coordinate x_i either x_i is fixed to l_i or u_i on the optimal facet or the width of the optimal facet in the direction of x_i is at least 1 since vertices have integer coordinates. Suppose x_i is not fixed on the optimal facet.

Then there exist optimal points x' and x'' such that $u_i - x'_i \geq 1$ and $x''_i - l_i \geq 1$. Then from inequality (3) it follows that $u_i - \omega_i \geq 1/(e+1)$ and $\omega_i - l_i \geq 1/(e+1)$. Using this property, we fix those edge flows which at the current approximate center are closer than $1/(2e)$ to a bound to that bound. For these edges the optimal flow value is the bound it is fixed to. The edges that are fixed may be removed from further consideration in the algorithm.

Dynamic Shift of Origin for Stability. If either the upper or the lower bound of an edge is large in magnitude there is a chance of numerical instability as the edge flow approaches a bound in the final stages of the algorithm. A typical machine has about 15 to 16 decimal digits of precision. However, if we have a capacity of 10^8 we effectively could loose half the digits of precision. For example, suppose the current flow in an edge is 10^{-8} less than the capacity which is 10^8 and the machine has 16 decimal digits of precision. Now if the flow moves further towards the capacity, the flow value would be set to the capacity as a result of numerical error thereby leading to a division by zero in the intermediate calculations since the algorithm assumes a strictly interior point. To remove instabilities of this kind, we apply a dynamic shift-of-origin changing those bounds on edges which have a flow close to them to zero and working with the offset of the flow on such edges. At the end of the algorithm, the origin is shifted back to get the correct flows. As a result, an edge flow can approach a bound to essentially within machine precision. Note that even though the analytic centers generated during the algorithm cannot be too close to a boundary, the intermediate points generated in moving from one center to the next get quite close to some of the boundaries. It is because of this that dynamic shift of origin is required for large problems.

5. Experimental Results and Analysis

MINFLOW, our implementation for minimum cost flow, was tested on several problems using a Sun Sparc-2 workstation running SunOS 4.1.1 with FPU version 3. It was experimentally observed that vector inner product and saxpy operations ran at about 0.7 to 1.0 MEGAFLOPS. In order to estimate the speed of pointer manipulations several examples of standard breadth-first-search using adjacency list representation were run, and the time for breadth-first-search was estimated to be about 3 microseconds per edge. All maximum flow problems were converted to minimum cost flow problems and then solved using MINFLOW. The following tables and graphs show the experimental data along with timing and iteration counts. I/O times are not included in the timing data. The time and iteration counts to find the optimal solution are averaged over the number of instances of each size that was run.

Seeds were selected according to the following rule for all families reported except mesh-standard-one, mesh-standard-two, mesh-standard-four and mesh-standard-eight. Suppose the total number of instances for a given size is I.

Then the seed for instance i, $1 \leq i \leq I$, is given by the formula $5+(i-1)*2$. For families mesh-standard-one, mesh-standard-two, mesh-standard-four and mesh-standard-eight the formula is $1+(i-1)*2$.

One of the referees pointed out that NETFLO uses a slow insertion sort while reading in the data and suggested that we run bench marks to see how this affects the comparisons between MINFLOW and NETFLO. From each family we selected two instances of moderate size and timed the initial insertion sort in the NETFLO code. It was found that for families capt-hi-bits, capt-hi-bits-mod, capt-lo-bits and capt-lo-bits-mod the percentage time spent in the initial insertion sort is in the range 60%–64%. For washing-line-moderate this percentage is 47%. For acyclic-dense, genrmf-long and grid-long this initial time is moderately small (in the range of 13%–25%). For the remaining eleven families, this time is negligible (below 8%). This fact should be taken into consideration while examining the timing data for NETFLO.

The data for each input class consists of:

- A table which gives the absolute timing statistics for MINFLOW, NETFLO and RELAXT III along with the standard deviation.
- A table which gives the iteration count statistics for MINFLOW, NETFLO and RELAXT III along with the standard deviation. Each iteration for MINFLOW is a PCCG iteration in which the work done is proportional to the number of edges. A NETFLO iteration consists of moving from one vertex to to another in the network simplex algorithm [**KH80**]. A RELAXT III iteration corresponds to a relaxation step as described in [**Ber91, BT90, BT88a, BT88b**].
- Graphs for comparison with NETFLO and RELAXT III codes. The average of the running time ratios are plotted against problem size represented by the number of nodes on a log-log scale. This is done because for small problems, both NETFLO and RELAXT III are typically much faster than MINFLOW but as the problem size increases the situation reverses for several input classes. For input sizes where the graph is above the abscissa of 1.0, MINFLOW is slower compared to the other code (NETFLO or RELAXT III) and in the region where the graph is less than the abscissa of 1.0, MINFLOW is faster. Note that we were unable to obtain some of the data points for comparison with RELAXT III because the RELAXT III code failed to solve those problems.

Note that we have shortened RELAXT III to RELAXT in the tables and graphs.

One of the referees also suggested modifying the parameters to the CAPT generator so that the number of sinks is roughly $N/2$ where N is the number of vertices minus two. So the CAPT families were run with two types of parameter settings: one, as per DIMACS recommendations, the results of which are given under capt-lo-bits and capt-hi-bits; and two, as per the referee's recommendations, the results of which are summarized under capt-lo-bits-mod and capt-hi-bits-mod. In particular, the parameters *flow* and *intlen* were chosen as *flow*= 2∗*intlen*= 2000 for capt-lo-bits-mod and capt-hi-bits-mod.

We have included the results for the maximum flow problem families for comparison purposes only. So, while examining the statistics for families acyclic-dense, washing-line-moderate, genrmf-long and genrmf-wide, which are maximum flow problems converted to mincost flow problems, the fact that *the simplex algorithm is known to perform poorly on mincost flow problems derived from maximum flow problems* should be borne in mind. Furthermore, inclusion of these results indicate that Interior Point methods work well uniformly irrespective of the origins of the problem.

The smaller problems give trends for the comparative growth rates for the three codes. Since it was too time consuming to run many instances for the large problem sizes, we typically ran 1 or 2 instances only. The main memory in our machine was sufficient to run problems with a little over a 100,000 arcs. For all input classes we ran as big an instance as would fit in virtual memory without thrashing. As a result only a few data points were obtained for input classes acyclic-dense, capt-lo-bits, capt-lo-bits-mod, capt-hi-bits and capt-hi-bits-mod for which the number of arcs grew quadratically with the number of nodes.

A cursory analysis indicates a growth rate of about $O(e^{1.5})$ for MINFLOW on all the problems.

For input families washing-line-moderate, genrmf-long, genrmf-wide, grid-den-sixteen, grid-inc-den, grid-square, and grid-wide, the graphs indicate that MINFLOW has a slower growth rate than both NETFLO and RELAXT III with increasing problem size.

For grid-den-eight, MINFLOW is faster than NETFLO and RELAXT III, even for small problem sizes.

For acyclic-dense and grid-long, MINFLOW has faster growth rate than both NETFLO and RELAXT III.

For capt-lo-bits, capt-lo-bits-mod, capt-hi-bits and capt-hi-bits-mod only four data points for each family have been obtained due to memory restrictions. The trend seems to be that MINFLOW has lower growth rate than NETFLO but higher than RELAXT III for these families.

For mesh-standard-one, mesh-standard-two, mesh-standard-four and mesh-standard-eight MINFLOW has lower growth rate than NETFLO but seems to have the same growth rate as RELAXT III.

