CENTER FOR
PARALLEL OPTIMIZATION

THE EFFICIENT PARALLEL SOLUTION OF GENERALIZED NETWORK FLOW PROBLEMS

by

Robert Hartley Clark

Computer Sciences Technical Report #933

May 1990

ACKNOWLEDGEMENTS

My sincere thanks go to my thesis advisor Robert Meyer for giving me support and criticism during the past three years and during the preparation of this thesis.

I would also like to thank Olvi Mangasarian and Michael Ferris for reading drafts of the thesis, and I would like to thank Stephen Robinson and Arne Thesen for being members of my examination committee.

This research was supported in part by The National Science Foundation under grant CCR-8709952 and by The Air Force Office of Scientific Research under grants AFOSR-86-0194 and AFOSR-89-0410.

THE EFFICIENT PARALLEL SOLUTION OF GENERALIZED NETWORK FLOW PROBLEMS

Robert Hartley Clark

Under the supervision of Professor Robert R. Meyer

ABSTRACT

This thesis is principally concerned with the discussion of two efficient parallel codes for the generalized network flow problem. Both of the codes are variants of the primal simplex method for generalized networks, and both were implemented by the author on a shared memory multiprocessor, the Sequent Symmetry S81. PGRNET, the first code, exploits parallelism in the pivoting and the pricing operation. Parallel pivoting is made possible by the disjoint nature of the basis graphs. Two processors may execute pivots simultaneously if the pivots involve updating different basis components (quasi-trees), and locking is used to ensure that processors have exclusive access to quasi-trees during the execution of a pivot. The second code, TPGRNET, exploits parallelism in the pricing operation and overlaps pricing and pivoting. Pivots are executed serially by a host processor, pivot arcs are selected from a collection of shared candidate lists by a selecting processor, and all remaining processors compute reduced costs in parallel and store pivot-eligible arcs in the shared candidate lists. TPGRNET is more efficient than PGRNET for problems having only a few quasi-trees in an optimal basis. Computational results for PGRNET for transportation problems with 30,000 nodes and over 320,000 arcs are given, and speedups over GRNET2, a state-of-the-art serial program range from 3.8 to 11.1 on 19 processors. Results for TPGRNET are given for large scale transportation and transshipment problems, and speedups over GRNET2 range from 2.6 to 8.8. A hybrid algorithm that can invoke PGRNET or TPGRNET is briefly described, and results are given, with speedups ranging from 6.1 to 13.2. Two test problems with more than a million variables were solved by PGRNET with speedups ranging from 7.0 to 11.0.

This thesis develops a technique for testing the "parallelizability" of a problem instance to determine a lower bound for the number of disjoint quasi-trees in any optimal basis of the problem. A technique is also given that can be used to generate a problem instance that is guaranteed to have a certain minimum number of quasi-trees in any optimal basis.

CONTENTS

I Introduction

1.1 Generalized Networks

The generalized network model can be used to optimize network problems found in the areas of investment planning, job scheduling, pure network optimization and others. The applications are characterized by networks for which each arc gains or loses flow at a fixed rate assigned to that arc. Profit from interest or dividends can be modeled by a network with gains, and loss from evaporation or seepage can be modeled by a network with losses. A generalized network without gains or losses is a pure network. Further discussion of applications can be found in [Glover, et al, 78] and [Mulvey and Zenios 85]. Serial and parallel algorithms for the convex generalized network flow problem are discussed in [Zenios 1986]. The linear generalized network flow problem can be formulated as follows:

$$\begin{array}{c} \min \quad cx \\ \text{s.t.} \quad Gx = b \\ 0 \le x \le u \end{array} \qquad \text{(GN)}$$

where the matrix $G \in R^{m \times n}$ is such that each column has no more than two nonzero-elements.

In order to simplify algorithms for GN, the variables can be scaled and the corresponding column can be multiplied by -1 (i.e. reflected) as needed so that columns of G with two non-zero elements are such that one of the elements is 1, and the other non-zero entry is then interpreted as a *flow multiplier.* If the

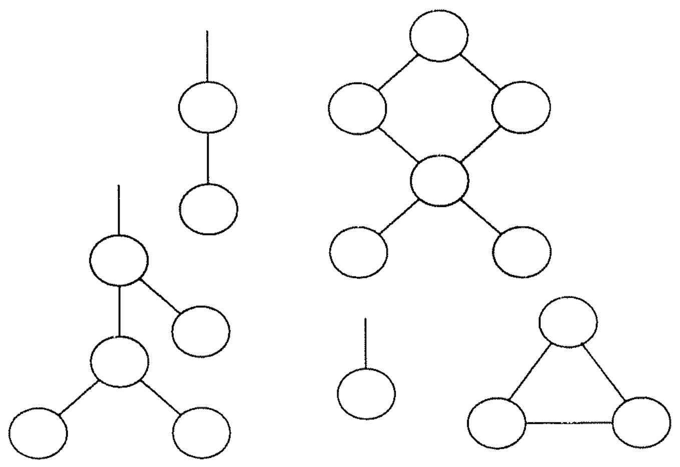

Figure 1.1 A Forest of Quasi-Trees

flow multiplier is -1, then the arc is a pure network arc. In general, the flow multiplier can be positive or negative and can have magnitude equal to 1, less than 1, or greater than 1. If the flow multiplier for a column is in the range $(-1, 0)$, the column corresponds to an arc that loses flow. If the flow multiplier is in the range $(-\infty, -1)$ then the column corresponds to an arc that gains flow. A column that has just one nonzero element corresponds to a *root-arc*, an arc that is incident to just one node. A specialization of the primal simplex algorithm for GN is described in [Jensen and Barnes 80] and [Kennington and Helgason 80]. As with the pure network flow problem, which we designate as PN, the simplex algorithm for GN can be executed on a graph. One difference between PN and GN is that the graph of any basis for PN consists of one rooted tree, while the graph of a basis for GN is a forest of quasi-trees, where a quasi-tree is a tree with exactly one additional arc (making it either a rooted tree or a tree with exactly one cycle). Figure 1.1 shows a forest of quasi-trees. Note in particular that any connected component of a graph corresponding to a basis cannot contain more than 1 root arc, so that the number of roots in the basis yields a lower bound on the number of quasi-trees in that basis. This observation is significant with respect to parallel algorithms, since individual quasi-trees may be updated independently.

1.2 A History of Codes

The following is a summary of the history of generalized network derived from a paper of J. Kennington and R. Muthukrishnan. The summary appears also in [Clark, et al, 89].

Glover, Klingman, and Stutz [Glover, et al, 73] developed the first specialized primal simplex code (NETG) which exploited this graphical structure. Many theoretical and computational improvements have been made to this system over the last fifteen years (see [Glover, et al, 78]) and [Elam, et al, 79]). A similar implementation was also developed in [Langley 73]. [Adolphson and Heum 81] presented computational results with their generalized code which used an extension of the threaded index method of [Glover, et al 74]. Brown and McBride presented the details of their generalized network code GENNET in [Brown and McBride 84]. In [Tomlin 84], Tomlin developed the first assembly language code which is part of Ketron's MPS III system. In [Engquist and Chang 85] data structures for the solution of GN are discussed that are primarily based on those of [Adolphson 82] and [Barr, Glover and Klingman 79]. These data structures were used to implement GRNET [Engquist and Chang 85], a serial version of the primal simplex algorithm for GN. [Engquist and Chang 85] establishes that on a CYBER 170/750, GRNET is about 50 times faster than

Code	Language	Authors	Year
NETG	FORTRAN	Glover, F., Klingman, D. Stutz, J.	1973
	FORTRAN	Langley, W.	1973
	FORTRAN	Adolphson, D., Heum, L.	1981
GENNET	FORTRAN	Brown, G., McBride, R.	1984
GWHIZNET	ASSEMBLER	Tomlin, J.	1984
GRNET	FORTRAN	Engquist, M., Chang, M.	1985
LPNETG	FORTRAN	Mulvey, J., Zenios, S.	1985
	FORTRAN	Ali, I., Charnes, A. Song, T.	1986
GRNET-K	FORTRAN	Chang, M., Engquist, M. Finkel, R., Meyer, R.	1987
PGRNET	FORTRAN	Clark, R., Meyer, R. Chang, M.	1987
GNO/PC	C	Nulty, W., Trick, M.	1988
GRNET-A	ASSEMBLER	Chang, M., Chen, M. Chen, C.	1988
GENFLO	FORTRAN	Muthukrishnan, R.	1988
GRNET2	FORTRAN	Clark, R., Meyer, R. Chang, M.	1989
TPGRNET	FORTRAN	Clark, R., Meyer, R. Chang, M.	1989

Table 1.1 Survey of Generalized Network Codes

MINOS [Murtagh and Saunders 78], a standard LP code, on problems of the form GN. Mulvey and Zenios, and Ali, Charnes and Song have developed implementations of the generalized network simplex algorithm discussed in [Mulvey and Zenios 85] and [Ali, et al, 86]. The first C language code is discussed in [Nulty and Trick 88]. Another assembly language code is discussed in [Chang, et al, 88]. The serial codes GENFLO and GRNET2 are discussed in [Muthukrishnan 88] and [Clark and Meyer 89] respectively. GENFLO is a modification of GENNET that permits up to two arbitrary non-zero entries in the system matrix. (GENNET requires that scaling be done to make one entry be equal to one). Computational results for GENFLO and GRNET2 will be given in Chapter IV. The first parallel generalized code, GRNET-K, was developed by Chang, Enquist, Finkel, and Meyer (see [Chang, et al, 87]) for the Wisconsin CRYSTAL Multicomputer, and the second (see [Clark and Meyer 87]) for the Sequent 21000, also at the University of Wisconsin. TPGRNET, a parallel algorithm that assigns distinct tasks to different processes is discussed in [Clark and Meyer 89] and results for this code are given in Chapter IV. A summary of the available software may be found in Table 1.1.

1.3 Overview of Dissertation

The primary goal of this dissertation is to present efficient parallel algorithms for GN and empirical results. Two algorithms illustrative of different approaches to parallelism are discussed in Chapter III and computational experience is given in Chapter IV. Another major goal of this work is to identify problems that can be solved easily by both serial and parallel methods. Having a clear understanding of what makes some instances of GN "easy" to solve might eventually lead to efficient algorithms for "difficult" instances, or it might motivate new methods for problem formulation that induce optimal behavior from existing algorithms. This broad goal is far from reached in this dissertation, but Chapter II contains some theoretical results that can help to classify problem instances as "easy" or "difficult." Computational experience has shown that problems having many quasi-trees in the optimal basis tend to be "easy" to solve. Lemma 1 gives a lower bound to the number of quasi-trees in any optimal basis of a bipartite or non-bipartite instance of GN. To give a good lower bound, it requires a feasible and nearly optimal primal/dual pair, and an optimal pair will not necessarily yield a good lower bound. Lemma 2 gives a lower bound for bipartite or non-bipartite problems and can be applied to a larger class of problems than Lemma 1, but Lemma 2, like Lemma 1, requires a feasible and nearly optimal primal/dual pair in order that it may be applied. Lemma 3 applies to a small class of problems, but it does not require a primal or a dual feasible solution to provide a lower bound to the number of quasi-trees in the optimal basis. Finally, Theorem 1 gives a method for generating instances that will have a certain minimum number of quasi-trees in the optimal basis by describing how to generate an instance that satisfies the hypotheses of Lemma 3. The generator can be used to test existing algorithms and is very similar to the actual generator discussed in Chapter IV. The lemmas and theorem of Chapter II make a first step toward revealing what makes instances "easy" or "difficult" by describing sufficient conditions for a problem to be "easy."

Chapter III begins with a discussion of a serial algorithm, GRNET2. The most important parallel algorithms discussed in Chapter III are PGRNET and TPGRNET, referred to above. PGRNET does pricing and pivoting in parallel, and TPGRNET does pricing in parallel and executes pivots serially. PGRNET (for Parallel GRNET) gives to each processor a subset of the arcs and puts each processor in charge of computing the reduced costs of arcs in its set, selecting pivot arcs from that set, and executing the pivots on the corresponding locked quasi-trees. All processors execute the same code in PGRNET. The second algorithm, TPGRNET (for Task Parallel GRNET), is a variant of PGRNET for *large-grained* problems, problems that have only a few quasi-trees in the optimal basis. TPGRNET uses many routines from PGRNET, but different processors perform different tasks in parallel. Most processors compute reduced costs, one selects pivot arcs, and one executes pivots without locking quasi-trees. A hybrid algorithm, called HYGRNET, that begins its execution in a PGRNET phase and can later change to a TPGRNET phase is discussed in Section 3.6.

In Chapter IV, it is shown that GRNET2 is a state-of-the-art serial program by comparing it to GENFLO [Muthukrishnan 88], a modification of GENNET [Brown and McBride 84]. Computational results are also given for PGRNET, TPGRNET and HYGRNET, a hybrid algorithm incorporating features from PGRNET and TPGRNET. The computational results of Chapter IV show that TPGRNET outperforms PGRNET for large-grained problems. All test problems are generated randomly. The three generators used are 1) MAGEN, the generator discussed in [Clark and Meyer 89], 2) NETGEN, the pure network generator discussed in [Klingman, et al, 74], and 3) GNETGEN, a modification of NETGEN that produces generalized networks with small numbers of quasi-trees in the optimal basis. The MAGEN test problems consist of three groups of problems with 10,000 nodes, and one group with 30,000 nodes. The largest problems contain 30,000 nodes and 300,000 arcs. Results for PGRNET and TPGRNET are given for all four groups. Speedups for PGRNET ranging from 2.9 to 11.1 (on 19 processors) were obtained with these test problems, and speedups for TPGRNET range from 3.7 to 6.8. Results for HYGRNET for a group of the largest problems are given, and speedups range from 6.1 to 13.2. The NETGEN pure network test problems consist of two groups. The largest problems have 50,000 nodes and 250,000 arcs. These problems are solved by TPGRNET, and speedups range from 3.4 to 4.7. The GNETGEN problems also consist of two groups of problems, the largest of which have 2,000 nodes and 50,000 arcs. These problems are solved by TPGRNET, and speedups range from 3.2 to 5.9 on 15 processors. The final group of problems were generated by MAGEN and have 30,000 nodes and more than a million variables. PGRNET speedups for these problems range from 7.0 to 11.0.

II Bounding the Number of Quasi-Trees

2.1 Introduction

The number of quasi-trees in the basic feasible solutions of a generalized network problem plays an important role in both the serial and parallel solution times required by the network simplex method. In the parallel case, one reason for this behavior is illustrated by Figure 2.1. The dashed lines (arcs $(2,3)$ and $(6,7)$) in the illustration indicate pivot-eligible non-basic arcs, and the large "$\times$" signs indicate arcs that could leave the basis after pivoting on $(2,3)$ and $(6,7)$. An important observation is that pivoting on $(2,3)$ requires updating only the quasi-trees rooted at 1 and 3, and pivoting on $(6,7)$ requires updating only the quasi-trees rooted at 5 and 7. Thus, the two pivots can be executed in parallel by having one processor pivot on arc $(2,3)$ and having another processor pivot on arc $(6,7)$.

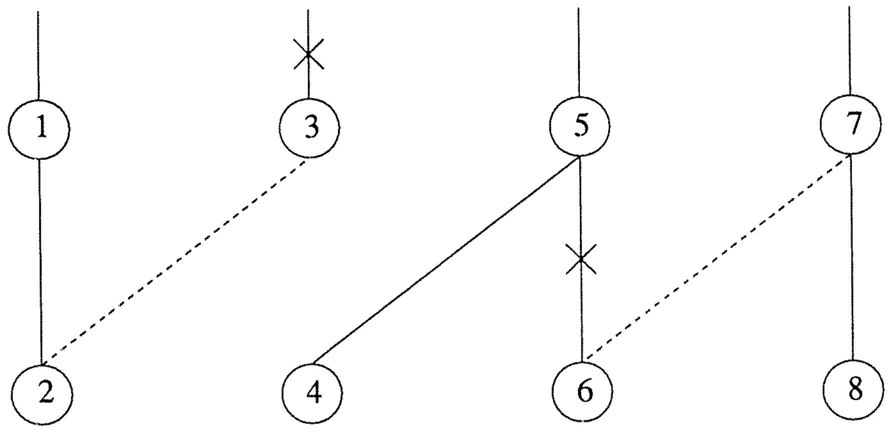

Figure 2.1 Two Pivot Eligible Arcs Connecting Disjoint Quasi-Trees

The parallel generalized network code GRNET-K [Chang, et al, 87] has been implemented on the CRYSTAL Multi-Computer [DeWitt, et al, 84] at the University of Wisconsin. The algorithm distributes different quasi-trees to different processors by way of message passing, and each processor executes pivots on pivot-eligible arcs that are "local" to its own set of quasi-trees. In the case of GRNET-K, the allocation of quasi-trees is maintained long enough to execute all pivots involving "local" pivot-eligible arcs, while in the case of PGRNET [Clark and Meyer 89], the allocation is constantly evolving, and each quasi-tree is allocated only long enough to allow the execution of one pivot. The Chaotic Column Partitioning (CCP) variant of the parallel GENFLO code is another implementation of the latter strategy. All of these codes can solve problems that have only one quasi-tree in the optimal basis or any intermediate basis, but all are most efficient when they are operating on many quasi-trees.

This makes it interesting and useful to develop tests that can be applied to an instance of GN in order to determine a lower bound for the number of quasi-trees in any optimal basis. Ideally the tests would consume very little CPU time and could be used in the construction of test problem generators that would produce problems with specified numbers of quasi-trees. Lemma 1, below, provides a test that applies to a certain class of transhipment or bipartite problems that have root arcs attached to source nodes. A disadvantage of Lemma 1 is that it requires a feasible primal/dual pair in order to be applied, and computational experience shows that the primal/dual pair must be nearly optimal in order for the lemmas to yield a good lower bound. A further disadvantage is that an optimal primal/dual pair might not yield a good approximation for a problem instance with heavy capacitation.

These two disadvantages can be put in perspective by looking at the structure of quasi-trees. Each quasi-tree is a tree with exactly one additional arc. The additional arc creates either a unique cycle or a root. Cycles seem to be rather rare, at least for bipartite problems, and this means that quasi-trees generally have root arcs. If a root arc is in the optimal basis or any intermediate basis, the root arc corresponds to a unique quasi-tree, and counting the number of root arcs in a basis yields a lower bound for the number of quasi-trees in that basis. Experimentation with the MAGEN [Clark and Meyer, 89] generator has shown that large supply at a given node is usually what forces the associated root arc into the basis, since the root arc acts as a slack variable that absorbs the excess supply. Example 3 in section 1.8 shows that the MAGEN approach for assigning supply to supply nodes may not always work. However, Lemma 1 tends to give good approximations to the number of quasi-trees when there is large supply. Example 1 in section 1.5 demonstrates how Lemma 1 can be used to show that a certain root arc will be in any optimal basis.

Lemma 2 of Section 2.6 is a generalization of Lemma 1 that can be applied to problems with relatively less supply at the supply nodes. The restriction of Lemma 1 that the supply at some of the supply nodes be equal to the capacities of the arcs emanating from those nodes is relaxed in Lemma 2. Lemma 3 of Section 2.10 uses Lemma 2 to show that problems with special loops will always have as many root arcs in the basis as special loops. Lemma 3 assumes that the problem under consideration has a very special form, but it does not need a feasible primal/dual pair in order to be applied. Example 4 of Section 2.9 illustrates an instance of GN that satisfies the hypotheses of Lemma 3. Finally, Theorem 1 of Section 2.11 builds upon the preceding lemma to validate a technique for generating problems that are guaranteed to have a certain number of quasi-trees in the optimal basis.

2.2 Notation

Throughout this work it will be assumed that there is an implied direction associated with each of the arcs, so that (i,j), for instance, has a "from" node of i and a "to" node of j. Arc (i,j) is different from arc (j,i). Source nodes may or may not have a root arc, and all attached non-root arcs lead away from source nodes. The constraint corresponding to a source node could be: $b_i = c_{ii}f_{ii} + \sum_j c_{ij}f_{ij}$. It will also be assumed that arc coefficients have been scaled in such a way that the "from" node i of arc (i,j) has a multiplier of 1 associated with it and the "to" node j has a multiplier of m_{ij} associated with it. Let c_{ij}, u_{ij} be defined as the cost and the upper bound on arc (i,j). A "root node" is any node that has an attached root arc. All of the theoretical results in this chapter apply only to a certain class, GN$'$, of problems in GN. In this class of problems, root arcs are uncapacitated. Root nodes have exactly one attached root arc, and the set of all root nodes in a given problem instance is R. Using the following definitions,

$$x^- := -\min[x,0], \quad x^+ := \max[x,0]$$

the primal and a dual formulation of GN$'$ are given by:

PRIMAL

$$\begin{array}{ll} \min & cx \\ \text{s.t.} & Gx = b \\ 0 \leq x_{ij} \leq u_{ij} & \forall\ i \neq j \\ 0 \leq x_{ii} & \forall\ i \in \mathrm{R} \end{array}$$

DUAL

$$\begin{array}{lll} \max b\mu - u(\rho - \mu\mathcal{G})^{-} & & \\ \mu_i \quad \text{unres.} & & \forall\ i \notin \mathrm{R} \\ \mu_i \quad \leq c_{ii} & & \forall\ i \in \mathrm{R} \end{array}$$

where ρ is the vector of arc costs corresponding to arcs that have finite capacities (the arcs with different "from" and "to" nodes), and $\mathcal{G}$ is the corresponding submatrix of G. Since the root arcs are unbounded, there are constraints on the dual variables corresponding to root nodes. A derivation of this dual objective function is given in Appendix 1.

2.3 The Strategy of Lemma 1

Lemma 1 uses a feasible primal/dual pair to get a lower bound for the number of quasi trees in the optimal solution of a problem P in GN$'$. It gets the lower bound by showing that certain root arcs of P have flow in any optimal solution. These root arcs are assumed to be attached to source nodes that have divergences equal to the sum of the capacities of the outgoing non-root arcs attached to that node. The lemma is based on the idea that if a root arc were removed (creating a perturbed problem P'), the dual variable at the corresponding node could be increased without destroying dual feasibility. If increasing this dual forces the value of the dual objective function for P' to be greater than the value of the primal objective function for a particular feasible solution of P, then this would indicate that removing the root arc had increased the optimal value of the primal objective function for P. This could only happen if every optimal primal solution for P had some flow on the root arc.

In more detail, the lemma utilizes a closed expression for the duality gap (referred to here as $DG(f, \pi)$) between the primal and dual objective values. Next, the lemma identifies a sum associated with a source node k for which $b_k = \sum_j u_{kj}$. The sum can be thought of as the amount by which the dual objective value for P could be increased when P is modified (by removing (k, k)) to create P'. Every root arc (k, k) for which this sum exceeds $DG(f, \pi)$ will have non-zero flow in every optimal solution of P. The number of root arcs having non-zero flow is a lower bound for the number of quasi-trees in the optimal basis. (The root arcs typically represent slack variables and therefore usually have zero cost. The lemma, however, does not require that the root arcs have zero cost.)

