SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 661

June 1985

ON THE COMPUTATIONAL BEHAVIOR OF A POLYNOMIAL-TIME NETWORK FLOW ALGORITHM*

By

Robert G. Bland
and
David L. Jensen**

*This work was supported in part under NSF grant #ENG-7910807 and ECS-8313853 to Cornell University and a Sloan Foundation research fellowship held by the first author.

**Department of Applied Mathematics and Statistics, SUNY Stony Brook

On the Computational Behavior of a Polynomial-Time Network Flow Algorithm*

Robert G. Bland

School of OR/IE, Cornell University, Ithaca, New York 14850

David L. Jensen**

Department of Applied Math and Statistics, SUNY, Stony Brook, New York 11794

Abstract

A variation on the Edmonds-Karp scaling approach to the minimum cost network flow problem is discussed. This algorithm, which scales costs rather than right-hand-sides, also runs in polynomial time. Large-scale computational experiments indicate that the computational behavior of such scaling algorithms may be much better than had been presumed. Within several distributions of square, dense, capacitated transportation problems, a cost scaling code, SCALE, exhibits linear growth in average execution time with the number of edges, while two network simplex codes, RNET and GNET, exhibit greater than linear growth. The test problems were generated by a routine, CAPT, which imposes certain symmetries on the distribution; they have 200-500 vertices and 10,000-62,500 edges. Although RNET is still faster than SCALE at the upper end of this range of CAPT-generated problems, SCALE is fastest of the three codes on a set of large (70,000-90,000 edges) dense transshipment problems that we generated using NETGEN.

Our experiments reveal that the behavior of median and mean execution times are predictable with surprising accuracy for all of the three CAPT distributions and all of the three codes tested. Moreover, for fixed problem size, individual execution times appear to behave as though they are approximately lognormally distributed with constant variance. The experiments also reveal sensitivity of the parameters in the models, and in the models themselves, to variations in the distribution of problems. This argues for caution in the interpretation of such computational studies beyond the realm in which the computations were performed.

Key words and phrases: network flow, scaling, polynomial algorithm, computation

Running head: Network Flow Algorithm

*This work has been supported in part by NSF grants ENG-7910807 and ECS-8313853 to Cornell University and a Sloan Foundation research fellowship held by the first author.

**Department of Applied Mathematics and Statistics, SUNY Stony Brook

Introduction

Minimum cost network flow problems are of great practical importance. Several different algorithms for these problems have been studied, implemented, and employed. Most of the algorithms that have received serious attention, and all of those presently in widespread use, fall into one of two categories: network simplex algorithms (primal and dual) and out-of-kilter algorithms (see [2,5,12,18,20,25]). The network simplex method, for which several sophisticated implementations are widely available, seems to be the method of choice at present. With it, users are able to solve problems with many tens of thousands of variables in a fraction of a minute on a fast mainframe computer.

The standard implementations of the network simplex method and of the out-of-kilter method share a negative theoretical property - they are not polynomial-time algorithms. However, Edmonds and Karp [4] showed in 1972 that by repeated scaling of the right-hand-side data one could employ the out-of-kilter algorithm iteratively to get an algorithm that runs in polynomial time.

Until very recently there has been almost no interest in employing the scaling approach of Edmonds and Karp in actual computation. In spite of favorable asymptotic worst-case performance, it was widely presumed that algorithms employing data scaling would not be nearly as fast in practice as the network simplex method.

The negative presumptions concerning data scaling were based on very scanty evidence. No large-scale tests were conducted. Only very recently have some researchers begun to reconsider data scaling as a computational tool for solving network flow problems (see Section 7). In this paper we examine the computational behavior of a minimum cost flow algorithm based upon cost scaling and the out-of-kilter method. Our computational experiments contrast the execution times of a polynomial-time cost scaling algorithm with two network simplex codes on square, dense, capacitated transportation problems generated by several different pseudo-random routines. The network simplex codes examined are GNET[1] (Bradley, Brown, and Graves, see [2]) and RNET[2] (Grigoriadis and Hsu, see [13]). Both were run with the input parameters set at their default values. We call our cost scaling out-of-kilter code SCALE. All three are FORTRAN codes, and were run on an IBM 3081 model D24 under the FORTVS compiler at optimization level 3 and language level 66.

The main experiment provides very persuasive statistical evidence that within the domains from which the generated problems were drawn, the rate of growth of average execution time with the number of edges is linear for the cost scaling code, and faster than linear for both of the network simplex codes. Even for the largest problems solved in our main experiment (62,500 edges), RNET is consistently faster than SCALE, which is much faster than GNET. However, both the relative execution times and the relative rates of growth of average execution times indicate that scaling can be much more effective than was previously believed, at least for special classes of problems. Furthermore, the effect of cost scaling on the out-of-kilter routine underlying SCALE was substantial. The out-of-kilter routine without cost scaling ran about ten times longer than with scaling on each of twelve test problems with ten significant bits in the cost coefficients.

In addition to the conclusion that data scaling is worthy of further consideration as a practical computational tool, there are two interesting sidelights to these experiments. One is the accuracy with which one can predict the behavior of average execution times, and, surprisingly, of individual execution times, for all three codes, <u>within the relevant problem domains</u>. Second, the data reinforce sharply the need to emphasize the prepositional phrase at the conclusion of the preceding sentence. One cannot expect to transport predictions based on one domain to another, even if it appears closely related to the first, and retain their significance. Even if the form of the prediction function continues to fit, the parameter values in the function may vary greatly.

In the main experiment we employed three capacitated transportation problem generators. These <u>CAPT</u> generators are described in the appendix. We also generated some problems using the widely available generator NETGEN of Klingman, Napier, and Stutz [19]. By control of certain input parameters we were able to generate problems from NETGEN on the same underlying space of graphs, with approximately the same distribution on total flow through the network. Except for the the presence of an "artificial" spanning tree with large costs and capacities, the distribution of costs and the range of capacities are the same as in the CAPT generated problems in the main experiment. However the generators differ in the distribution of capacities within those ranges. The "artificial" spanning tree contains between 0.80%

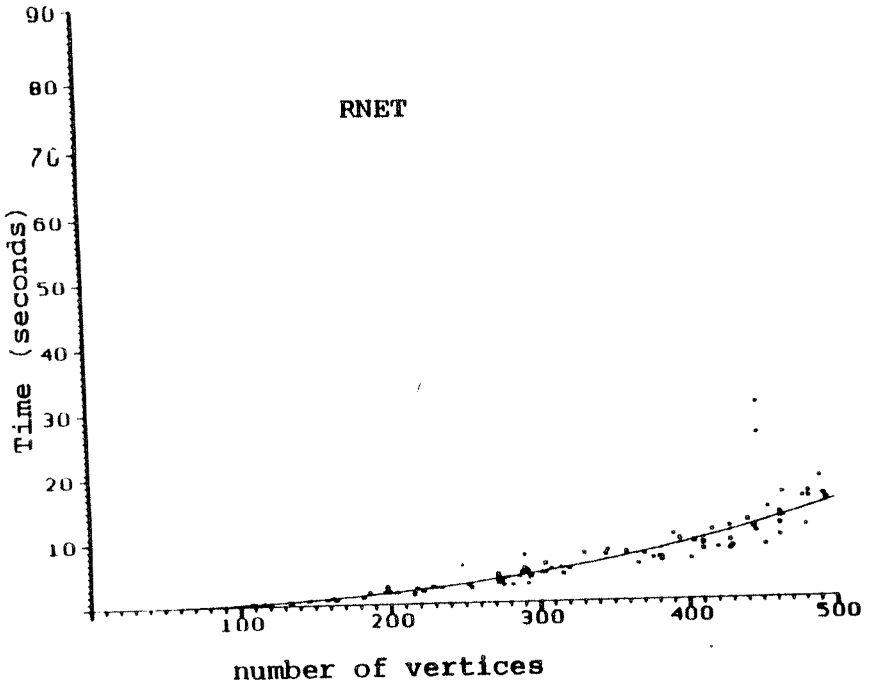
and 3.96% of the edges; on average it contains 1.61%. These differences result in substantially different estimates of the parameter values in the functions that predict execution time.

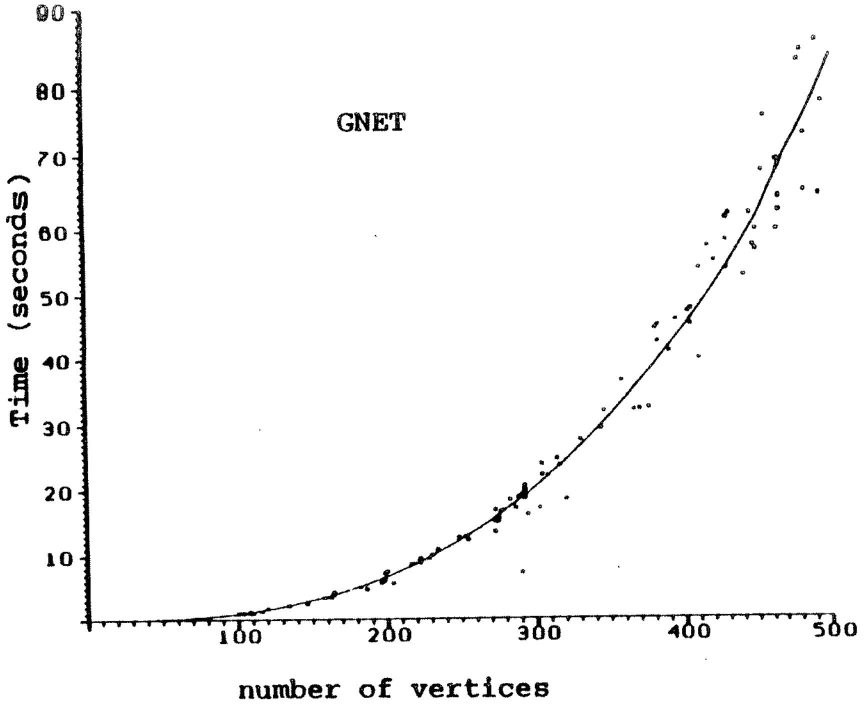
For each of the three codes and each of the three CAPT-generated problem distributions, both median and mean execution times over problems with $|V|$ vertices and Λ significant cost bits, can be estimated accurately by a function of the form

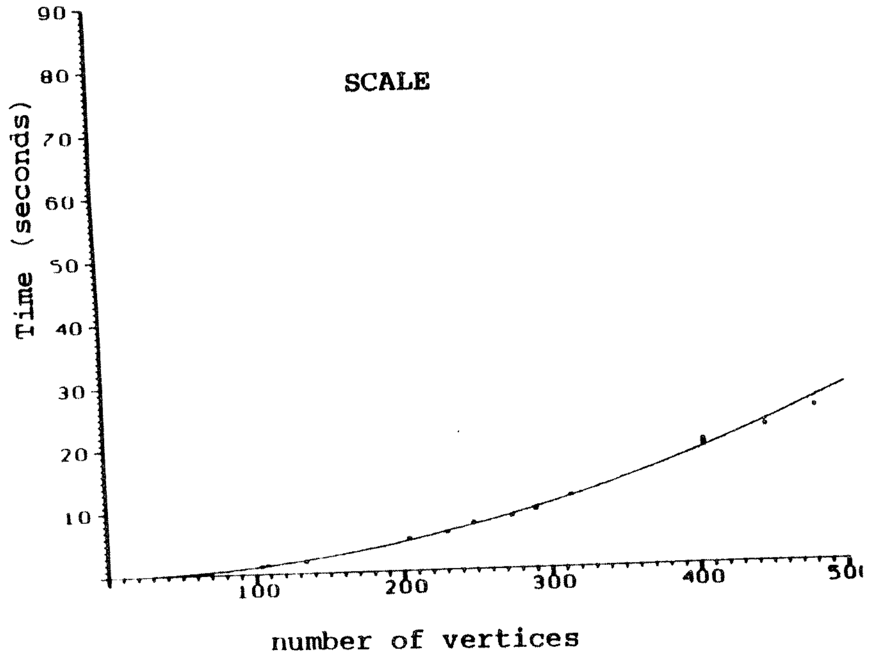
$$c|V|^{b_1}\Lambda^{b_2} \tag{0.1}$$

where c is a constant. The leading constant c is slightly larger for mean execution time than for median execution time, but the exponents are the same. For RNET and GNET times are insensitive to Λ, and b_2 can be fixed at zero. For SCALE b_2 ranges between 0.737 and .793 over the three distributions. The exponent b_1 ranges between 1.976 and 2.078 for SCALE, between 2.313 and 2.450 for RNET and between 2.713 and 2.743 for GNET. In these problems $|E|$, the number of edges, is equal to $|V|^2/4$, so we can substitute to express (0.1) as a model in which $|E|$ and Λ are the independent variables. Then we see from the parameter estimates above that estimated average execution times for SCALE grow almost exactly linearly with the number of edges, while the rates of growth for RNET and for GNET are faster than linear.

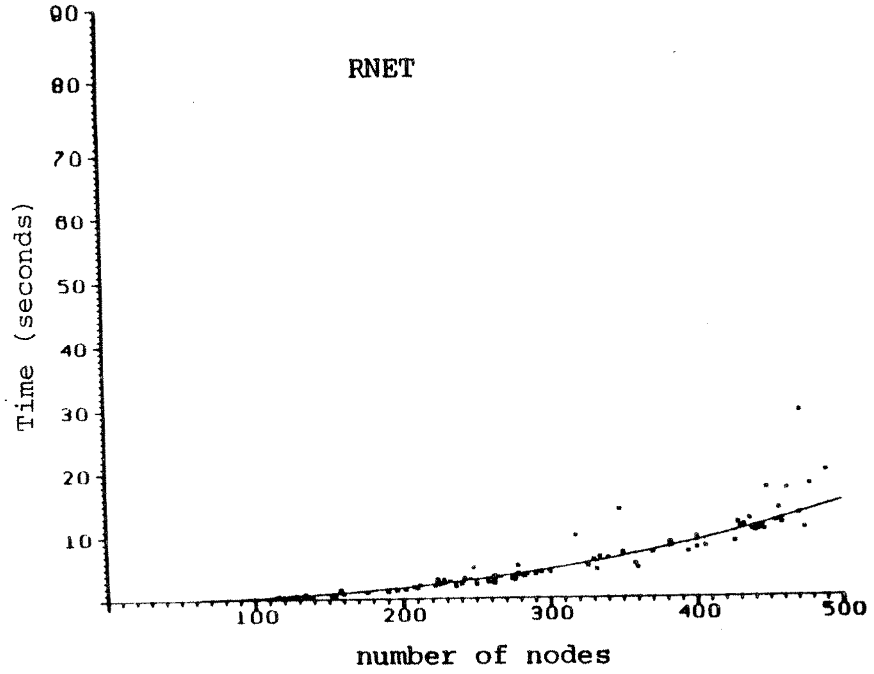
For each of SCALE, RNET, and GNET Figure 0.1 presents a plot of observed execution times versus the number of vertices, and a fitted curve of the form (0.1). The problem samples here are from distribution #1. See the appendix for descriptions of the three problem distributions. The plotted curves are fitted to minimize the sum of the squares of the deviations between the logarithm of observed execution time and the logarithm of the evaluation of (0.1). Figures B.1.1 and B.1.2 in Appendix B of the report [1] present the same information for each of distributions #2 and #3, respectively; they are very similar in appearance to the figures presented here for distribution #1. Since SCALE has two independent variables in the model, we made separate plots for each of the seven values of $\Lambda \in \{4,5,6,7,8,9,10\}$ that arise in the data. (Keep in mind that the model was fitted to all observations simultaneously with Λ as a parameter.)