6. Conclusions

The implementation is quite robust, numerically stable, and fairly insensitive to variations in costs and capacities. Furthermore, we can handle problems with non-integer (real) costs since the stopping criterion does not depend on integrality of costs.

Size		#	Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
64	2016	13	19.2	0.8	0.7	1.0	0.0	0.1
128	8128	13	141.0	3.7	2.8	9.5	0.1	0.7
256	32640	13	1049.1	21.0	21.3	28.3	0.7	14.7

Size		#	Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
64	2016	13	349.9	794.7	103.5	15.8	93.0	46.4
128	8128	13	812.8	2761.5	177.8	63.8	353.9	88.0
256	32640	13	2002.8	4861.9	457.5	53.7	669.5	226.6

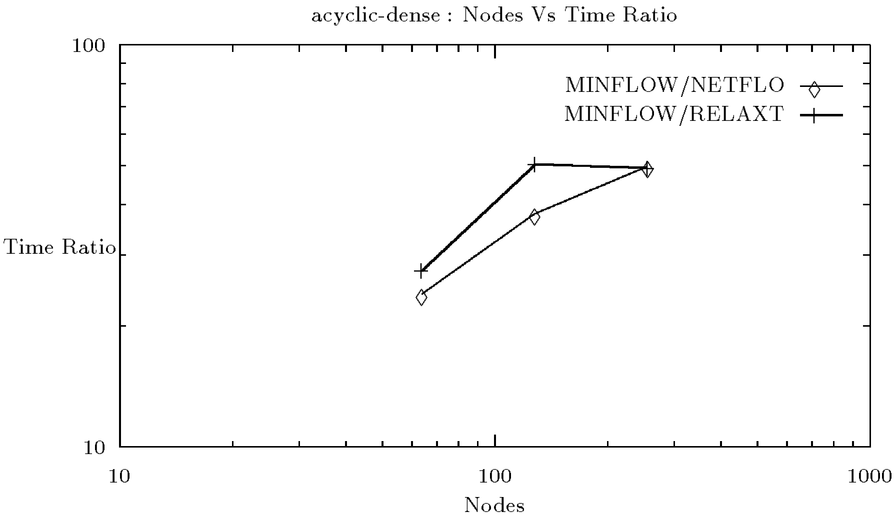

FIGURE 1. Statistics for family **acyclic-dense**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
66	127	10	1.7	0.1	0.0	0.2	0.0	0.0
130	373	10	5.3	0.3	0.2	0.4	0.0	0.0
258	1001	10	16.4	1.0	0.9	1.3	0.1	0.2
514	3006	10	67.1	5.4	4.5	15.4	0.3	0.7
1026	8071	10	226.8	27.7	23.6	37.3	1.2	3.3
2050	24276	10	708.8	193.7	136.2	90.5	6.4	20.0
4098	65002	1	3464.6	1088.6	621.7	0.0	0.0	0.0

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
66	127	10	124.6	86.4	25.2	19.2	9.0	7.7
130	373	10	209.4	261.4	76.7	24.4	37.3	24.0
258	1001	10	338.5	580.6	168.9	29.0	108.1	54.6
514	3006	10	627.8	1282.2	357.5	145.4	134.4	69.4
1026	8071	10	1034.3	2696.7	747.1	178.9	165.4	113.3
2050	24276	10	1412.6	5712.0	1538.2	181.9	476.3	232.2
4098	65002	1	3119.0	11063.0	2820.0	0.0	0.0	0.0

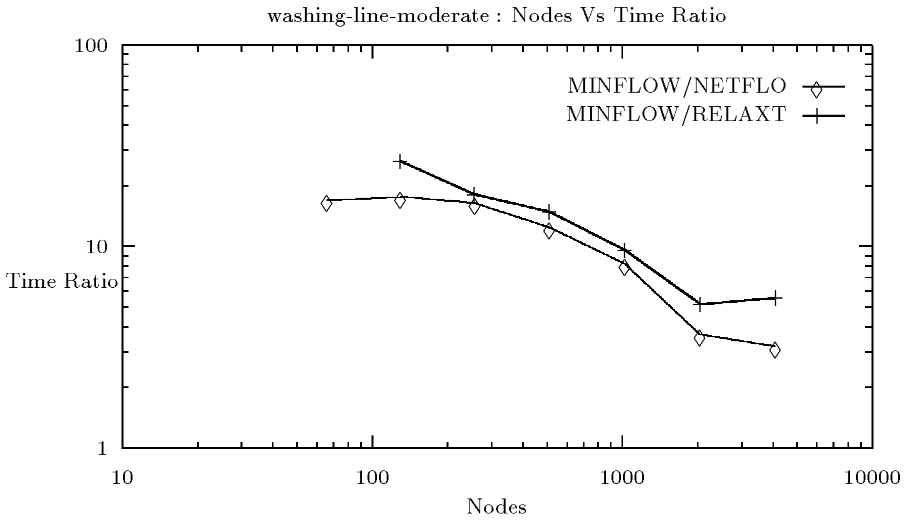

FIGURE 2. Statistics for family **washing-line-moderate**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
72	255	10	1.8	0.1	0.1	0.1	0.0	0.0
192	752	10	7.0	0.4	0.5	0.6	0.0	0.0
256	1008	10	10.5	0.6	0.9	0.7	0.0	0.0
575	2390	10	30.7	3.0	4.5	0.7	0.1	0.5
1152	4956	10	80.5	10.8	19.8	3.9	0.3	1.0
2254	9933	10	182.2	41.2	80.7	6.8	1.5	3.6
4096	18368	10	406.7	120.9	276.0	17.0	5.5	24.7
9100	41760	10	1260.0	628.8	1500.2	59.3	33.3	123.6
18432	85872	10	6858.3	2539.0	6958.9	1427.3	65.5	293.3

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
72	255	10	83.6	126.5	41.9	5.5	6.2	7.1
192	752	10	119.0	387.3	123.1	8.2	21.3	13.9
256	1008	10	124.0	496.5	149.8	7.8	21.1	14.3
575	2390	10	165.3	1259.4	394.2	7.8	42.6	41.7
1152	4956	10	225.1	2684.5	857.4	20.9	83.6	54.1
2254	9933	10	250.9	5397.5	1871.4	15.6	159.0	85.8
4096	18368	10	298.5	9641.5	3374.9	20.8	298.7	301.0
9100	41760	10	392.0	22591.8	7763.6	30.8	846.4	590.1
18432	85872	10	730.4	48517.3	16825.2	146.9	862.1	682.3