2.4 A Quasi-Tree Lower Bound

For notational convenience, Lemma 1 requires that there is no more than one arc between any pair of distinct nodes (i, j). The results of Lemmas 1 and 2 remain valid when this restriction is removed. The proof of Lemma 3 requires the removal of this restriction.

Lemma 1:

Let $P \in \mathrm{GN}'$ be a transportation or transshipment problem. Let f and π be respectively primal and dual feasible solutions for P with the property that for all $i \in R$, either $\pi_i = c_{ii}$, or $f_{ii} = 0$. Let

$$DG(f, \pi) = \sum_{ij} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^+ + \sum_{ij} [u_{ij} - f_{ij}](c_{ij} - \pi_i - m_{ij}\pi_j)^- .$$

Let $b_k = \sum_j u_{kj}$ for some source node $k \in R$, and let $DI(k, \pi) > DG(f, \pi)$ where

$$DI(k,\pi) := \sum_{j \neq k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j)^+.$$

Then (k, k) will have non-zero flow in any optimal solution of P.

Proof:

Throughout this proof, it will be assumed that $i \neq j$, unless stated otherwise. Assume that P is perturbed to create P' by removing the root arc (k, k), where $k \in R$ is some node for which $b_k = \sum_{(k,j)} u_{kj}$. Since k is no longer a rooted node, the associated dual is unrestricted (for P'). Let $\pi'_i = \pi_i \ \forall \ i \neq k$, and let $\pi'_k = \max_j(c_{kj} - m_{kj}\pi_j)$. The dual objective function can be expressed as follows:

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- = & \sum_i b_i\pi'_i - \sum_{ij} u_{ij}(c - \pi'G)^-_{ij} \\ = & \sum_i b_i\pi'_i - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\ & - \sum_{j \neq k} u_{kj}(c_{kj} - \pi'_k - m_{kj}\pi_j)^- \end{aligned}$$

where $I = \{(i,j) | i \neq k, i \neq j\}$. The value of π'_k is large enough so that $(c_{kj} - \pi'_k - m_{kj}\pi_j) \leq 0 \ \forall \ j$. So,

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- = & \sum_i b_i\pi'_i - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\ & + \sum_{j \neq k} u_{kj}(c_{kj} - \pi'_k - m_{kj}\pi_j). \end{aligned}$$

The last sum involves all (k,j). Since $\pi'_k = (\pi'_k - \pi_k) + \pi_k$,

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- = & \sum_i b_i\pi_i + b_k(\pi'_k - \pi_k) \\ & - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\ & + \sum_j u_{kj}(\pi_k - \pi'_k) + \sum_j u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j). \end{aligned}$$

Since $b_k(\pi'_k - \pi_k) = \sum_j u_{kj}(\pi'_k - \pi_k)$, the dual variable π'_k can be eliminated from the expression for the dual objective function.

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- = & \sum_i b_i\pi_i - \sum_{ij\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\ & + \sum_j u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j). \end{aligned}$$

The last sum involves all (k,j). The next expression splits this last sum by separating the positive and negative terms.

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- = & \sum_i b_i\pi_i - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\ & - \sum_j u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j)^- \\ & + \sum_j u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j)^+ \\ = & \sum_i b_i\pi_i - \sum_{ij} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- + DI(k,\pi). \end{aligned}$$

The remaining steps of the proof relate the dual objective function to the primal objective function (evaluated at f). The identity

$$b_t = \begin{cases} f_{tt} + \sum_{tj} f_{tj} + \sum_{jt} m_{jt} f_{jt}, & \text{if node } t \text{ has an attached root arc, or} \\ \sum_{tj} f_{tj} + \sum_{jt} m_{jt} f_{jt}, & \text{otherwise} \end{cases}$$

is used eliminate the right hand side vector b.

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- = &\sum_{i\in R} f_{ii}\pi_i + \sum_{ij} f_{ij}\pi_i + \sum_{ij} m_{ij}f_{ij}\pi_j \\ &- \sum_{(i,j)} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- + DI(k,\pi) \\ = &\sum_{i\in R} f_{ii}\pi_i + \sum_{ij} f_{ij}c_{ij} - \sum_{ij} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j) \\ &- \sum_{ij} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- + DI(k,\pi). \end{aligned}$$

By splitting the sum $\sum_{ij} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)$ into its negative and positive terms, the dual objective becomes:

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- = &\sum_{i\in R} f_{ii}\pi_i + \sum_{ij} f_{ij}c_{ij} - \sum_{ij} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^+ \\ &- \sum_{ij} [u_{ij} - f_{ij}](c_{ij} - \pi_i - m_{ij}\pi_j)^- + DI(k,\pi) \\ = &\sum_{i\in R} f_{ii}\pi_i + \sum_{ij} f_{ij}c_{ij} - DG(f,\pi) + DI(k,\pi). \end{aligned}$$

By assumption, either $\pi_i = c_{ii}$, or $f_{ii} = 0 \ \forall \ i \in R$. So,

$$\sum_{i\in R} f_{ii}\pi_i = \sum_{i\in R} f_{ii}c_{ii}.$$

Making this final substitution relates the value of the dual objective function to the value of the primal objective function, given by $\sum_{i\in R} f_{ii}c_{ii} + \sum_{ij} f_{ij}c_{ij}$.

$$\begin{aligned} b\pi' - u(\rho - \pi'\mathcal{G})^- &= \sum_{i\in R} f_{ii}c_{ii} + \sum_{ij} f_{ij}c_{ij} - DG(f,\pi) + DI(k,\pi) \\ &\geq cf^* - DG(f,\pi) + DI(k,\pi), \end{aligned}$$

where f^* is an optimal primal solution. Since $-DG(f,\pi) + DI(k,\pi) > 0$, the dual objective value of P' is strictly greater than the the optimal objective value of P. This means that arc (k,k) must have a non-zero flow in any optimal basis of P, because the above analysis shows that removing the arc perturbs the optimal primal objective value of P. If arc (k,k) had zero flow in some optimal solution, then removing it would have no effect on the primal objective value. QED.

It's important to point out that if either $\pi_i = c_{ii}$, or $f_{ii} = 0 \ \forall \ i \in R$, as assumed in Lemma 1, then $DG(f,\pi)$ is precisely equal to the duality gap between the primal and dual objective functions evaluated at f and π respectively. This is because the proof of the lemma establishes both $b\pi' - u(\rho - \pi'\mathcal{G})^- = b\pi - u(\rho - \pi\mathcal{G})^- + DI(k,\pi)$ and $b\pi' - u(\rho - \pi'\mathcal{G})^- = \sum_{i \in R} f_{ii}c_{ii} + \sum_{ij} f_{ij}c_{ij} - DG(f,\pi) + DI(k,\pi)$. (It is also easy to derive this directly without reference to the perturbed problem.) Also notice that the proof of Lemma 1 establishes that no feasible solution satisfies $f_{kk} = 0$, $DI(k,\pi) > DG(f,\pi)$, and the other hypotheses, since such a solution would be feasible for P' but would have an objective value dominated by the objective value of the perturbed dual solution. However, Example 1, on the following pages, gives a problem with a root node k for which the hypotheses of Lemma 1 are satisfied, but for which another feasible solution exists with $f_{tt} = 0 \ \forall \ t$. This establishes that the class of problems addressed by Lemma 1 does not consist merely of problems for which $f_{kk} > 0$ for root nodes k in every feasible solution.

2.5 Example 1 (The Application of Lemma 1)

Figure 2.3 is the graph of a generalized network flow problem. The node divergences are indicated next to the nodes. The cost, the multiplier and the capacity for each of the arcs are indicated in brackets next to each arc. Lemma 1 can be used to show that arc $(1, 1)$ will be in any optimal basis of the network.

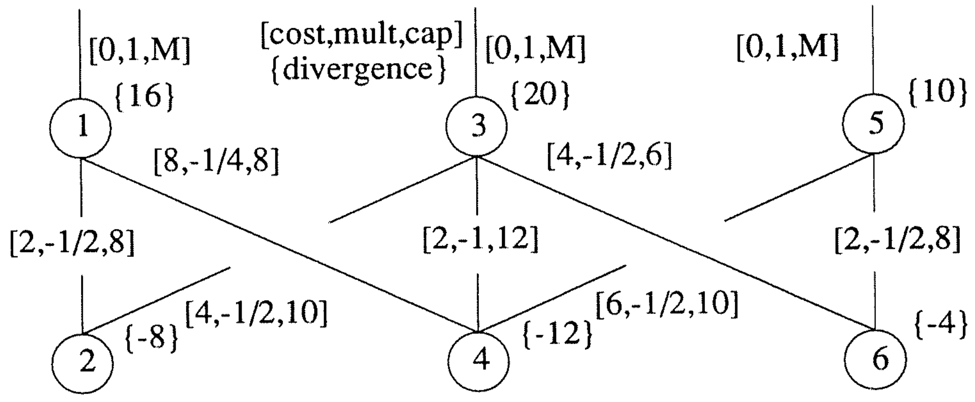

Figure 2.3 Example for Lemma 1

Let the primal feasible solution f be given (see Fig. 2.4)by $f_{34} = 12, f_{12} = 8, f_{56} = 8, f_{32} = 8, f_{11} = 8, f_{55} = 2$, and flows of 0 for all remaining arcs. Let the dual feasible solution be given by $\pi_1 = 0, \pi_3 = 0, \pi_5 = 0, \pi_2 = -6, \pi_4 = -2, \pi_6 = -12$.

For this problem $DI(1, \pi)$ is easy to compute because the corresponding sum has just one non-zero term:

$$\begin{aligned}\sum_{(1,j)} u_{1j}(c_{1j} - \pi_1 - m_{1j}\pi_j)^+ &= u_{14}(c_{14} - \pi_1 - m_{14}\pi_4) \\ &= 8(8 - 0 - (-1/4)(-2)) \\ &= 60.\end{aligned}$$

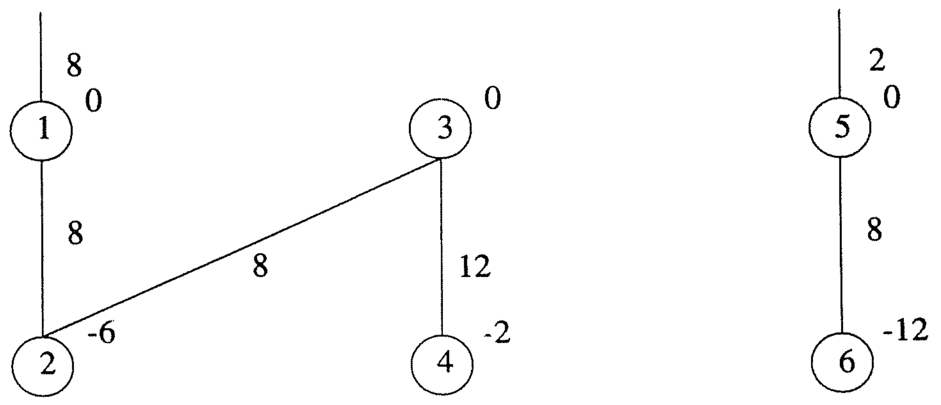

Figure 2.4 Primal and Dual Feasible Solutions

The duality gap, $DG(f, \pi)$, can be computed as follows:

$$\sum_{ij} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^+ - \sum_{ij} [u_{ij} - f_{ij}](c_{ij} - \pi_i - m_{ij}\pi_j)^-$$

$$= \quad f_{32}(c_{32} - \pi_3 - m_{32}\pi_2) - [u_{36} - f_{36}](c_{36} - \pi_3 - m_{36}\pi_6)$$

$$= \quad 8(4 - 0 - (-1/2)(-6)) - [6 - 0](4 - 0 - (-1/2)(-12))$$

$$= 20.$$

Since $60 > 20$, lemma 1 shows that arc $(1, 1)$ will be in any optimal basis. The particular primal solution chosen here is optimal. The primal objective value is given by:

$$\sum_{i \in \{1,3,5\}} c_{ii} f_{ii} + \sum_{(i,j)} c_{ij} f_{ij} = 88.$$

The dual feasible solution used to verify the conditions of the lemma is not optimal. The optimal dual solution also satisfies the conditions and can be computed by using the set of arcs $(1, 1), (1, 2), (3, 2), (3, 4), (5, 5), (5, 6)$ as a basis, and the dual variables corresponding to this basis can be computed by starting at the roots and moving down the quasi-trees: $\pi_1 = 0,\ \ \pi_2 = (2 - \pi_1)/(-1/2) =$

-4, $\pi_3 = 4 - (-1/2)\pi_2 = 2$, $\pi_4 = (2 - \pi_3)/(-1) = 0, \pi_5 = 0, \pi_6 = (2 - \pi_5)/(-1/2) = -4$. These dual variables are indicated in Figure 2.5.

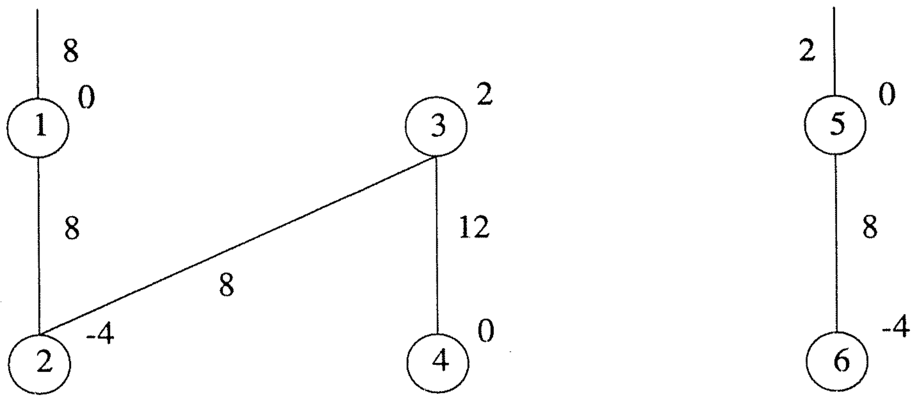

Figure 2.5 Optimal Primal and Dual Solutions

The dual objective value is:

$$b\pi - u(\rho - \pi\mathcal{G})^{-} = 88.$$

Since the value of the primal objective function equals the value of the dual objective function, the primal solution is optimal. (Note that the value of $DI(1, \pi)$ for this solution is 64.) Notice that arc $(1, 1)$ is in the optimal basis. This is consistent with the result of Lemma 1. However, the root arcs of this problem are not forced into the basis simply to achieve feasibility, since the solution indicated in Figure 2.6 is primal feasible and has no flow on the root arcs.

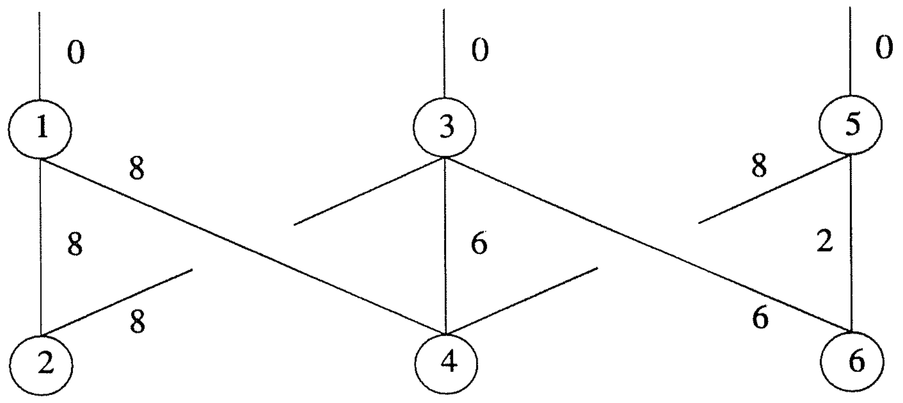

Figure 2.6 Feasible Flow Involving No Root Arcs

2.6 A Lower Bound For Problems Without Matched Supply

The MAGEN generator is discussed in detail in Section IV. It generates bipartite problems in GN$'$ that have a collection of root nodes with their supply equal to the sum of the capacities of the outgoing arcs. Computational experience with modified versions of MAGEN has shown that the supply at a root node (i.e. a node with an attached root arc) does not have to equal the sum of the capacities of the outgoing arcs in order to force the attached root arc into the basis. Most of the test problems in groups 1 through 4 (Chapter IV) have the same number of quasi-trees if the supply at the root nodes is set at 3/4 the usual value. Hence, we develop a generalization of Lemma 1 that does not require a match of supply and capacity of outgoing arcs at the root nodes. Lemma 2 gives the generalization.

Lemma 2:

Let $P \in \mathrm{GN}'$ be either a transportation or transshipment problem. Let f and π be respectively primal and dual feasible solutions for P, such that for all $i \in R$, either $\pi_i = c_{ii}$, or $f_{ii} = 0$. Let $DG(\cdot,\cdot)$ be defined as above, let k be a source node with attached root arc, and let π' be chosen such that $\pi'_k \geq \pi_k$, and $\pi'_s = \pi_s \ \forall \ s \neq k$. Let F be defined as the set of arcs (i,j) for which $(c - \pi G)_{ij} \leq 0$. Let $\overline{F}_k$ be defined as the set of arcs (k,j) for which $(c - \pi G)_{kj} > 0$ and $(c - \pi' G)_{kj} \leq 0$. Let

$$\sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j) + (\pi'_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) > DG(f,\pi).$$

Then (k,k) will have non-zero flow in any optimal solution of P.

Proof:

See Appendix 2.

2.7 Example 2 (The Application of Lemma 2)

Figure 2.7 is the graph of a generalized network flow problem. The node divergences are indicated by the nodes. The cost, the multiplier and the capacity for each of the arcs are indicated in brackets next to each arc. Lemma 2 can be used to show that arc $(1,1)$ will be in any optimal basis of the network, even though the divergence at node 1 is exceeded by the sum of the capacities of the adjacent non-root arcs.

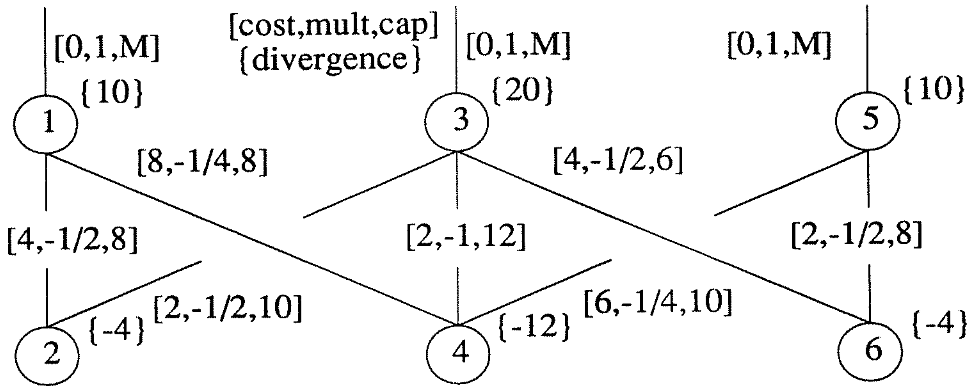

Figure 2.7 Generalized Network With Moderate Supply

A feasible primal/dual pair is given by: $f_{34} = 12, f_{56} = 8, f_{32} = 8, f_{11} = 10, f_{55} = 2, \pi_1 = \pi_3 = \pi_5 = 0, \pi_2 = -4, \pi_4 = -2, \pi_6 = -4$. The resulting graph is shown in Figure 2.8.

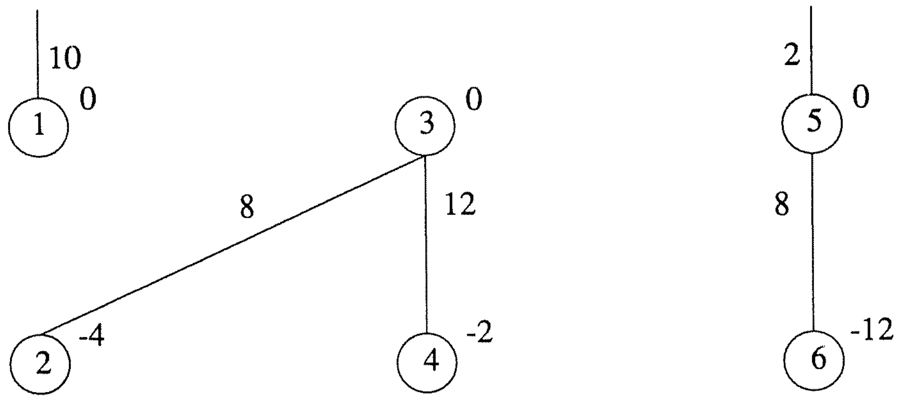

Figure 2.8 Primal and Dual Feasible Solutions

Satisfying the hypotheses of Lemma 2 requires identifying a value, π'_1 which will make the the final hypothesis of Lemma 2 hold. The set $\overline{F}_1$ is the set of all arcs that are not in F but have a non-positive reduced cost when π_1 is replaced by π'_1. The simplest way to chose π'_1 is to make it as large as possible without forcing $b_1 - \sum_{(1,j)\in F\cup\overline{F}_1} u_{1j} < 0$. (The value of π'_1 found in this way will not necessarily yield a maximum value for the left hand side of the inequality of Lemma 2). If $\pi'_1 = 2$, then $\overline{F}_1 = \{(1,2)\}$, and the left hand side of the inequality of Lemma 2 can be computed as follows:

$$\begin{aligned}\sum_{(1,j)\in\overline{F}_1} u_{1j}(c_{1j} - \pi_1 - m_{1j}\pi_j) + (\pi'_1 - \pi_1)(b_1 - \sum_{(1,j)\in F\cup\overline{F}_1} u_{1j}) \\ = 8(4 - 0 - (-1/2)(-4)) + (2-0)(10-8) \\ = 20.\end{aligned}$$

Computing $DG(f,\pi)$ in the usual way yields $DG(f,\pi) = 14$. Since $20 > 14$, Lemma 2 shows that arc $(1,1)$ will be in any optimal basis. In this example, as in example 1, the primal feasible solution f is optimal. The primal objective value is given by:

$$\sum_{i\in\{1,3,5\}} c_{ii}f_{ii} + \sum_{(i,j)} c_{ij}f_{ij} = 56.$$

The graph of Figure 2.8 can have arc $(3,3)$ added to it with a flow of 0 to make the total graph consist of 3 rooted trees. The dual variables corresponding this primal solution can be computed by back substitution: $\pi_1 = 0$, $\pi_3 = 0$, $\pi_2 = (2-\pi_3)/(-1/2) = -4$, $\pi_4 = (2-\pi_3)/(-1) = -2$, $\pi_5 = 0$, $\pi_6 = (2-\pi_5)/(-1/2) = -4$. These dual variables are the same as the duals in Figure 2.8. The dual objective value is: $b\pi - u(\rho - \pi\mathcal{G})^- = 56$. Since the primal objective function equals the dual objective function, the primal solution is optimal. Notice that arc $(1,1)$ is in the optimal basis. As with example 1, a primal feasible flow does not have to put flow on the root arcs. The solution indicated in Figure 2.9 is primal feasible and has no flow on the root arcs.