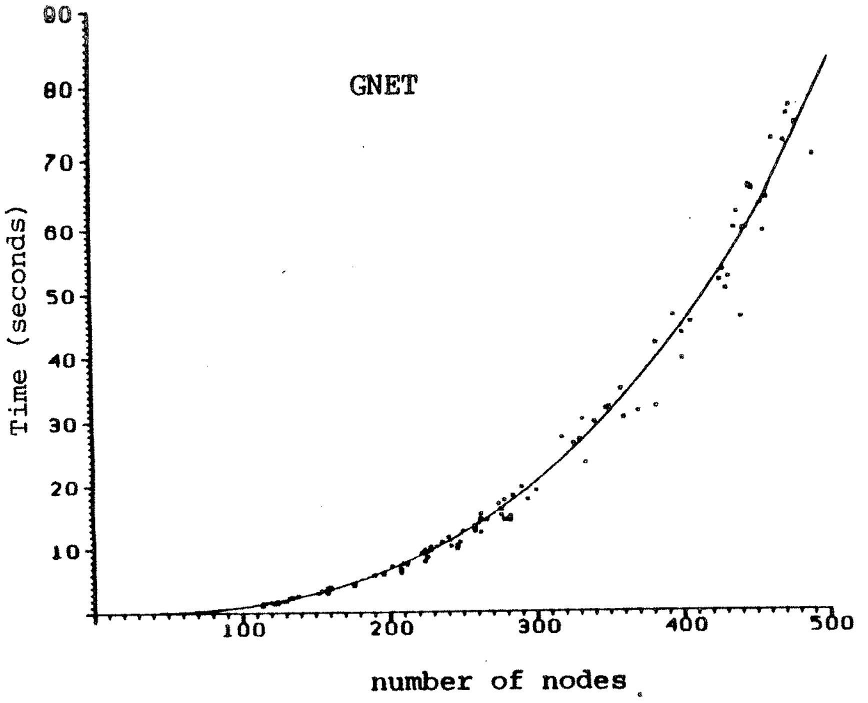






<u>Figure 0.1</u> Observed execution times and fitted curves. Distribution #1.





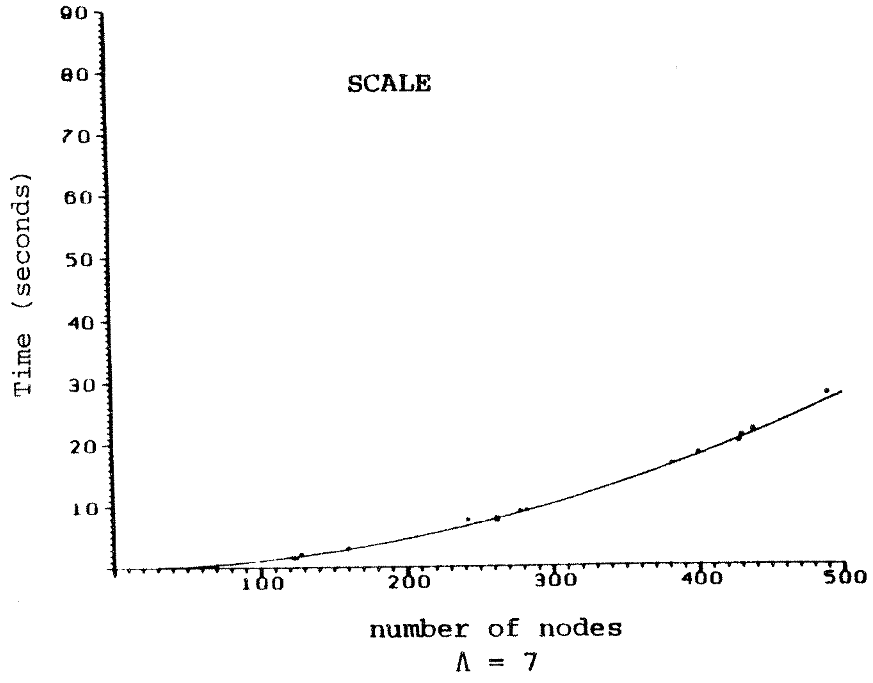


Figure 0.2 Predicted median execution time as a function of $|V|$, and a sample of observed execution times independent of the prediction. Distribution #1.

The plot for SCALE in Figure 0.1 includes only the observations with $\Lambda = 7$. The plots for the other values of Λ appear in Figure B.1.3 of [1]. It is clear from the plots in Figure 0.1 (and B.1 of [1]) that the fit is close in each case. Figure 0.2 is still more persuasive of the goodness of the models as predictors of average and median execution times (cf. Figure B.2 of [1]). The curves in these plots indicate predicted median execution times, but the observed times plotted here are from samples that are independent of the earlier ones on which the predictions were based.

It is apparent from the Figures that (0.1) is a good model of median execution time for these problem distributions. More detailed analysis of residuals from prediction in these samples indicates that for fixed $|E|$ and Λ, individual execution times behave as though they are are distributed, approximately, as the product of the estimate (0.1) of the median with a lognormal random variable with median equal to one and constant variance.

In the sample of 25 NETGEN problems on the same underlying space of graphs produced by the CAPT generators, model (0.1) still fits, but we get drastically different estimates of some of the parameters. SCALE's relative performance is much better in this NETGEN sample than under the CAPT generators. It is even better still on fifty nonbipartite fully dense NETGEN problems with as many as 90,000 edges. Of twenty problems here with more than 70,000 edges, SCALE was fastest of the three codes on twelve, RNET was fastest on six, and GNET was fastest on two.

The sensitivity of our results to the problem generator should give pause to anyone producing empirical or analytical estimates of performance of an algorithm under some convenient generator or distribution, with the intention of explaining computational experience with "real-life" problems. We may develop better insight and begin to explain, but we must be very cautious not to extend the significance of these studies beyond their proper domains.

In the first part of this paper, Sections 1-4, we describe the general form of the cost scaling algorithm employed in the computational study. It is the variation on Edmonds-Karp [4] that scales costs rather than right-hand-sides, and it uses a maximum flow routine as its computational engine. We show why it is polynomial-bounded in the worst case, and indicate how to implement it so that the bound is $|V|^4\Lambda$, where $|V|$ is the number of vertices. We have learned recently that Hans Röck [26] has described the same variation on the Edmonds-Karp algorithm. The exposition and implementation here differ from [26]. In Section 5 we discuss the main experiment, including its basic design and the statistical analysis of the data. Section 6 concerns our computational tests on NETGEN problems, and Section 7 outlines several recent results and their relationship to this work. Section 8 summarizes our conclusions. Sections 5-8 can be read without reading Sections 1-4.

The form of the CAPT generators employed in our main experiment is discussed in the appendix. Additional tables and plots in support of the statistical analysis of Section 5 can be found in the report [1].

Our notation and terminology are mostly standard. The symbol Z denotes the integers. The symbols $\lfloor \; \rfloor$ and $\lceil \; \rceil$ denote the functions that take a real number β to $\lfloor\beta\rfloor$, the greatest integer less than or equal to β, and $\lceil\beta\rceil$, the least integer greater than or equal to β. A directed graph $G = (V,E)$ with vertex set V and edge set E is assumed to be finite, and (for convenience of notation) to be loopless and without multiple edges. It may have oppositely directed edges. Beginning in Section 3 we assume that our minimum cost flow problems have nonpositive costs; since we make no assumption on the signs of the upper and lower bounds on the flow variables, there is no loss of generality here. The reason for this assumption, as opposed to, say, the commonly used assumption of nonnegative costs, is to make the maximum flow problem a direct special case. This is extremely useful in the exposition of our algorithm, since it solves the minimum cost flow problem as a sequence of maximum flow problems. One further departure from convention is our association of the terms cycle and path with ordered pairs (S^+,S^-), where S^+ (respectively, S^-) is the set of forward (reverse) edges when the cycle or path, as usually defined, is traversed in one of the two possible directions.

The problem generators and the codes explicitly deal with networks in which the original (bipartite) transportation network is augmented by source and sink vertices, s and t, and edges connecting s to each supply point and each demand point to t. When we discuss $|V|$ and $|E|$, they refer to the numbers of vertices and edges in the original bipartite network only.

1. Maximum Flows

A minimum cost network flow problem is a linear programming problem of the form

$$\begin{array}{ll} \text{minimize} & ax \\ \text{subject to} & Ax = b \qquad\qquad (P) \\ & l \leq x \leq u \end{array}$$

where A is the vertex-edge incidence matrix of a directed graph $G = (V,E)$. The column of A corresponding to $e = (i,j) \in E$ is denoted $A(e)$ and has entries of $+1$ in the row corresponding to vertex j, the head of e, -1 in the row corresponding to vertex i, the tail of e, and zero elsewhere. (Some authors use the negative of this matrix.) The head and tail of an edge are called its ends. An instance of problem (P) can be described by a five-tuple (G,a,l,u,b). The vectors a, l, and $u \geq l$ in Z^E are costs, lower bounds, and upper bounds, respectively, and the demand vector $b \in Z^V$ is assumed to have the sum of its entries equal to zero, since this is necessary for the conservation of flow constraints $Ax = b$ to have a solution. A vector $x \in \mathbb{R}^E$ is called a flow if $Ax = b$; it is called a circulation if $Ax \equiv 0$. A flow x is feasible if $l \leq x \leq u$.

The special case of (P) in which $b \equiv 0$, and for some $e^* \in E$, $a(e^*) = -1$ and $a(e) = 0$ for all $e \in E \backslash e^*$ will be called the maximum e^*-flow problem. We will occasionally describe a maximum e^*-flow problem instance by a four-tuple (G,l,u,e^*). This formulation differs slightly from the usual formulation, as in [5], in that we are maximizing flow on a bounded edge e^*.

The maximum e^*-flow problem is easier to solve than the minimum cost flow problem (P). Indeed the solution of (P) by the out-of-kilter method can be viewed as an implicit succession of maximum flow calculations. This viewpoint becomes explicit in the next section. Another reason for separate consideration of the maximum flow problem is its relation to the "Phase I" problem for (P). One can determine a feasible solution of (P), or show that none exists, by solving a maximum flow problem in a network only slightly larger than the original. This transformation is well known; see the routine TRANS and its discussion in Section 3 for details. A few definitions concerning directed graphs will be helpful in our discussion of an algorithm for the maximum e*-flow problem.

In the directed graph $G = (V,E)$ let $Q = (v_0,e_1,v_1,e_2,\ldots,e_k,v_k)$ be a non-null alternating sequence of vertices and edges such that $v_i \neq v_j$ for all $0 \leq i < j \leq k - 1$, $e_i \neq e_j$ for all $1 \leq i < j \leq k$, and the ends of each e_i are v_{i-1} and v_i. Such a sequence is usually called: a <u>cycle</u> (or circuit) if $v_k = v_0$ and $k > 0$; or a <u>path</u> if $k = 0$ or $k > 0$ and $v_k \neq v_i$, for all $0 \leq i \leq k - 1$. We will use these terms instead to denote the natural partition of the edge set $F = \{e_1,\ldots,e_k\}$ of Q into <u>forward</u> edges

$$F^+ = \{e_i \in F: e_i = (v_{i-1},v_i)\},$$

and <u>reverse</u> edges

$$F^- = \{e_i \in F: e_i = (v_i,v_{i-1})\}.$$

We will usually denote a cycle by $C = (C^+,C^-)$ and a path by $P = (P^+,P^-)$. The <u>signed incidence vector</u> of a cycle C is the vector $z \in \{-1,0,+1\}^E$ having $C^+ = \{e \in E: z(e) = +1\}$ and $C^- = \{e \in E: z(e) = -1\}$. For $e^* \in E$ a cycle C having $e^* \in C^+$ is called an <u>e*-cycle</u>.

The <u>cut</u> $D(S)$ of $G = (V,E)$ determined by a nonempty proper subset S of V is the ordered pair $(D^+(S), D^-(S))$, where

$$D^+(S) = \{(i,j) \in E: i \in \bar{S},\ j \in S\}$$

is the set of <u>forward</u> edges of $D(S)$ and

$$D^-(S) = \{(i,j) \in E: i \in S,\ j \in \bar{S}\}$$

is the set of <u>reverse</u> edges of $D(S)$, and $\bar{S}$ denotes $V\backslash S$, the complement of S in V. Note that our designation of forward and reverse edges in $D(S)$ is the opposite of the implicit designation in [5]; this yields a nice symmetry in our description of the Augmenting Cycle Theorem for the maximum flow problem. When referring to an arbitrary cut and no ambiguity is possible we may write $D = (D^+, D^-)$. For $e^* \in E$ we say that D is an e^*-cut if $e^* \in D^+$. Recall that the signed incidence vector $y \in \{-1,0,1\}^E$ of a cut $D(S)$ has $D^+(S) = \{e \in E: y(e) = +1\}$, $D^-(S) = \{e \in E: y(e) = -1\}$, and can be obtained by adding those rows of A corresponding to vertices in S.

Given a maximum e^*-flow problem (G,l,u,e^*) and a feasible flow x, it will be convenient to partition (or color) the edge set E of G into red (R), yellow (Y), blue (B), and white (W) edges as follows:

$$\begin{aligned} R &= \{e^*\} \cup \{e \in E\backslash e^*: l(e) = x(e) < u(e)\}; \\ Y &= \{e \in E\backslash e^*: l(e) < x(e) = u(e)\}; \\ B &= \{e \in E\backslash e^*: l(e) < x(e) < u(e)\}; \\ W &= \{e \in E\backslash e^*: l(e) = x(e) = u(e)\}. \end{aligned} \tag{1.1}$$

Since $R \cup B$ is the subset of edges with excess capacity and $Y \cup B$ is the set of edges with excess flow, we can rephrase the usual definitions of augmenting cycles and saturated cuts. An e^*-cycle C is augmenting with respect to x if $C^+ \subseteq R \cup B$ and $C^- \subseteq Y \cup B$. An e^*-cut D is saturated with respect to x if $D^+ \subseteq R \cup W$ and $D^- \subseteq Y \cup W$. Ford and Fulkerson exploited the special relationship between cycles and cuts in their approach to the maximum flow problem. Their Augmenting Path Theorem becomes the following Augmenting Cycle Theorem under our formulation of maximum flows.

Theorem 1.2 [5] Given a maximum flow problem (G,l,u,e^*) and a feasible flow x such that $x(e^*) < u(e^*)$, then either there is an augmenting e^*-cycle or a saturated e^*-cut with respect to x, but not both.

In terms of the coloring (1.1), this Augmenting Cycle Theorem says that if $x(e^*) < u(e^*)$ then exactly one of the following holds:

there is an e^*-cycle C with $C^+ \subseteq R \cup B$ and $C^- \subseteq Y \cup B$ (1.3a)

or

there is an e^*-cut D with $D^+ \subseteq R \cup W$ and $D^- \subseteq Y \cup W$. (1.3b)

This is true, of course, for any choice of a partition of E as $R \cup Y \cup B \cup W$ and $e^* \in R \cup Y$; it is just Minty's celebrated Colored Arc

Lemma [22] with an extra color, yellow, to incorporate the effect of reversing the orientation of red edges.

The Augmenting Cycle Theorem leads to a finite algorithm to solve the maximum e*-flow problem. Starting from a feasible flow x (which can be obtained by solution of a maximum flow problem with an obvious feasible flow - see Section 3) determine a cycle C as in (1.3a) or a cut D as in (1.3b). If an augmenting cycle C is discovered, increase flow by some $\delta > 0$ on C^+, decrease flow by δ on C^-, and iterate. If a saturated cut is discovered, stop. There is a natural way to determine which alternative holds, and with some insight it leads to nice bounds on the number of iterations. First we need to introduce some terminology concerning paths.