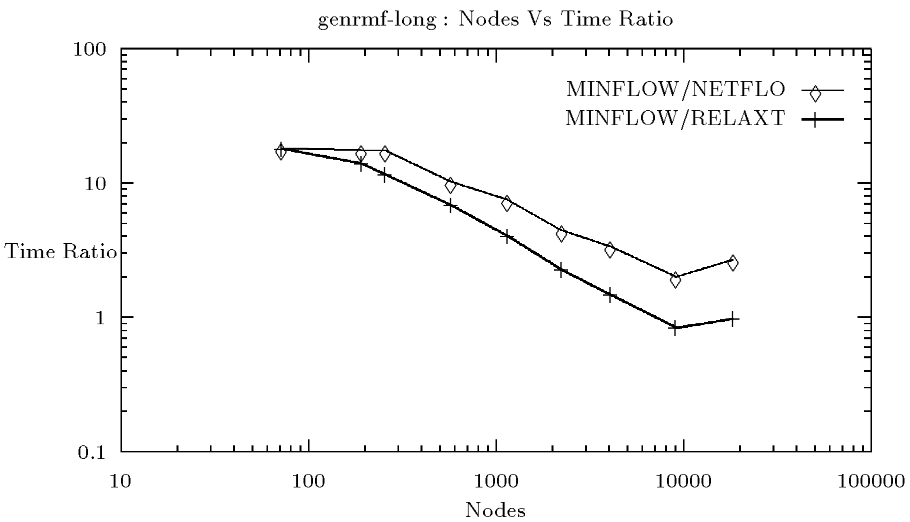

FIGURE 3. Statistics for family **genrmf-long**

Size		#	Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
72	276	10	2.1	0.1	0.1	0.2	0.0	0.0
147	602	10	5.9	0.3	0.4	0.5	0.0	0.0
300	1280	10	17.0	0.8	1.2	0.9	0.0	0.1
676	3003	10	54.5	3.3	7.0	1.0	0.1	0.5
1024	4608	10	96.5	5.3	16.6	2.1	0.2	1.0
1936	8844	10	243.2	15.4	58.5	12.7	0.5	0.0
3920	18256	10	623.1	59.7		16.2	2.4	
16807	80262	1	10468.2	1308.9		0.0	0.0	

Size		#	Iterations			Iterations (Std.dev)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
72	276	10	124.2	120.4	38.2	9.8	2.2	1.5
147	602	10	179.2	281.0	137.8	12.8	3.7	25.2
300	1280	10	270.6	586.6	330.3	12.9	24.3	22.6
676	3003	10	390.4	1820.8	740.8	8.5	91.9	63.2
1024	4608	10	498.3	2909.8	1142.3	16.5	115.9	56.5
1936	8844	10	708.3	5284.7	2206.0	48.5	208.0	0.0
3920	18256	10	917.2	11785.8		32.9	419.2	
16807	80262	1	3959.0	78634.0		0.0	0.0	

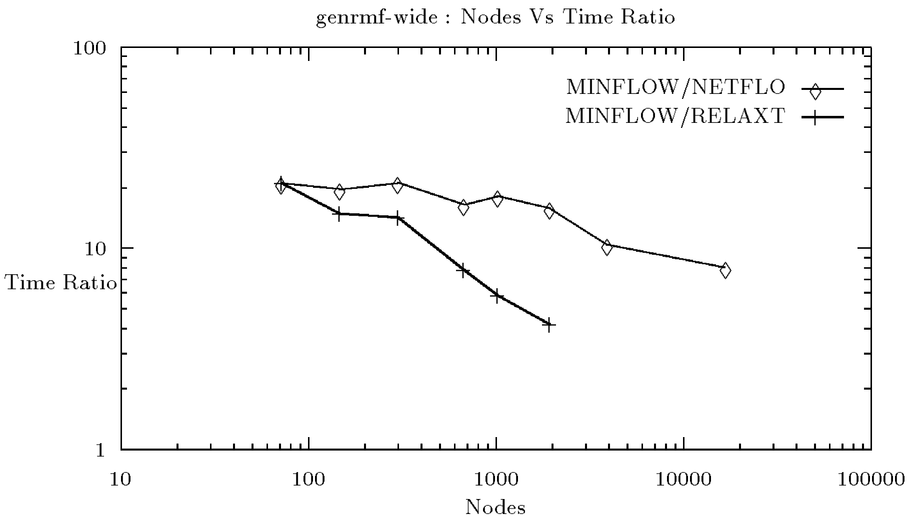

FIGURE 4. Statistics for family **genrmf-wide**

Size		#	Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	2048	10	26.5	2.8	18.2	1.7	0.1	8.8
512	4096	10	56.1	9.9	75.2	1.8	0.4	11.1
1024	8192	10	148.1	34.5	313.4	12.5	0.9	47.1
2048	16384	10	338.5	134.7	1215.2	25.9	7.5	116.2
4096	32768	10	707.2	716.0	4658.0	17.5	27.7	619.1
8192	65536	2	1856.6	2504.3	25876.0	23.7	59.1	1063.1
16384	131072	1	4139.1	10068.9	102480.5	0.0	0.0	0.0
32768	262144	1	16566.1			0.0		

Size		#	Iterations			Iterations (Std.dev)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	2048	10	150.2	5878.6	53197.5	20.0	205.7	21464.5
512	4096	10	137.5	15952.7	170049.1	14.2	736.8	26027.1
1024	8192	10	199.8	29290.1	521763.9	18.2	537.4	73833.2
2048	16384	10	235.7	68639.6	1353897.0	40.0	2465.5	131036.8
4096	32768	10	267.4	373951.4	2154553.1	20.5	11898.2	521322.2
8192	65536	2	252.5	331879.0	10645328.0	14.5	2165.0	232605.0
16384	131072	1	354.0	745617.0	29090153.0	0.0	0.0	0.0
32768	262144	1	1576.0			0.0		

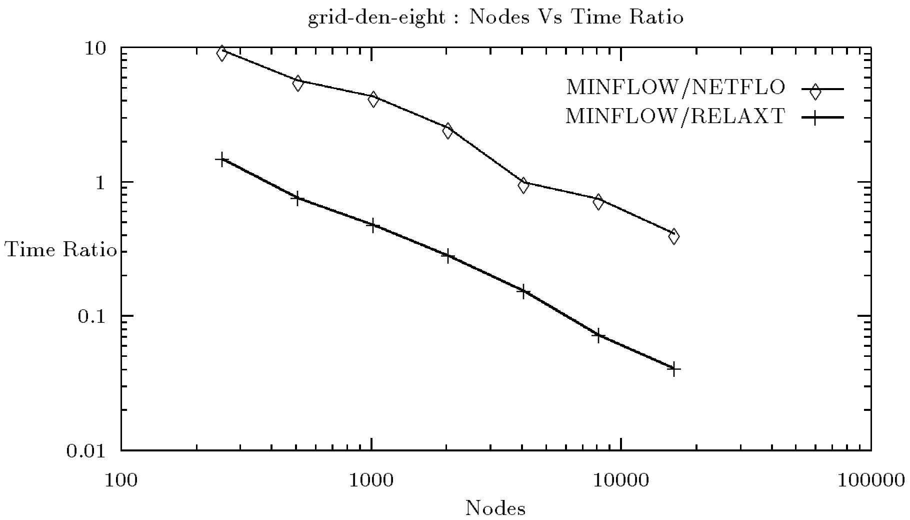

FIGURE 5. Statistics for family **grid-den-eight**

Size		#	Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	4096	8	55.5	5.6	97.7	1.6	0.2	9.5
512	8192	10	132.5	18.7	328.8	7.5	0.3	39.3
1024	16384	10	317.7	74.6	1232.1	16.9	1.6	99.6
2048	32768	10	759.7	287.8	4041.4	20.6	6.0	265.7
4096	65536	10	1666.5	1250.7	15730.9	84.5	38.7	1358.1
8192	131072	2	5517.7	4998.6	58637.1	218.2	63.1	2081.5
16384	262144	1	13211.3	20026.6		0.0	0.0	

Size		#	Iterations			Iterations (Std.dev)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	4096	8	202.7	14467.7	165768.2	16.8	701.0	16869.1
512	8192	10	251.9	32430.8	413513.6	17.0	455.5	46136.0
1024	16384	10	347.1	75449.5	1110089.5	16.2	1416.7	129243.0
2048	32768	10	431.7	177102.0	2585272.4	12.8	3244.6	229859.9
4096	65536	10	470.1	459134.6	6166728.4	38.7	13582.9	820072.0
8192	131072	2	697.0	842259.0	14155666.5	4.0	6441.0	577765.5
16384	262144	1	744.0	1949450.0		0.0	0.0	