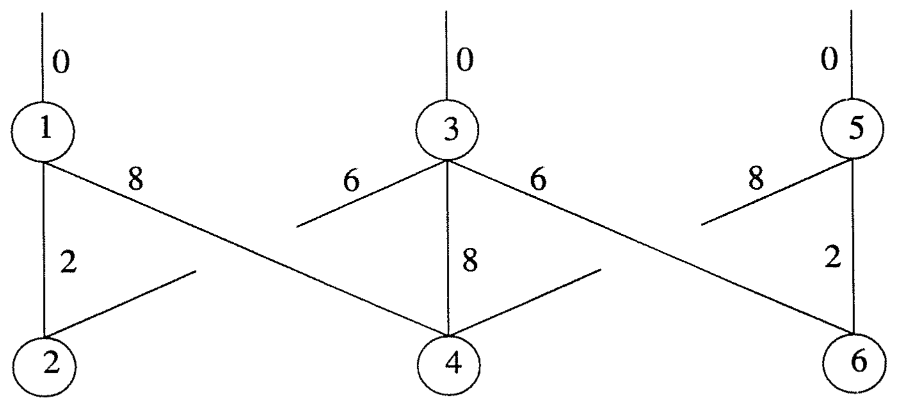

Figure 2.9 Feasible Flow Involving No Root Arcs

2.8 Example 3 (A Shortcoming of the MAGEN Strategy)

Many of the results discussed in Chapter IV are results for bipartite problems generated by MAGEN [Clark and Meyer 89]. This generator employs a simple technique in attempting to force root arcs into the optimal basis. In practice, this has the effect of creating problems with many or few quasi-trees in the basis at optimality, depending on the number of root arcs that were forced into the basis. For the purposes of this discussion, the generator can be described as follows:

The MAGEN Generator

Let RG1(r) denote the following random generator:

Start with a prespecified number of source and sink nodes. Let $SRCS$ be the set of all sources, and let $SNKS$ be the set of all sinks. (The number of source nodes must be greater than or equal to r). Initially, set $b_i = b_j = 0$ for all

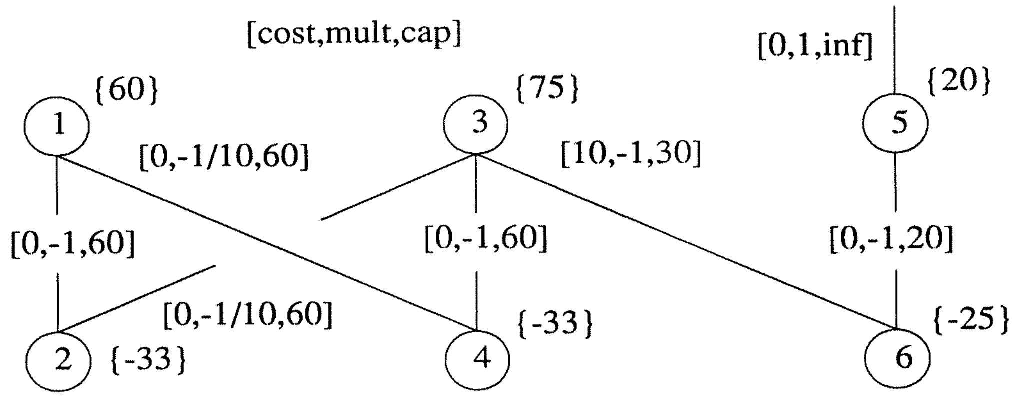

Figure 2.10 Gen. Net. Consistent With MAGEN Generation Technique

$i \in SRCS$ and $j \in SNKS$. Let $0 < \alpha < 1$. For each source node, do the following:

1 Generate a non-negative random integer between some bounds to indicate the number of arcs that will emanate from this node.

2 For each arc (i, j) that must be generated, generate its "to" node randomly, generate its non-negative cost randomly within some bounds, generate its multiplier within some bounds, and generate its non-negative upperbound (capacity) randomly within some bounds. The lower bound on all of these arcs is zero. Set $b_i := b_i + \alpha u_{ij}$, and set $b_j := b_j + \alpha u_{ij} m_{ij}$.

Now chose r source nodes. For each source node k, do the following:

3 Increase b_i so that it equals the sum of the capacities of the outgoing arcs generated so far.

4 Add the arc (k, k) with arbitrary non-negative cost and positive multiplier to the set of arcs. Arc (k, k) has no upper bound, and has a lower bound of zero.

The generalized network in Figure 2.10 demonstrates that the MAGEN technique is not guaranteed to force a quasi-tree into the basis. The node divergences are indicated within the nodes, and all problem data for this example is consistent with the MAGEN technique. The optimal solution is given in Figure 2.11. The flows on the arcs have been rounded to one decimal place.

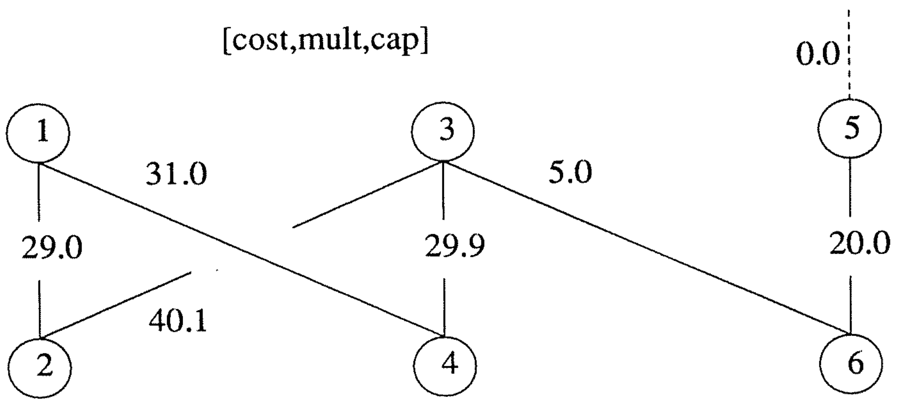

Figure 2.11 Optimal Flow (Rounded) With Zero Flow On Root

The divergence at node 5 is equal to the sum of the capacities of the outgoing arcs, but arc $(5,5)$ is not in the optimal basis. This shows that the technique used in MAGEN to create quasi trees is not guaranteed to work, despite the fact that the technique generally works in practice. Theorem 1 below will describe a very similar generator, $RG2(r)$, that uses a technique that is guaranteed to force root arcs into the basis. The technique is very similar to steps 3 and 4 of $RG1(\cdot)$ above, but it involves adding two more arcs to the instance of GN$'$.

2.9 Example 4 (Satisfying the Hypotheses of Lemma 3)

Lemma 3 below gives sufficient conditions for an instance of GN$'$ to have at least a certain number of quasi-trees in the optimal solution. It shows that under some simple assumptions, the hypotheses of Lemma 1 can be satisfied. The lemma assumes that the instance under consideration has two "paired" or "parallel" arcs associated with all root nodes. The arcs are "parallel" in the sense that they emanate from the same node and lead to the same node. The arcs have a large enough capacity so that either one of them can satisfy the demand of the "to" node, and one of the arcs has a large enough cost (and an appropriate multiplier) so that it will have a strictly positive reduced cost in any optimal solution. This, together with a large supply at the "from" node, makes it possible to show that all of the hypotheses of Lemma 2 are satisfied by any optimal solution, and therefore all root arcs will have non-zero flow in any optimal basis.

The following instance of GN$'$ satisfies the hypotheses of Lemma 3. The cost, multiplier, and capacity for each of the arcs are given in brackets next to the arc. The brackets for the "parallel" arcs have been designated by the superscripts 1 and 2.

If the root arc $(5,5)$ is removed from the graph and the resulting problem is solved, the flows (rounded to one decimal place) are: $f_{12} = 60.6, f_{14} = 89.3, f_{32} = 57.3, f_{34} = 22.6, f_{36} = 0.0, f_{54} = 10.0, f_{56}^1 = 29.7, f_{56}^2 = 0.2$. The optimal value of the objective function for this perturbed problem is 913.1. If arc $(5,5)$ is not removed from the graph the optimal flows are: $f_{12} = 60.6, f_{14} =$

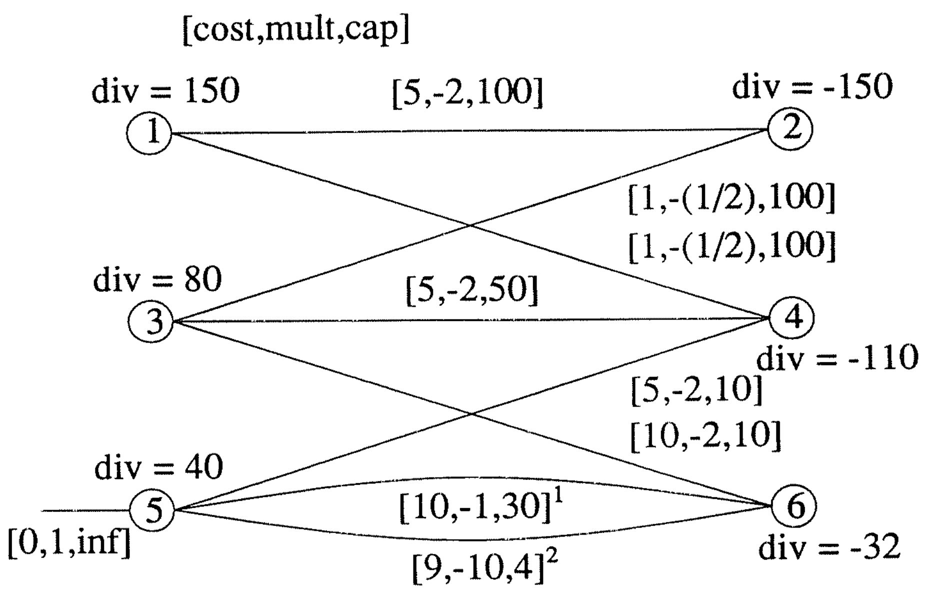

Figure 2.12 Generalized Network Satisfying the Hypotheses of Lemma 3

89.3, $f_{32} = 57.3, f_{34} = 22.6, f_{36} = 0.0, f_{54} = 10.0, f^1_{56} = 0.0, f^2_{56} = 3.2, f_{55} =$ 26.8, and the optimal value of the objective function is 642.1. Since $(5,5)$ is uncapacitated and has non-zero flow in this optimal solution, $(5,5)$ is in the optimal basis. Lemma 3 shows that $(5,5)$ must have non-zero flow in any optimal solution.

Note that a problem with parallel arcs can be transformed into a problem without parallel arcs by adding additional intermediate nodes, but this would complicate the proof of Lemma 3.

2.10 Sufficient Conditions For Root Arcs To Have Flow

Lemma 3:

Let P be a transportation or transshipment instance of GN$'$. Let P have s source nodes, and let r of them have an attached uncapacitated root arc. Each root node k, is assumed to have two "parallel" arcs. For root node k, the common "to" node is node j. The superscripts 1 and 2 will be used to distinguish the data associated with these arcs. Let $m_{ij} < 0 \ \forall \ (i,j)$ with $i \neq j$, including the paired arcs. Let $c^1_{kj} > c^2_{kj} \geq c_{kk}$, $m^1_{kj} \geq m^2_{kj}$, and let $u^1_{kj} > u^2_{kj} > 0$. Let $\sum_{J_j} u_{ij} m_{ij} > b_j > u^2_{kj} m^2_{kj}$, where J_j is the set of all arcs incident to node j other than the paired arcs. Let $b_k = u^1_{kj} + \sum_{J_k} u_{kj}$, where J_k is the set of all arcs from k except the paired arcs. If P has an optimal solution, then the optimal solution will have at least r quasi-trees in the basis.

Proof:

As in the proof of Lemma 1, $i \neq j$ throughout this proof, unless stated otherwise.

We will show that the hypotheses of Lemma 2 (as extended to allow parallel arcs) are satisfied. Let f and π be an optimal primal/dual pair. Since f and π are optimal, complementary slackness requires that for all $i \in R$, either $f_{ii} = 0$ or else $\pi_i = c_{ii}$. This satisfies one of the hypotheses of Lemma 2.

Since $u^2_{kj} m^2_{kj} < b_j$, f^2_{kj} must be such that $f^2_{kj} < u^2_{kj}$, otherwise the constraint $f^1_{kj} m^1_{kj} + f^2_{kj} m^2_{kj} + \sum_{i \in J_j} u_{ij} m_{ij} = b_j$ will force some arcs to have a negative flow. Since $f^2_{kj} < u^2_{kj}$ and f is optimal, the reduced cost on this arc must be non-negative. So,

$$c^2_{kj} - \pi_k - m^2_{kj}\pi_j \geq 0. \tag{1}$$

Since $\sum_{i \in J_j} u_{ij} m_{ij} > b_j$, primal feasibility requires that either $f^1_{kj} > 0$ or $f^2_{kj} > 0$ or both. The next step in this proof involves showing that $f^2_{kj} > 0$. To do this, assume that $f^2_{kj} = 0$. Then $f^1_{kj} > 0$, and this implies that $c^1_{kj} - \pi_k - m^1_{kj}\pi_j \leq 0$. Since $c^1_{kj} > c_{kk}$, and since the dual feasibility of π requires that $c_{kk} \geq \pi_k$, $c^1_{kj} - \pi_k > 0$. So,

$$0 < c^1_{kj} - \pi_k \leq m^1_{kj}\pi_j,$$

and thus,

$$\pi_j < 0.$$

The negativity of π_j then leads to the following contradiction:

$$\begin{gathered}
m^1_{kj} \geq m^2_{kj} \\
-m^1_{kj}\pi_j \geq -m^2_{kj}\pi_j \\
c^1_{kj} - \pi_k - m^1_{kj}\pi_j > c^2_{kj} - \pi_k - m^2_{kj}\pi_j \\
0 \geq c^1_{kj} - \pi_k - m^1_{kj}\pi_j > c^2_{kj} - \pi_k - m^2_{kj}\pi_j \geq 0.
\end{gathered}$$

The conclusion is that $f^2_{kj} > 0$ in any optimal solution f. Since $0 < f^2_{kj} < u^2_{kj}$, the reduced cost on this arc must be 0, i.e.,

$$c^2_{kj} - \pi_k - m^2_{kj}\pi_j = 0. \tag{2}$$

Thus the second paired arc is in the set F. Since $c^2_{kj} \geq c_{kk} \geq \pi_k$, equation (2) yields $m^2_{kj}\pi_j \geq 0$, or $\pi_j \leq 0$. Since $m^1_{kj} \geq m^2_{kj}$, we have $-m^1_{kj}\pi_j \geq -m^2_{kj}\pi_j$, and thus $c^1_{kj} - \pi_k - m^1_{kj}\pi_j > c^2_{kj} - \pi_k - m^2_{kj}\pi_j = 0$.

Now to use Lemma 2, set

$$\pi'_k = c^1_{kj} - m^1_{kj}\pi_j$$

With this choice of π'_k, the set $\overline{F}_k$ contains the first paired arc, and $\pi'_k > \pi_k$, which satisfies another of the hypotheses of Lemma 2. Putting everything together satisfies the inequality of Lemma 2.

$$\begin{aligned} u^1_{kj} &> u^2_{kj} \\ u^1_{kj}(c^1_{kj} - \pi_k - m^1_{kj}\pi_j) &> u^2_{kj}(c^1_{kj} - \pi_k - m^1_{kj}\pi_j) \\ &= u^2_{kj}(\pi'_k - \pi_k) \\ &= (u^2_{kj} + u^1_{kj} + \sum_{J_k} u_{kj} - b_k)(\pi'_k - \pi_k) \\ &\geq (\sum_{(k,j)\in F\cup\overline{F}_k} u_{kj} - b_k)(\pi'_k - \pi_k). \end{aligned}$$

The last step is valid because $u_{kj} \geq 0 \ \forall \ (k,j)$ and because $F \cup \overline{F}_k$ is a subset of the set of all arcs (other than the paired arcs) that emanate from node k. Finally,

$$u^1_{kj}(c^1_{kj} - \pi_k - m^1_{kj}\pi_j) + (b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj})(\pi'_k - \pi_k) > 0 = DG(f,\pi).$$

This satisfies the last of the hypotheses of Lemma 2. Since k is an arbitrary root node, the above discussion must apply to all root nodes, and this means that the root arc associated with each root node will have positive flow in any optimal basis. Q.E.D.

From the proof of Lemma 3, the optimal flow on the first paired arc is 0, so this arc could be deleted from the network without changing the optimal solution. However, deleting it would produce a network in which the total capacity of the outgoing non-root arcs at k was less than b_k, and the goal of our approach is to avoid this trivial case.

2.11 A Guaranteed Generator

In this section, a random generator for generalized network problems will be described. Theorem 1 will prove that the random generator will generate problems with at least r (a prespecified number) quasi-trees.

The Generator

Let RG2(r) denote the following random generator:

Start with a prespecified number of source and sink nodes. Let $SRCS$ be the set of all sources, and let $SNKS$ be the set of all sinks. (The number of source and sink nodes must both be greater than or equal to r). Start with $b_i = b_j = 0$ for all $i \in SRCS$ and $j \in SNKS$. Let $0 < \alpha < 1$. For each source node, do the following:

1 Generate a non-negative random integer between some bounds to indicate the number of arcs that will emanate from this node.

2 For each arc (i, j) that is generated, generate its "to" node randomly within bounds, generate its non-negative cost randomly within some bounds, gen-

erate its negative multiplier within some bounds, and generate its non-negative upperbound (capacity) randomly within some bounds. Set $b_i := b_i + \alpha u_{ij}$, and set $b_j := b_j + \alpha u_{ij} m_{ij}$.

Now choose r source nodes. The set R for this problem instance will consist only of these r nodes, since all root arcs will be generated explicitly. For each source node k selected, do the following:

3 Increase b_k so that it equals the sum of the capacities of the outgoing arcs generated so far.

4 Select a "to" node j that has not previously been selected (so that no demand node has more than one incident set of paired arcs). Increase the demand at node j by setting $b_j < \sum_{J_j} u_{ij} m_{ij}$. Randomly select the negative arc multiplier m^2_{kj} within some bounds. Set $m^1_{kj} \leq m^2_{kj}$.

5 Chose a capacity u^2_{kj} with $u^2_{kj} > b_j / m^2_{kj}$. Set $u^1_{kj} > u^2_{kj}$. Randomly chose the non-negative costs c^1_{kj} and c^2_{kj} with $c^1_{kj} > c^2_{kj}$.

6 Set $b_k := b_k + u^1_{kj}$.

7 Add the arc (k, k) with $c_{kk} \leq c^2_{kj}$, $m_{kk} = 1$ and $u_{kk} = \infty$ to the set of arcs.

Theorem 1:

The generation procedure RG2(r) will generate a problem having at least r quasi-trees in the optimal basis.

Proof:

All costs are assumed to be non-negative, and since all variables have non-negative flow, the problem is not unbounded. The problem is feasible because a feasible solution can be found as follows:

a For each of the arcs generated in step 2 above, set $f_{ij} = \alpha u_{ij}$.

b For each root node k, set $f^1_{kj} = 0$, $f^2_{kj} = (b_j - \sum_{i \in J_j} f_{ij} m_{ij}) / m^2_{kj}$. (This will not violate the capacity of this arc, because $u^2_{kj} > b_j / m^2_{kj}$.)

c For each of the root arcs generated in step 7, set $f_{ii} = b_i - \sum j f_{ij} - f^2_{kp}$, where p is the "to" node of the paired arc incident to node i.

Since the problem is feasible and is not unbounded, an optimal solution exists.

All of remaining hypotheses of Lemma 3 are explicitly satisfied during the generation procedure, so the r selected root arcs will have positive flow in any optimal solution. Q.E.D.

III Algorithms

3.1 Introduction

In this chapter, serial and parallel algorithms for GN are discussed. Section 3.2 gives a description of GRNET2, a modification of the serial generalized network code GRNET of Chang and Engquist. The principal algorithmic difference between these codes is that GRNET2 uses a gradual penalty method [Grigoriadis 84] rather than the big M method. There are also differences in the treatment of candidate lists. Section 3.4 gives a description of PGRNET, a "parallel pivoting" version of the GRNET2 algorithm that uses the parallelization technique discussed in Section 2.1. Section 3.5 discusses TPGRNET, a "parallel pricing overlapped with pivoting" algorithm that was developed to solve large-grained problems not handled well by PGRNET. It will be shown in Chapter IV that the two algorithms PGRNET and TPGRNET are complementary in the sense that PGRNET solves small-grained problems (problems with many small quasi-trees in an optimal basis) more efficiently than TPGRNET and TPGRNET solves large grained problems more efficiently than PGRNET. Section 3.6 describes a hybrid algorithm, HYGRNET, that combines PGRNET and TPGRNET and generally yields results that are better than either of the two algorithms alone. Section 3.7 discusses some of the other codes in the literature.

3.2 GRNET2

Figure 3.1 gives a flow chart for GRNET2. In the figure, "l.t." designates *list_threshold*, a candidate list parameter. The following is a summary of the

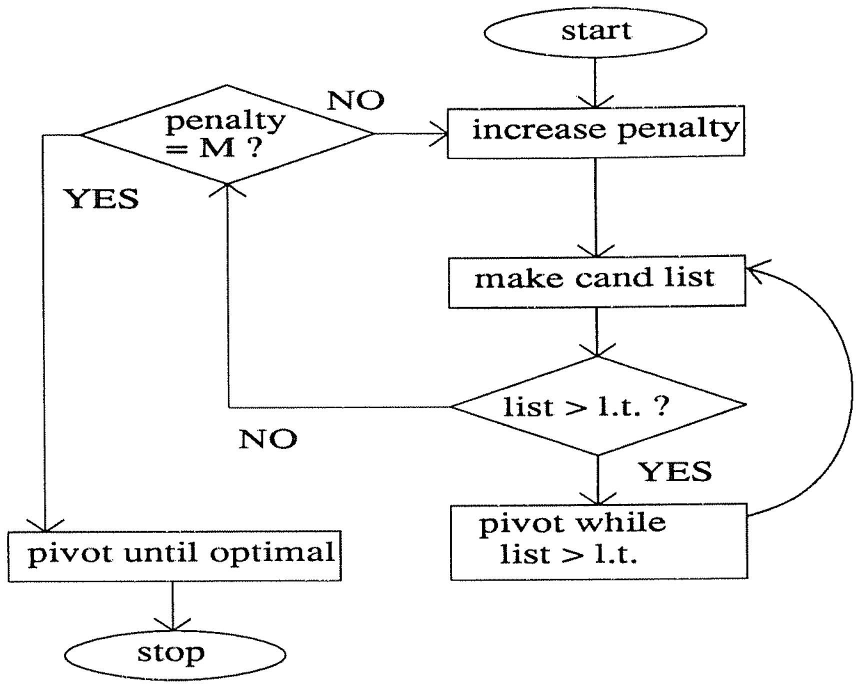

Figure 3.1 Flow Chart For Serial Algorithm GRNET2

serial algorithm GRNET2.