Say that path P is an $s - i$ path if s is the initial vertex and i is the final vertex in the corresponding sequence Q; let the length of P be $|P^+| + |P^-|$. Say that the $s - i$ path P is s-i augmenting with respect to the feasible flow x if $P^+ \subseteq R \cup B$ and $P^- \subseteq Y \cup B$, where the coloring is as in (1.1). Let $e^* = (t,s)$. Given a feasible flow x with $x(e^*) < u(e^*)$, the determination of whether there exists an augmenting e*-cycle is clearly equivalent to the determination of whether there is an augmenting s-t path. A natural approach is to begin with $V^0 = \{s\}$, and determine recursively the set V^k of vertices i such that there is an augmenting s-i path of length less than or equal to k. This continues until either

$$t \in V^k \tag{1.4a}$$

or

$$k = |V| - 1. \tag{1.4b}$$

If $t \notin V^{|V|-1}$ then the cut $D(V^{|V|-1})$ is saturated. (Obviously, if for some $1 \leq k < |V| - 1$ we find $V^k = V^{k-1}$, then we can stop since $|V|^{k+i} = |V|^{k-1}$ for all $i \geq -1$.) If $t \in V^k \backslash V^{k-1}$, then an s-t augmenting path of length k has been traced. Furthermore there are no s-t augmenting paths with fewer edges, which gives special significance to this breadth-first-search procedure. Suppose, as usual, the size of each augmentation δ is the maximum permitted by the bounds l and u, and all augmentations are on shortest augmenting s-t paths. Edmonds and Karp [4] showed that the number of augmentations until the algorithm halts with a saturated e*-cut and a maximum e*-flow must then be fewer than $|V||E|/2$. The search itself takes at most $O(E)$ steps giving a bound of $O(|V||E|^2)$ on the total running time. Dinits (see [25,chapter 9]) showed how to improve this to $O(|V|^2|E|)$ by performing more quickly a set of augmentations that saturates the subnetwork consisting of layers $V^0, V^1, \ldots, V^k$ of vertices (where k is the length of a shortest s-t augmenting path) and only edges of the form

$$(i,j) \quad i \in V^h, \quad j \in V^{h+1} \quad \text{and} \quad x(i,j) < u(i,j)$$

or

$$(i,j) \quad i \in V^{h+1}, \; j \in V^h \quad \text{and} \quad x(i,j) > l(i,j).$$

This layered network is saturated if the new flow admits no more augmentations of length k. Many further improvements have been based upon faster ways of saturating the layered network (see [25, chapter 9] and its references, and also the recent papers [27,28,30]). We employed in our code a version of the particularly simple method of Malhotra, Kumar, and Maheshwari [21], which gives a bound of $O(|V|^3)$ on the total running time.

2. The Maximum Flow Problem As a Subproblem of the Minimum Cost Network Flow Problem

The Augmenting Cycle Theorem for Maximum Flows (1.2), on which the maximum e*-flow algorithm is based, is generalized by the following well-known Augmenting Cycle Theorem for Minimum Cost Network Flows.

Theorem 2.1. Let x be a feasible flow in the minimum cost flow problem (G,a,l,u,b) ,and let R, Y, B, and W partition the edges of G as in (1.1). Then either

(a) there is a cycle C such that

C has $C^+ \subseteq R \cup B$, $C^- \subseteq Y \cup B$, and (2.2a)

$\Sigma\{a(e): e \in C^+\} - \Sigma\{a(e): e \in C^-\} < 0$

or

(b) there is a set of vertex weights $y \in Z^V$ such that:

$(a - yA)(e) > 0$ implies $x(e) = l(e)$, and (2.2b)

$(a - yA)(e) < 0$ implies $x(e) = u(e)$,

but not both.

Given (P) and x we will say that a cycle C as in (2.2a) is a negative cost augmenting cycle.

Suppose that (P) is, in fact, a maximum e*-flow problem. To see the relationship of (2.1) to (1.2) first note that every negative length cycle C in a maximum e*-flow problem must have $e^* \in C^+$, since e^* is the only edge with non-zero cost. Any e*-cycle C satisfying alternative (1.3a) of Theorem 1.2, also satisfies alternative (a) in Theorem 2.1. On the other hand, a vector $y \in Z^V$ satisfying (2.2b) with respect to x having $x(e^*) < u(e^*)$ must have $(a - yA)(e^*) \geq 0$, implying $y(t) > y(s)$, where $e^* = (t,s)$. Let $w \in \{0,1\}^V$ have $w(v) = 1$ whenever $y(v) \geq y(t)$ and $w(v) = 0$ whenever $y(v) < y(t)$. Then $y(v) - y(v')$ is positive (respectively, negative) whenever $w(v) - w(v')$ is positive (negative), and so (2.2b) is also satisfied by w in place of y. Furthermore, wA is the signed incidence vector of a saturated e*-cut and satisfies alternative (1.3b) of Theorem 1.2.

Recall that Theorem 1.2 is just Minty's Colored Arc Lemma with the coloring specified by (1.1). The Colored Arc Lemma can be used iteratively to provide an algorithmic proof of Theorem 2.1. Recall that we have assumed the data specifying the network are integral. We assume further that an initial integral feasible flow x and initial integral vertex weights y are given. Since x is feasible the following sets partition E:

$$\begin{aligned}
R &= \{e \in E: l(e) = x(e) < u(e) \text{ and } (a - yA)(e) = 0\};\\
Y &= \{e \in E: l(e) < x(e) = u(e) \text{ and } (a - yA)(e) = 0\};\\
B &= \{e \in E: l(e) < x(e) < u(e) \text{ and } (a - yA)(e) = 0\};\\
W &= \{e \in E: (a - yA)(e) \neq 0 \text{ or } l(e) = u(e)\}.
\end{aligned} \tag{2.3}$$

We also define the following subset of W:

$$W^* = \{e \in E: x(e) = l(e) \text{ and } (a - yA)(e) > 0 \text{ ; or } x(e) = u(e) \text{ and } (a - yA)(e) < 0 \text{ ; or } l(e) = u(e)\}.$$

If $W = W^*$, so $R \cup Y \cup B \cup W^* = E$, then $a - yA$ satisfies alternative (b) of Theorem (2.1). Otherwise, choose $e^* \in W \backslash W^*$. Either $x(e^*) > l(e^*)$, or $x(e^*) < u(e^*)$, depending on whether $(a - yA)(e^*) > 0$ or $(a - yA)(e^*) < 0$. Assume the latter. The edge e^* enters $R \cup Y \cup B \cup W^*$ if either $(u - x)(e^*)$ is decreased to 0 or $(a - yA)(e^*)$ is increased to 0. Consider applying the Colored Arc Lemma to $(R + e^*, Y, B, W \backslash e^*)$ and e^*. The signed incidence vector z of a cycle C satisfying alternative (a) may be used to update x to $x + z$. This decreases $(u - x)(e^*)$, since $z(e^*) = 1$. By the definition of the coloring, we may be assured that $R \cup Y \cup B \cup W^*$, defined with respect to $x + z$ and y, contains $R \cup Y \cup B \cup W^*$, defined with respect to x and y. A vector w, with wA the signed incidence vector of a cutset satisfying alternative (b), may be used to update y to $y - w$. This increases $(a - yA)(e^*)$, since $wA(e^*) = 1$. Here $R \cup Y \cup B \cup W^*$, defined with respect to x and $y - w$, contains $R \cup Y \cup B \cup W^*$, defined with respect to x and y. Thus, after at most $(u-x)(e^*) + (yA-a)(e^*) - 1$ applications of the Colored Arc Lemma, $e^* \in R \cup Y \cup B \cup W^*$, which has increased properly. Similarly, in the case $x(e^*) > l(e^*)$, the Colored Arc Lemma may be applied to the 4-tuple $(Y + e^*, R, B, W \backslash e^*)$ and e^* to obtain a reduction in either $(x - l)(e^*)$ with z satisfying (a) or $(a-yA)(e^*)$ with wA satisfying (b).

This method for determining an optimal pair from an initial coloring is a specialization of the out-of-kilter method as it applies to a problem with an initial feasible flow. The out-of-kilter method, in its full generality, was developed by Minty [22] and Fulkerson [8].

Determining which alternative of the Colored Arc Lemma holds is equivalent to determining which alternative holds in Theorem 1.2 in the graph $\hat{G} = (V, R \cup Y \cup B)$. The edge e^* will enter $R \cup Y \cup B \cup W^*$ after at most $|(a - yA)(e^*)|$ maximum e^*-flow subproblems. Repetition of this process until $W^* = W$ leads to an optimal solution of (P), as indicated by satifaction of alternative (b) of Theorem 2.1. (A similar approach can be based upon shortest path calculations instead of maximum flows.)

Of particular interest will be the case where $W \backslash W^* = \{e^*\}$ and $(a - yA)(e^*) = -1$. Then the incumbent solutions x and y can be updated, using the solutions z and w resulting from the solution of a single maximum e^*-flow subproblem, to a pair $x^* \equiv x + z$ and $y^* \equiv y - w$ satisfying alternative (b) of Theorem 2.1. This observation will be of central importance in the development of a scaling algorithm for the minimum cost flow problem (P) in the next section. Motivated by these observations we introduce the following definitions. A pair $(x \in \mathbb{R}^E, y \in \mathbb{R}^V)$ with $Ax = b$ will be called almost optimal if $|W\backslash W^*| = 1$; it will be called optimal if $|W\backslash W^*| = 0$. In accordance with the Colored Arc Lemma, a cycle C is augmenting with respect to a coloring (R,Y,B,W) if $C^+ \subseteq R \cup B$ and $C^- \subseteq Y \cup B$. A cut D is saturated with respect to (R,Y,B,W) if $D^+ \subseteq R \cup W$ and $D^- \subseteq Y \cup W$. A path P is augmenting with respect to (R,Y,B,W) if $P^+ \subseteq R \cup B$ and $P^- \subseteq Y \cup B$.

3. The Cost Scaling Algorithm

Given that the maximum e*-flow problem can be solved efficiently, we wish to develop an efficient algorithm for the solution of the minimum cost network flow problem (P). For a vector $d \in Z^E$, denote by $P(d)$ the variation on (P) in which the original cost vector a is replaced by d. In this notation, the original problem $P = P(a)$. The cost scaling method generates a sequence of cost functions that converge to a. This sequence, $\{a^i\}_{i=0}^q$, has the following properties.

It is easy to find an optimal pair (x^0, y^0) for $P(a^0)$. (3.1)

An optimal pair (x^{i+1}, y^{i+1}) for $P(a^{i+1})$ can be easily obtained from an optimal pair (x^i, y^i) for $P(a^i)$. (3.2)

The integer q is small with respect to the size of the input of $P(a)$. (3.3)

We have remarked previously that an optimal flow x^* with respect to the cost function that is identically zero can be obtained by solving a maximum e*-flow problem in a (relatively small) transformed graph. Moreover (x^*,y^*) is an optimal pair, where $y^* \equiv 0$. Thus (3.1) can be accomplished by setting $a^0 \equiv 0$.

A few definitions will simplify the description of a sequence $\{a^i\}$ with $a^0 \equiv 0$ that satisfies (3.2) and (3.3). The replacement of $P(d)$ by $P(d')$, denoted by $P(d) \leftarrow P(d')$, is called a perturbation. A perturbation is <u>scalar</u> if $d' = \alpha d$ for some $\alpha \in Z_+$; it is <u>simple</u> if for some choice of $e^* \in E$ we have $d'(e) = d(e)$ for all $e \in E \backslash e^*$ and $|d'(e^*) - d(e^*)| = 1$. We will now see how to accomplish (3.2) and (3.3) starting from $a^0 \equiv 0$ and using only a small number $(\Lambda - 1)$ of scalar perturbations, each with $\alpha = 2$, and at most $|E|$ simple perturbations between successive scalar perturbations. The parameter $\Lambda \equiv 1 + \lfloor \max \{\log_2(|a(e)|): e \in E\} \rfloor$ is the number of bits in the cost coefficients.

Two easy lemmas relate scalar and simple perturbations to the desired properties (3.2) and (3.3).

<u>Lemma 3.4</u> If (x,y) is an optimal pair for $P(d)$ and $\alpha \in \mathbb{R}_+$, then $(x, \alpha y)$ is optimal for $P(\alpha d)$.

Lemma (3.4) is clear since scaling by positive numbers does not alter the condition (2.2b). So scalar perturbations $P(a^i) \leftarrow P(a^{i+1})$ satisfy (3.2).

<u>Lemma 3.5</u> If (x,y) is an optimal pair for $P(d)$ and $P(d) \leftarrow P(d')$ is a simple perturbation, then (x,y) is either optimal or almost optimal for $P(d')$. Thus an optimal pair (x^*, y^*) for $P(d')$ can be obtained from (x,y) by solution of (at most) a single maximum flow subproblem.

That a single maximum flow subproblem suffices when (x,y) is not optimal for $P(d')$ follows immediately from the discussion of almost optimal pairs in Section 2. Note also that the size of the maximum flow problem is no larger than the size of $P(d)$. Hence simple perturbations $P(a^i) \leftarrow P(a^{i+1})$ also satisfy (3.2). Now we need only show that one can get from $a^0 \equiv 0$ to a by a short (in the sense of (3.3)) list of simple perturbations and scalar perturbations. It will be convenient to assume

$$a \leq 0 \quad \text{and} \quad a \not\equiv 0. \tag{3.6}$$

Certainly $a \not\equiv 0$ is reasonable since otherwise $a = a^0$. If $a(e) > 0$ for $e = (v_1, v_2)$, then we can replace e by its "opposite" $e' = (v_2, v_1)$ with $a(e') = -a(e)$, $u(e') = -l(e)$ and $l(e') = -u(e)$, so there is no loss of generality in (3.6).

It will now be convenient to index the edges. Write $E = \{e_1, \ldots, e_{|E|}\}$ and consider each entry $a(e_j)$ of a to be a (nonpositive) Λ-bit binary number. Let $q = \Lambda(1 + |E|) - 1$. Define a^i, $0 \le i \le q$, by

$$a^{j(1+|E|)+k}(e_h) = \begin{cases} \left\lceil a(e_h)/2^{\Lambda-(j+1)} \right\rceil , & 1 \le h \le k \\ 2\left\lceil a(e_h)/2^{\Lambda-j} \right\rceil , & k+1 \le h \le |E| \end{cases}$$

for all $j = 0, \ldots, (\Lambda - 1)$ and $k = 0, \ldots, |E|$. Note that $a^0 \equiv 0$, $a^q = a$; and for consecutive a^i and a^{i+1} either they are equal, or the replacement of $P(a^i)$ by $P(a^{i+1})$ is a simple or scalar perturbation. It is a scalar perturbation if $i+1$ is an integer multiple of $|E|+1$.

Think of each $a(e_h)$ as a Λ-bit binary number, and for convenience of notation, let $a^{-1} \equiv 0$. Then $a^{j(1+|E|)-1}(e_h)$, $0 \le j \le \Lambda$, is a j-bit number obtained by striking off the least significant $\Lambda - j$ bits of $a(e_h)$. So $2^{\Lambda-j}a^{j(1+|E|)-1}$ represents a to j significant bits. Or, considering the process in reverse, for $1 \le p \le |E|$ the vectors $a^{j(1+|E|)-1-p}$ are obtained by overwriting a zero in the least significant place starting from the $|E|$-th entry of $a^{j(1+|E|)-1}$ and working toward the first. Then the trailing zero is dropped from each entry of $a^{(j-1)(1+|E|)-1}$.

The remarks above demonstrate that the sequence of cost functions $\{a^i\}$ satisfies (3.1-3.3). This leads to an algorithm that halts with a pair (x^q,y^q) optimal for $P(a)$. The algorithm solves at most $\Lambda|E|$ maximum e^*-flow subproblems, each corresponding to a simple perturbation, and performs $(\Lambda - 1)$ scalar perturbations. Note that each of the maximum flow subproblems that arises in the course of this procedure has at most $|E|$ edges and $|V|$ vertices, since it is associated with the subgraph $(V, R \cup Y \cup B)$ of G. Also the magnitudes of the bounds in these subproblems are the same as in (P). So, given a polynomial-time maximum flow routine, e.g. the $O(|V|^3)$ algorithm of [21], each of the subproblems is solved in time that is bounded above by a fixed polynomial function of the size of the input of (P). Furthermore the number of subproblems is less than or equal to $\Lambda|E|$, which is also polynomial in the input size. Keep in mind that the input of (P) includes, among other things, each entry $a(e)$ of the cost vector a, whose binary representation has length

$1 + \lfloor \log_2(|a(e)|) \rfloor$ (plus a sign bit) if $a(e) \neq 0$. So this cost scaling algorithm for (P) runs in polynomial time.

Of course we will not need as many as $\Lambda|E|$ simple perturbations unless there are no zeros in the binary expansions of the costs $a(e)$. However our discussion until now was merely intended to show how the scaling approach of Edmonds and Karp [4] yields a polynomial-time cost scaling algorithm. In the statement of the minimum cost network flow algorithm MCNF at the end of this section, the "degenerate" simple perturbations that change nothing are skipped. In the next section we will indicate how to modify the approach described above for better computational performance. In particular we will make all of the simple perturbations corresponding to the same position in the $|E|$ bit-strings simultaneously, and solve the resulting (non-simply) perturbed problem as a sequence of no more than $|V|$ maximum flow problems.