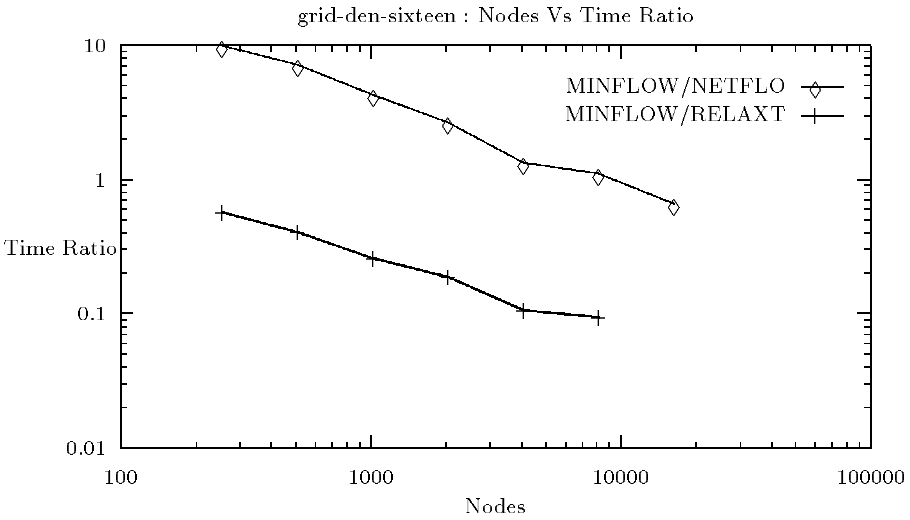

Figure 6. Statistics for family **grid-den-sixteen**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	4096	10	54.9	5.6	97.3	1.8	0.2	9.3
512	11586	10	199.6	29.9	485.1	10.6	0.6	63.6
1024	32768	10	720.0	183.5	2537.8	23.0	3.9	199.1
2048	92682	10	3005.6	1297.4	12802.8	118.2	20.9	457.9
4096	262144	1	12669.0	12088.7	40994.6	0.0	0.0	0.0

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	4096	10	201.9	14467.7	165768.2	16.3	701.0	16869.1
512	11586	10	312.4	47781.6	454180.0	20.8	742.0	81517.8
1024	32768	10	527.8	186138.8	1387404.4	21.1	3880.1	165211.7
2048	92682	10	1046.4	637883.4	3666692.0	37.4	34136.0	221686.4
4096	262144	1	1248.0	2514153.0	8293935.0	0.0	0.0	0.0

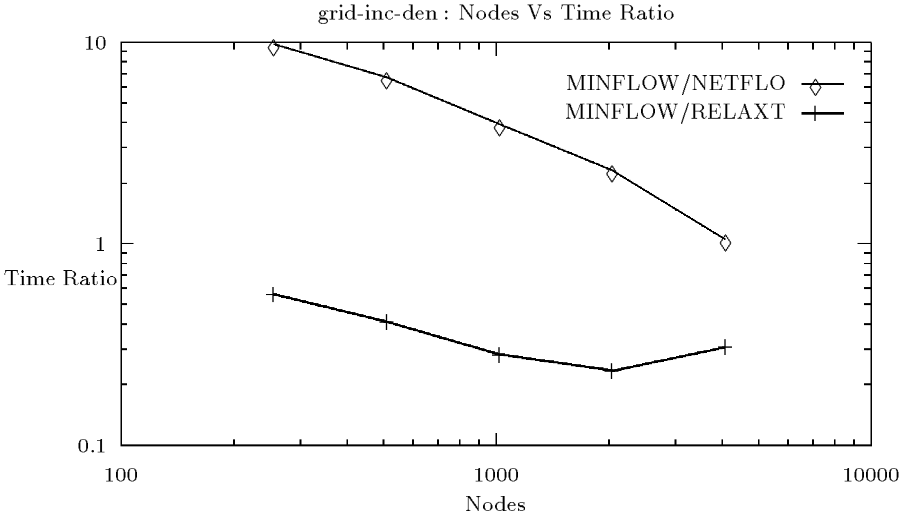

FIGURE 7. Statistics for family **grid-inc-den**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
258	512	10	16.5	0.2	0.4	2.4	0.0	0.1
531	1058	10	43.3	0.5	1.3	4.7	0.0	0.1
1026	2048	10	97.5	1.3	4.1	34.3	0.1	0.5
2118	4232	10	280.1	3.4	17.4	20.0	0.3	2.1
4098	8192	10	779.2	9.8	67.8	94.3	0.7	12.6
8283	16562	2	2567.6	36.2	315.5	261.5	4.6	12.0
16386	32768	1	6938.3	118.0	1244.4	0.0	0.0	0.0
33126	66248	1	21711.0	558.9	5608.4	0.0	0.0	0.0

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
258	512	10	309.7	346.5	1357.5	37.0	43.1	334.7
531	1058	10	433.0	826.9	3353.7	47.8	79.2	570.0
1026	2048	10	514.4	1968.3	7620.6	179.0	123.9	1323.2
2118	4232	10	814.9	4916.6	13886.0	46.6	297.1	4367.4
4098	8192	10	1221.3	11509.7	29922.8	134.2	625.8	11600.2
8283	16562	2	2112.5	30226.0	136411.5	208.5	1814.0	3973.5
16386	32768	1	2831.0	70167.0	377766.0	0.0	0.0	0.0
33126	66248	1	4597.0	197148.0	1020525.0	0.0	0.0	0.0

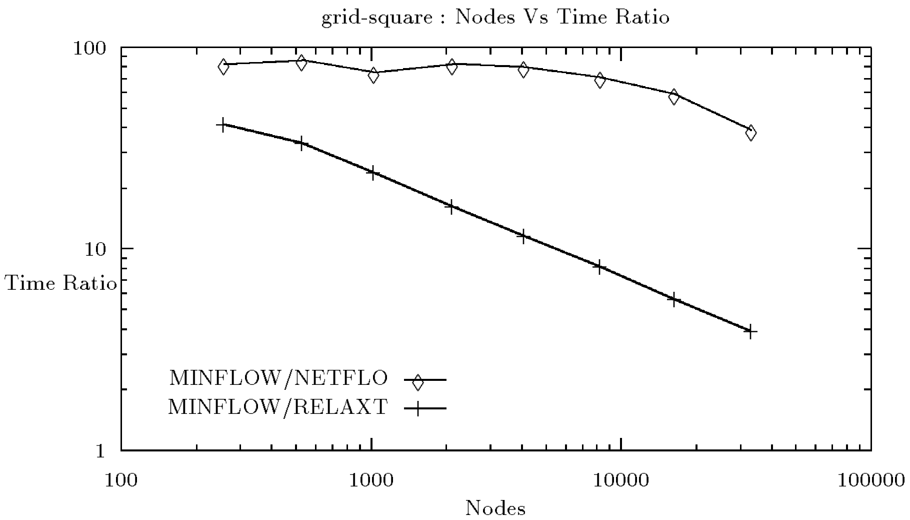

FIGURE 8. Statistics for family **grid-square**

Size		#	Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
258	512	10	16.6	0.2	0.4	2.3	0.0	0.1
514	1008	10	43.6	0.5	1.0	5.0	0.0	0.2
1026	2000	10	132.3	1.0	2.5	5.8	0.0	0.2
2050	3984	10	364.4	2.2	6.6	43.6	0.1	1.1
4098	7952	10	978.6	5.1	19.7	42.2	0.4	2.9
8194	15888	2	3367.1	10.5	43.4	147.7	0.5	3.3
32770	63504	1	36619.2	57.6	374.4	0.0	0.0	0.0