The (serial) GRNET algorithm is:

INITIALIZATION

Set initial flows and penalty on the artificial arcs. Partition the set of arcs into roughly equal sized segments for pricing during the next stage.

STAGE 1 (*serial pivoting with candidate lists*)

Develop a separate candidate list for each segment of the arc list. If the number of arcs stored in a candidate list is greater than some number *list_threshold*, select the "best" pivot-eligible arc (if any) from this list, execute the pivot and go to the next candidate list. If the number of arcs is less than or equal to *list_threshold*, scan the corresponding segment of the arc list to make a new candidate list. If it is not possible to make a candidate list from that segment with more than *list_threshold* entries and the penalty on the artificial arcs has reached its maximum value go to STAGE 2. Otherwise increase the penalty on the artificial arcs, recompute duals, create new candidate lists and continue STAGE 1.

STAGE 2 (*verification of optimality*)

Scan the whole arc list for pivot-eligible arcs. If a pivot-eligible arc is found, then the pivot is immediately executed (no candidate list is constructed). If a complete sweep through the entire arc list can be made without finding any pivot-eligible arcs, then optimality has been reached.

GRNET in its original form uses an all artificial starting basis discussed in [Glover, et al, 74]. An artificial (root) arc with *big* M penalty is attached to each node, providing an initial basis of m quasi-trees with just one node and one arc per quasi-tree. Each artificial arc is given a flow that satisfies the constraint corresponding to the node. GRNET2 uses a starting procedure motivated by the gradual penalty method (GPM) discussed in [Grigoriadis 1984]. Again, a root arc with an appropriate flow is attached to each node, but the gradual penalty method gives a moderate initial penalty to the artificial arcs and then gradually increases the penalty. (As discussed below, this results in a dramatic improvement in performance in certain problem classes.) GRNET uses one candidate list and searches the entire arc list when refreshing the candidate list. GRNET2 uses a candidate list strategy similar to that of GRNET, but GRNET2 partitions the arc set into segments, and a separate candidate list is maintained for each segment of the arc list. Successive pivot arcs are selected from successive candidate lists (which correspond to distinct segments in the partition). To do a pivot, GRNET2 selects a candidate list as the source of the next pivot-eligible arc. A pivot arc is selected from that candidate list if the number of arcs remaining in the candidate list is greater than *list_threshold.* If the number of arcs in the list is less than or equal to *list_threshold*, the list is purged and pricing is done in the corresponding segment until a new candidate list is filled. If a sweep through the segment yields *list_threshold* or fewer pivot-eligible arcs, the penalty is increased by popping a new value off of a stack, dual variables are recomputed, and pivots are executed until there are again *list _threshold* or fewer pivot-eligible arcs in some segment. This gradual penalty method is continued until the penalty has reached its maximum (*big M*) value.

The strategy of maintaining multiple candidate lists was motivated by research with parallel algorithms. The parallel code PGRNET, which maintains a separate candidate list for each processor, yielded super linear speedup over an older version of GRNET2 having just one candidate list. The current version of GRNET2 (with multiple candidate lists) is up to 45% faster than the old version, for large-scale problems having more than 300,000 arcs.

The usual motivation for using gradual penalty method is that the procedure can utilize original costs more effectively than the *big M* method. In the case of generalized network flow problems, the gradual penalty method has the added advantage that pivot arcs are initially less likely to be chosen in such a way that the outgoing arc will be an artificial arc. Under the *big M* method, the initial penalty on the artificial arcs is so large that pivots tend to be chosen essentially to reduce the flow on the artificial arcs, and therefore pivots tend to cause artificial arcs (which are root arcs) to leave the basis quickly. This tends to reduce rapidly the number of quasi-trees in the basis. Under the gradual penalty method, the penalty on the artificial arcs is more moderate, and artificial arcs are less likely to leave the basis immediately. This has the result that root-arcs remain in the basis for a longer time, and quasi-trees are more numerous and smaller than they would be under the *big M* method. Since quasi-trees are smaller, there is less work involved in executing pivots. Moreover, since quasi-trees are more numerous, algorithms like PGRNET that rely on the disjoint nature of the basis for parallelism can run more efficiently. Finally, since pivots are not chosen merely to achieve feasibility, more pivots result in advancement toward optimality, and the overall number of pivots is significantly reduced. The computational experience described in [Grigoriadis 1984] shows that the gradual penalty method can give a 15% improvement in CPU time for pure network flow problems. Our computational experience shows that our version of the gradual penalty method reduces CPU time by a factor of 13 to 29 (not 13% to 29%) for generalized network flow problems with heavy capacitation. The reduction in computing time decreases as the number of quasi-trees in the optimal basis is increased, but almost all problems can be solved more quickly by the gradual penalty method.

3.3 Parallel Algorithms

In developing distributed versions of GRNET2, a number of different strategies were tried. Strategy 1 , a *tree allocation* strategy, involves partitioning the set of quasi-trees, and executing pivots on *local arcs* in parallel. A *local arc* is an arc that is incident to two nodes belonging to the same subset of the partition. This scheme was used on the University of Wisconsin CRYSTAL multicomputer, and is discussed in [Chang, et al, 1987]. It eliminates the need for locking quasi-trees and is suited to distributed memory environments. While this approach succeeds for certain classes of generalized networks, it has some shortcomings. In particular, the set of *local arcs* may be a very small fraction of the whole arc set. This means that the set of potential pivot arcs is a small fraction of the whole arc set. The scheme does not scale well because the set of *local arcs* diminishes as the number of subsets increases, but the greatest disadvantage to the scheme is that the set of quasi-trees must be periodically repartitioned so that pivot- eligible *cross arcs* become *local arcs*. This is a time consuming task, and it is difficult to quickly partition the quasi-trees in such a way that each subset contains roughly the same number of nodes and each subset contains a substantial number of pivot-eligible *local arcs*. Strategy 2, an *arc allocation* strategy, partitions the set of arcs, rather than the nodes corresponding to quasi-trees. Pivot arcs are selected from different segments of the partition in parallel, and all arcs (not just "local" arcs) are potential pivot arcs. To execute pivots correctly in parallel, no more than one processor may update the data structures of a quasi-tree at any given time. To facilitate this exclusive access to quasi-trees, quasi-trees are locked with hardware locks immediately before executing a pivot involving the quasi-trees, and they are unlocked immediately after. From a locking standpoint, the quasi-tree allocation strategy of [Chang, et al, 87] is analogous to assigning *long-term* locks (thereby reducing communication costs in a loosely-coupled system) on node subsets to processors, whereas the arc allocation strategy utilizes *short-term* locks (in effect only for the time required to perform a single pivot) on the appropriate nodes corresponding to the quasi-trees incident to the selected pivot arc. Strategy 3, another *arc-allocation* strategy, allows parallel pricing of the arc subsets, but does pivoting serial (*avoiding the need for any locking*) and concurrently with pricing. Since the values of all duals must be available to all processors in order to permit the parallel pricing of all arcs (not just *local arcs*), the shared memory multiprocessor is an ideal if not a necessary architecture for the implementation of *arc allocation* algorithms.

Section 2.4 contains a brief summary of PGRNET, an implementation of Strategy 2 that works nicely when there are many quasi-trees in the optimal basis. PGRNET does parallel pricing to find pivot arcs, locks the quasi-trees at the ends of the pivot arcs to ensure exclusive access to quasi-trees, and executes pivots in parallel. The greatest disadvantage to PGRNET is that processors must temporarily lock one or two quasi-trees before executing a pivot involving those quasi-trees. If there are only a few quasi-trees in the basis, this means that only a few pivots can be executed in parallel. PGRNET rejects pivot-eligible arcs that are found to connect quasi-trees that are locked (i.e. currently in the process of being modified), so the work invested in finding these arcs is wasted. Quasi-trees are more likely to be locked if there are just a few of them. Pricing can always be done in parallel, regardless of the number of quasi-trees in a given basis, but pricing is more likely to be done with old dual values if there are few quasi-trees and they are changing in parallel. The use of stale dual information and the frequent loss of good pivot- eligible arcs can greatly reduce the efficiency of PGRNET if the optimal basis has only a few quasi-trees.

TPGRNET, an implementation of Strategy 3, will be described in more detail in Section 3.5, and results for implementations of PGRNET and TPGRNET on the Sequent Symmetry S81 will be compared in Chapter IV.

3.4 PGRNET (Parallel GRNET)

The flow chart for PGRNET in Figure 3.2 is very similar to the flow chart of Figure 3.1, because PGRNET is based on GRNET2. PGRNET, however, has parallel portions of code that are emphasized by parallel arrows. As in Figure 3.1, "l.t." designates *list_threshold*, a candidate list parameter.

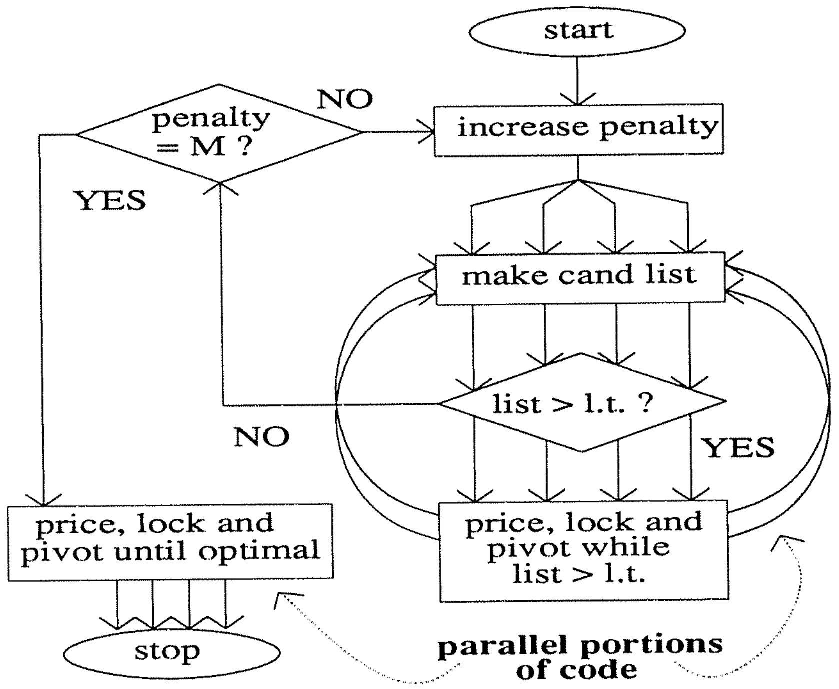
start
increase penalty
penalty = M ?
NO
YES
make cand list
list > l.t. ?
NO
YES
price, lock and pivot until optimal
price, lock and pivot while list > l.t.
stop
parallel portions of code

Figure 3.2 Flow Chart For Parallel Algorithm PGRNET

The (parallel) PGRNET algorithm can be summarized:

INITIALIZATION

In parallel, generate the initial flows on the artificial arcs. Divide the problem arcs into roughly equal-sized segments for pricing in the next stage.

STAGE 1 (*parallel pivoting with candidate lists*)

Asynchronously and in parallel scan the segments of the arc set to develop multiple candidate lists. Pivot arcs are chosen from the candidate lists, and quasi-trees are locked before pivots are made. When, for a particular segment of the arc set, it is not possible to develop a candidate list with more than *list_threshold* entries, check the penalty on the artificial arcs. If this penalty has reached its maximum value go to STAGE 2. Otherwise assign a new value to the penalty of the artificial arcs, and update the duals in parallel and continue asynchronous pivoting.

STAGE 2 (*parallel verification of optimality*)

Scan the segments of the arc list in parallel to locate pivot-eligible arcs. If a pivot-eligible arc is found, lock the associated quasi-trees, and execute the pivot (if the quasi-trees were successfully locked). If an entire sweep through the segments can be made without finding any pivot-eligible arcs, optimality has been reached.

Arcs are divided evenly between segments. If there are n arcs and P segments, then segment 1 has arcs (1) through (n/P), processor 2 gets arcs $(n/P + 1)$ through $(2n/P)$ and so forth. (A more sophisticated allocation of the non-artificial arcs might try to guess the topology of the optimal solution, and thereby assign arcs to specific partitions. If the optimal topology is known, lock contention can be reduced significantly by collecting in the same subset of the partition, all of the arcs that connect nodes in a given quasi-tree or group of quasi-trees. This idea could be used to solve perturbed problems efficiently. Given the optimal quasi-tree structure of some solved problem, subsets of the arc set could contain arcs that are local to certain collections of the optimal quasi-trees. A small perturbation of the data would hopefully change the optimal topology by very little, and therefore the initial arc allocation might improve the solution time significantly.) The dual variables of all the nodes, the predecessor threads, the successor threads and all other tree functions required by the generalized network simplex method are assumed to be stored in shared memory and are available to all processors. It is important to emphasize that only the acquisition of problem data (i.e. generating data or reading data) is done serial, and the solution process is entirely parallel. The number of partitions is equal to the number of processors, and all processors execute the same set of tasks. Each processor refreshes its own candidate list, selects pivot arcs from the list, locks quasi-trees to prevent corruption of tree structures, and executes pivots. We say that PGRNET is an example of "uniform parallelism" for this reason. The results below show that uniform parallelism is the best solution strategy for generalized network flow problems, as long as the number of quasi-trees in the optimal solution is not too small.

The program that each processor executes is almost identical to GRNET2. During a *parallel pivoting* stage, Stage 1, each processor makes its own candidate list of pivot- eligible arcs. The candidate lists are made in the same way that candidate lists are made in GRNET2. Each processor p choses its next pivot arc from its candidate list by selecting the pivot-eligible arc with the greatest reduced cost in absolute value. If the quasi-trees at the ends of the arc have not been locked by another processor, p locks the quasi-trees to keep other processors from interfering with the tree update, performs the pivot and removes the arc from the candidate list. If the quasi-trees are already locked, processor p removes the arc from the candidate list and chooses another arc. The dual update part of the pivot operation has also been parallelized and this parallelization is described below. When the candidate list belonging to p has not more than *list_threshold* arcs, p develops a new candidate list. If the new candidate list also has not more than *list_threshold* entries, processor p sets a flag in shared memory to indicate that it is having difficulty finding pivot-eligible arcs. This flag is checked frequently by all processors, and when it is set, processor 1 checks to see if the penalty on the artificial arcs is *big* M. If the penalty is *big* M, all processors enter Stage 2. If the penalty is smaller, then the processors increment the penalty on the artificial arcs and cooperate in recomputing the dual variables. Then all processors develop new candidate lists.

Stage 2 of PGRNET corresponds to the *verification of optimality* stage in GRNET2. The verification of optimality is done in parallel, and all processors execute the same tasks. Optimality is achieved by performing any remaining pivots. Processors sweep through their segments looking for pivot-eligible arcs, but no candidate lists are developed. If processor p finds a pivot-eligible arc, it locks the quasi-trees at either end of the arc, executes the pivot, and indicates to the other processors that they must restart their sweep (by setting flags in a shared array). This restart mechanism is needed because a pivot executed by processor p might cause an arc owned by another processor to become pivot-eligible. If processor p finds that one of the trees at the ends of a pivot-eligible arc is locked, it sets the other processors restart flags and restarts its own sweep. Each processor checks its restart flag frequently during Stage 2, and when a processor finds that its flag has been set, it marks the arc in its segment that was last priced out, and continues pricing. If the processor prices out all of its arcs up to the marked arc without finding any to be pivot-eligible and without finding its restart flag to be set, that processor informs the others that none of its arcs are pivot-eligible. Optimality is reached when all processors make a sweep through their arcs without finding their restart flags set, and without finding any arcs that are pivot-eligible.

The dual update is performed recursively and in parallel. A processor p that is updating the dual variables in a subtree S inspects a tree function $t(\cdot)$ that specifies how many nodes are in the subtree. It then sets a shared variable $size\_lim$ so that $size\_lim = t(root)/nproc$ where $nproc$ is the number of processors and $root$ is the root node of S. (The root node of S is the usual root node if S is a rooted tree. If S is not a rooted tree, then S has a unique cycle, and $root$ will be one of the nodes in the cycle.) Next, processor p traverses S and updates the dual variables in all subtrees of S that have fewer than some small number $tree\_threshold$ of nodes. The root nodes of the larger subtrees are put onto a shared queue. All processors that are computing reduced costs check frequently to see if there are any nodes on this queue. When a node appears on the queue, some processor q takes it off and checks to see if the associated subtree is smaller than *size_lim*. If so, then q updates all of the dual variables in that subtree. If not, then q traverses the subtree (without recomputing *size_lim*) and puts the root nodes of subtrees with *tree_threshold* or more nodes on the queue. The efficiency of this scheme relies on the subtrees of S being broad rather than deep. Ideally, p will chop S into *nproc* pieces, all having fewer than *size_lim* nodes. If this happens, then all processors will work in parallel updating the duals, and little CPU time will be spent putting root nodes onto the queue. If, however, the original subtree is badly skewed, then subtrees that are put on the queue may have to be chopped up by other processors. Despite some potentially inefficient aspects of the parallel dual update, computational experience has shown that it improves the overall efficiency of PGRNET by about 15% for certain problem classes. The parallel dual update is also used by TPGRNET, and the effects there are much more significant.

3.5 TPGRNET (Task-Parallel GRNET)

In this section, a class of problems is discussed that PGRNET handles poorly, and an algorithm, TPGRNET, is described that solves these problems more efficiently. The most significant factor in the efficiency of PGRNET is the number of quasi-trees in the optimal basis. If this number is small, the problem is large-grained (because quasi-trees tend to be large), and if this number is large, the problem is small-grained (because quasi-trees tend to be small). If a problem being solved by PGRNET has large granularity, it is more likely that a processor p will find the quasi-trees at the ends of a pivot arc to be locked. If they are locked, p must reject the arc and look for another in its candidate list. By the time the quasi-trees are freed, often all of the best arcs from the candidate list have been rejected and a pivot is executed on an arc with a relatively small reduced cost. This tends to increase the total number of pivots required to reach optimality. Candidate lists are exhausted quickly because many arcs that are chosen turn out to connect locked quasi-trees. This increases the total amount of time that is spent developing candidate lists. For these reasons, large-grained problems are solved inefficiently by PGRNET. (The performance of PGRNET is also degraded when nodes are distributed unevenly between the quasi-trees. A problem can be small-grained in the sense that it involves a large number of quasi-trees, but if most of the nodes belong to just one quasi-tree, that quasi-tree will be locked most of the time. In this case, the processors may compete for access to this one quasi-tree. Candidate lists are exhausted quickly because most pivot-eligible arcs have an end in this quasi-tree.)

Some alternatives to rejecting arcs with an end in a locked quasi-tree are to store them in a temporary stack, or simply to wait until the quasi-tree becomes available. Computational experience indicates that these alternatives are less efficient than the algorithm used by PGRNET, which simply discards these arcs.

A better approach to solving large-grained problems is to eliminate the need for the locking operation by parallelizing candidate list development and prioritization, but doing pivots serial (and concurrently with the development of candidate lists). Since different processors perform different tasks and execute different code, we say that TPGRNET is an example of "specialized parallelism." The algorithm is divided into two main stages. During Stage 1, the tasks are distributed as described above, and Stage 2 is exactly the same as Stage 2 of PGRNET. Figure 3.3 illustrates the flow of information during Stage 1, and figure 3.4 gives a flow chart for TPGRNET. (In both of these figures, dotted arrows indicate the direction of the flow of information.)

The (parallel) TPGRNET algorithm is:

INITIALIZATION

In parallel, generate the initial flows on the artificial arcs. Divide the problem arcs into roughly equal-sized segments for pricing during the next stage. (Same as the INITIALIZATION stage of PGRNET)

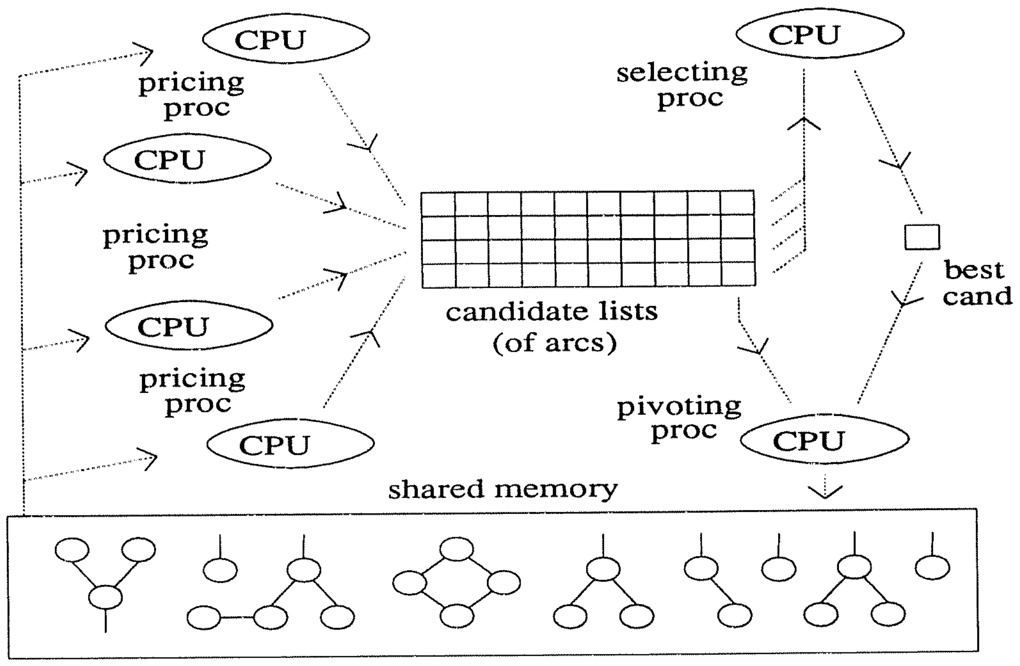

Figure 3.3 Flow of Information in TPGRNET, Stage 1

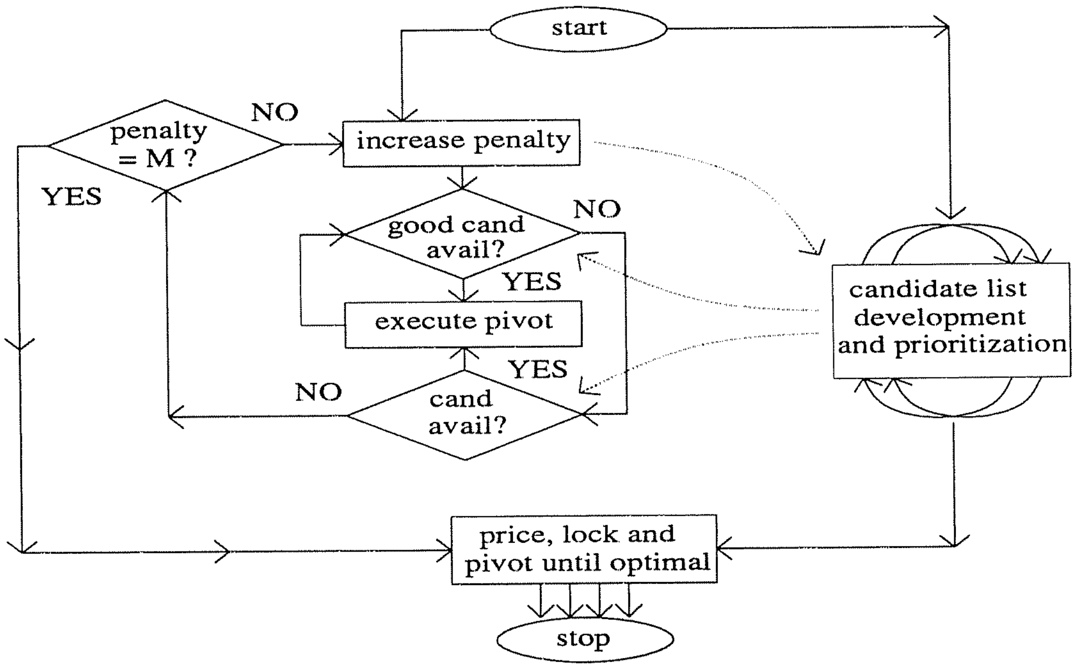

Figure 3.4 Flow Chart for Parallel Algorithm TPGRNET

STAGE 1

(*parallel candidate list development overlapped with serial pivoting*)

A set of candidate lists is developed and prioritized in parallel. This process is continued during the pivot, which concurrently modifies some of the duals being used in candidate list development. When the pivot is completed, the next arc to enter the basis is selected by using the "best" arc from the candidate list (if this arc has a sufficiently good reduced cost) or a different arc if this is not possible. The latter case occurs very infrequently, and under conditions to be described below, may trigger an increase in the penalty cost or an exit to stage 2.