For those familiar with the interpretation of the out-of-kilter method in terms of "kilter diagrams" (e.g., see [20]), cost scaling as above can be given the following interpretation in which the data are unscaled, but the "kilter step" changes over time. The approximation of $P(a)$ by $P(a^{j(1+|E|)-1})$ corresponds to making the horizontal (here it is blue) part of the kilter step not a line segment as usual, but a band below the horizontal axis of height $2^{\Lambda-j}$. The scalar perturbation corresponds to shrinking the height of the band. The mechanism of the simple perturbations corresponds to the shrinking being done one edge at a time.

In our formal description of scaling algorithms for (P) , based on the remarks above, executable statements are delimited by semicolons, labels are delimited by colons, and comments are italicized. We will have subroutine calls, including, of course, some to a maximum flow algorithm, MAXFLOW. MAXFLOW accepts as inputs the parameters $G = (V,E)$, l, u, and e^* and returns as outputs an optimal pair x, y. In general the parameters being passed in a subroutine call will be listed parenthetically with inputs and outputs separated by a semicolon, as in:

```
CALL MAXFLOW(G,l,u,e*;x,y).
```

We will make use of another routine TRANS, which enables us to solve the Phase I problem for (P) by a single call of MAXFLOW. TRANS will also be used later for another purpose. TRANS accepts as input $G = (V,E)$, l, u, b from (P), and outputs a transformed network $G' = (V',E')$, l', u', b', $e' \in E'$ and a feasible flow x' in the transformed network. For our later purposes it will be convenient to include among the inputs to TRANS subsets L, U, X that partition E, and a vector $x : E \to Z$. For our present purpose we will take $L = E$, $U = X = \emptyset$, and $x \equiv 0$. TRANS is described in Figure 3.1.

```
PROCEDURE TRANS($G=(V,E),l,u,b,L,U,X,x$ ; $G'=(V',E'),l',u',b',x',e'$):

   i)   two additional vertices, $s'$ and $s$, are added to the
        graph so that $V' \equiv V \cup \{s,s'\}$;
   ii)  $x'(e)$ is set to $l(e)$ for all $e \in L$, $x'(e)$ is set to $u(e)$
        for all $e \in U$, $x'(e)$ is set to $x(e)$ for all $e \in X$;
   iii) the discrepancy $\delta(v)$ in the conservation of flow equations
        for each vertex $v$ is calculated:

             $\delta(v) = b(v) + \Sigma\{x'(e): e = (v,v')$ and $e \in E\}$
                        $- \Sigma\{x'(e): e = (v',v)$ and $e \in E\}$;

   iv)  for each vertex $v$ with $\delta(v) > 0$ an edge
        $(v,s')$ is added with flow $x'(v,s') = -\delta(v)$,
        while for each vertex $v$ with $\delta(v) < 0$
        an edge $(s,v)$ is added with flow $x'(s,v) = \delta(v)$;
   v)   a distinguished edge $e' \equiv (s',s)$ is added with flow
        $x'(s',s) = \Sigma\{\delta(v): \delta(v) < 0\}$;
   vi)  SET $b'(v)=b(v)$ for all $v \in V$; $b'(s)=0; b'(s')=0$;
        SET $u'(e)=u(e)$ and $l'(e)=l(e)$ for all $e \in E$;
        SET $u'(e)=0$ and $l'(e)=x'(e')$ for all $e \in E'\backslash E$;
END TRANS
```

Figure 3.1 Transformation subroutine.

The Phase I problem for (P) can be solved, essentially, by solving an e'-maximum flow problem starting from x' in (G',l',u',b',e'), as output by TRANS. Specifically we solve

$$\begin{array}{ll} \text{maximize} & x(e') \\ \text{subject to} & A'x = b' \\ & l' \leq x \leq u', \end{array} \tag{3.7}$$

where A' is the vertex-edge matrix of G'. Technically (3.7) is not a maximum flow problem, since b' can be nonzero. But since x' is feasible for (3.7) we can convert (3.7) to the correct form by translation of the flow variables that are passed to and from MAXFLOW. We CALL MAXFLOW $(G',l'-x',u'-x',e';x,y)$. If $x(e') + x'(e') < 0$, then (P) has no feasible flow, and if $x(e') + x'(e') = 0$, then $x + x'$ restricted to E is a feasible flow in (P). This is a standard approach to the Phase I problem.

We can now state formally a scaling algorithm, MCNF, for problem (P).

```
Algorithm MCNF(G=(V,E),a,l,u,b ; x,y):
   1. Get an initial feasible solution for (V,E), l, u, b:
      CALL TRANS ((V,E),l,u,b,E,∅,∅,0;(V',E'),l',u',b',x',e');
      CALL MAXF ((V',E'),l' - x',u' - x',e';x,y);
           recall (V',E') is the transformed graph used to
           produce an initial solution.
      IF $x(e') \neq -x'(e')$ THEN STOP;
          the restriction $y_V$ of $y$ to $V$ gives the incidence
           vector of a cut $y_V A$ that blocks any feasible flow.
      SET $x \equiv x' + x$ restricted to E;
   2. Initialization:
      IF $a \equiv 0$ RETURN (x,0);
      SET $\Lambda \equiv 1 + \max\{\lfloor \log_2(-a(e)) \rfloor : e \in E\}$;
      SET $i \equiv \Lambda-1$; $\tilde{a} \equiv 0$; $\hat{a} \equiv \lceil (a/2^i) \rceil$;
```

```
    3. Perform maximum flow iterations to solve $\tilde{a} + \hat{a}$
           (Solve $\{0,-1\}$ perturbation problem):
       SET $U^i \equiv U \equiv \{e \in E: \hat{a}(e) = -1\}$;
       i)  Select $e^* \in U$ to be removed from $W$:
           If $U$ is empty GOTO 4;
           SELECT $e^* \in U$;
           SET $U \equiv U \setminus e^*; \tilde{a}(e^*) = \tilde{a}(e^*) - 1$;
           IF $x(e^*) = u(e^*)$ or $(\tilde{a} - yA)(e^*) \geq 0$ GOTO 3.i;
       ii) Adjust $x$ and $y$ so that $e^*$ leaves $W \setminus W^*$:
           SET $E'' = R \cup Y \cup B$
               where $E$ is colored $(R,Y,B,W)$ with respect to
               flow $x$, bounds $l$ and $u$, cost $\tilde{a} - yA$
               and $e^*$;
           CALL MAXFLOW $((V,E''), l - x, u - x, e^*; z, w)$;
           SET $x \equiv x + z$; $y \equiv y - w$;
         GOTO 3.i;
    4. Either stop or scale $\tilde{a}$ and $y$ and repeat 3:
       SET $i \equiv i - 1$;
       IF $i < 0$ RETURN $(x,y)$;
               $(x,y)$ is an optimal pair.
       SET $\tilde{a} \equiv 2\tilde{a}$;
       SET $\hat{a} \equiv \lceil (a/2^i) \rceil - \tilde{a}$;
       SET $y \equiv 2y$;
       GOTO 3;
END MCNF
```

Figure 3.2 Cost Scaling Algorithm for Minimum Cost Flows

4. Variations and Implementation

The algorithm MCNF described in Section 3 can be improved, both in its practical performance and its worst-case bound, by altering the third step. Step (3) perturbs the current vector $\tilde{a}$ of approximate cost coefficients with $\Lambda - i - 1$ bits to $\tilde{a} + \hat{a}$, the approximation with $\Lambda - i$ bits, one entry at a time. Increased efficiency results from perturbing the entire vector at once. Given an optimal pair (x,y) for $P(a)$ and a nonpositive perturbation vector $\hat{a}$, the procedure PERTURB described in Figure 4.1 can be used to calculate an optimal pair (x',y') for $P(a + \hat{a})$ faster than by repeating a maximum flow calculation for each edge e having $\hat{a}(e) \neq 0$.

```
PROCEDURE PERTURB (G=(V,E),a,l,u,b,x,y,$\hat{a}$ ; x',y'):
  1. *Initialize by setting edges to bounds
       so that (R,Y,B,W) partitions E:*
     SET $U \equiv \{e \in E: (a - yA)(e) + \hat{a}(e) < 0$ and $x(e) < u(e)\}$;
      *A is the vertex-edge incidence matrix of (V,E).*
     SET $X \equiv E \setminus U$;
     CALL TRANS ((V,E),l,u,b,$\emptyset$,U,X,x;(V',E'),l',u',b',x',e');
     SET $a'(e) \equiv (a + \hat{a} - yA)(e)$ for all $e \in E$,
         $a'(e) \equiv \min\{|U| + 1,|V|\} \cdot \min\{\hat{a}(e): e \in E\}$ for $e = e'$,
         $a'(e) \equiv 0$ for all other $e \in E'$;
     SET $y' \equiv 0$;
  2. *Perform maximum flow iterations until x'(e') = 0:*
     SET $E'' \equiv R \cup Y \cup B$
         where E' is colored (R,Y,B,W) with
         respect to x',l',u', and $a' - y'A$;
     CALL MAXFLOW ((V',E''),l' - x',u' - x',e';z,w);
     SET $x' \equiv x' + z$; SET $y' \equiv y' - w$;
     IF $x'(e') \neq 0$ and $(a'-y'A')(e') \neq 0$ GOTO 2;
        *A' is the vertex-edge incidence matrix of (V',E').*
  3. *x' restricted to E and y + y' restricted to V are
     returned:*
     SET $x' \equiv x'$ restricted to E;
     SET $y' \equiv y'$ restricted to V;
     SET $y' \equiv y + y'$;
         *(x', y') is an optimal pair for P(a + $\hat{a}$)*
END PERTURB
```

Figure 4.1 Perturbation Algorithm

It is here that we again make use of the routine TRANS. Initially (x,y) is an optimal pair. After adding the perturbation vector $\hat{a}$ to the transformed cost vector $(a - yA)$, TRANS is called in step (1). The flow on each edge e having $(a + \hat{a} - yA)(e) < 0$ is changed to $u(e)$. Two new vertices s and s' and a new edge $e' = (s',s)$ are added, as are edges of the form (s,i) or (i,s'), $i \in V$, as necessary to create conservation of flow. Upon return from TRANS, every edge except the new edge e' is in $R \cup Y \cup B$. MAXFLOW is called in step (2) until the flow $x'(e') = 0$. At that time (x',y') is an optimal pair for the transformed problem, and the restrictions of x' and y' to the original edge set E and vertex set V, respectively, form an optimal pair for $P(a + \hat{a})$.

The magnitude of $a'(e')$ fixed in step (1) is chosen to be large enough to cause the flow on e' to be zero in any optimal solution of $P(a')$, but small enough to give a small upper bound $|V|$ on the number of times that step (2) can be iterated. This then limits the number of calls of MAXFLOW. That $a'(e')$ is large enough is a consequence of the following lemma with $(a + \hat{a} - yA)$ from step (1) in the role of a in the lemma. This Lemma and its use here are similar to Corollary 3.3 in in Ford and Fulkerson [5] and its use there in the treatment of the minimum cost flow problem.

Lemma 4.1 Let (P) be a minimum cost flow problem in $G = (V,E)$, and let $e' \in E$. For any feasible flow x, define $K(x) \equiv \{e \in E: a(e) = 0$, or $a(e) < 0$ and $x(e) = u(e)$, or $a(e) > 0$ and $x(e) = l(e)\}$, and $\bar{K}(x) \equiv E \backslash K(x)$. Suppose there exists a feasible flow x^0 such that $e' \in K(x^0)$ and

$$|a(e')| > (|V|-1)\ |a(e)| \quad \text{for all} \quad e \in \bar{K}(x^0). \tag{4.2}$$

Then for every optimal flow x^* we have $e' \in K(x^*)$; i.e. $x^*(e') = u(e')$ if $a(e') < 0$ and $x^*(e') = l(e')$ if $a(e') > 0$, or, equivalently, $x^*(e') = x^0(e')$ if $a(e') \neq 0$.

Proof Let x^0 be a feasible flow as in the hypothesis, and let x^* be an optimal flow violating the conclusion, so either

$$a(e') < 0 \quad \text{and} \quad x^*(e') < u(e')$$

or

$$a(e') > 0 \quad \text{and} \quad x^*(e') > l(e'). \tag{4.3}$$

The nonzero vector $(x^0 - x^*)$ is in the null space of A, the vertex-edge incidence matrix of G, and $(x^0 - x^*)(e') \neq 0$. Hence there is a cycle C in G containing e' and having

$$C^+ \subseteq \{e \in E: (x^0 - x^*)(e) > 0\}$$

and

$$C^- \subseteq \{e \in E: (x^0 - x^*)(e) < 0\}.$$

Now $e \in C^+$ implies $x^0(e) > x^*(e) \geq l(e)$. So

$$e \in K(x^0) \cap C^+ \Rightarrow a(e) \leq 0. \tag{4.4a}$$

Similarly $e \in C^-$ implies $x^0(e) < x^*(e) \leq u(e)$; so

$$e \in K(x^0) \cap C^- \Rightarrow a(e) \geq 0. \tag{4.4b}$$

Now consider

$$a(C) = \sum_{e \in C^+} a(e) - \sum_{e \in C^-} a(e) . \tag{4.5}$$

By (4.4) each term in (4.5) that comes from an $e \in K(x^0)$ is nonpositive, and by (4.3) the term that comes from e' is negative, it is $-|a(e')|$. Now, since a cycle has at most $|V|$ edges, and C has at least one edge, e', in $K(x^0)$, $|C \cap \bar{K}(x^0)| \leq |V| - 1$. Then from (4.2) we see that the contribution to (4.5) from those $e \in \bar{K}(x^0)$ has magnitude

$$\left| \sum_{e \in C^+ \cap \bar{K}(x_0)} a(e) - \sum_{e \in C^- \cap \bar{K}(x_0)} a(e) \right| < |a(e')| .$$

Therefore $a(C) < 0$, which contradicts the optimality of x^*, since C is augmenting with respect to x^*.■

In the application of Lemma 4.1 to PERTURB note that $(a + \hat{a} - yA)(e) > 0$ implies $(a - yA)(e) > 0$ which then implies $x(e) = l(e)$. Similarly $(a + \hat{a} - yA)(e) < 0$ implies either $(a - yA)(e) < 0$ or $0 \leq (a - yA)(e) < 1$ and $\hat{a}(e) = -1$. So either $x(e) = u(e)$, or $|(a + \hat{a} - yA)(e)| \leq |\hat{a}(e)|$.

This approach improves the bound on the scaling algorithm by reducing the bound on the number of maximum flow subproblems solved. Lemma 4.1 allows us to conclude that step (2) of procedure PERTURB is executed at most $|V|$ times. Since PERTURB will be called at most Λ times, at most $\Lambda|V|$ maximum flow subproblems are solved from within PERTURB, plus one more in the initialization. On the other hand, step (3.ii) of algorithm MCNF can be executed $|U^i|$ times during each major iteration. Since $|U^i|$ can be as large as $|E|$, MCNF may have to solve as many as $\Lambda|E| + 1$ maximum flow subproblems to solve P(a).

A heuristic reason for expecting better behavior from the use of PERTURB is provided by the following observation. In setting all the flows on edges with negative transformed cost in the perturbed problem to their upper bounds, the total amount of flow assigned to the special edge e' may be significantly lower than the total of these upper bounds, due to cancellation of flow at the vertices. As a result, the total flow adjustment in any major iteration of the scaling algorithm is expected to be lower when all the edges are perturbed at the beginning of the iteration, rather than sequentially.

A second modification of the cost scaling algorithm applies the scaling in base n instead of base 2. We use the algorithm given in Figure 4.2 to solve the perturbation problem where $\hat{a}$ takes values in $\{0,-1,\ldots,1-n\}$. The bound on the number of iterations of step (2) of PERTURB required to attain a flow with $x'(e') = 0$ then becomes $(n - 1)|V|$.