Size		#	Iterations			Iterations (Std.dev)		
Nodes	Arcs		MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
258	512	10	309.7	346.5	1357.5	37.0	43.1	334.7
514	1008	10	463.7	686.4	1726.6	40.0	110.8	936.8
1026	2000	10	776.8	1335.1	790.2	33.6	95.0	238.3
2050	3984	10	1140.2	2353.1	842.3	154.5	137.5	374.0
4098	7952	10	1595.0	5096.2	1500.9	107.1	633.1	878.0
8194	15888	2	2767.5	9056.0	1382.0	147.5	849.0	158.0
32770	63504	1	6500.0	25243.0	6023.0	0.0	0.0	0.0

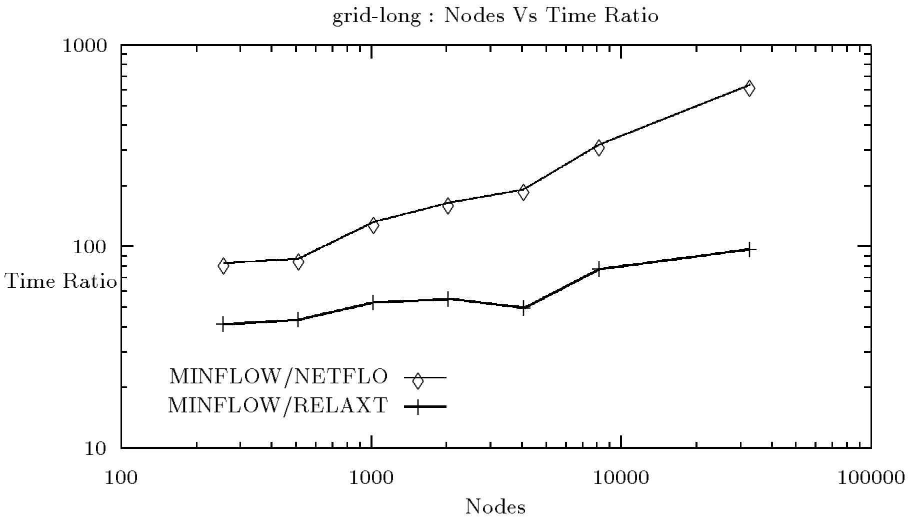

FIGURE 9. Statistics for family **grid-long**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
258	512	10	16.4	0.2	0.4	2.3	0.0	0.0
514	1040	10	41.4	0.6	1.5	5.3	0.0	0.1
1026	2096	10	106.1	1.4	6.0	14.6	0.1	0.7
2050	4208	10	222.2	3.9	25.3	25.2	0.3	3.3
4098	8432	10	529.0	12.1	111.7	33.7	0.8	10.8
8194	16880	2	1327.4	41.7	589.3	99.7	0.2	70.2
16386	33776	1	2925.1	166.2	2501.6	0.0	0.0	0.0
32770	67568	1	6617.7	719.1	12477.0	0.0	0.0	0.0
65538	135152	1	20737.5	2578.5	66517.5	0.0	0.0	0.0

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
258	512	10	309.7	346.5	1357.5	37.0	43.1	334.7
514	1040	10	387.1	1090.7	4444.6	49.6	139.7	656.5
1026	2096	10	501.9	2618.6	17748.9	51.3	312.5	3199.2
2050	4208	10	559.9	6132.2	55576.4	52.7	523.0	9823.2
4098	8432	10	688.1	14204.1	156956.9	48.0	770.5	20308.4
8194	16880	2	845.5	33238.5	550904.5	78.5	138.5	85058.5
16386	33776	1	932.0	73542.0	1263055.0	0.0	0.0	0.0
32770	67568	1	1080.0	170984.0	4595085.0	0.0	0.0	0.0
65538	135152	1	1469.0	346385.0	13321493.0	0.0	0.0	0.0

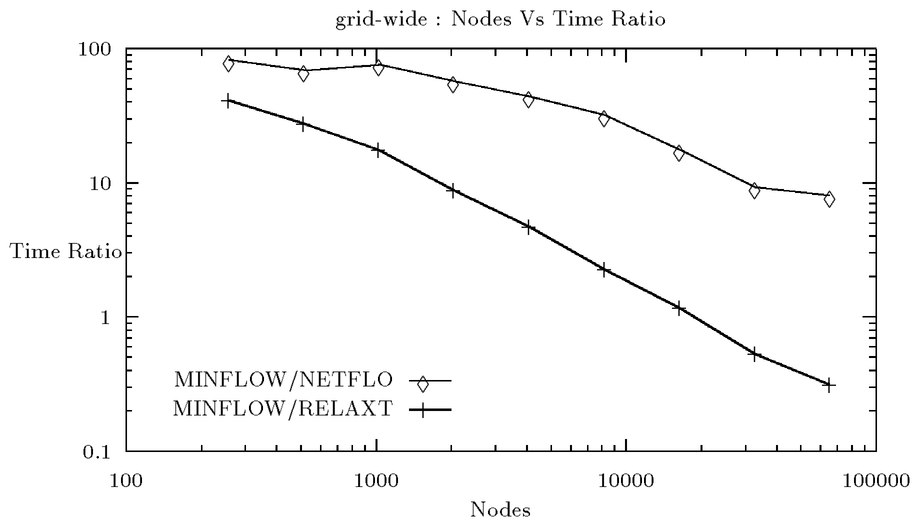

FIGURE 10. Statistics for family **grid-wide**

Size		#	Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	16.1	1.5	0.5	2.6	0.0	0.0
258	8321	10	87.4	11.5	2.0	13.3	0.4	0.0
514	33025	10	546.0	93.8	7.9	69.5	2.1	0.1
1026	131585	2	4273.5	812.9	33.8	270.1	43.9	0.4

Size		#	Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	92.8	1.6	0.3	35.6	0.5	0.5
258	8321	10	175.3	4.2	3.1	44.8	2.6	2.3
514	33025	10	378.5	21.6	20.7	46.7	3.2	3.7
1026	131585	2	1198.0	90.5	83.0	29.0	5.5	7.0

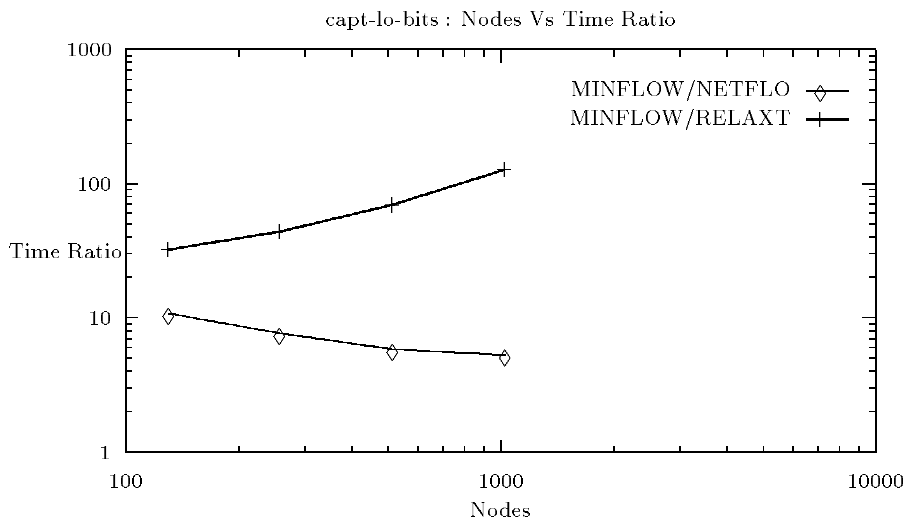

FIGURE 11. Statistics for family **capt-lo-bits**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	24.7	1.6	0.5	2.0	0.1	0.0
258	8321	10	133.0	11.7	2.2	11.3	0.4	0.0
514	33025	10	801.4	94.1	8.7	54.7	2.3	0.1
1026	131585	2	3513.7	820.5	38.5	125.0	5.2	0.1