STAGE 2 (*parallel verification of optimality*)

Scan the segments of the arc list in parallel to locate pivot-eligible arcs. If a pivot-eligible arc is found, lock the associated quasi-trees, and execute the pivot (if the quasi-trees were successfully locked). If an entire sweep through the segments can be made without finding any pivot-eligible arcs, optimality has been reached. (Same as STAGE 2 of PGRNET)

The remainder of this section will give a detailed description of the tasks performed by the individual processors during Stage 1 of TPGRNET. The pricing processors have the task of computing reduced costs and storing pivot-eligible arcs (in candidate lists of length 10 stored in the shared memory). When processor p finds a pivot-eligible arc, it recomputes the reduced cost of the first element in its candidate list to see if the new arc has a larger reduced cost in absolute value. If it does, the new arc number gets stored in the first element of the array, and the previous entry is overwritten. Experience has shown that saving the previous entry yields no improvement in efficiency. If the new arc has a smaller reduced cost than the first arc in the candidate list, the new arc gets stored at a random location in the list. The pricing processors stay in a loop that includes three operations. First, there is the pricing operation. This uses most of the processor's CPU time. Second, there is a check to see if the pivoting processor has put a subtree on the dual-update queue (because of space limits this is not shown in the figures). Third, there is a check to see if Stage 1 of the algorithm has finished.

The pivot selecting processor has the task of scanning the collection of candidate lists of the pricing processors to locate the pivot-eligible arc with the largest reduced cost and storing that arc in a single shared variable called *best_cand*. This processor stays in a loop that has three operations. First, the processor checks to see if *best_cand* is empty. If so, the processor looks in the first entry of some candidate list to find an arc to put in *best_cand*. Second, the processor traverses the candidate lists to see if there is a pivot-eligible arc that has a reduced cost larger than the arc in *best_cand*. Third, there is a check to see if Stage 1 of the algorithm has finished.

The pivoting processor stays in a loop in which it selects pivot arcs for itself (as described below), executes pivots, and directs the increases in the penalty on the artificial arcs. Whenever possible, the pivoting processor selects its pivot arc from *best_cand*, but before adopting an arc from *best_cand*, a check is made to see that the arc is pivot-eligible and to see if the reduced cost is sufficiently large in absolute value. If the arc in *best_cand* has a small reduced cost, or if there is no arc in *best_cand*, the pivoting processor looks at the first entry of each of the candidate lists to find a pivot-eligible arc. If a pivot-eligible arc is found, the pivot is executed. If no pivot-eligible arc is found, then either the penalty on the artificial arcs is increased, or Stage 2 is begun (if the penalty has reached *big M* and cannot be increased). The pivoting processor has the task of directing the parallel update of dual variables during the execution of pivots and after the penalty on the artificial variables has been increased. During both of these operations, the pivoting processor can put the root node of subtrees onto the dual update queue, and the pricing processors will then adopt the task of updating the duals on those subtrees.

3.6 HYGRNET (Hybrid GRNET)

The hybrid algorithm begins its computation with PGRNET. All processors execute the same code, and all processors own a segment of the arc list. When a processor p selects an arc from its candidate list and is unable to lock the associated quasi-trees, it increments an entry, corresponding to p, of an array, *lost_arcs*, in shared memory. It then checks the entries corresponding to the other processors. If the value of the entry for all processors is 48 or more then processor p sets a shared flag, indicating that the threshold for quasi-tree contention has been reached, and the program will convert to TPGRNET after the next step is made in the gradual penalty method. Each time that a processor creates a new candidate list, it divides its entry in *lost_arcs* by 2.

The (parallel) HYGRNET algorithm is:

INITIALIZATION

All processors cooperate in generating the initial flows on the artificial arcs. Then all processors go to STAGE 1P.

STAGE 1P (*parallel pivoting with candidate lists*)

Processors scan their segment of the arc list to develop candidate lists. Pivot arcs are chosen from the candidate lists, and quasi-trees are locked before pivots are made. When a processor is unable to lock the quasi-trees associated with a selected arc, it increments its entry in the shared array *lost_arcs*. If the entries for all processors in *lost_arcs* are greater than or equal to 48, the processors will begin STAGE 1T at the start of the next step in the GPM. When any processor cannot develop a candidate list with more than (*listsize*/2) entries, all processors either cooperate with processor 1 to increase the penalty on the artificial arcs, or they all proceed to STAGE 2.

STAGE 1T (*parallel candidate list development with serial pivoting*)

The pricing processors sweep through their lists of arcs and store pivot-eligible arcs in candidate lists. The pivot selecting processor scans the candidate lists of the pricing processors and puts the best pivot-eligible arc in *best_cand*. The pivoting processor selects the next pivot arc from the shared variable *best_cand* whenever *best_cand* is not empty and the arc stored in *best_cand* is pivot-eligible and has a reduced cost greater than 1.0 in absolute value. Otherwise, the pivoting processor attempts to obtain the next pivot arc from the first entry of the candidate lists of the pricing processors. If the pivoting processor cannot find a pivot-eligible arc in *best_cand* and cannot find a pivot-eligible arc in the first entry of the candidate lists, it either oversees the increase in the penalty on the artificial arcs, or else directs the other processors to go to STAGE 2.

STAGE 2 (*parallel verification of optimality*)

Processors scan their segment of the arc list simultaneously to locate any remaining pivot-eligible arcs. If a processor finds a pivot-eligible arc, it locks the trees involved, performs the pivot and restarts the sweep of the other processors. If all processors make a sweep through their lists without being interrupted and without finding any pivot-eligible arcs, optimality has been reached.

3.7 Comparison With Other Work

The first parallel variant of GRNET was developed on the U.W. CRYSTAL Multicomputer. The code is referred to as GRNET-K in [Chang, et al, 87]. This code is executed on K processors, and uses the "node partitioning" strategy described above. Processors (1) through (K-1) solve a *local* problem by pivoting on pivot-eligible arcs that are incident to *local* nodes. Processor (K) computes reduced costs (using possibly old dual variables) on *cross arcs* and develops a candidate list. After all of processors (1) through (K-1) have solved their local problems to optimality, processor (K) directs a redistribution of quasi-trees so that some of the pivot-eligible *cross arcs* become *local* arcs. This approach yields good results for multi-period problems. Speedups over GRNET reported in [Chang, et al, 87], range from 8.25 to 14.17 on 13 processors. Speedups for problems without the block diagonal structure range from 1.52 to 1.57 on 6 processors.

NETPAR is a parallel network simplex code for the pure network flow problem discussed in [Peters 1988a]. The code uses a task partitioning algorithm in which one processor executes all pivots, and all other processors compute reduced costs and store pivot-eligible arcs in a shared queue. This scheme is modified in [Peters 1988b] to give to one processor the task of executing pivots, to another processor the task of selecting arcs, and to all other processors the task of computing reduced costs. The modified code, PARNET, compares favorably with NETFLO [Kennington and Helgason, 1980], a standard serial code for the regular network flow problem. PARNET, when run on three processors, is 10 to

20 times faster than NETFLO for the standard collection of problems generated by NETGEN [Klingman, et al, 74]. PARNET yields linear speedups for a variety of large scale problems generated by NETGEN.

PARNET and TPGRNET have a similar distribution of tasks. Both programs have one pivoting processor, one pivot selecting processor, and both programs give the task of pricing (i.e. computing reduced costs) to the other processors. However, the two programs have slightly different pivot selection strategies. Under the PARNET strategy, the pricing processors develop queues of pivot-eligible arcs. When a pricing processor finds a pivot-eligible arc, it puts it at the head of the queue. The pivot selecting processor looks at the first three elements in a queue when selecting an arc for the pivoting processor. Under the TPGRNET strategy, a pricing processor compares the reduced costs of a newly found pivot- eligible arc with the reduced cost of the first arc on its candidate list. If the new arc has a larger reduced cost it is put at the beginning of the list, and the arc that was previously at the beginning of the list is overwritten. Otherwise, the newly found arc is stored at a random location elsewhere in the list. This strategy has been found to be somewhat faster, for generalized network flow problems, than the PARNET strategy.

Since the distribution of tasks is very similar in PARNET and TPGRNET, and since PARNET yields a linear speedup for almost all test problems, one would hope that the speedups from TPGRNET would also be linear. The results given in section 3 show that TPGRNET does not yield linear speedup for all test problems. The difference in the performance of the two algorithms is possibly due to differences in the nature of the two algorithms. PARNET does the ratio test and the flow update with integer operations, while TPGRNET does these operations in double precision. This means that pivots are, on the average, more expensive for TPGRNET than they are for PARNET. PARNET computes reduced costs with integer arithmetic while TPGRNET computes reduced costs in double precision. Experience has shown that the dual variables in large grained generalized network problems change rather frequently. A change in the dual variables after the execution of a pivot can make the arcs stored in a candidate list no longer pivot-eligible, and the CPU time that was spent finding the arcs is wasted. It's possible that TPGRNET spends more time doing pricing with incorrect duals than PARNET, although further studies are needed to verify this possibility.

GENFLO [Muthukrishnan 88] is a parallel generalized network program written for the Sequent Balance 21000 and the Sequent Symmetry S81. GENFLO does not scale to force one element 1 in each column. For pure network problems, a serial version of GENFLO is competitive with NETFLO [Kennington and Helgason 80] and GENNET [Brown and McBride 84], and it is competitive with GENNET for generalized network flow problems. The parallel version of GENFLO is similar to PGRNET in its distribution of tasks. It was tested using a group of generalized network problems generated by GNETGEN, a modification of NETGEN [Klingman, et al, 1974]. GENFLO speedups ranged from about 2 to 3 (on 8 processors on a Sequent Symmetry) for these problems. A forthcoming joint paper [Clark, et al, 89] will provide a comparison of the performances of these approaches on the GNETGEN test problems as well as the MAGEN test problems described below.

IV Computational Results

4.1 Code Parameters for GRNET2, PGRNET and TPGRNET

GRNET2 and PGRNET generate and maintain candidate lists of pivot-eligible arcs during Stage 1. Both GRNET2 and PGRNET maintain *num_pricers* candidate lists of length *listsize* = 60. For GRNET2, *num_pricers* = 19 for transportation problems, and *num_pricers* = 5 for transshipment problems. For PGRNET, *num_pricers*= *nproc*, the number of processors, regardless of the nature of the problem being solved. Appendix 4 gives a detailed description of the technique used by GRNET2 and PGRNET to develop and maintain candidate lists, and it discusses the code parameter *list_threshold* , a threshold parameter that indicates when a candidate list should be purged. For both GRNET2 and PGRNET, *list_threshold*= (*listsize*/2). For TPGRNET, the number of candidate lists is specified by the algorithm to be $nproc-2$. The lists have length 10. Both PGRNET and TPGRNET use the same value for *tree_threshold* (used in parallelizing the dual update), namely 5. All three codes increment the cost on the artificial arcs in the same way. Assuming that the arc costs are in the range [1,100], the initial penalty on the artificial arcs is set at 20, and the initial increment between penalties is 5. Later, the increment increases by 10 and then by 20. The last increment between penalties changes the penalty from 200 to *big* M, and *big* M in our implementation is 9,999,999.

4.2 MAGEN (Massive Generator)

The test problems in groups 1 through 4 are generated randomly with MAGEN, a generator similar to GTGEN [Chang and Engquist 1986]. The generator has the feature that the user may specify, roughly, the number of quasi-trees in the optimal basis. This is accomplished with a technique discussed in [Chang and Engquist 1986]. A processor first selects a source node and, in two steps, generates all of the arcs incident to that node. In the first step, the processor generates all of the arcs that will begin at that node and end at one of the sink nodes. As these arcs are generated, the divergences at the source and sink nodes of the arcs are adjusted so that a feasible flow exists. In the second step, the processor determines whether or not a generalized root arc will be generated for the source node. To do this, it checks a randomly generated number, and if this number is smaller than a user specified parameter $\alpha \leq 1$, two things are done. First, a generalized root arc for the source node is generated. Second, the divergence at the node is adjusted so that it equals the sum of the capacities of the outgoing arcs. This tends to force the generalized root arc into the optimal basis, creating a quasi-tree. The number of quasi-trees in the optimal basis will be roughly $\alpha \cdot (num\_sources)$. By adjusting α, one can specify, approximately, the number of quasi-trees in the optimal basis. The random seed used for all runs is 0731246890, and the remaining input data for the Group 1 problems is given in Table 4.1. The row labeled "Arcs per node" gives a range for the number of arcs that will eminate from a given source node. The input data for groups 2 through 4 is the same, but the number of arcs per source node for Group 2 is 39-45, and for Group 3 is 80-86. The input data for Group 4 is given in Table 4.2. The input parameter *zfrac* roughly indicates the percent of arcs that will have flow at upper bound in an optimal solution. The optimal objective function values for groups 1 through 4 are given in Appendix 3.

	Problems					
Character.	**1.00**	**1.01**	**1.03**	**1.05**	**1.10**	**1.50**
Nodes	10,000	10,000	10,000	10,000	10,000	10,000
Sources	5,000	5,000	5,000	5,000	5,000	5,000
Sinks	5,000	5,000	5,000	5,000	5,000	5,000
Arcs per node	18-24	18-24	18-24	18-24	18-24	18-24
Cost range	1-100	1-100	1-100	1-100	1-100	1-100
Mult Range	.90-.98	.90-.98	.90-.98	.90-.98	.90-.98	.90-.98
Cap max	1,000	1,000	1,000	1,000	1,000	1,000
zfrac	0.50	0.50	0.50	0.50	0.50	0.50
α	0.00	0.01	0.03	0.05	0.10	0.50

Table 4.1 Input data for Group 1

	Problems					
Character.	**4.00**	**4.01**	**4.03**	**4.05**	**4.10**	**4.50**
Nodes	30,000	30,000	30,000	30,000	30,000	30,000
Sources	15,000	15,000	15,000	15,000	15,000	15,000
Sinks	15,000	15,000	15,000	15,000	15,000	15,000
Arcs per node	18-24	18-24	18-24	18-24	18-24	18-24
Cost range	1-100	1-100	1-100	1-100	1-100	1-100
Mult Range	.90-.98	.90-.98	.90-.98	.90-.98	.90-.98	.90-.98
Cap max	1,000	1,000	1,000	1,000	1,000	1,000
zfrac	0.50	0.50	0.50	0.50	0.50	0.50
α	0.00	0.01	0.03	0.05	0.10	0.50

Table 4.2 Input data for Group 4

4.3 Definition of Speedup and the Target Computer

The parallelism of the algorithms discussed here is measured by calculating the speedup for various problems. For a given parallel code, the speedup for P processors is defined in the following way:

$$\text{speedup} = \frac{\text{CPU time required by GRNET2}}{\text{CPU time required by } P \text{ processors}}$$

The parallel results below are taken from a Sequent Symmetry S81, a shared memory multiprocessor. The machine used in this research is configured with 20 Intel 80386 processors, each with a Weitek 1167 floating-point accelerator. It has a 40 Mbyte physical memory, and each processor has a 16 kbyte cache. The time sharing system allows the user to request up to 19 processors for the execution of a program. Sequent provides a parallel programming library that includes commands for forking processes, locking shared variables and killing processes. The DYNIX operating system on the Sequent Symmetry S81 provides the user time and the system time used in the execution of a program. The system time reported can depend on the number of users on the system. For this reason, we report only the user time as the CPU time.

4.4 Problem Organization and Problem Sizes

The MAGEN [Clark and Meyer 89] problems are divided into four groups. Groups 1 through 3 have 10,000 nodes. The problems in Group 1 have about 100,000 arcs, the problems in Group 2 have about 200,000 arcs, and the problems in Group 3 have about 400,000 arcs. The problems in Group 4 have 30,000 nodes and more than 300,000 arcs. All problems were solved by both PGRNET and

TPGRNET, and each group contains problems with varying numbers (ranging from 1 to 7376) of quasi-trees in the optimal basis. The NETGEN test problems consist of three groups. The Group 5 problems are bipartite and non-bipartite problems taken from the (new) standard problem set. These problems have 5,000 nodes and about 25,000 arcs. Group 6 is a group of bipartite problems with 50,000 nodes and 250,000 arcs. The problems in groups 5 and 6 are solved by TPGRNET, and speedups range from 3.4 to 8.8 on 15 processors. Group 7 is taken from the standard set [Kingman, et al, 74] of test problems. These bipartite and non-bipartite problems are solved by a number of different codes. The GNETGEN problems consist of three groups. Group 8 is a set of relatively small test problems, and serial results for three codes are given for this group. Group 9 has three sets of problems, and each problem within a set has a different percent of capacitation. All of the problems in Group 9 have roughly the same ratio of nodes to arcs. Group 10 has two sets of problems, again with varying percentages of capacitated arcs. The results of Group 10 are intended to indicate the efficiency of an algorithm as the ratio of nodes to arcs changes. The GNETGEN problems are solved by TPGRNET, and speedups range from 2.6 to 5.9 on 15 processors. The final group of test problems, Group 11, was generated by MAGEN. These problems have 30,00 nodes and more than 1 million arcs. Speedups for PGRNET for these problems are as high as 11.0.

4.5 MAGEN Test Problems

Figure 4.3 shows a collection of speedup graphs for the Group 1 problems listed in Table 4.3. All problems have 10,000 nodes and more than 100,000 arcs. Each column in the table corresponds to a different problem, and the problem number is given at the top of the column. The last two digits of the problem number are $\alpha \times 100$, where α is the quasi-tree parameter for the generator described above. Since α is different for each of these problems, each problem has a different number of quasi-trees in its optimal basis. The graphs show speedup as a function of the number of processors for the algorithm PGRNET. According to the table, problem 1.50 is the one with the greatest number of quasi-trees in this group of problems, and both the table and the graph show that this is the problem for which PGRNET yields the best speedup (even though the serial time for this problem is the smallest of the group). The great number of pivots and CPU time required to solve problem 1.00 and the fact that PGRNET yields a maximum speedup of only 3.2 for this problem indicate the need to develop an alternate algorithm (like TPGRNET) for large grained problems. However, the speedups for all other problems are greater than 5.3 (on 19 processors). This means that PGRNET yields a very substantial reduction in wall clock time for a fairly wide range of problems. Also notice that, except for problem 1.00, the graphs in Figure 4.3 have a slope indicating that the speedup has not reached its maximum at 19 processors. This means that the wall clock time might be reduced further by increasing the number of processors beyond 19.

Using 19 processors, PGRNET required more pivots than GRNET2 to solve each of the problems in Group 1. This may be due to the fact that some of the best pivot arcs from candidate lists are rejected, since an arc is removed from a candidate list in PGRNET whenever it is found to have an end in a locked quasi-tree. This conjecture is supported by the fact that the difference in pivots required diminishes as the number of quasi-trees increases, and by the diminished number of pivots required by TPGRNET, to be discussed below. The result is that PGRNET makes poorer choices for pivot arcs.