The last modification of MCNF combines step (1) of MCNF, the initialization, with the first iteration of step (3). After the combined iteration, one has either detected that (P) is infeasible, or produced an optimal solution to the subproblem P(a) in which the magnitude of each a(e) is equal to the value of the most significant bit in a(e). PERTURB is called with an initial flow of $x' \equiv u$. This initial flow is likely to be infeasible. If the flow x returned by PERTURB is not feasible, then by Lemma (4.1) there is no feasible solution of (P). This happens precisely when $(a'-y'A')(e') = 0$ upon exiting from PERTURB in Step 2. This exit condition can only occur in the initial return from PERTURB. Since all subsequent calls will pass a feasible flow to PERTURB, Lemma 4.1 guarantees it returns with $x'(e') = 0$ and $(a'-yA')(e') < 0$.

Figure 4.2 describes MCNF2, the variant of MCNF that incorporates the three modifications discussed above.

```
Algorithm MCNF2(G=(V,E),a,l,u,b ; x,y):
  1. Initialization:
     SET $\Lambda \equiv 1 + \max\{\lfloor \log_n(-a(e)) \rfloor : e \in E\}$;
     SET $i \equiv \Lambda - 1$; $\tilde{a} \equiv 0$; $\hat{a} \equiv \lceil (a/n^i) \rceil$;
     SET $x \equiv u$; $y \equiv 0$
  2. Perform maximum flow iterations to solve $\tilde{a} + \hat{a}$
        (Solve $\{0,-1,-2,\ldots,1-n\}$ perturbation problem):
     SET $x' \equiv x$; $y' \equiv y$;
     CALL PERTURB $((V',E'),\tilde{a},l',u',b',x',y',\hat{a};x,y)$;
  3. Either stop or scale $\tilde{a}+\hat{a}$ and $y$ and repeat 2:
     IF x is not feasible,STOP;
        There is no feasible solution.
     SET $i \equiv i - 1$;
     IF $i < 0$ STOP;
       (x,y) is an optimal pair
     SET $\tilde{a} \equiv n(\tilde{a} + \hat{a})$;
     SET $\hat{a} \equiv \lceil (a/n^i) \rceil - \tilde{a}$;
     SET $y \equiv ny$;
     GOTO 2;
END MCNF2
```

Figure 4.2 Modified Cost Scaling Algorithm

In the following sections we report on a computational study of SCALE, an implementation of MCNF2. This implementation is flexible enough to allow scaling in any base. Although the leading constant in the worst-case bound for the scaling algorithm is best when the scaling base is two, we found in preliminary testing that our implementation of the scaling code solved the tested problems faster when the base was set to four. All of the data reported here are for base four.

An efficient implementation of the algorithm MCNF2 requires an efficient subroutine MAXFLOW to solve the subproblems generated by the subroutine PERTURB. Of the known maximum flow algorithms one of the simplest to implement is due to Malhotra, Kumar, and Maheshwari [21]. Although this MKM algorithm is the most efficient known for dense graphs, there are others that are asymptotically more efficient in the worst case on sparse graphs. These algorithms, however, require more sophisticated data structures for implementation. For problems with a small number of vertices, the algorithm of [21] is practically more efficient. Our tests focus on fully dense square transportation problems, and so it is natural to employ the algorithm of [21]. Note that a fully dense transportation problem with, say, 200 sources and 200 sinks will have 40,000 edges. The initial maximum flow subproblems will be dense. As SCALE iterates, the later subproblems will become rather sparse, and it would be useful to be able to take advantage of the sparsity. But the number of vertices is not large enough to cover the overhead of more complicated data structures.

The specific implementation of the MKM algorithm employed here differs in two respects from [21]. First, the selection of a vertex v^* of minimum potential is refined to select v^* to lexicographically minimize (potential (v), layer number (v)). This allows for a simpler procedure to remove saturated vertices. Second, incidence lists rather than linked lists are used to maintain the appropriate partition of the edges at each vertex v into blocks consisting of those edges: (1) between v and the next layer; (2) between v and the preceding layer; (3) incident with v but absent from the present layered network. For our special use of the MKM algorithm as a subroutine in MCNF2 it is also necessary to partition the last block into edges that are present in the current maximum flow subproblem (blue) and these that are not (white). The use of incidence lists permits some savings in storage with no additional computational expense.

The other codes used in these tests are GNET, developed by Bradley, Brown, and Graves [2], and RNET, developed by Grigoriadis and Hsu [13]. Both are primal simplex codes that use list structures for efficient maintenance of the basis and its inverse. They differ primarily in the method of pivot selection. GNET uses a Big M method, along with a candidate queue and a vertex scan to obtain edges for basis entry. RNET uses a pseudo-random pricing strategy along with a gradual penalty method for costing artificial edges.

The simplex codes are both extreme point codes. As a result, the data structures used to store the current solution have length $|V|$. The basis tree is represented using either three or four vertex length arrays. The capacities and costs are stored in an $|E|$-length array. GNET sorts the edges so that one $|V|$- length array and one $|E|$-length array can be used to store the endpoints. The time for this sort is not included in our GNET execution times. RNET uses two $|E|$-length arrays for the same purpose to avoid the initial sort.

SCALE requires one $|E|$-length array for the capacities, one for the costs, and one for the current flow. In addition, SCALE requires two $2|E|$-length arrays for maintenance of the layered network. In the current implementation, two more $|E|$-length arrays are used to store the endpoints of a given arc. All other storage for SCALE is either $|V|$-length arrays, of which there are twenty, or constant storage. Thus, SCALE requires nine edge-length arrays, while GNET requires only three, and RNET requires four. The factor of two to three times the storage requirements could be a serious impediment to using this implementation of MCNF2 for problems with many edges. But, for the time being we will be concerned only with computational speed.

5. The Computational Experiment

The broad purpose with which we began this work was: to understand better the computational behavior of network flow algorithms, including those based on both the simplex method and the out-of-kilter method with data scaling, and specifically, to contrast their behavior in order to test the widely held presumption that data scaling does not yield computationally effective algorithms. For specific implementations of these methods and within specific classes of problems, we planned to collect data from Monte Carlo experiments. The data would then be used for comparisons of execution times of the different codes on problems with similar characteristics, and in an attempt to model execution times as a function of problem characteristics.

The code SCALE, based on the cost scaling algorithm MCNF2 of the previous section, and two network simplex codes GNET (Bradley, Brown, and Graves [2]) and RNET (Grigoriadis and Hsu [13]), were employed. We used two different network simplex codes in the experiments because earlier work indicates clearly the sensitivity of performance of the simplex method to such factors as pivot selection strategy.

We restricted the generated problems to capacitated transportation problems that are square (equal numbers of sources and sinks). This property, and the form of the CAPT generators that we devised, were chosen to impose symmetry on the problem distributions. Indeed each of the three generators is essentially characterized by a symmetry condition (see the appendix for details on the CAPT generators). We also restricted the samples to completely dense (bipartite) networks. This is consistent with the symmetry desired, and was imposed, instead of some mechanism to create, say, a sparse symmetry, because of the character of the issue under examination, and the specific implementation of MCNF2 employed. This implementation, SCALE, seems much better suited to dense networks than sparse ones. Preliminary tests bore this out. Since the conventional wisdom has been that scaling algorithms would fare very poorly in actual computation, one wants to test this particular implementation in a "non-hostile" domain. Keep in mind the necessity of restricting the eventual conclusions to the domain from which the problems were drawn. Test problems can be meaningful only so long as one does not attempt to impute fine interpretations of the algorithm's performance to settings different from the one in which the problems arose. That SCALE is likely to fare poorly on sparse problems is apparent from its performance on NETGEN problems. Table B.1 in Appendix B of [1] gives execution times of the three codes on 35 benchmark NETGEN problems [19]. (The benchmark times were produced to facilitate comparison of our work with work performed in other machine environments.) The times on the benchmark NETGEN problems, which are all sparse, contrast strikingly with the times on the dense NETGEN problems that we generated, and report on in Section 6; SCALE does very badly on all of the benchmark problems. A different implementation might be better suited to sparsity.

The main experiment was designed to have four phases: (1) preliminary testing; (2) collection of large sets of observations from three different distributions; (3) modeling and validation; and (4) evaluation of the predictive power of the models. Immediately below are brief remarks about the four phases. These are followed by detailed comments on the statistical analysis conducted in phases (3) and (4). See Draper and Smith [3] for reference material on the regression analysis.

In the first phase of the experiment we gathered preliminary data to assist in the determination of appropriate ranges on the input parameters to the three CAPT generators. Each generator together with a specification of the intervals from which the five generator parameters are selected (uniformly), determines a problem distribution (see the appendix). Of the five generator parameters, the two to which the observed execution times were clearly sensitive were: the number of vertices, $|V|$, which determines the number of edges, $|E| = |V|^2/4$; and the number of bits in the costs, Λ. We fixed the intervals from which these parameters would be selected uniformly in the second phase of the experiment to $|V| \in [100,500] \cap 2Z$ and $\Lambda \in [4,10] \cap Z$. This results in the costs on the $|E| \in [10000,62500]$ edges being selected uniformly from the interval $[0,2^\Lambda - 1]$. The expected total flow from sources to sinks in the test problems is approximately $50|E|$ in distributions #1 and #2, and $25|E|$ in distribution #3. The preliminary data also provides an inkling of qualitative changes in algorithmic behavior outside of the chosen ranges for which considerable data was eventually compiled and studied. Some additional details on this preliminary phase can be found in the appendix following the description of the generators.

In the second phase of the experiment large sets of data were collected. One hundred problems were generated from each of the three distributions specified in the first phase. Every problem was solved to optimality by all three codes. Total execution times and generator parameter values were recorded in nine data sets, one for each combination of the three distributions and three codes. The analysis conducted in the third phase of the experiment is based on these data sets. We also created additional data sets with the seeds (for reproducibility), and such additional information as execution times for each iteration of SCALE, and number of saturated edges at termination, for later, more detailed, analysis.

In the third phase of the experiment execution times were modeled as functions of the generator parameters. Regressions yielded estimates of the model parameters. Log-linear models of execution times as functions of $|V|$ (or, equivalently, $|E|$) and, for SCALE only, Λ, were selected. The set of one hundred observations and the fitted curve for each of the nine data sets were given in Figure 0.1 (and B.1 of [1]). The significance, fit, and precision of each of the nine regression functions were evaluated using standard techniques. The results were favorable in each case. The assumptions on error that underlie the statistical methods were supported by examination of residuals. Confidence intervals were calculated for the estimated parameters, for predicted median execution times and for individual execution times.

In the fourth phase of the experiment we collected nine additional sets of data independent of those on which the regressions were performed in the third phase. These were used to test the predictive power of the nine models. For each of the nine combinations of a code and distribution, the curve of predicted median execution time as a function of $|V|$ (and Λ for SCALE), and the new set of one hundred observations was displayed in Figure 0.2 (and B.2 of [1]). The quality of the predictions is striking in each case.

Modeling and Validation

To develop models of the execution times of SCALE, GNET, and RNET as functions of the generator parameters, various regressions were performed for each of the nine data sets from the second phase of the experiment. In each case a log-linear form seemed to be the best of several forms examined. A complete set of thirty-two log-linear regressions, one for every subset of the five generator parameters, was then run for each of the nine data sets. Analysis of the regressions led to the conclusion that

$$T = \varepsilon\, e^{\beta_0} |V|^{\beta_1} \Lambda^{\beta_2} \tag{5.1}$$

is an appropriate model of the random variable T, observed execution time. Here e is the base of the natural logarithm, and, as usual, $|V|$ is the number of vertices, and Λ is the number of bits in the cost coefficients. The dependent random variable T and the nonnegative random variable ε should be regarded to be functions $T(|V|,\Lambda)$ and $\varepsilon(|V|,\Lambda)$, respectively, of the independent variables $|V|$ and Λ. Later we will address the issue of whether the random variables $\varepsilon(|V|,\Lambda)$ are independent and identically distributed. These random variables represent inherent variability in observed execution time for problems with fixed $|V|$ and Λ. Estimates b_0, b_1, b_2, respectively, of the parameters β_0, β_1, β_2, were determined by the regression for each combination of code and problem distribution. In the cases of GNET and RNET β_2 was taken to be zero and only β_0 and β_1 were estimated, so (5.1) reduced to

$$T = \varepsilon e^{\beta_0} |V|^{\beta_1} . \tag{5.2}$$

Although the estimate b_2 of β_2 was nonzero when included in the regression, the relative amount of error explained by it, as measured by its t-value, was insufficient to justify its inclusion in the models for GNET and RNET.

Parts of the statistical analysis are founded upon the assumption:

the random variables $\varepsilon'(|V|,\Lambda) \equiv \log \varepsilon(|V|,\Lambda)$
are independent and identically distributed (5.3a)
with mean zero and (unknown) variance σ^2 .

In other places we appeal to an even stronger condition:

the random variables $\varepsilon'(|V|,\Lambda) \equiv \log \varepsilon(|V|,\Lambda)$
are independent and <u>normally</u> distributed (5.3b)
with mean zero and (unknown) variance σ^2 .

The assumption (5.3b) implies that the error ε is a lognormal random variable. We will make clear where either of these assumptions is involved, and we will examine their validity.

By logarithmic transformations of (5.1) and (5.2) the dependent variable in the regressions becomes $Y \equiv \log T$ and the independent variables become $\log(|V|)$ and $\log(\Lambda)$. We then fitted the linear models

$$Y = \beta_0 + \beta_1 \log(|V|) + \beta_2 \log(\Lambda) + \varepsilon' \tag{5.4}$$

for SCALE and

$$Y = \beta_0 + \beta_1 \log(|V|) + \varepsilon' \tag{5.5}$$

for GNET and RNET.

In this discussion of the regressions, and henceforth, the term "parameter" will refer to a parameter in the model, as opposed to a generator parameter.

Table B.2 in Appendix B of [1] summarizes the information produced from the regressions in which the selected models (5.4) and (5.5) were fitted to the data. Table 5.1 below abstracts from Table B.2 of [1] the estimated value b_i of each β_i in (5.4) or (5.5), for the nine data sets.

Table 5.1

Parameter Estimates

	b_0	b_1	b_2
Distribution #1			
SCALE	-10.495	1.976	0.793
RNET	-11.648	2.313	
GNET	-12.632	2.743	
Distribution #2			
SCALE	-10.864	2.061	0.749
RNET	-12.343	2.439	
GNET	-12.586	2.737	
Distribution #3			
SCALE	-10.922	2.078	0.737
RNET	-12.384	2.450	
GNET	-12.599	2.713	

So, for example, we see that $b_0 = -10.495$, $b_1 = 1.976$, and $b_2 = 0.793$ for SCALE and distribution #1. Thus the fitted model estimates the mean of the logarithm of SCALE execution times on problems from distribution #1 by the function

$$-10.495 + 1.976 \log(|V|) + 0.793 \log(\Lambda).$$

This corresponds to an estimate of median execution time by the function

$$\hat{T}_{med} = e^{-10.495} |V|^{1.976} \Lambda^{0.793}. \tag{5.6}$$

Tests of Fit

Once a regression is run, one wants to determine whether it has fitted the data well. Data points with the same values of the independent variables are grouped and the group means are calculated. Let $\tilde{Y}$ denote the vector obtained from the observation vector Y by replacing each entry by its group mean, and let $\hat{Y}$ denote the estimation vector obtained from the original linear regression. If a new regression were run on $\tilde{Y}$, the resulting estimates of the parameters and, therefore, the estimation vector $\hat{Y}$, would be the same as in the regression on Y. Consequently, error of the form $Y - \tilde{Y}$ cannot be accounted for as a function of the independent variables, whereas error of the form $\tilde{Y} - \hat{Y}$ can be. Since the vectors $Y - \tilde{Y}$ and $\tilde{Y} - \hat{Y}$ are orthogonal and their sum is the vector $Y - \hat{Y}$ of residuals, the sum of the squared lengths of $Y - \tilde{Y}$ and $\tilde{Y} - \hat{Y}$ is the sum of the squares of the residuals. The model fits the data well if the length of $\tilde{Y} - \hat{Y}$ is small relative to the length of $Y - \tilde{Y}$, so that a relatively large part of the residual sum of squares can be explained as <u>pure error</u> from $Y - \tilde{Y}$, as opposed to <u>lack-of-fit error</u> from $\hat{Y} - \tilde{Y}$. Under the additional assumption that the errors ε' are normally distributed with constant variance, the squared lengths of $Y - \tilde{Y}$ and $\tilde{Y} - \hat{Y}$ are independent chi-squared random variables, and the ratio of their squared lengths normalized by the appropriate degrees of freedom has an F-distribution. This F-statistic is used to test the hypothesis that Y is a linear function of the independent variables. The hypothesis is rejected at a specified level of confidence, $(1 - \alpha)$, if the ratio exceeds the $(1 - \alpha)$ percentile of the appropriate F distribution.