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	207.0	335.6	171.3	17.5	34.4	11.9
258	8321	10	312.5	681.7	369.1	22.4	41.0	23.0
514	33025	10	510.2	1316.6	765.1	39.8	45.2	17.1
1026	131585	2	674.5	2702.5	1574.0	29.5	42.5	12.0

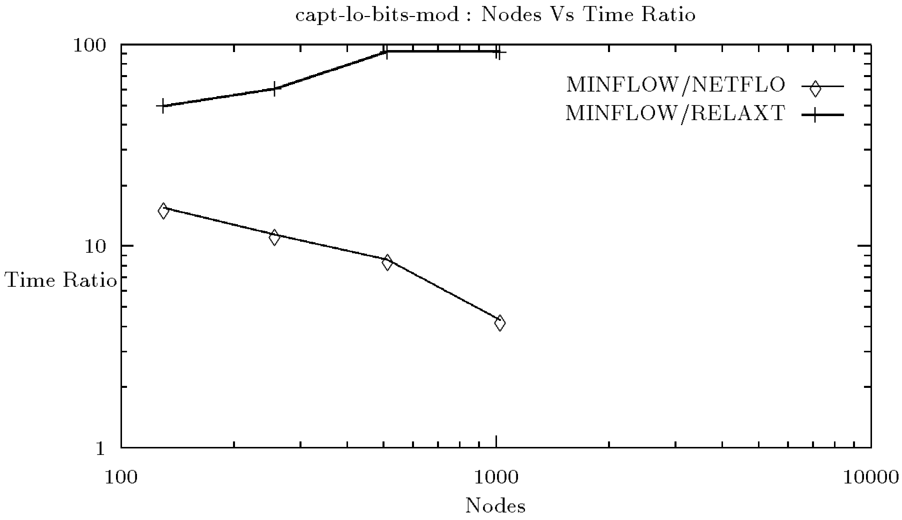

FIGURE 12. Statistics for family **capt-lo-bits-mod**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	16.1	1.5	0.5	2.4	0.0	0.0
258	8321	10	81.5	11.7	2.0	10.3	0.4	0.0
514	33025	10	529.7	93.6	8.0	97.6	2.3	0.1
1026	131585	2	3983.9	771.6	32.2	452.3	1.7	0.0

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	97.1	1.8	0.2	24.2	0.4	0.4
258	8321	10	154.5	4.2	3.0	40.1	2.1	2.4
514	33025	10	351.9	20.0	17.6	56.1	5.9	5.6
1026	131585	2	1137.0	89.0	73.5	91.0	1.0	1.5

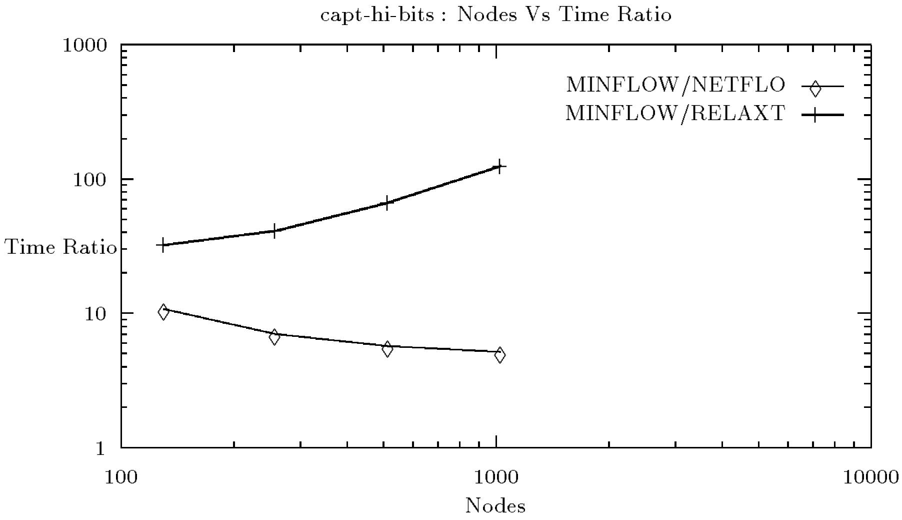

FIGURE 13. Statistics for family **capt-hi-bits**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	26.3	1.6	0.6	3.0	0.0	0.0
258	8321	10	142.8	11.7	2.2	17.2	0.4	0.2
514	33025	10	895.9	94.3	8.7	60.9	2.4	0.0
1026	131585	2	3713.4	852.9	37.6	228.2	1.3	0.2

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
130	2113	10	203.7	361.1	159.1	16.2	39.8	14.7
258	8321	10	278.4	758.9	322.9	21.1	46.9	23.1
514	33025	10	455.7	1525.1	657.2	14.3	67.2	33.7
1026	131585	2	594.5	3197.0	1234.5	5.5	75.0	24.5

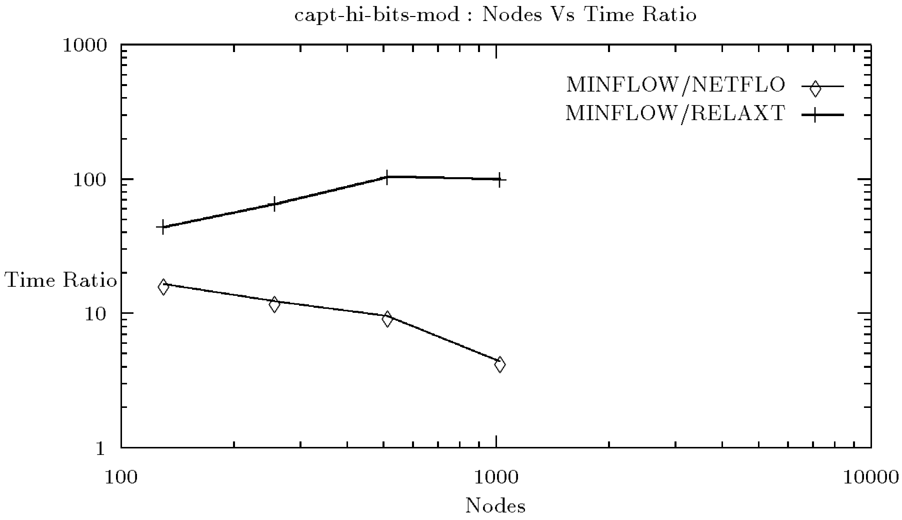

Figure 14. Statistics for family **capt-hi-bits-mod**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	512	3	9.8	0.4	0.2	0.1	0.0	0.0
529	1058	3	26.1	1.3	0.7	0.1	0.0	0.1
1024	2048	3	62.5	4.2	1.4	0.5	0.2	0.1
2116	4232	3	171.5	16.2	4.4	7.8	1.0	0.5
4096	8192	3	436.7	55.8	10.4	25.9	1.8	0.8
8281	16562	3	1199.1	239.8	27.3	54.0	2.0	2.8
16384	32768	3	3060.4	1046.9	78.0	98.4	27.2	8.0
33124	66248	3	8887.6	4667.0	169.1	90.9	138.3	26.2
65536	131072	3	24706.9	19616.1	540.2	182.2	1031.3	49.0