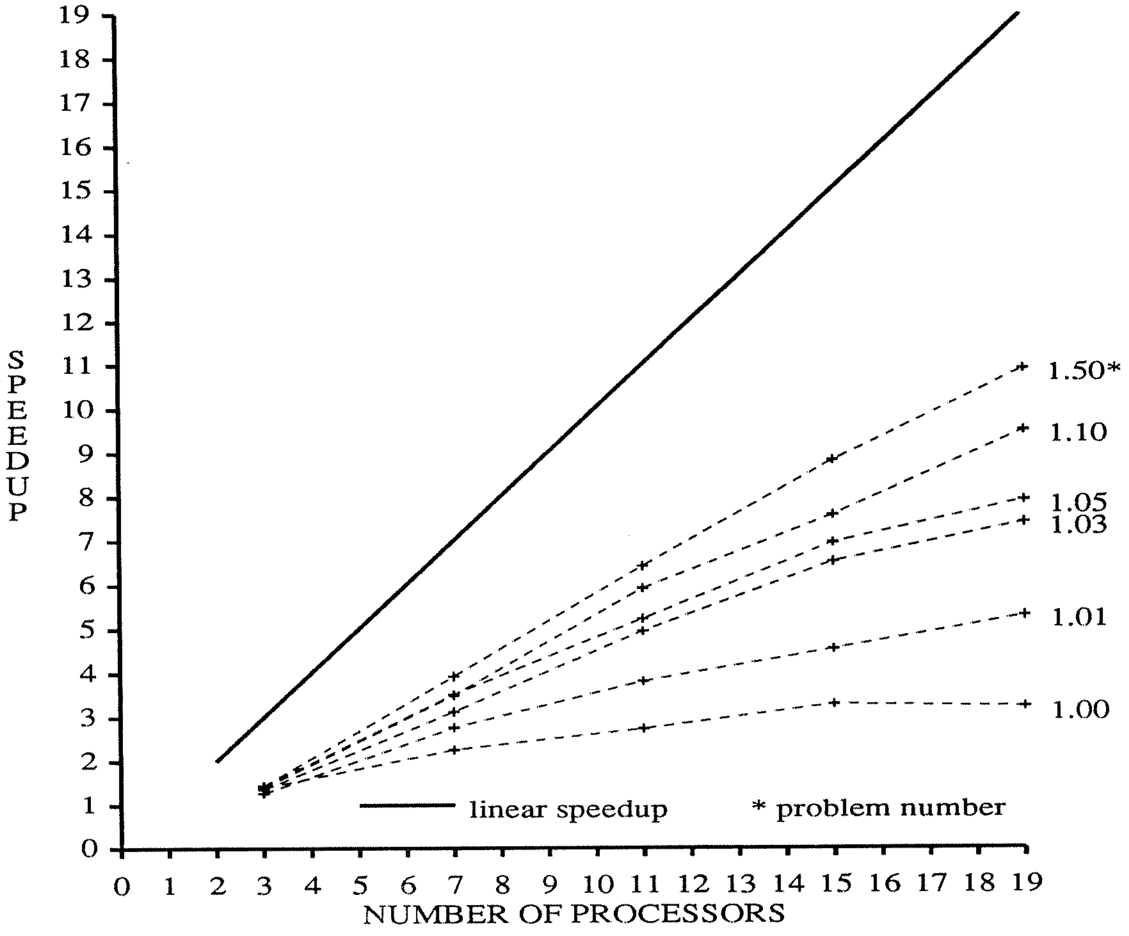

Figure 4.3 Speedups for PGRNET

Problem #	1.00	1.01	1.03	1.05	1.10	1.50
# qtrees	1	43	147	258	494	2,467
# nodes	10,000	10,000	10,000	10,000	10,000	10,000
# arcs	107,290	107,333	107,437	107,548	107,784	109,757
# pivots seq	101,630	108,359	101,717	97,875	92,407	75,458
# pivots 19 pr	111,843	115,680	106,496	103,064	96,075	76,367
CPU secs. seq	2,796	1,974	1,076	833	638	404
CPU secs. 19 pr	864	372	145	105	67	37
Speedup 19 procs	3.2	5.3	7.4	7.9	9.5	10.9

Table 4.3 Results for PGRNET for Group 1

Figure 4.4 and Table 4.4 give results for TPGRNET for the Group 1 problems. Note that TPGRNET solves all problems with fewer pivots than both GRNET2 and PGRNET, and uses substantially fewer pivots to solve the large grained problems. Profiles of TPGRNET and GRNET2 have indicated that TPGRNET uses up to 68% of its CPU time doing pricing (i.e. computing reduced costs), while GRNET2 spends only 12% of its CPU time on pricing. The heavy emphasis that TPGRNET puts on pricing apparently results in better choices for pivot arcs and ultimately results in a smaller number of pivots needed for solution. PGRNET also does a great deal of pricing relative to GRNET2. PGRNET can spend up to 59% of its CPU time doing pricing, but this is a smaller percentage than for TPGRNET. Note that GRNET2 solves problem 1.00 in 2796 seconds and problem 1.50 in 404 seconds. This means that the solution time for problem 1.00 is about 6 times greater than the solution time for problem 1.50. However, the number of pivots needed for problem 1.00 is only about 4/3 times the number of pivots needed for problem 1.50. The difference in CPU time relative to the number of pivots is explained by the fact that the quasi-trees developed during the solution of problem 1.00 are larger than the quasi-trees for 1.50. Therefore, the pivots used to solve problem 1.00 are, on average, much more expensive than the pivots used to solve problem 1.50. Looking at the speedup graphs for these problems, one sees that the behavior of TPGRNET is fairly uniform for these problems. The speedup on 19 processors for TPGRNET ranges from a minimum of 3.5 to a maximum of 6.2, while the speedup for PGRNET ranges from a minimum of 3.2 to a maximum of 10.9. The shape of the speedup graphs for the problems in this group indicates that the maximum speedup for TPGRNET was reached when TPGRNET was run on about 15 processors.

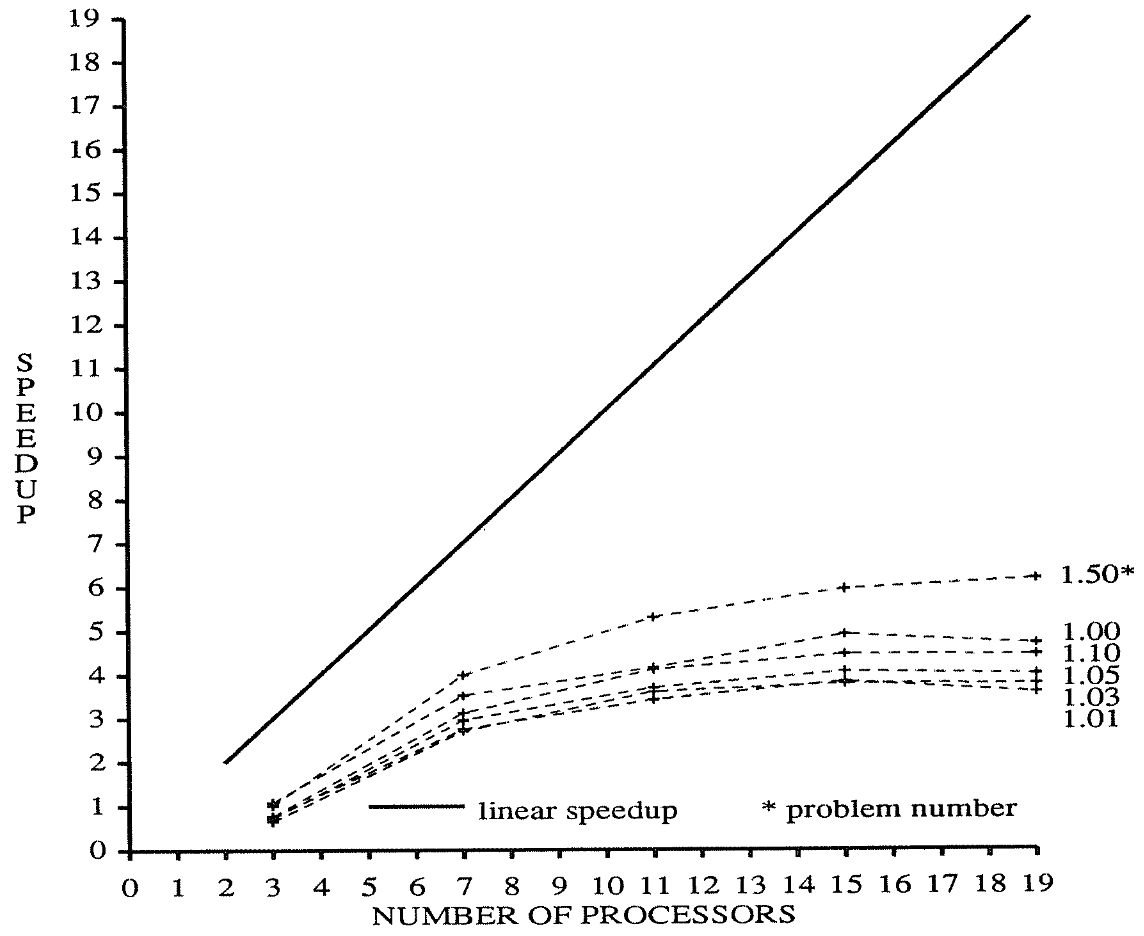

Figure 4.4 Speedups for TPGRNET

Problem #	1.00	1.01	1.03	1.05	1.10	1.50
# qtrees	1	43	147	258	494	2,467
# nodes	10,000	10,000	10,000	10,000	10,000	10,000
# arcs	107,290	107,333	107,437	107,548	107,784	109,757
# pivots seq	101,630	108,359	101,717	97,875	92,407	75,458
# pivots 19 pr	88,144	95,798	92,026	90,292	86,534	73,775
CPU secs. seq	2,796	1,974	1,076	833	638	404
CPU secs. 19 pr	595	551	284	208	143	65
Speedup 19 procs	4.6	3.5	3.7	4.0	4.4	6.2

Table 4.4 Results for TPGRNET for Group 1

Figures 4.5 - 4.6 and tables 4.5 - 4.6 present results for PGRNET and TPGRNET for Group 2, a set of problems having 10,000 nodes and more than 200,000 arcs and for Group 3, where the number of arcs is increased to more than 400,000. The speedup for PGRNET and TPGRNET improves slightly when the number of arcs is increased, but otherwise the results are similar to those of Group 1, except that for TPGRNET for Group 3 the maximum speedup doesn't seem to have been reached when 19 processors are used. This is due to the fact that the Group 3 problems have many more arcs than the problems in Groups 1 and 2, relative to the number of nodes. So more processors are "needed" to do pricing. (We note that each pricing processor in TPGRNET can each compute reduced costs for only about 9 arcs between pivots at the beginning of the run. This happens because the quasi-trees in the starting basis consist of just one node and one generalized root-arc, and therefore pivots are executed by the pivoting processor very quickly. Assuming that there are 17 pricing processors, the total number of arcs priced out between pivots in TPGRNET is initially only about 153, regardless of the number of arcs in the problem. This represents a very small fraction of all of the arcs, especially for the problems in Group 3. For small grained problems the pricing processors can compute only about 10 reduced costs between pivots at the end of the run. For the large grained problems, the pricing processors can each compute more reduced costs between pivots, but only rarely are 17 processors able to price out as many as 10% of all the arcs between pivots.)

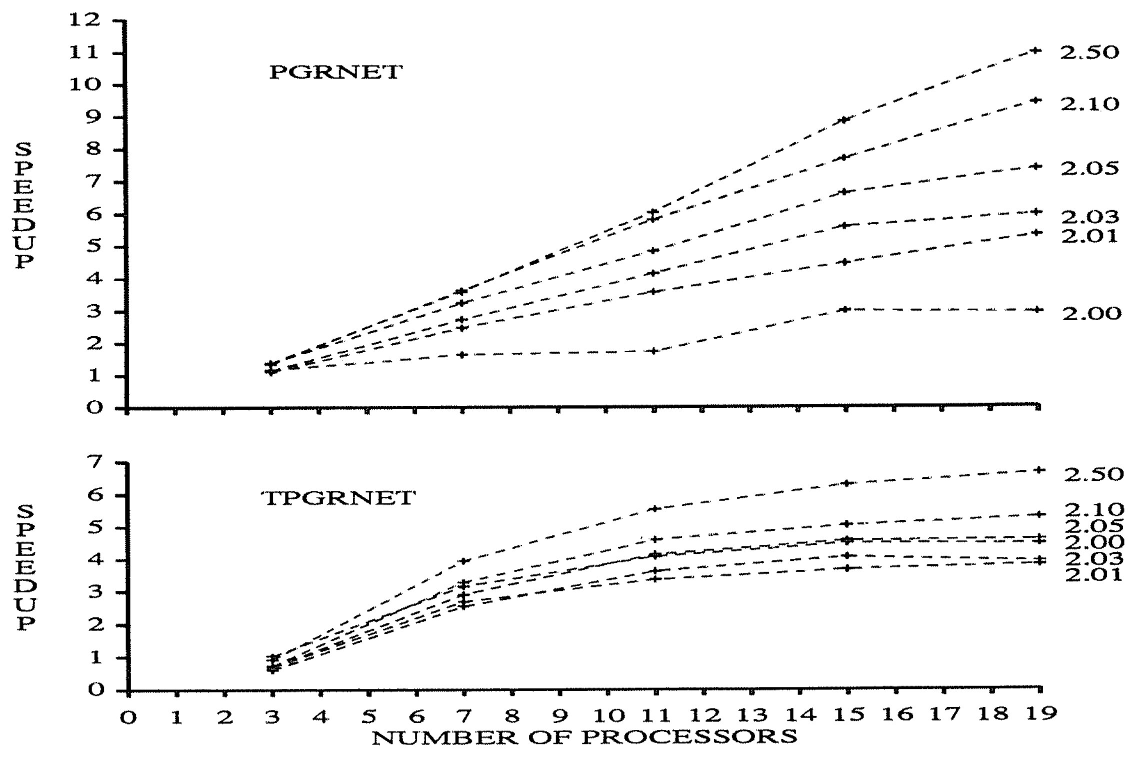

Figure 4.5 Speedups for PGRNET and TPGRNET

Problem #	2.00	2.01	2.03	2.05	2.10	2.50
# qtrees	1	46	133	233	508	2,490
# nodes	10,000	10,000	10,000	10,000	10,000	10,000
# arcs	212,627	212,673	212,760	212,860	213,135	215,117
# pivots seq	185,105	195,430	184,746	180,115	166,392	138,635
pivs PGRNET	205,149	212,594	199,859	190,724	174,818	140,260
pivs TPGRNET	156,885	169,054	165,166	160,899	153,504	134,183
CPU secs. seq	5,003	3,470	2,007	1,703	1,201	765
CPU PGRNET	1,700	652	336	230	127	69
CPU TPGRNET	1,118	909	512	371	227	115
Spdp PGRNET	2.9	5.3	5.9	7.4	9.4	11.0
Spdp TPGRNET	4.4	3.8	3.9	4.5	5.2	6.6

Table 4.5 Results for PGRNET and TPGRNET for Group 2

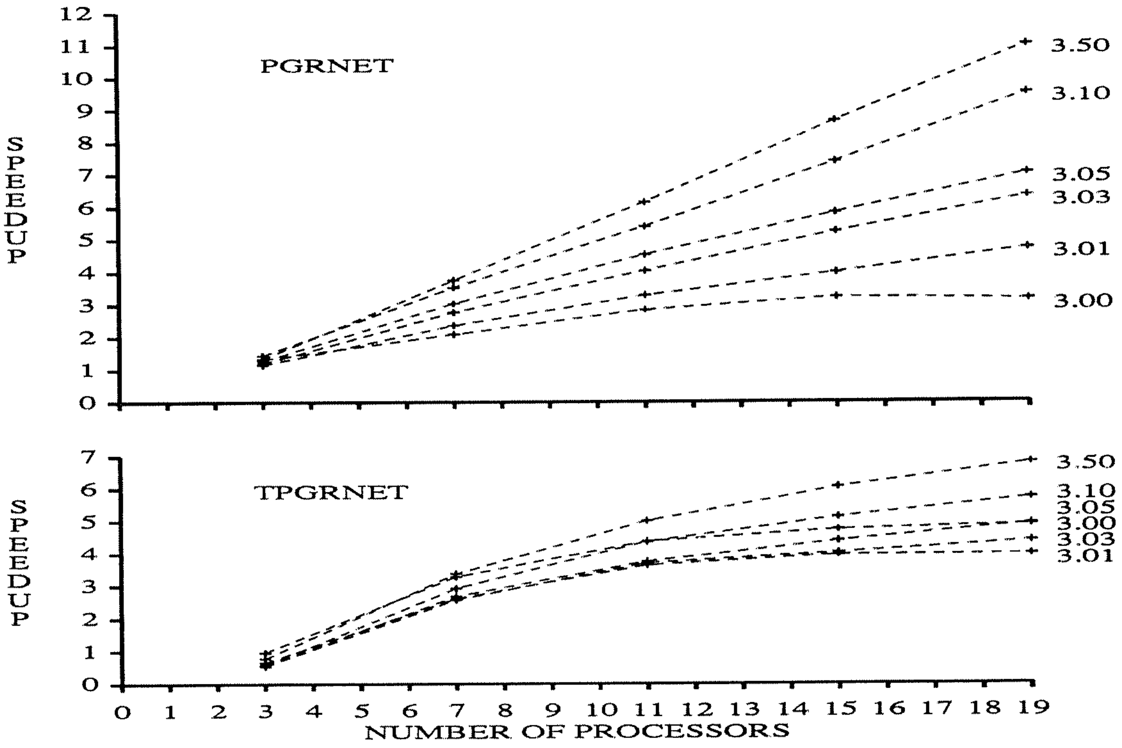

Figure 4.6 Speedups for PGRNET and TPGRNET

Problem #	3.00	3.01	3.03	3.05	3.10	3.50
# qtrees	1	41	142	235	487	2,560
# nodes	10,000	10,000	10,000	10,000	10,000	10,000
# arcs	417,397	417,438	417,539	417,632	417,884	419,957
# pivots seq	340,153	373,375	347,419	333,389	312,852	260,266
pivs PGRNET	378,806	415,128	379,045	362,048	330,367	264,381
pivs TPGRNET	283,695	314,301	304,675	295,536	284,233	250,043
CPU secs. seq	9,124	7,814	4,337	3,444	2,578	1,531
CPU PGRNET	2,877	1,649	682	488	270	138
CPU TPGRNET	1,867	1,966	989	700	450	225
Spdp PGRNET	3.1	7.7	6.3	7.0	9.5	11.0
Spdp TPGRNET	4.8	3.9	4.3	4.9	5.7	6.8

Table 4.6 Results for PGRNET and TPGRNET for Group 3

Figure 4.7 and Table 4.7 present results for the two parallel algorithms for Group 4, a set of problems with 30,000 nodes and about 320,000 arcs. The ratio of the number of arcs to the number of nodes is roughly the same as for Group 1, but the speedups for Group 4 are generally higher. This is probably due to the greater number of quasi-trees in the optimal bases of the Group 4 problems. PGRNET gives a speedup of more than 6 for most problems. PGRNET outperforms TPGRNET in terms of CPU time for all but problem 4.00. Both PGRNET and TPGRNET do pricing in parallel, but only PGRNET does pivoting in parallel. This parallel pivoting aspect of PGRNET must be giving it the superior timing results, because TPGRNET does fewer pivots than PGRNET for all problems. PGRNET yields an impressive speedup of 11.1 over GRNET2 for 4.50, because the quasi-trees are numerous in the optimal basis (and in the intermediate bases). This speedup might improve even further if problem 4.50 were made larger. For problem 4.00, TPGRNET gives a speedup of 4.8 on 19 processors. Since the speedup for PGRNET for 4.00 is only 3.8, this suggests that TPGRNET solves extremely large-grained problems more efficiently than PGRNET.

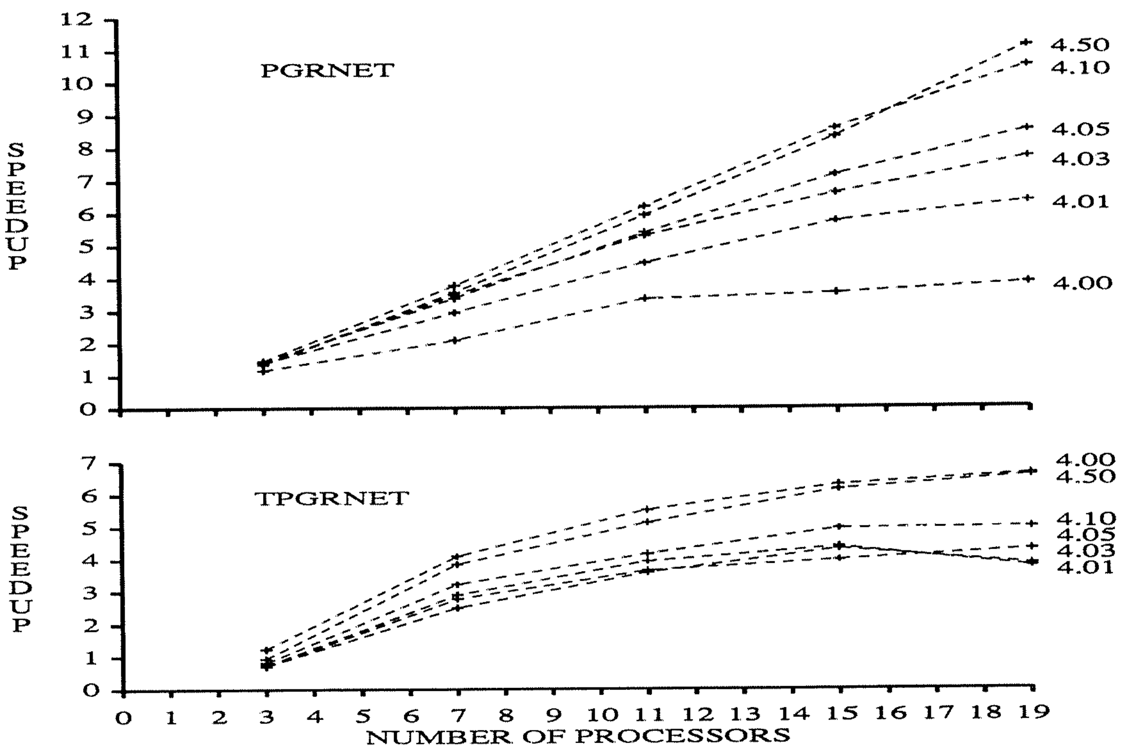

Figure 4.7 Speedups for PGRNET and TPGRNET

Problem #	4.00	4.01	4.03	4.05	4.10	4.50
# qtrees	1	139	459	776	1,490	7,376
# nodes	30,000	30,000	30,000	30,000	30,000	30,000
# arcs	322,289	322,428	322,748	323,065	323,779	329,665
# pivots seq	328,711	347,420	320,937	308,284	288,856	228,819
pivs PGRNET	365,633	383,483	345,325	326,323	300,496	231,507
pivs TPGRNET	265,774	305,979	288,279	272,322	263,411	221,544
CPU secs. seq	22,434	9,679	4,525	3,096	2,340	1,388
CPU PGRNET	5,829	1,527	589	364	223	124
CPU TPGRNET	3,390	2,571	1,177	721	470	211
Spdp PGRNET	3.8	6.3	7.6	5.2	10.4	11.1
Spdp TPGRNET	6.6	3.7	3.8	4.2	4.9	6.5

Table 4.7 Results for PGRNET and TPGRNET for Group 4

Figure 4.8 and Table 4.8 give results for the hybrid algorithm HYGRNET for Group 4. HYGRNET has been tuned to make the most of the two component algorithms PGRNET and TPGRNET for the MAGEN problems. Since PGRNET gives better performance than TPGRNET for problems 4.03 through 4.50, TPGRNET is never invoked during the solution of these problems. The results given in Table 4.8 for these problems are slightly superior to the results for the corresponding problems in Table 4.7. This is explained by a slightly improved pricing strategy that was added to the PGRNET component of HYGRNET, which puts an upperbound (of 8,000) on the number of arcs that may be priced in the process of creating a candidate list. Problems 4.00 and 4.01 are the only problems for which TPGRNET is invoked, and fewer than 2% of all pivots for these problems are executed by TPGRNET. The speedup graph for problem 4.01 shows a loss in efficiency at 15 processors (relative to 11 and 19 processors). This is caused by HYGRNET changing its strategy from PGRNET to TPGRNET too early, and it shows a need for more experimentation with the threshold mechanism. The speedups (for HYGRNET on 19 processors) for problems 4.00 and 4.01 are 6.1 and 7.6, which is a considerable improvement over the results for PGRNET for these problems. The results show that invoking TPGRNET at the end of the solution process can make a substantial improvement in runtime.