The original data sets contain enough observations with repeated values of $|V|$ to conduct meaningful lack-of-fit tests for GNET and RNET. The model (5.4) for SCALE has two independent variables, $\log(|V|)$ and $\log(\Lambda)$, and there were very few repeats of the pair of values. Hence it was necessary to generate three new data sets for SCALE with repeated values of the pair $(|V|,\Lambda)$. The problems were generated as in the second phase of the experiment, but with five repetitions of each of the twenty-one combinations of $|V| \in \{100, 300, 500\}$ and $\Lambda \in \{4,5,6,7,8,9,10\}$. The lack-of-fit tests on these data sets for SCALE do not result in rejection of the hypothesis that $Y = \log(T)$ is linear in $\log(|V|)$ and $\log(\Lambda)$ at the 90% level of confidence for any of the three distributions. Similarly, using the original data sets for GNET we are unable to reject the linearity hypotheses. For RNET, the hypothesis can be rejected with 90% confidence for the data set from distribution #1, but not for either of the other two. We generated a new data set for RNET from distribution #1 with ten repetitions each of $|V| \in \{100,200,300,400,500\}$, and ran a new regression and a new goodness-of-fit test. The estimated parameter values are within two standard deviations of the estimates from the original for RNET, and the F-value of .8393 with 3 and 45 degrees of freedom is well below the critical level at 90% confidence. Table B.3 in Appendix B of [1] contains the results of the goodness-of-fit tests.

Since none of the nine models exhibited significant lack-of-fit, we are sufficiently well-satisfied with the adequacy of the models to examine their significance and precision. The plots in Figure 0.1 convey, better than any single statistic, the significance and precision of the models. However, we will report below on conventional measures.

Under the assumption (5.3b) that the errors ε' are independent identically distributed normal random variables, an F-test can be used to measure the significance of the regression. Technically, one tests the hypothesis that all parameters β_i, $i > 0$, are zero. The hypothesis is rejected with $100(1 - \alpha)\%$ confidence if the F-value, mean square due to regression, $(\Sigma(\hat{Y}_i - \bar{Y})^2)$ divided by mean square about regression, $(\Sigma(Y_i - \hat{Y}_i)^2/\nu)$, is larger than the $100(1 - \alpha)\%$ point of the $F(1,\nu)$ distribution. Here $\bar{Y}$ is the mean observation and ν is the number of degrees of freedom, $|V|$-2 for the RNET and GNET data sets, or $|V|$-3 for the SCALE data sets. The calculated F-values appear in Table B.2 of Appendix B of [1]. They are all in the range 1924-19069, while the critical F value at 90% confidence is less than ten. Thus all nine regressions are found to be significant.

The correlation coefficient R, $-1 \leq R \leq 1$, between the prediction vector $\hat{Y}$ and the observation vector Y is one conventional measure of precision in regression analysis. R is the cosine of the angle between $(Y - \bar{Y}\cdot 1_{|V|})$, the vector of differences of the observed values from their mean, $\bar{Y}$, and $(\hat{Y} - \bar{Y}\cdot 1_{|V|})$, the vector of differences of predicted values from their mean, which is also $\bar{Y}$. R^2 can be regarded to be the fraction of observed variation about the mean $\bar{Y}$ that is explained by the regression. All nine R^2 values are greater than .976.

Given that the errors ε' are independent and identically distributed random variables, s^2, the mean square error about regression, is an unbiased estimator, based on ν degrees of freedom, of the variance of ε'; $s^2 = \Sigma (Y_i - \hat{Y}_i)^2/\nu$. One can then estimate the standard deviation of β_i from s and the observed values of the independent variables. Under the additional assumption (5.3b) that the ε' are normal, the resulting <u>standard error</u> s_i of b_i can be used to construct confidence intervals for β_i since b_i has a t-distribution. With $100(1 - \alpha)\%$ confidence β_i is in the interval $b_i \pm s_i\, t(\nu, 1 - \alpha/2)$. The 90% confidence intervals for all of the estimated parameters are given in Table 5.2 below.

<u>Table 5.2</u>

Confidence Intervals for the Parameters β_i in (5.4) and (5.5)

	β_0	β_1	β_2
<u>Distribution #1</u>			
SCALE	(-10.694 , -10.296)	(1.946 , 2.006)	(0.748 , 0.838)
RNET	(-12.139 , -11.157)	(2.227 , 2.399)	
GNET	(-12.913 , -12.351)	(2.693 , 2.793)	
<u>Distribution #2</u>			
SCALE	(-11.100 , -10.628)	(2.021 , 2.101)	(0.694 , 0.804)
RNET	(-12.834 , -11.852)	(2.353 , 2.525)	
GNET	(-12.774 , -12.398)	(2.704 , 2.770)	
<u>Distribution #3</u>			
SCALE	(-11.164 , -10.680)	(2.041 , 2.115)	(0.681 , 0.793)
RNET	(-12.907 , -11.861)	(2.357 , 2.543)	
GNET	(-12.894 , -12.304)	(2.660 , 2.766)	

The narrow ranges of the confidence intervals are indicators of good precision. Note in particular that zero is not in the interval for β_2 in any of the models for SCALE. This supports the inclusion of Λ in those models.

In the preliminary regressions for GNET and RNET, which included Λ as an independent variable, as well as $|V|$, zero was in the 90% confidence intervals for β_2. This is why Λ was dropped from those models.

Examination of Residuals

Let us now address the assumptions (5.3) on the term ε' in the models (5.4) and (5.5). The stronger of the two assumptions, (5.3b), is that for each combination of a code and a problem distribution, the random variables $\varepsilon'(|V|, \Lambda)$ are independent normals with mean zero and constant, but unknown, variance, σ^2. If (5.3b) were correct, then we could write ε' in place of $\varepsilon'(|V|, \Lambda)$, and ε'/σ would be standard normal. Then we could estimate σ by s, computed in the regressions. Assumption (5.3b) implies further that the studentized residuals, $(Y - \hat{Y})/s$, are t-distributed with 97 or 98 degrees of freedom, 97 for the SCALE samples and 98 for the RNET and GNET samples. The t-distribution with 97 or 98 degrees of freedom is almost exactly standard normal. So, for each of the nine samples, the 100 studentized residuals should be consistent with a standard normal distribution, as should the aggregation of all 900 studentized residuals. Figure 5.1 supports this. Part (a) of Figure 5.1 is a plot of the 900 studentized residuals plotted against $|V|$, part (b) is a histogram, and part (c) is a normal probability plot. (One GNET observation from distribution number one has its studentized residual less than -10, and does not show up in the plots; all of the other residuals appear.)

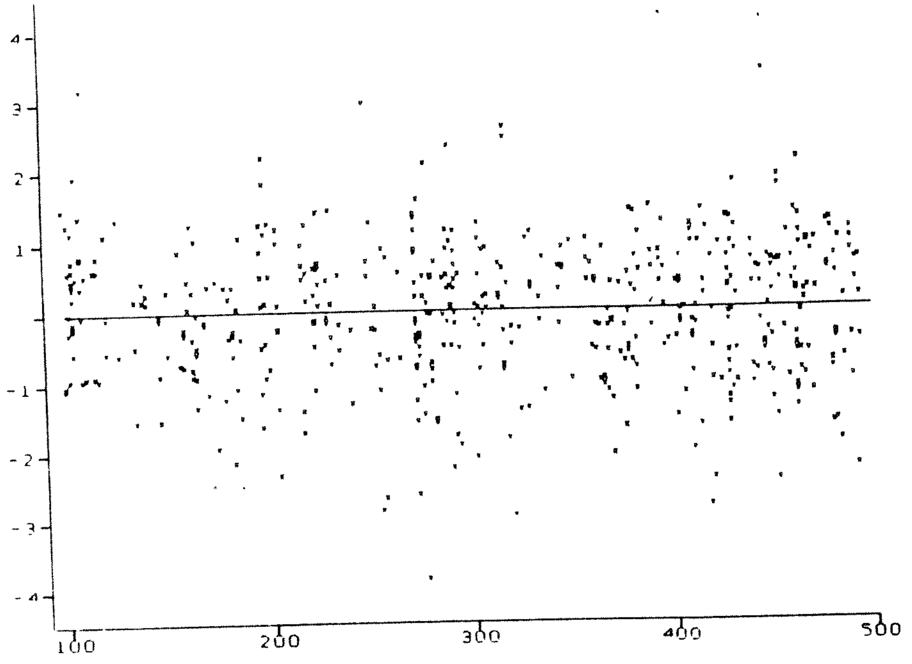
A normal probability plot for a sample $Z_1, \ldots, Z_n$ is a plot of the pairs

$$(Z_{i(j)}, F^{-1}(i(j) - 1/2)/n),$$

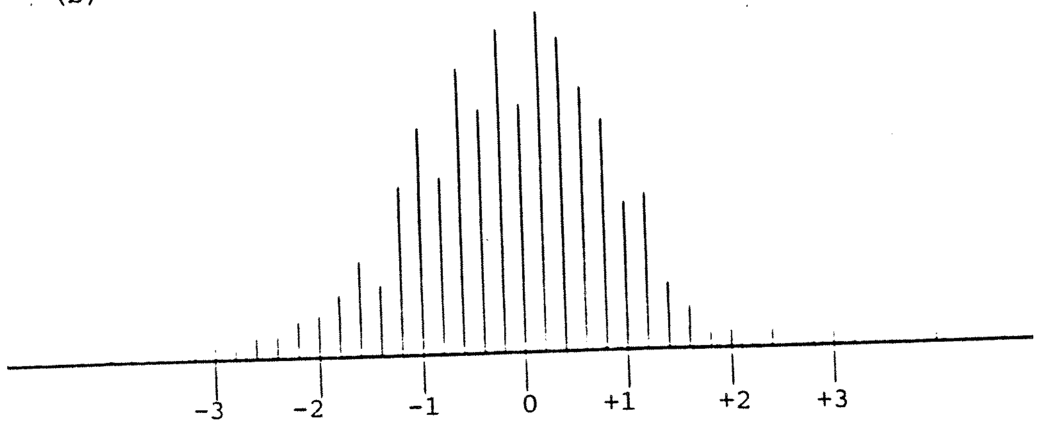
where $Z_{i(1)} \leq Z_{i(2)} \leq \cdots \leq Z_{i(n)}$ and F is the standard normal cumulative distribution function. If the sample is drawn from the standard normal distribution, the plot should be close to a straight line with slope one and intercept zero.

Figures (5.1a) and (5.1b) appear to be consistent with (5.3b), and Figure (5.1c) is rather convincing. At the resolution in this probability plot there is no deviation from the 45° line in the interval from -2 to +1 standard deviations, and the only substantial deviations are well into the tails.

(a)



(b)



(c)

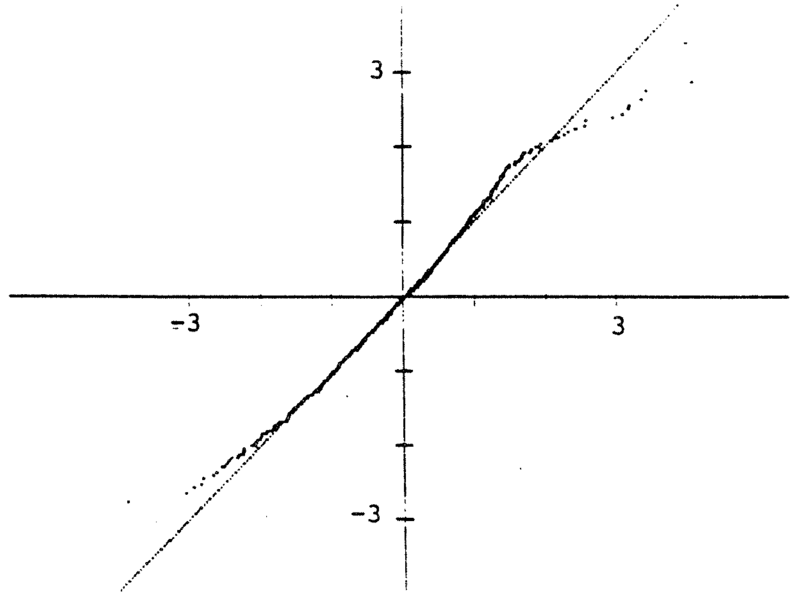


Figure 5.1 Plots of aggregated studentized residuals.

(a) Residuals versus number of vertices

(b) Histogram of residuals.

(c) Probability plot: inverse normal versus ordered residuals.

Figures B.3, B.4, and B.5 in Appendix B of [1] present separate probability plots from the nine samples. The resemblance to standard normal behavior here is not as striking as with the aggregated data, but again the resemblance is reasonably good, except in the tails. For comparison we generated nine sets of 100 approximately standard normal variates by normalizing sums of forty-eight uniform [0,1] variates. Figure B.6 of [1] presents the probability plots for these samples; they are difficult to distinguish from the plots from our nine samples, except in the tails.

The examination of residuals leads to the conclusion that the normality assumption (5.3b) is approximately correct, except in the tails. If the $\varepsilon'(|V|, \Lambda)$ were exactly standard normal then we could calculate confidence intervals on individual execution times. In the next subsection we will see that the comparison of the new observations from the fourth phase of the experiment with the confidence intervals based on (5.3b) further supports the conclusion that (5.3b) is approximately correct.

Symmetry about zero of ε' permits us to estimate the median execution time T_{med} by

$$\hat{T}_{med} = e^{b_0}|V|^{b_1}\Lambda^{b_2} \tag{5.7a}$$

for SCALE, and

$$\hat{T}_{med} = e^{b_0}|V|^{b_1} \tag{5.7b}$$

for GNET and RNET.

The fitted curves in Figure 0.1 are from these estimates of T_{med}. Under the stronger normality assumption on ε' one can estimate mean execution time T_{mean} by

$$\hat{T}_{mean} = e^{\frac{1}{2}s^2}\hat{T}_{med}. \tag{5.8}$$

For our nine samples, (5.8) never deviates from (5.7) by more than 1 or 2 percent for SCALE and GNET, nor by more than 6 or 7 percent for RNET.

Prediction

In order to examine the usefulness for prediction of the models of execution time, we ran new sets of one hundred problems for each of the three generators. Figures 0.2 (and B.2 of [1]) give the plots for the three codes of execution time in seconds versus the number of vertices, $|V|$. The plotted curve represents predicted median execution time based on the parameter estimates from the regression on the earlier data set. The dots indicate the observations from the new independent sample of one hundred problems. For SCALE we have partitioned the observations according to the value of $\Lambda \in \{4,5,6,7,8,9,10\}$ and plotted each separately. In each case the predictions of T_{med} are very good.

Based on the normality assumption (5.3b), we can construct confidence intervals for individual execution times. We examined the relationship of the observations plotted in Figures 0.2 (and B.2 of [1]) to the 95% confidence intervals. Note that the observations here are independent of the observations on which the calculation of the confidence intervals is based. In each of the nine cases at least 91 out of 100 observations are within the calculated 95% confidence intervals. In total, 860 of 900 observations, 95.6%, fall within the intervals. Figure 5.2 depicts the same kind of information as Figure 0.2, except here the plotted observations are those generated for lack-of-fit tests of RNET and SCALE, and the upper 97.5th and lower 2.5th percentiles are also plotted. These plots are especially interesting because of the replication of the independent variables, and, again, because they are independent of the observations from which the predictions were calculated. The normality assumption (5.3b) on ε' in (5.4) and (5.5) implies that the standard deviation of T should be proportional to T_{med}. In this respect, as well as the proportion of observations in the 95% confidence intervals, Figure 5.2 appears to be consistent with assumption (5.3b).