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	512	3	219.3	1016.7	672.0	6.6	58.9	30.6
529	1058	3	282.3	2531.7	1760.3	6.8	90.7	205.5
1024	2048	3	355.7	5833.7	3110.3	5.4	138.0	215.3
2116	4232	3	527.3	13784.7	6949.3	2.6	469.3	194.3
4096	8192	3	751.3	28993.3	13129.7	12.0	1005.1	483.9
8281	16562	3	1115.7	67341.3	27472.7	26.4	499.2	282.2
16384	32768	3	1558.3	151026.0	57056.7	15.0	2766.8	640.7
33124	66248	3	2378.7	351469.7	113439.3	35.7	6161.2	1252.2
65536	131072	3	3483.3	779332.3	226449.3	33.2	8654.5	1996.9

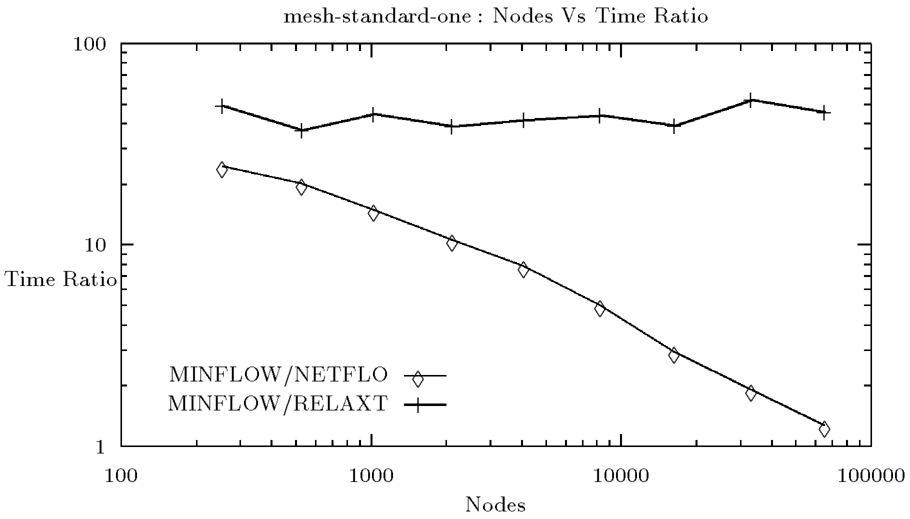

FIGURE 15. Statistics for family **mesh-standard-one**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	1024	3	16.7	0.9	0.4	0.4	0.0	0.0
529	2116	3	44.9	3.6	1.1	1.3	0.1	0.0
1024	4096	3	110.0	13.0	2.2	1.7	0.3	0.3
2116	8464	3	306.5	54.2	6.2	6.4	0.9	0.3
4096	16384	3	761.2	211.8	15.2	8.7	5.4	1.5
8281	33124	3	2035.9	930.4	46.5	14.9	6.1	4.3
16384	65536	3	5958.4	3892.7	100.3	39.2	20.4	21.4
33124	132496	3	16779.6	18285.5	346.3	553.8	1163.5	14.2
65536	262144	3	33900.3			23984.5		

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	1024	3	269.3	3028.3	903.7	1.2	109.4	74.4
529	2116	3	377.0	8143.3	1979.7	5.0	281.3	162.1
1024	4096	3	530.0	18902.3	3614.7	11.3	180.9	414.5
2116	8464	3	764.0	44960.0	8056.0	5.7	795.4	348.5
4096	16384	3	1091.7	100814.0	17023.3	2.5	539.2	1135.2
8281	33124	3	1659.3	236127.3	38345.0	27.5	1227.2	367.5
16384	65536	3	2444.0	526127.0	71911.7	23.3	3452.9	3701.2
33124	132496	3	3751.3	1249134.7	156234.0	146.4	23765.4	2685.1
65536	262144	3	3626.3			2564.2		

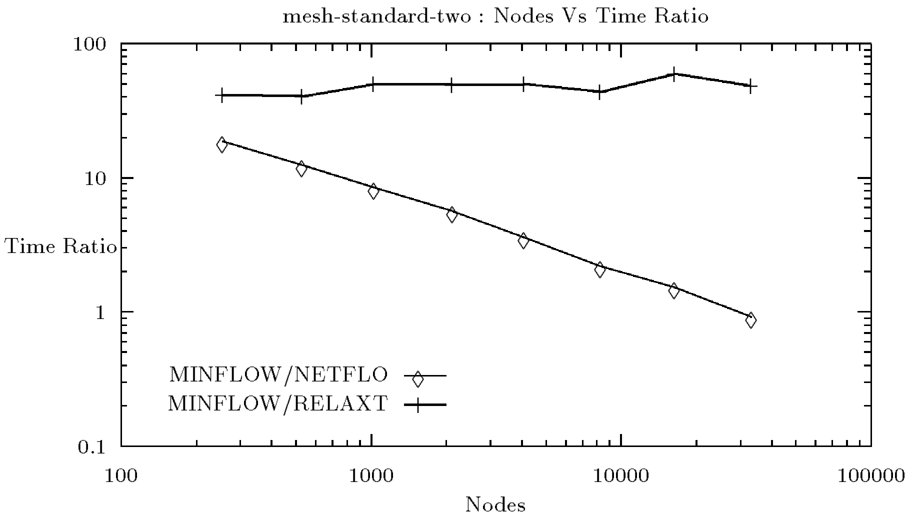

FIGURE 16. Statistics for family **mesh-standard-two**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	2048	3	27.7	1.9	0.7	0.4	0.0	0.0
529	4232	3	74.3	7.9	1.8	1.2	0.1	0.1
1024	8192	3	190.4	31.7	4.0	14.5	1.1	0.1
2116	16928	3	472.0	147.7	10.1	13.7	2.1	0.9
4096	32768	3	1143.2	605.5	29.1	26.6	11.8	3.1
8281	66248	3	3295.4	2736.6	67.8	102.2	57.1	2.2
16384	131072	3	9191.0	12084.4	182.8	180.4	92.7	4.2
33124	264992	3	18096.9			12796.4		

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	2048	3	301.3	6717.7	975.0	17.2	158.1	82.0
529	4232	3	414.3	19854.0	2092.0	4.8	505.1	193.8
1024	8192	3	625.7	50809.7	4354.0	12.7	1087.9	141.4
2116	16928	3	890.0	126310.3	10060.7	21.5	758.8	603.5
4096	32768	3	1265.0	286119.0	20588.3	28.3	3267.3	2547.8
8281	66248	3	1993.0	664587.7	42485.0	39.9	3579.5	423.3
16384	131072	3	2974.0	1500231.7	87171.7	32.5	11059.1	2153.2
33124	264992	3	3084.0			2180.8		