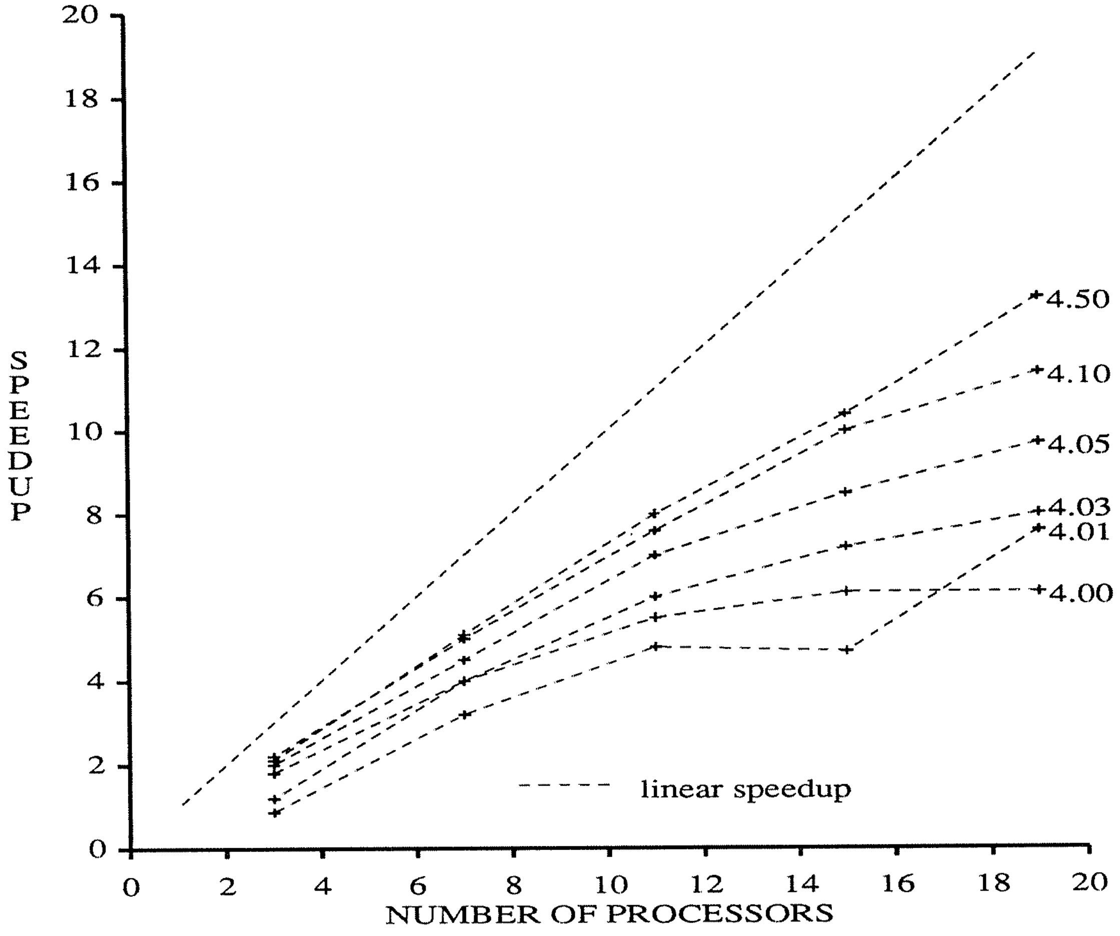

Figure 4.8 Speedups for HYGRNET

Problem #	4.00	4.01	4.03	4.05	4.10	4.50
# qtrees	1	139	459	776	1,490	7,376
# nodes	30,000	30,000	30,000	30,000	30,000	30,000
# arcs	322,289	322,428	322,748	323,065	323,779	329,665
# pivots seq	329,745	351,614	320,739	307,838	288,610	234,935
# pivots 19 pr	315,644	356,240	333,642	319,365	296,009	232,063
CPU secs. seq	21,966	8,694	3,859	2,867	2,144	1,322
CPU secs. 19 pr	3,566	1,134	479	294	187	100
Speedup 19 procs	6.1	7.6	8.0	9.7	11.4	13.2

Table 4.8 Results for HYGRNET for Group 4

4.6 Results for NETGEN problems

Pure network problems are a class of problems in GN. For these problems, the multipliers are equal to -1 for all arcs, and algorithms for GN can be applied. Though HYGRNET can solve pure networks, computational experience has shown that TPGRNET is faster. This suggests that the heavier emphasis that TPGRNET puts on pricing (relative to PGRNET) is important for this class of problems. Table 4.9 gives the NETGEN input data for problems 101-103 and 121-123 of the (new) standard problem set. They are listed as N101-N103 and N121-N123 to indicate that they correspond to NETGEN problems. Problems N101-N103 are bipartite problems, and problems N121-N123 are transshipment problems. The random seeds for these problems and the optimal objective function values are given in Appendix 3.

	Problems					
Character.	**N101**	**N102**	**N103**	**N121**	**N122**	**N123**
Nodes	5,000	5,000	5,000	5,000	5,000	5,000
Arcs	25,000	25,000	25,000	25,000	25,000	25,000
Sources	2,500	2,500	2,500	50	250	500
Sinks	2,500	2,500	2,500	50	250	500
% Cap	100	100	100	100	100	100
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100
Bnd Range	1-1k	1-1k	1-1k	1-1k	1-1k	1-1k
Supply	250k	2,500k	6,250k	250k	250k	250k

Table 4.9 NETGEN Input Data for Group 5

The data in Table 4.10 shows that TPGRNET (on 15 processors) yields a speedup over GRNET2 of at least 5.1 for the bipartite problems, and a speedup of at least 3.8 for the transshipment problems. The dashed line on the curve for problem N121 indicates that the leftmost data point is extrapolated from other data (the correct left data point has been ommitted, and would otherwise be plotted at (1,674.3) CPU seconds). As with the MAGEN problems, TPGRNET gets much of its speed by solving problems with fewer pivots than GRNET2.

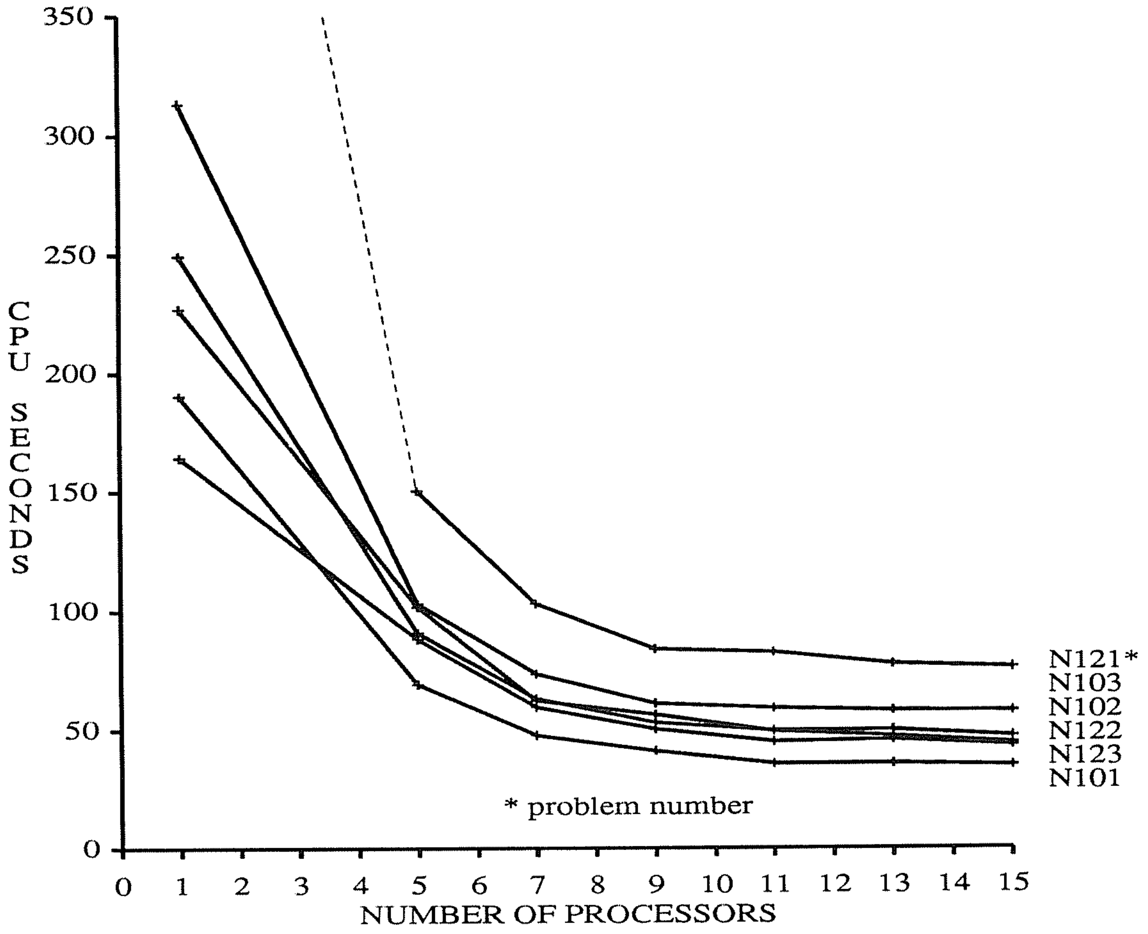

Figure 4.10 TPGRNET CPU times for NETGEN Group 5

Problem #	N101	N102	N103	N121	N122	N123
# qtrees	166	1	1	1	1	1
# nodes	5,000	5,000	5,000	5,000	5,000	5,000
# arcs	25,336	25,387	25,355	25,000	25,000	25,000
# pivots seq # pivots 19 pr	12,589 9,838	16,698 13,898	20,634 17,562	35,792 19,475	15,623 12,386	13,528 11,602
CPU secs. seq CPU secs. 19 pr	190 34	249 47	313 57	674 76	227 44	164 43
Speedup 15 procs	5.5	5.2	5.4	8.8	5.1	3.8

Table 4.10 TPGRNET results for NETGEN Group 5

The problems whose characteristics are given in Table 4.11 are used as test problems for NETPAR in [Peters 88b]. These are enlarged versions of N101, N102 and N103. The random seeds and optimal objective function values are given in Appendix 3.

	Problems		
Character.	**N201**	**N202**	**N203**
Nodes	50,000	50,000	50,000
Arcs	250,000	250,000	250,000
Sources	25,000	25,000	25,000
Sinks	25,000	25,000	25,000
% Cap	100	100	100
Cost Range	1-100	1-100	1-100
Random Seed	13502460	04281922	44820113
Supply	250k	2500k	6250k
Bnd Range	1-1k	1-1k	1-1k

Table 4.11 NETGEN Input Data for Group 6

Figure 4.12 and Table 4.12 give results for problems N201, N202 and N203. Speedups range from 3.4 to 4.7, and overall execution times are competitive with or slightly worse than those of NETPAR [Peters 88b].

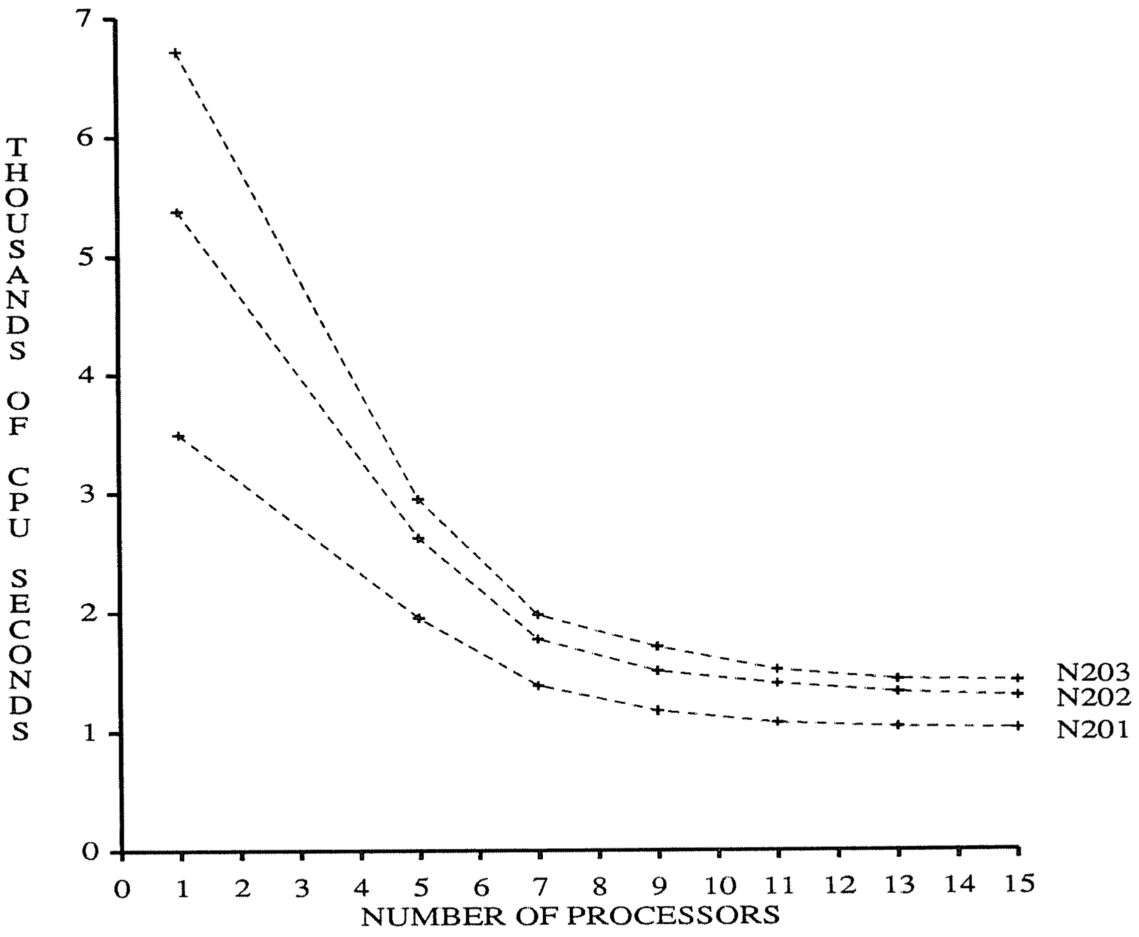

Figure 4.12 Results for TPGRNET

Problem #	N201	N202	N203
# qtrees	6	6	5
# nodes	50,000	50,000	50,000
# arcs	253,537	253,388	253,364
# pivots seq # pivots 19 pr	143,229 113,027	140,985 114,041	144,888 118,919
CPU secs. seq CPU secs. 19 pr	3,497.2 1,017.7	5,376.2 1,289.3	6,718.7 1,419.0
Speedup 15 procs	3.4	4.1	4.7

Table 4.12 TPGRNET results for NETGEN Group 6

The following NETGEN problems help to establish that the serial generalized network code GENFLO [Muthukrishnan 88] is a state-of-the-art generalized network code by making a direct comparison between GENFLO and GENNET [Brown and McBride 84]. In Section 4.7, a direct comparison will be made between GRNET2 and GENFLO will be given that will provide an indirect comparison between GRNET2 and GENNET.

Table 4.13 gives results for Group 7, a collection of problems generated by NETGEN. As with the problems N201 through N203, the problem numbers have the prefix "N", to indicate that they were generated by NETGEN. The numerical suffix indicates the standard NETGEN problem number. All times are in seconds, and all runs were made on an IBM 3081-D24. Although the number of pivots executed by MPSX (the IBM proprietary mathematical programming system) is roughly equal to the number of pivots executed by NETFLO [Kennington and Helgason 80], NETFLO is roughly 68 times faster than MPSX due to the fact that it utilizes the tree structure of bases and uses integer arithmetic. GENNET uses a good pricing strategy that reduces the number of pivots by a factor of three, compared to MPSX. Overall, GENNET timings are about 54 times faster than MPSX. The timings and the number of pivots for GENFLO are similar to those of GENNET. GENFLO solves these problems with fewer pivots than GENNET, but CPU times are about 25% slower, possibly due to the fact that more floating point arithmetic is done in pricing, the ratio test, and the flow update. This means that modifying GENNET to derive GENFLO caused a loss in efficiency of about 25%.

	Size		MPSX		GENFLO		GENNET		NETGEN	
Problem	nodes	arcs	pivots	time	pivots	time	pivots	time	pivots	time
N15	400	4,500	2,818	30.60	1,288	1.41	1,307	1.19	2,073	0.47
N18	400	1,306	2,077	12.00	593	0.49	578	0.39	1,079	0.24
N19	400	2,443	4,229	29.40	688	0.71	765	0.53	1,305	0.23
N22	400	1,416	3,052	18.00	613	0.52	504	0.33	1,284	0.29
N23	400	2,836	7,073	57.60	492	0.47	604	0.45	1,156	0.22
N26	400	1,382	4,286	24.60	511	0.42	500	0.27	917	0.14
N27	400	2,676	11,829	95.40	628	0.55	826	0.46	1,730	0.28
N28	1,000	2,900	3,313	38.40	1,487	1.39	1,732	1.24	3,524	0.93
N29	1,000	3,400	3,744	43.80	1,889	1.59	1,996	1.18	4,570	1.12
N30	1,000	4,400	4,954	60.00	1,947	1.87	1,969	1.31	4,346	1.04
N31	1,000	4,800	6,232	81.00	2,171	2.13	2,347	1.47	4,798	1.13
N33	1,500	4,385	5,836	103.20	2,645	2.83	2,521	2.01	6,113	2.16
N34	1,500	5,107	6,503	110.40	2,498	2.50	2,943	2.10	7,640	2.37
N35	1,500	5,730	7,026	115.80	3,017	3.35	3,310	2.82	7,384	2.30
Total			72,972	820.20	47,919	12.92	21,902	15.75	20,467	20.23

Table 4.13 Serial results for NETGEN problems (IBM 3081-D24)

4.7 Results for GNETGEN Problems

NETGEN has been modified by F. Glover to generate generalized network flow problems. The modified generator is called GNETGEN. Table 3.14 gives the GNETGEN input data (except for the random seed) for the small test problems G1 through G7 (Group 8). The random seed for all of these problems is 13502460. The prefix "G" for these problems indicates that they were generated by GNETGEN. The numerical suffix corresponds to the problem numbers in Table 2.2 in [Muthukrishnan 88].

Problem	G1	G2	G3	G4	G5	G6	G7
Nodes	200	200	200	300	400	400	1,000
Arcs	1,500	4,000	6,000	4,000	5,000	7,000	6,000
Sources	100	5	15	135	20	30	20
Sinks	100	195	50	165	100	50	100
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100	1-100
Gain Range	.5-1.5	.5-1.5	.25-.95	.5-1.5	.3-1.7	.5-1.5	.4-1.4
Supply	100k	100k	100k	100k	100k	100k	200k
% Cap	0	100	100	0	0	100	100
Bnd Range	-	1-2k	1-2k	-	-	1-2k	4-6k

Table 4.14 Input data for GNETGEN Group 8

Table 4.15 gives results for MPSX, GENNET and GENFLO for problems G1 through G7. For these problems, GENNET is about 12 times faster than MPSX and GENFLO about 11 times faster. GENFLO solves these problems with about 40% fewer pivots than MPSX.

	Size		MPSX		GENNET		GENFLO	
Pr	nds	arcs	pivots	time	pivots	time	pivots	time
G1	200	1,500	1,151	7.80	590	0.62	533	0.95
G2	200	4,000	550	3.00	443	0.22	358	0.23
G3	200	6,000	2,058	18.60	1,448	2.07	954	1.53
G4	300	4,000	4,112	47.40	2,703	3.50	2,106	4.23
G5	400	5,000	1,870	26.20	1,229	2.06	897	2.23
G6	400	7,000	1,408	16.80	1,591	1.59	1,171	1.68
G7	1k	6,000	2,811	40.20	3,160	3.30	2,352	3.60
Total			13,960	160.00	11,164	13.36	8,371	14.45

Table 4.15 Serial results for GNETGEN Group 8 (IBM 3081-D24)

Tables 4.16 through 4.18 give the input data for the GNETGEN problems G8 through G22 (Groups 9 and 10). These problems are generated with the same input data as problems 1 through 15 in Table 4.1a and 4.1b in [Muthukrishnan 88]. The generator also requires the following data: # transshipment sources: 0, # transshipment sinks: 0, % skeleton arcs with maximum cost: 35, cost range: 1-100, supply: 100k, bound range: 1k-2k, multiplier range: 0.5-1.5.

	Problems					
Character.	**G8**	**G9**	**G10**	**G11**	**G12**	**G13**
Nodes	2,000	2,000	2,000	4,000	4,000	4,000
Arcs	13,000	13,000	13,000	26,000	26,000	26,000
Sources	150	150	150	150	150	150
Sinks	600	600	600	600	600	600
% Cap	100	50	0	100	50	0
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100
Bnd Range	1k-2k	1k-2k	1k-2k	1k-2k	1k-2k	1k-2k
Mult Range	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5

Table 4.16 Input data for problems G8-G13

	Problems		
Character.	**G14**	**G15**	**G16**
Nodes	6,000	6,000	6,000
Arcs	39,000	39,000	39,000
Sources	150	150	150
Sinks	600	600	600
% Cap	100	50	0
Cost Range	1-100	1-100	1-100
Bnd Range	1k-2k	1k-2k	1k-2k
Mult Range	0.5-1.5	0.5-1.5	0.5-1.5

Table 4.17 Input data for problems G14-G16

	Problems					
Character.	**G17**	**G18**	**G19**	**G20**	**G21**	**G22**
Nodes	2,000	2,000	2,000	2,000	2,000	2,000
Arcs	25,000	25,000	25,000	50,000	50,000	50,000
Sources	150	150	150	150	150	150
Sinks	600	600	600	600	600	600
% Cap	100	50	0	100	50	0
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100
Bnd Range	1k-2k	1k-2k	1k-2k	1k-2k	1k-2k	1k-2k
Mult Range	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5	0.5-1.5

Table 4.18 Input data for problems G17-G22

Table 4.19 gives a comparison of GENFLO and GRNET2 for problems G1 through G22. The two programs give nearly the same performance for these test problems, despite the fact that the two codes have different pricing strategies.