Assumption (5.3b) implies that for a fixed code the inherent variability in execution times for problems of fixed size is lognormal. In further work we expect to generate more data for the purpose of examining

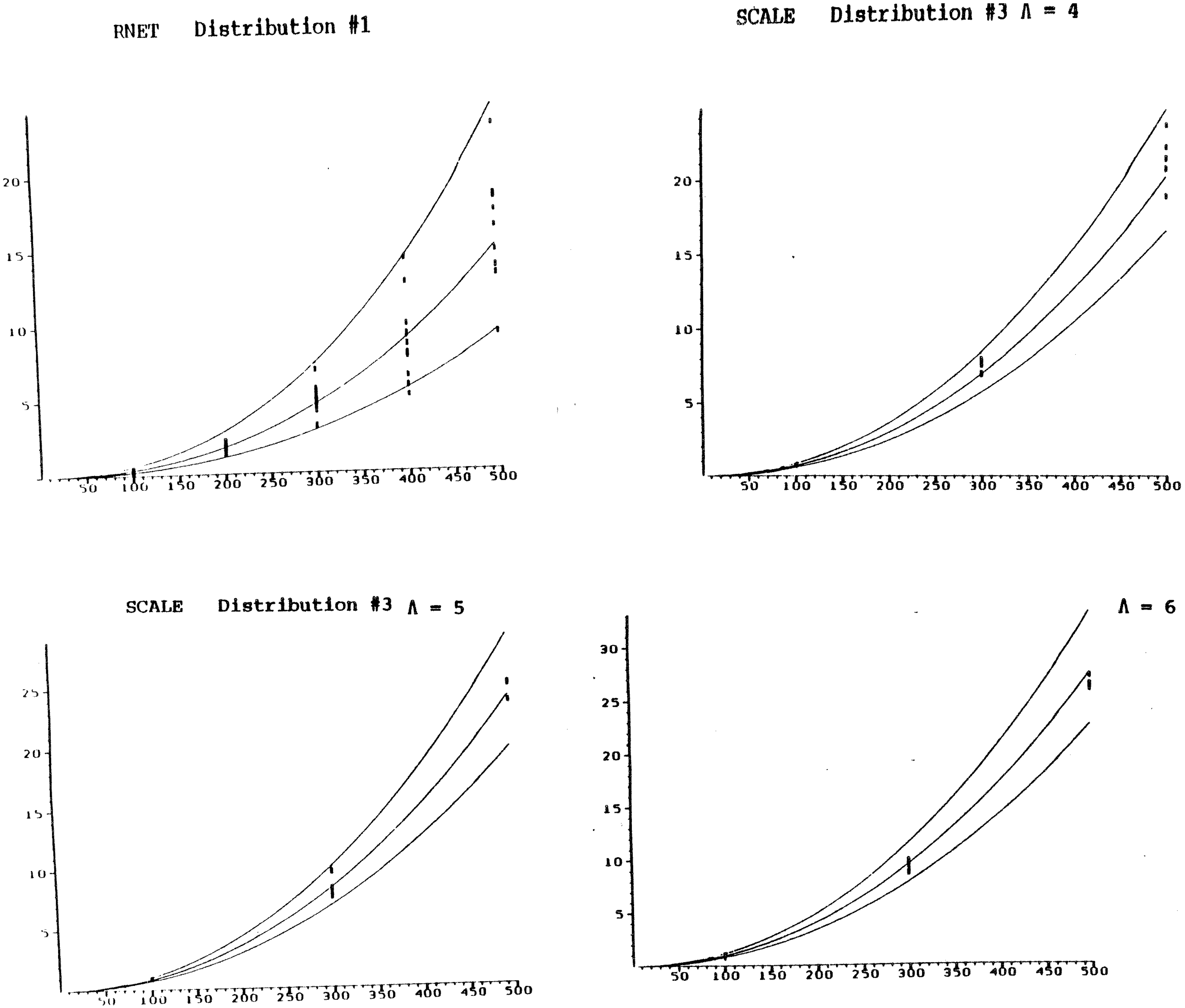


Figure 5.2 95% confidence intervals for individual execution times and observations from the lack-of-fit tests.

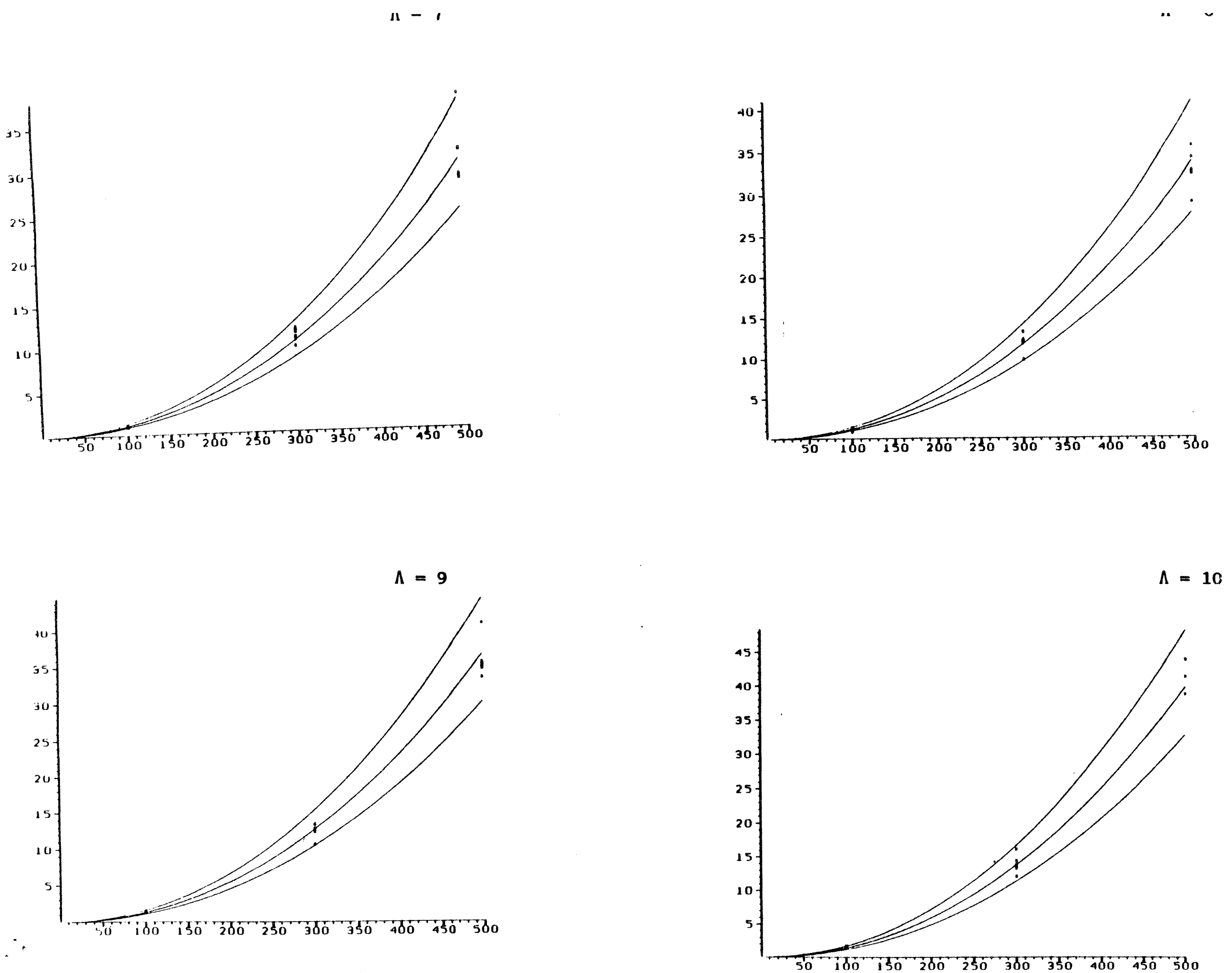


Figure 5.2 (continued)

with greater care the accuracy of the lognormal approximation. It would also be interesting to learn whether this type of behavior is exhibited in other mathematical programming settings.

6. Fully Dense NETGEN Problems

In addition to the set of benchmark NETGEN problems noted earlier, seventy-five fully dense NETGEN problems were run. Three different sets of data containing twenty-five problems each were collected. The problems in the first of the three data sets are fully dense capacitated transportation problems. These problems are similar to those generated by CAPT in terms of the number of vertices, total flow through the network, and edge cost and capacities. The second data set has parameters drawn from the same ranges, but now the generated network can contain transshipment vertices. Each of these problems has between 75 and 350 vertices. The third data set, like the second, has transshipment vertices. The problems in this data set, however, are all large problems having between 350 and 425 vertices.

Table 6.1

Estimates of parameters in models (5.4) and (5.5) for three sets of NETGEN problems

	b_0	b_1	b_2
Set #1			
SCALE	-10.459	2.176	0.483
RNET	-14.116	2.935	
GNET	-13.444	2.912	
Set #2			
SCALE	-11.125	2.418	0.224
RNET	-15.602	3.279	
GNET	-13.670	2.979	
Set #3			
SCALE	-19.941	3.648	0.965
RNET	-19.428	3.865	
GNET	-15.970	3.370	

The models were not significant for Set #3.

We ran regressions to fit the earlier models to this data. The resulting parameter estimates are indicated in Table 6.1, and summaries of the regressions are given in Table B.4 in Appendix B of [1]. Since these samples are smaller than those from the CAPT generators, we can expect that the regressions will not fit as well, and, that the parameter estimates will be less accurate, even if the fit is satisfactory. Indeed we found that for the third of these new NETGEN data sets the fit was too poor to make meaningful use of the model. For each of the other two new NETGEN data sets the fit and the precision, though not as good as for the larger CAPT data sets, were sufficiently good to use the resulting models of median execution time. In particular it is interesting to make some comparisons with the CAPT distributions.

Consider the first of the NETGEN data sets. Like the CAPT generated problems discussed earlier, these are dense capacitated transportation problems with an equal number of sources and sinks, between one hundred and five hundred vertices, and like distributions #1 and #2, approximately fifty units of flow per edge, on average. The costs are determined as in the CAPT generators, except high costs (and high capacities) are assigned to all the edges in a single spanning tree. The selection of capacities here differs from CAPT, but the range from which they are drawn is the same, except on the edges of the special spanning tree. The special spanning tree constitutes between 0.80% and 3.96% of the edges, 1.61% on average. Although this distribution of problems resembles distributions #1-3 closely, the computational behavior of the algorithms changes noticeably. First observe that here the estimate $b_2 = 0.48$ for SCALE is much smaller than for any of the three CAPT distributions, where the minimum was 0.737. The estimate here of $b_1 = 2.94$ for RNET is much larger than previously, where the maximum was 2.450. These differences are well beyond a few standard errors. Most striking is that for the NETGEN problems in this set with more than 400 vertices, SCALE was faster than RNET. The crossover point in the estimates of median execution time in Figure 6.1 for $\Lambda = 4$ is at approximately 275 vertices (18,900 edges), where the predicted median times for both RNET and SCALE are approximately 11 seconds, and for $\Lambda = 10$ at approximately 487 vertices (59,300 edges), where the predicted times are approximately 59 seconds.

In the second and third new sets of twenty-five NETGEN problems, the underlying graph is nonbipartite, though still dense. For the second set b_2 is even smaller for SCALE than for the bipartite problems; b_1 is noticeably larger for SCALE and for RNET. As noted earlier the fit of the regression on the third set was not adequate.

Table 6.2 below gives the average execution times of the three codes on the three sets of twenty-five dense NETGEN problems. It appears from these data that SCALE's very poor performance relative to GNET and RNET on the benchmark NETGEN problems is attributable, in part, to the sparsity of those problems.

Table 6.2

Average execution times (seconds) on dense NETGEN problems

	RNET	GNET	SCALE
Set 1	23.985	39.830	24.491
Set 2	13.445	16.268	13.624
Set 3	44.608	63.070	41.333

Each set has 25 fully dense capacitated problems. Set 1 consists of transportation problems. Set 2 consists of nonbipartite problems with 2,500 - 60,000 edges, and Set 3 consists of nonbipartite problems with 60,000 - 90,000 edges.

7. Relationship to Other Recent Results

Very recently several reports [9,10,11,14,15,16,23,24,26,29] on the use of data scaling in network algorithms have appeared. Several of these lend credence to our conclusion that scaling is potentially useful in computation.

Edmonds and Karp [4] raised the issue of whether there is an algorithm for minimum cost flows for which the number of arithmetic operations is bounded above by a polynomial function of $|V|$ and $|E|$, independent of the logarithms of costs, capacities,and demands. Some people refer to an algorithm with this property as "strongly polynomial" or "genuinely polynomial," although one must be careful not to attach significance to those terms at the level of Turing machine computation. Éva Tardos [29] has recently given the first strongly polynomial algorithm for the minimum cost network flow problem. Her algorithm uses cost scaling to create a sequence of problems $\{P(a^i)\}$, each of which is solved by an out-of-kilter routine. Tardos manages to remove any dependence of running-time on $\log(|a(e)|)$ by a very clever, and beautifully elementary, scheme for choosing the a^i. After each iteration, i.e. solution of one $P(a^i)$, certain variables are fixed in value by invoking a result like Lemma 4.1; these variables are deleted to reduce the original problem P to P_R. The objective function that approximates a_R in the next iteration is obtained by first projecting into the circuit space (the null space of the vertex-edge matrix) of the reduced graph, then scaling the projection (if it has entries of large magnitude) so that the largest magnitude of any entry is now approximately $|V||E|^{1/2}$, and then rounding up to integers. The projection and scaling have the combined effect of insuring that one, or more, variables can be fixed and deleted after each iteration. Since scaling keeps the magnitudes of the costs on the order of $|V||E|^{1/2}$, the out-of-kilter method, as implemented in SCALE, for example, will require no more than $O(|V|^4 \log(|V||E|^{1/2}))$ computations. The total number of iterations can be at most $|E|$, so the total running time is no worse than $O(|E||V|^4 \log(|V||E|^{1/2}))$.

In actual computation this worst-case analysis does not look encouraging. In a large "real-world" problem one might expect that Λ would be very small compared to $|E| \log(|V||E|^{1/2})$, which would make direct application of SCALE, or a similar algorithm, preferable from the point-of-view of worst-case analysis. (Indeed for big "real-world" problems Λ might be expected to be small compared with $\log(|V||E|^{1/2})$. Under this restriction direct application of the out-of-kilter method is already polynomial-time.) Galil and Tardos[11] have now given a variation on Tardos's original approach [29] that yields substantial improvement in the worst-case bound. It exploits ideas of Fujishige [7], which avoid the projections of [29], and scales right-hand-sides, rather than costs, as in Edmonds-Karp [4]. Consequently, the computation involves repeated solution of shortest path problems (with nonnegative edge lengths). The running time is $O(|V|^2\log(|V|)(S(|E|,|V|)))$, where $S(|E|,|V|)$ denotes the running time of the subroutine used for the single source shortest path subproblems. If the Fredman-Tarjan [6] algorithm is used as the subroutine, then the overall bound on the running time is $O(|V|^2\log(|V|)(|E|+|V|\log(|V|))$. This is better than the bound on SCALE, or any other implementation of MCNF2 using a known maximum flow subroutine, for all ranges of density of the graph, so long as $O(\Lambda)$ is larger than $O(\log(|V|))$. On problem domains such as those investigated in the main experiment here, the bound for SCALE is $O(|V|^4\Lambda)$ and the bound for the Galil-Tardos algorithm is $O(|V|^4\log(|V|))$; furthermore, Λ and $\log(|V|)$ have approximately the same (small) range of values. We have seen that SCALE's observed running times within these distributions are much faster than the worst-case bound predicts, they are close to the square root of the bound. It would be interesting to see how the Galil-Tardos algorithm performs on these, and other, problems.

A reference in Tardos [29] brought to our attention a 1980 paper by Hans Röck [26]. Röck also exposits the variation on the Edmonds-Karp scaling algorithm that scales costs rather than right-hand-sides, and his suggested implementation also has an $O(|V|^4\Lambda)$ bound on running time. The direct approach of doing what we have called the simple perturbations one-at-a-time gives a bound of $O(|E||V|^3\Lambda)$. Within each of the Λ iterations Röck aggregates these simple edge perturbations into blocks of edges with a common tail. He then perturbs the entire block at once and solves the perturbed problem by a maximum flow computation. This gives a bound on the work per iteration of the same order as for our approach. However, for the reasons given at the end of Section 4, it appears to be computationally attractive to aggregate all of the simple perturbations at each iteration. Indeed, very early in the development of SCALE, we tried on several problems the same aggregation that Röck describes. On those problems it was slower than aggregating everything.