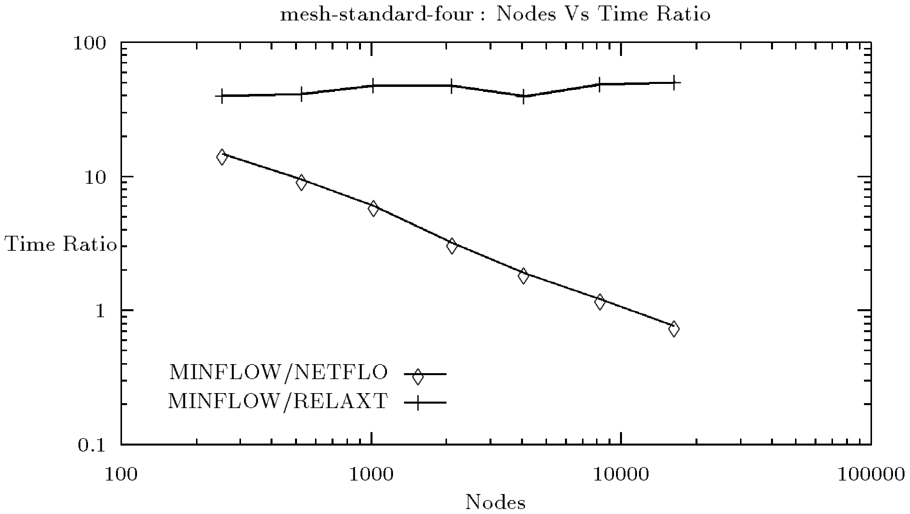

Figure 17. Statistics for family **mesh-standard-four**

Size			Avg. Time (seconds)			Std. Dev. Time (seconds)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	4096	3	51.4	3.6	1.5	2.5	0.1	0.0
529	8464	3	132.1	16.4	3.5	2.4	0.3	0.1
1024	16384	3	327.2	66.6	7.4	16.6	1.0	0.4
2116	33856	3	801.0	347.3	19.5	18.5	5.7	0.2
4096	65536	3	2148.2	1651.9	48.9	56.9	35.3	3.2
8281	132496	3	5160.2	6977.0	116.6	230.6	35.5	6.0
16384	262144	3	10374.1			7363.3		

Size			Iterations			Iterations (Std.dev)		
Nodes	Arcs	#	MINFLOW	NETFLO	RELAXT	MINFLOW	NETFLO	RELAXT
256	4096	3	307.7	12141.3	1152.0	15.3	470.2	84.2
529	8464	3	470.0	41434.0	2794.0	19.6	921.1	211.6
1024	16384	3	649.3	114986.0	5397.3	17.4	1190.6	125.2
2116	33856	3	933.3	309586.0	11669.7	13.3	6483.7	472.8
4096	65536	3	1363.0	711413.3	24505.0	26.2	9406.9	1226.1
8281	132496	3	1983.7	1672519.7	50600.7	42.6	986.2	1407.7
16384	262144	3	2154.3			1526.1		

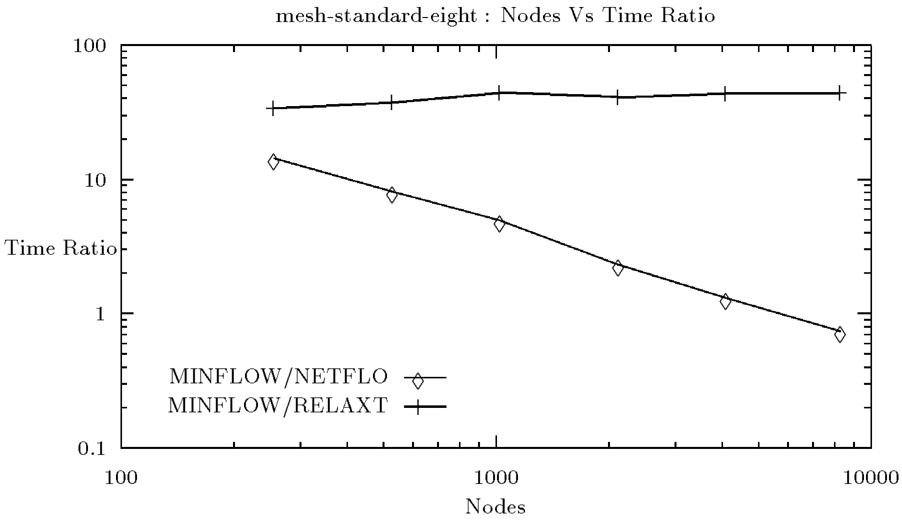

FIGURE 18. Statistics for family **mesh-standard-eight**

References

[A+85] J. Aronson et al. The projective transformation algorithm of Karmarkar: A computational experiment with assignment problems. Technical Report 85-OR-3, Southern Methodist University, Department of Operations Research,Southern Methodist University, Dallas, Texas, 1985.

[AM91] A. Armacost and S. Mehrotra. Computational comparison of the network simplex method with the affine scaling method. *OpSearch*, 28:26–43, 1991.

[Ber91] D.P. Bertsekas. *Linear Network Optimization: Algorithms and Codes.* The MIT Press, 1991.

[BT88a] D.P. Bertsekas and P. Tseng. RELAX: A computer code for minimum cost network flow problems. *Annals of Operations Research*, 13:127–190, 1988.

[BT88b] D.P. Bertsekas and P. Tseng. Relaxation methods for minimum cost ordinary and generalized network flow problems. *Operations Research*, 36:93–114, 1988.

[BT90] D.P. Bertsekas and P. Tseng. RELAXT-III: A new and improved version of the relax code. Technical Report P-1990, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, MIT, Cambridge, Massachusetts, 1990.

[Cha52] A. Charnes. Optimality and degeneracy in linear programming. *Econometrica*, 20:160–170, 1952.

[FF62] Lestor R. Ford Jr. and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

[GL89] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, 1989.

[Kar84] Narendra Karmarkar. A new polynomial algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[KH80] J.L. Kennigton and R.V. Helgason. *Algorithms for Network Programming.* John-Wiley & Sons, 1980.

[KY91] J.A. Kalinski and Y. Ye. A decomposition variant of the potential reduction algorithm for linear programming. Technical Report 91-11, University of Iowa, Department of Management Sciences, University of Iowa, Iowa City, Iowa, 1991.

[Raj89] A. Rajan. An empirical comparison of KORBX against REALXT, a special code for network flow problems. Technical Report (Not available), AT&T Bell Laboratories, Holmdel, NJ, 1989.

[Ren88] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming*, 40:59–93, 1988.

[RV90] M.G.C. Resende and G. Veiga. An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks. Technical Report (Not available), AT&T Bell Laboratories, Murray Hill, NJ, 1990. (To appear in Siam Journal on optimization).

[TCR90] C.E. Leiserson T.H. Cormen and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[Vai89] Pravin M. Vaidya. A simple Newton-type method for finding the analytical center of a polytope. In Nimrod Megiddo, editor, *Progress in Mathematical Programming : Interior Point and Related Methods*, pages 79–90. Springer-Verlag, New York, 1989.

[Vai90] Pravin M. Vaidya. An algorithm for linear programming which requires $\mathrm{O}(((m+n)n^2+(m+n)^{1.5}n)\mathrm{L})$ arithmetic operations. *Mathematical Programming*, 47:175–201, 1990.

[Vai91] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing provably good preconditioners. Technical Report (In preparation), Department Of Computer Science, University of Illinois at Urbana-Champaign, 1991.

Department of Computer Science, University of Illinois, Urbana, Illinois
E-mail address: joshi@cs.uiuc.edu

Department of Computer Science, University of Illinois, Urbana, Illinois
Current address: Department of Mathematics and Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel
E-mail address: arthur@cs.bgu.ac.il

Department of Computer Science, University of Illinois, Urbana, Illinois
E-mail address: vaidya@cs.uiuc.edu