Subsequent to Tardos's discovery of the first strongly polynomial minimum cost flow algorithm, Jim Orlin [24] gave others, using right-hand-side scaling. An especially interesting feature of Orlin's approach is that it produces strongly polynomial algorithms that are specializations of the dual network simplex method. A little earlier he had given a dual network simplex algorithm that was polynomial, but not strongly [23].

Ikura and Nemhauser [16] have reported on computational experiments with a polynomial (not strongly) algorithm for the transportation problem. Their algorithm, originally presented in [15], uses right-hand-side scaling, and solves the subproblems by a dual-simplex method. They report substantial savings in execution time from the use of scaling within their algorithm. They also compare execution times of their algorithm and NETFLO, a primal network simplex code developed by Kennington and Helgason (see [18]), on 60 transportation problems with 5000-24000 edges. The test problems were generated by NETGEN. Ikura and Nemhauser report that their dual algorithm is competitive with NETFLO on small problems but 30-60% slower on large ones.

Hung and Murphy [14] have conducted computational tests on uncapacitated transportation problems of an implementation of the Edmonds-Karp (right-hand-side) scaling algorithm. They report that it ran two to eight times slower than GNET.

The broadest investigation yet of data scaling for network algorithms (not only for network flows) is in the paper [9] by Harold Gabow. Gabow examines a variety of problems, including maximum flow and minimum cost flow with all capacities equal to one. He shows that when N, the largest magnitude in the data, is of the same order as $|V|$, then data scaling enables one to improve on the asymptotic worst-case behavior of the best known algorithms for: the assignment problem; single-source shortest paths in networks with no negative length directed cycle; and minimum cost flow with unit capacities. In [10] Gabow reported on computational comparisons of his scaling algorithm for the assignment problem and the Hungarian (primal-dual) algorithm. The observed behavior was consistent with the improvement in the worst-case bound achieved by scaling.

Gabow's suggestion of using capacity scaling to solve maximum flow problems is a most intriguing one. He presents a simple algorithm with a bound $O(|V||E| \log(|V|))$. This approach would be most natural within the subroutine MAXFLOW of our minimum cost network flow algorithm. The use of scaling at that more fundamental level of the computation has a natural esthetic appeal. Moreover, his approach might be able to take advantage of the sparsity of the maximum flow subproblems that arise in the later iterations of our algorithm, without imposing data structures with too much overhead for a problem with small $|V|$, but large $|E|$.

8. Conclusions

The principal conclusion of this study is that scaling algorithms are worthy of further consideration as computational tools for the solution of network flow problems. The presumption that scaling-based algorithms would be non-competitive, even within restricted classes of flow problems, was ill-founded.

To be sure, we have not shown that SCALE is computationally faster than RNET on any broad class of problems. (Although this appears to be the case for the class of large dense NETGEN problems discussed in Section 6.) Moreover, SCALE's memory requirements could be problematic. However, its relative performance so far surpasses expectations, that it seems to argue for further study of other scaling-based algorithms.

There are two additional conclusions that emerged from this study - one concerning the predictability of computational behavior within the problem distributions studied, and another concerning unpredictability across problem distributions. Although these are secondary to the original purpose of the study, they may have broader implications for computational mathematical programming than the main conclusion concerning the potential efficacy of data scaling.

APPENDIX

Problem Generators

Two pseudo-random minimum cost network flow problem generators were used in conducting the tests reported on here. One generator, NETGEN [19], has been used widely to benchmark codes. This generator has several parameters which determine the structure of the problems generated. The user specifies the following.

- the number of supply, demand, transshipment supply, transshipment demand, and pure transshipment vertices in the problem. These numbers can be determined from the input parameters srcs, sink, tsrcs, tsnk and nodes.
- the number of edges (approximately) in the generated graph. This number is input as the value of the parameter arcs.
- the amount of flow to be sent from supply to demand vertices. This number is input as the value of the parameter supply.
- a range on the upper bounds. These numbers are input as values of the parameters max and min.
- a percentage on the number of edges having the maximum upper bound. This number is input as the value of the parameter pcap.

NETGEN then generates a feasible problem by, essentially, creating a tree that will be a feasible basis. The edges in this "skeleton" tree are capacitated in such a way that the specified amount of flow may be sent from the supplies to the demands using only edges in the tree. (Note that this implies that there are feasible solutions to these problems in which a small percentage of the edges in the graph are at their upper bounds.) A user-specified percentage of the tree edges, pcst, are set to the highest cost - presumably so that the initial tree can be made to resemble an artificial basis. The second phase of the procedure adds edges so that the total number of edges is approximately that specified in the input parameter arcs.

In addition to NETGEN, a capacitated transportation problem generator called CAPT was used to generate problems. Like NETGEN, CAPT first generates a preliminary flow, which will be feasible in the problem being generated. Unlike NETGEN, this preliminary solution is not basic; it is likely to be an interior point of the generated network flow polyhedron. Three different methods are used to generate this feasible flow, each is based on a symmetry condition. In order to discuss the methods used we introduce the following notation.

Let S and D denote the set of supply and demand vertices in the generated graph, $G = (V,E)$, so that $V = S \cup D \cup \{s,s'\}$ and $E = \{(v,v'): v \in S , v' \in D\} \cup \{(s,v): v \in S\} \cup \{(v',s'): v' \in D\} \cup \{(s',s)\}$. Edges of the form (v,v') will be called <u>transportation edges</u>, edges of the form (s,v) will be called <u>supply edges</u>; edges of the form (v',s') will be called <u>demand edges</u>.

One way to generate a preliminary feasible flow is to choose a point uniformly in the unit simplex having a coordinate for each transportation edge. The coordinates of the point then give the proportion of the total flow which is to be allocated to this edge in the preliminary feasible flow. The sum of the preliminary flow values corresponding to the transportation edges incident with a particular supply vertex gives the value of the preliminary flow on the corresponding supply edge. The preliminary flows on demand edges are computed in a similar manner. Problem distributions based on this approach will be called <u>edge symmetric</u>. Distribution #1 in the main experiment is edge symmetric.

Once the preliminary flow is fixed, the capacitated transportation problem is determined by perturbing the values of the preliminary flow to obtain upper bounds for the supply and transportation edges, and lower bounds for the demand edges. The method used here is to add or subtract an amount of flow chosen uniformly from a user-specified interval. This is referred to as an <u>additive perturbation</u>. The generator also has the capability to use instead a multiplicative perturbation on the supply and demand edges. Multiplicative perturbations were only used in the first preliminary phase of this experiment.

Problem generation is controlled by several parameters. The numbers of sources and sinks are specified and will be referred to as <u>NSRC</u> and <u>NSNK</u>, respectively. In the experiments reported on here, NSRC and NSNK have been set equal. The <u>average</u> total amount of flow through each source in the preliminary flow is chosen uniformly from 1 to <u>FLOW</u>. Another input parameter, <u>INT</u>, is used to specify the upper bound on the magnitude of the additive perturbations to the preliminary flow. The fraction of INT which serves as an upper bound on the perturbation of transportation edges is given by FRAC. Individual perturbations are chosen uniformly from 0 to the upper bound. Finally, costs are chosen uniformly from the integers between 0 and $2^{BIT}-1$. The parameter Λ in the models of execution time is shorthand for BIT. Table A.1 gives the parameter ranges used in distribution #1.

Table A.1

CAPT Parameter Ranges for Distribution #1

PARAMETER	RANGE
NSRC = NSNK	[50,250]
FLOW	100·[1,NSRC]
INT	[1,FLOW]·0.05
FRAC	[1,100]/(100·NSRC·0.05)
BIT	[4,10]

[p,q] denotes an integer chosen uniformly between p and q.

As a consequence of these choices, the expected flow through the network is just over 49 $|E|$. The expected excesses of total supply over total demand, and total capacity over total demand, are a little more than 2.5%, and 14%, respectively. This can be expected to result in rather tightly constrained problems from CAPT, while NETGEN tends to produce very loosely constrained problems.

Our preliminary tests indicated that RNET might be adversely sensitive to loosening constraints in CAPT generated problems, particularly through increase of the CAPT parameter FRAC. This phenomenon might be related to RNET's worse performance on the dense NETGEN problems than on the CAPT problems.

The selection of an appropriate range of values for BIT (= Λ) was given careful consideration. Computational experiments concerning network flows often choose costs uniformly on the integers between 0 and 100, which corresponds, approximately, to $\Lambda = 7$. By selecting Λ uniformly in $[4,10] \cap Z$, we have $\Lambda = 7$ on average, but we also get to put SCALE through its paces in problems with Λ as high as ten and expected average cost in excess of 500, and we get to evaluate how SCALE execution times vary with Λ. One can easily imagine interesting problems where some of the edge costs exceed 2^{10}. From a practical standpoint it is not really max $\log(|a(e)|)$ that determines Λ, it is the number of significant bits in the vector a, since a could be rescaled if the number of significant bits were less than max log(|a(e)|). One can still imagine interesting and practically important problems with more than ten significant bits, perhaps problems with |E| extremely large and with large ranges of values on the capacities. However, since we are focusing on a restricted problem distribution anyway, $\Lambda \in [4,10]$ seemed reasonable.

Distributions #2 and #3 are generated in a manner similar to distribution #1, with a different mechanism for constructing the preliminary interior solution. One method is to first select the flow on a particular supply edge uniformly from some prespecified range. This flow is then distributed among the demand vertices by choosing a point uniformly in the unit simplex having a coordinate for each demand vertex. The proportion of the supply going to each sink is then given by the value of the corresponding coordinate. The flow on a demand edge is the sum of the flows on the transportation edges incident with the corresponding demand vertex. Unlike the first method, the flows generated by this procedure are not symmetric with respect to supplies and demands. That is, the joint distribution of the flows on the supply edges is different from the joint distribution of the flows on the demand edges. Problem distributions based on this approach will be called the supply symmetric. Distribution #3 in the main experiment is supply symmetric. The parameter ranges used are as in Table A.1, except that the preliminary flow on each supply edge is selected uniformly in the range 50·[1, NSRC].

A related method symmetrizes the flows by first using the supply symmetric scheme, then using the analogous demand symmetric scheme. The preliminary flow is the sum of these two flows. Problem distributions based on this approach will be called vertex symmetric. Distribution #2 in the main experiment is vertex symmetric.

Acknowledgment

We would like to thank Bob Bechhofer, Tom Santner, and Lionel Weiss for their valuable advice concerning the statistical analysis, and David Heath for the use of his program GRAPH for the IBM PC.

This work was first reported as part of the second author's Ph.D. dissertation [17].

Footnotes

[1] GNET, copyright 1975, G.H. Bradley, G.G. Brown, and G.W. Graves, see [2]. It should be noted that the GNET code was designed to exploit special structure. Its particular construction of candidate queues within the pricing step is well-suited to transportation problems with few sources, but not to transportation problems with a large number of sources. It is likely that the running times for GNET on the test problems in our main experiment could be improved somewhat by a simple alteration of the pricing routine. However it is unlikely that such a simple alteration would curb the observed rate of growth of execution times with problem size.

[2] RNET, version no. 3.61, copyright 1977, M.D. Grigoriadis and T. Hsu, see [13]).

REFERENCES

1. R.G. Bland and D.L. Jensen, "A report on the computational behavior of a polynomial-time network flow algorithm", Cornell Univ. School of OR/IE Tech. Report No. 661 (1985).

2. G.H. Bradley, G.G. Brown and G.W. Graves, "Design and implementation of large scale primal transshipment algorithms", *Management Science* 24,1 (1977) 1-28.

3. N.R. Draper and H. Smith, *Applied regression analysis*, second edition, (Wiley, New York, 1981).

4. J. Edmonds and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems", *J. ACM* 19,2 (1972) 248-264.

5. L.R. Ford, Jr. and D.R. Fulkerson, *Flows in networks* (Princeton University Press, Princeton, N.J. 1962).

6. M.L. Fredman and R.E. Tarjan, "Fibonacci heaps and their uses",in Proc. 25th Symposium on the Foundations of Computer Science (1984) 338-346.

7. S. Fujishige, "An $O(m^3 \log n)$ capacity-rounding algorithm for the minimum-cost circulation problem: a dual framework for the Tardos algorithm", Report no. 253, University of Tsukuba, Japan (1983).

8. D.R. Fulkerson, "An out-of-kilter algorithm for minimal cost flow problems", *J. SIAM* 9,1 (1961) 18-27.

9. H.N. Gabow, "Scaling algorithms for network problems", *J. Computer and Systems Science* 31,2 (1985) 148-168.

10. H.N. Gabow, "On the theoretic and practical efficiency of scaling algorithms for network problems", presented at the Fall 1984 ORSA/TIMS meeting, Dallas.

11. Z. Galil and É. Tardos, "An $O(n^2 \log n(m+n\log n))$ minimum cost flow algorithm", MSRI report no. 04518-86 (1986).

12. F. Glover, D. Karney, and D. Klingman, "Implementation and computational comparisons of primal, dual and primal-dual computer codes for minimum cost network flow problems", *Networks* 4 (1974) 191-212.

13. M.D. Grigoriadis and T. Hsu, "RNET - The Rutgers minimum-cost network flow subroutines", *SIGMAP Bulletin of the ACM* 26 (1979) 17-18; see also M.D. Grigoriadis, "An efficient implementation of the network simplex method", *Math. Programming Study 26*, eds: G. Gallo and C. Sandi (1986) 83-111.

14. M.S. Hung and C. Murphy, "Implementation of Edmonds and Karp's scaling algorithm", presented at the Spring 1984 ORSA/TIMS meeting, San Francisco.

15. Y. Ikura and G.L. Nemhauser, "A polynomial-time dual simplex algorithm for the transportation problem", SORIE Technical Report No. 602, Cornell University (1983).

16. Y. Ikura and G.L. Nemhauser, "Computational experience with a polynomial-time dual simplex algorithm for the transportation problem", SORIE Technical Report No. 653, Cornell University (1985).

17. D.L. Jensen, "Coloring and duality: combinatorial augmentation methods", Dissertation, Cornell University (Ithaca, New York 1985).

18. J.L. Kennington and R.V. Helgason, *Algorithms for network programming* (Wiley and Sons, New York, 1980).

19. D. Klingman, A. Napier, and J. Stutz, "NETGEN: a program for generating large-scale assignment, transportation, and minimum cost network flow problems", *Management Science* 20,5 (1974) 814-821.

20. E.L. Lawler, *Combinatorial optimization: networks and matroids*, (Holt-Rinehart-Winston, New York 1976).

21. V.M. Malhotra, M. Kumar, and S.N. Maheshwari, "An $O(|V^3|)$ algorithm for finding maximum flows in networks", *Information Processing Letters*, 7,6 (1978) 277-278.

22. G.J. Minty, "Monotone networks", *Proc. Roy. Soc. London, Ser. A* 257(1960) 194-212.

23. J.B. Orlin, "On the simplex algorithm for networks and generalized networks", to appear in *Mathematical Programming*.

24. J.B. Orlin, "Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem", Sloan Working Paper No. 1615-84, M.I.T. (1984).

25. C.H. Papadimitriou and K. Stieglitz, *Combinatorial optimization: algorithms and complexity* (Prentice-Hall, Englewood Cliffs, New Jersey, 1982).

26. H. Röck, "Scaling techniques for minimal cost network flows", in: V. Page, ed., *Discrete structures and algorithms* (Carl Hansen, Munich, 1980).

27. D. Sleator, "An O(n m log n) algorithm for maximum network flow", Dissertation., Tech. Report STAN-CS-80-831, Stanford University (1980).

28. D. Sleator and R.E. Tarjan, "A data structure for dynamic trees", *J. Computer and System Sciences*.

29. É. Tardos, "A strongly polynomial minimum cost circulation algorithm", to appear in *Combinatorica*.

30. R.E. Tarjan, "A simple version of Karzanov's blocking flow algorithm", *O.R. Letters* 2,6 (1984) 265-268.