	Size		GENFLO		GRNET2	
Problem	**nodes**	**arcs**	**pivots**	**time**	**pivots**	**time**
G8	2,000	13,000	4,634	50.2	3,808	48.5
G9	2,000	13,000	4,454	44.2	3,998	46.8
G10	2,000	13,000	5,108	60.5	3,892	51.9
G11	4,000	26,000	9,145	99.3	7,375	105.2
G12	4,000	26,000	9,815	123.6	7,460	115.1
G13	4,000	26,000	9,897	125.0	7,690	121.9
G14	6,000	39,000	13,653	141.0	10,456	152.0
G15	6,000	39,000	12,900	126.3	10,059	158.9
G16	6,000	39,000	13,262	145.4	10,245	142.2
G17	2,000	25,000	6,629	119.2	5,440	81.3
G18	2,000	25,000	6,596	93.4	5,260	78.4
G19	2,000	25,000	6,186	89.2	6,369	98.0
G20	2,000	50,000	8,500	174.2	7,913	133.8
G21	2,000	50,000	9,208	204.5	10,343	194.9
G22	2,000	50,000	8,601	198.4	9,608	194.8
Total			128,588	1,794.4	109,916	1,723.7

Table 4.19 Results for GNETGEN Groups 9,10 (Sequent: 1 proc)

To give a further comparison of GENFLO and GRNET2, results for the Group 4 problems are given in Table 4.20. The serial version of GENFLO outperforms GRNET2 on problem 4.50 in terms of CPU time, but GRNET2 generally outperforms GENFLO for the more difficult problems in terms of both CPU time and the number of pivots. The results of tables 4.19. and 4.20 indicate that GRNET2 is competative with GENFLO, and the results of Table 4.13 indicate that GENFLO is competative with GENNET. This makes it reasonable to claim that GRNET2 is a state-of-the-art generalized network flow code.

Problem #	**4.00**	**4.01**	**4.03**	**4.05**	**4.10**	**4.50**
# qtrees	1	139	459	776	1,490	7,376
# nodes	30,000	30,000	30,000	30,000	30,000	30,000
# arcs	322,289	322,428	322,748	323,065	323,779	329,665
pivs GENFLO	***	411,720	337,907	313,982	289,937	244,635
pivs GRNET2	328,711	347,420	320,937	308,284	288,856	228,819
CPU GENFLO	***	36,523	10,524	5,121	2,436	1,038
CPU GRNET2	22,434	9,679	4,525	3,096	2,340	1,388

*** Did not finish after 14 hours.

Table 4.20 Results for MAGEN Group 4 (Sequent: 1 proc)

Table 4.21 gives CPU times for TPGRNET (run on various numbers of processors) for problems G8 through G22. TPGRNET is faster than the parallel versions of GENFLO for these test problems, and it is more robust in the sense that CPU times usually decrease monotonically as the number of processors is increased. The serial times reported in the table are taken from GRNET2 if they have the "T" prefix, and they are taken from GENFLO if they have the "O" prefix. The serial time is always taken from the faster of the two codes. The totals from the bottom of Table 4.21 are graphed in Figure 4.21.

Table 4.22 gives speedup results for problems G8 through G22, and the results are graphed for problems G14 through G16 and problems G20 through G22. Problems G14 through G16 have the smallest ratio of arcs to nodes, and TPGRNET yields the smallest speedup for these problems. TPGRNET gives an average speedup of 5.4 on 15 processors for problems G20 through G22. These are the problems for which the arcs/nodes ratio is the largest. Since TPGRNET gets most of its parallelism from pricing out arcs in parallel and gets only limited parallelism from parallel pivoting, the dependence of the efficiency of TPGRNET on the arcs/nodes ratio is understandable.

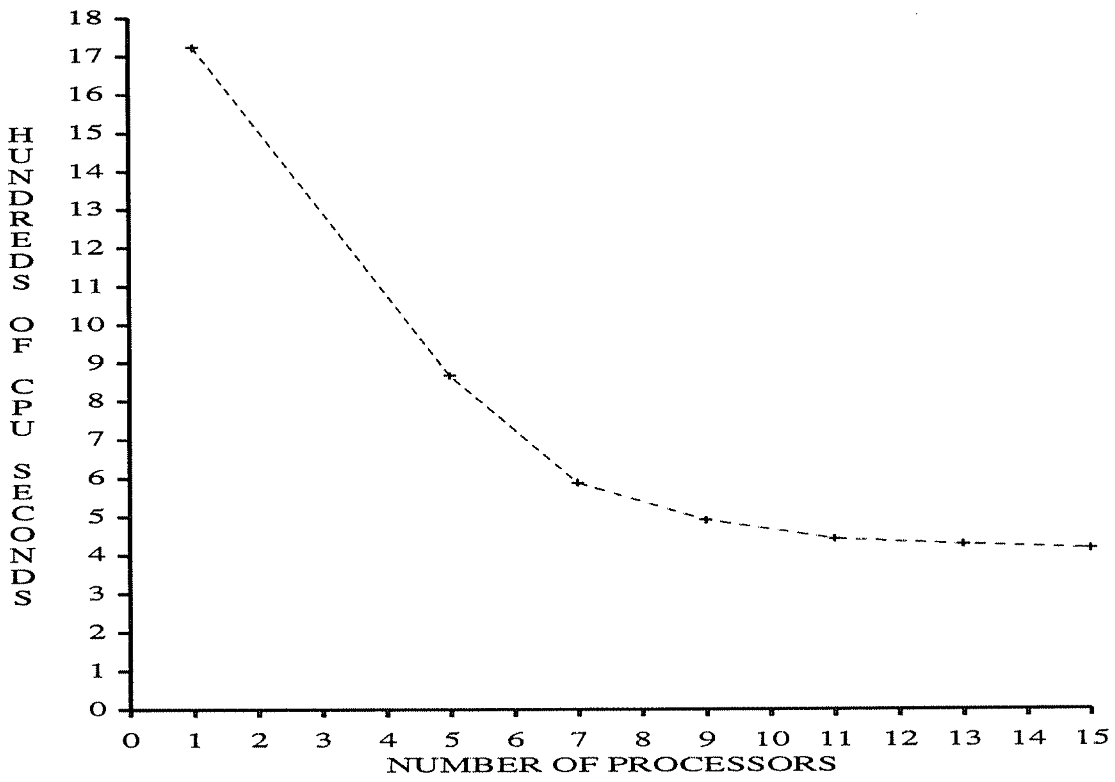

Figure 4.21 Composite results for TPGRNET

			Number		of		Processors		
Prb	nds×arcs	cap	1	5	7	9	11	13	15
G8	2k×13k	100	T51.9	29.3	19.8	16.2	16.3	16.9	15.8
G9	2k×13k	50	O44.2	24.6	16.7	14.0	12.6	13.1	13.8
G10	2k×13k	0	T48.5	25.2	17.5	14.2	12.9	13.2	13.3
G11	4k×26k	100	T121.9	58.3	43.8	34.3	35.6	33.8	31.8
G12	4k×26k	50	T115.1	61.2	41.1	36.4	33.5	31.6	31.4
G13	4k×26k	0	O99.3	56.8	38.7	34.0	29.0	29.9	29.6
G14	6k×39k	100	T142.2	87.5	58.4	50.3	44.3	45.1	43.8
G15	6k×39k	50	O126.3	87.0	65.5	51.6	48.1	46.5	47.0
G16	6k×39k	0	O141.0	89.5	61.3	51.3	45.8	40.8	40.8
G17	2k×25k	100	O89.2	39.0	25.2	20.2	19.2	18.1	16.8
G18	2k×25k	50	T78.4	41.3	29.8	22.5	22.5	21.2	19.0
G19	2k×25k	0	T81.3	41.8	25.7	21.9	18.4	18.1	18.2
G20	2k×50k	100	T194.8	77.5	49.3	44.3	35.0	34.8	33.0
G21	2k×50k	50	T194.9	83.4	50.8	42.2	36.5	31.7	33.5
G22	2k×50k	0	T133.8	64.4	44.5	37.9	32.1	32.0	27.9
Totals			1723	866	588	491	441	426	415

Table 4.21 TPGRNET results for GNETGEN Groups 9,10 (Sequent)

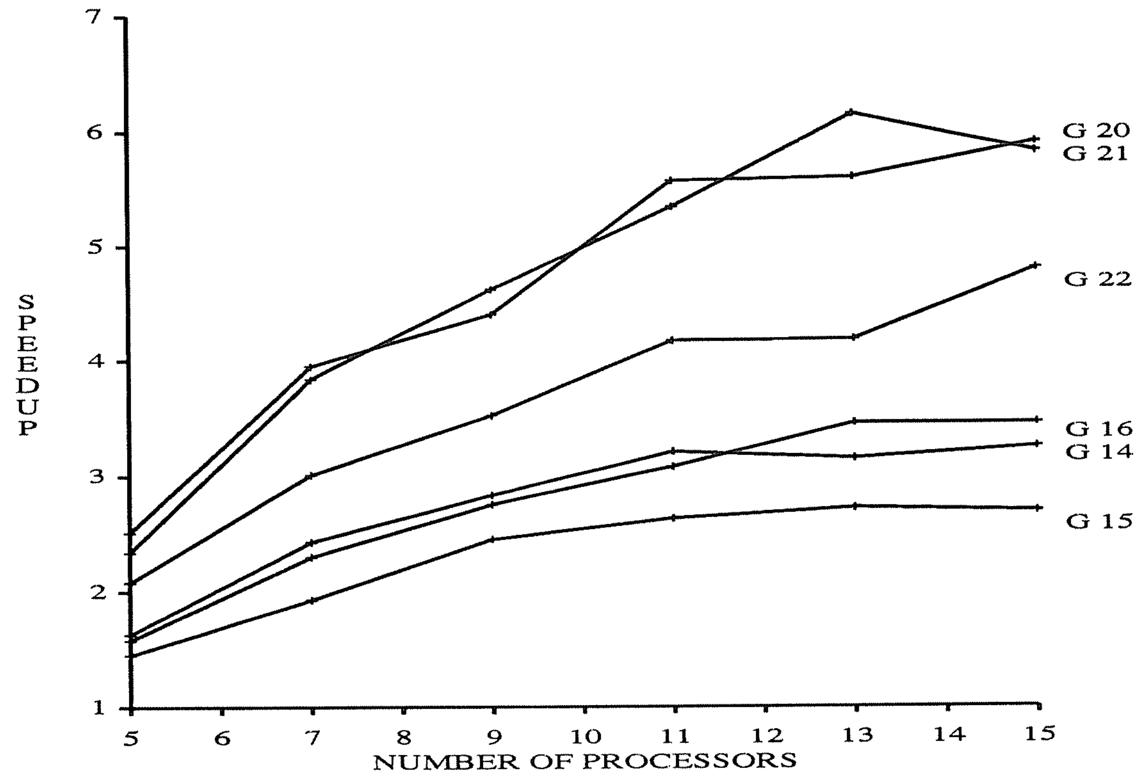

Figure 4.22 Speedups for TPGRNET

				Number of Processors					
Prb	nds×arcs	cap	qtree	5	7	9	11	13	15
G8	2k×13k	100	4	1.7	2.6	3.2	3.1	3.0	3.2
G9	2k×13k	50	1	1.7	2.6	3.1	3.5	3.3	3.2
G10	2k×13k	0	2	1.9	2.7	3.4	3.7	3.6	3.6
G11	4k×26k	100	3	2.0	2.7	3.5	3.4	3.6	3.8
G12	4k×26k	50	1	1.8	2.8	3.1	3.4	3.6	3.6
G13	4k×26k	0	3	1.7	2.5	2.9	3.4	3.3	3.3
G14	6k×39k	100	5	1.6	2.4	2.8	3.2	3.1	3.2
G15	6k×39k	50	7	1.4	1.9	2.4	2.6	2.7	2.6
G16	6k×39k	0	2	1.5	2.3	2.7	3.0	3.4	3.4
G17	2k×25k	100	9	2.2	3.5	4.4	4.6	4.9	5.3
G18	2k×25k	50	3	1.8	2.6	3.4	3.4	3.6	4.1
G19	2k×25k	0	3	1.9	3.1	3.7	4.4	4.4	4.4
G20	2k×50k	100	2	2.5	3.9	4.3	5.5	5.5	5.9
G21	2k×50k	50	3	2.3	3.8	4.6	5.3	6.1	5.8
G22	2k×50k	0	1	2.0	3.0	3.5	4.1	4.1	4.7
Average Speedup				1.9	2.9	3.5	3.8	4.0	4.1

Table 4.22 TPGRNET speedups for GNETGEN Groups 9,10 (Sequent)

4.8 Other Measures of Performance

Figure 4.23 and Table 4.23 further help to explain the behavior of PGRNET. A "collision" occurs when a processor tries to lock a quasi-tree that has already been locked by another processor. Table 4.23 gives the number of pivots that PGRNET uses to solve the problems in MAGEN Group 1, along with the number of collisions that occur during the solution of each problem. The 19 processor version of PGRNET had 1,044,340 collisions for problem 1.00 and it solved this problem with 111,843 pivots. This means that there were more than 9 collisions per pivot for this problem, and this can explain the poor speedup results from PGRNET for this problem. The 3 processor version had only about 1/10th as many collisions, because there were fewer processors to compete with each other. For problems 1.03, 1.05 and 1.50, PGRNET had fewer collisions than pivots, and the number of collisions decreased as the granularity of the problem and the number of processors were decreased. The ratios of collisions to pivots for these problems are listed in table 4.23 and graphed in Figure 4.23.

One way to measure the effectiveness of the TPGRNET algorithm is to count the number of times that pivot arcs are chosen from *best_cand* and the number of times that pivot arcs are chosen from the candidate lists. This gives a measure of the effectiveness of the TPGRNET pricing scheme, which overlaps pricing and pivoting, and dedicates all but two processors to pricing. Table 4.24 gives this data for the 3 and 19 processor versions of TPGRNET solving the problems in MAGEN Group 1. Figure 4.24 gives the percent of pivot arcs that are taken from *best_cand* (the variable in shared memory in which the pivoting processor attempts to find its pivot arcs). The 3 processor version takes a slight majority of its pivot arcs from *best_cand*. The 19 processor version takes a vast majority of pivots from *best_cand*. This is because more pricing can be done by the 17 pricing processors, and there is thus a high probability that the pivoting processor will find a pivot-eligible arc in *best_cand* almost every time that it looks there.

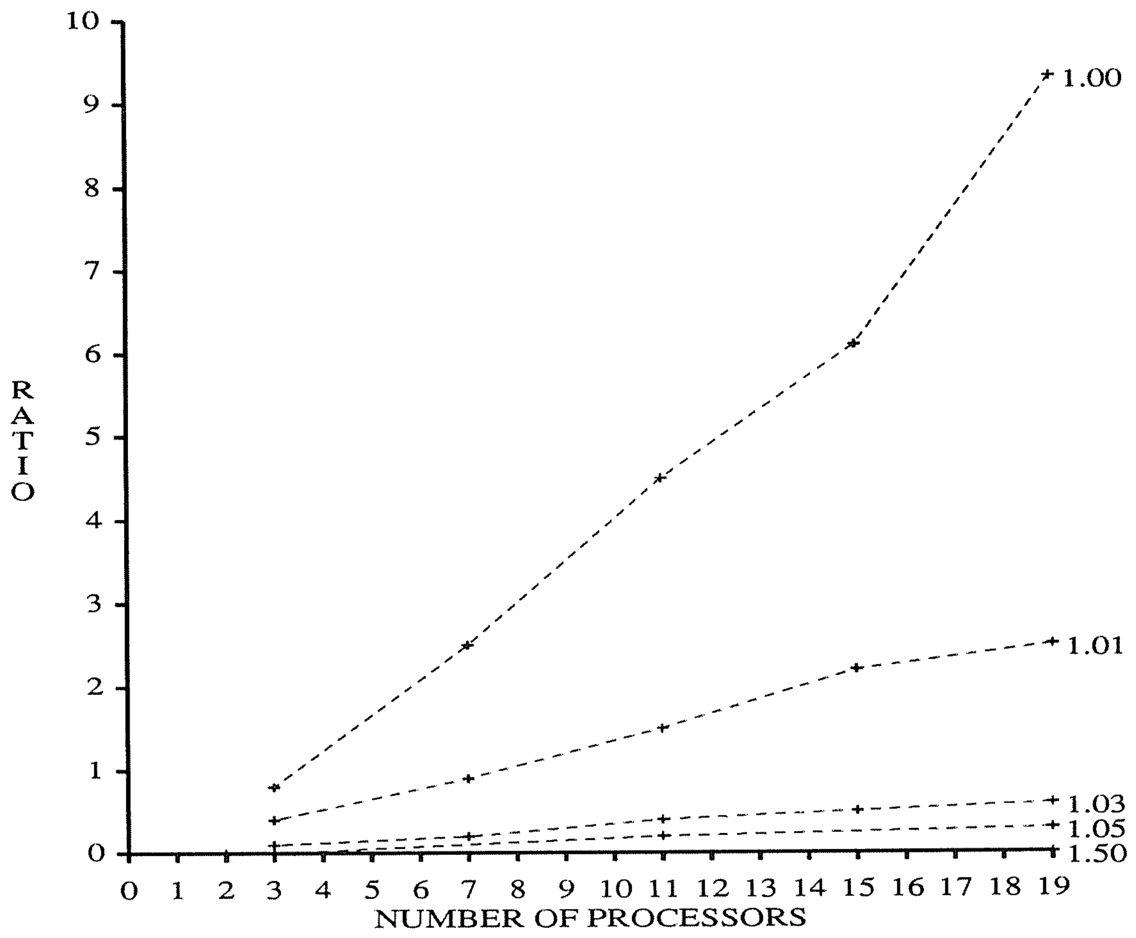

Figure 4.23 Ratio of collisions to # pivots for PGRNET

Problem #	1.00	1.01	1.03	1.05	1.50
collisions 19 procs	1,044,340	296,144	70,768	40,072	615
pivots 19 procs	111,843	115,680	106,496	103,064	76,367
ratio 19 procs	9.3	2.5	0.6	0.3	0.0
collisions 3 procs	94,875	56,714	15,084	5,964	31
pivots 3 procs	117,075	126,604	116,083	109,244	76,847
ratio 3 procs	0.8	0.4	0.1	0.0	0.0

Table 4.23 Collisions, pivots and collision/pivot ratio

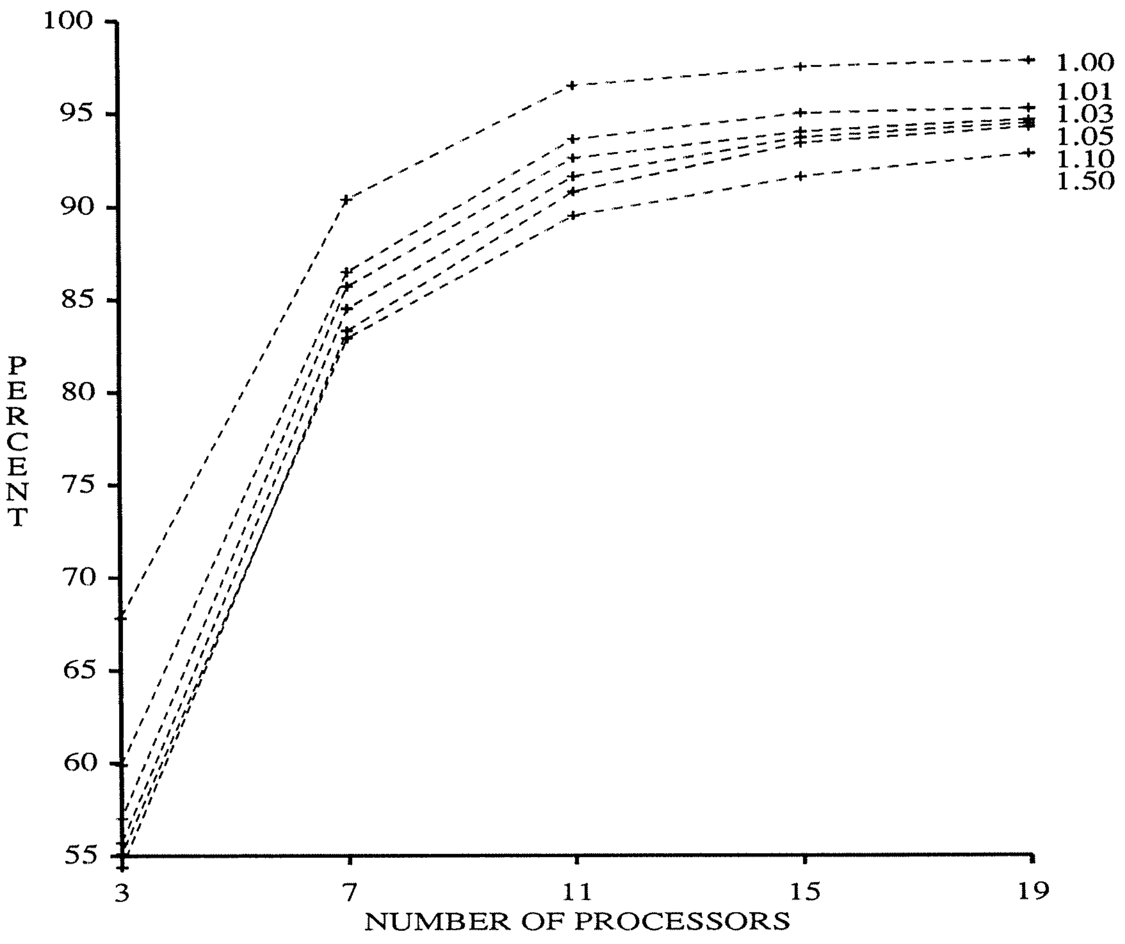

Figure 4.24 TPGRNET pivot arcs taken from *best_cand*

Problem #	1.00	1.01	1.03	1.05	1.10	1.50
best_cand 19 procs	86,252	91,282	87,131	85,275	81,567	68,530
cand lists 19 procs	1,877	4,512	4,879	4,989	4,944	5,194
STAGE 2 pivots 19 procs	15	4	16	28	23	51
best_cand 3 procs	81,862	82,767	75,812	70,810	64,388	46,884
cand lists 3 procs	38,837	55,267	57,028	56,174	53,876	38,131
STAGE 2 pivots 3 procs	9	7	9	2	0	2

Table 4.24 # pivot arcs taken from *best_cand* and cand lists

4.9 Results for Massive Problems

Figure 4.25 and tables 4.25 and 4.26 show results for two problems with more than a million variables. The MAGEN input data for these problems is given in Appendix 3, along with the optimal objective function value. The problem reported in table 4.25 was generated with $zfrac = 0.50$ (the same as $zfrac$ for the problems in groups 1 through 4), and the problem in table 4.26 has $zfrac = 0.01$. Problem 4.99.heavy is heavily capacitated in the sense that roughly half of all arcs have their flow at upper bound in an optimal solution, and problem 4.99.light is lightly capacitated in the sense that only about 1/100th of all arcs have their flow at upper bound in an optimal solution. Both of these problems are small grained, so they can be solved quite efficiently by PGRNET. All of the CPU times given in the table are CPU seconds. Due to the large number of quasi-trees in the optimal bases of these problems, one would expect the speedups on 19 processors to be somewhat higher than they are, 11.0 for the heavily capacitated problem and 7.0 for the lightly capacitated problem. One possible factor is that the system time for the major page faults might be included, to some extent, in the user time for these problems. The results given in Table 4.25 and Table 4.26 may also be affected somewhat by the presence of other users on the system at the time that the runs were made.

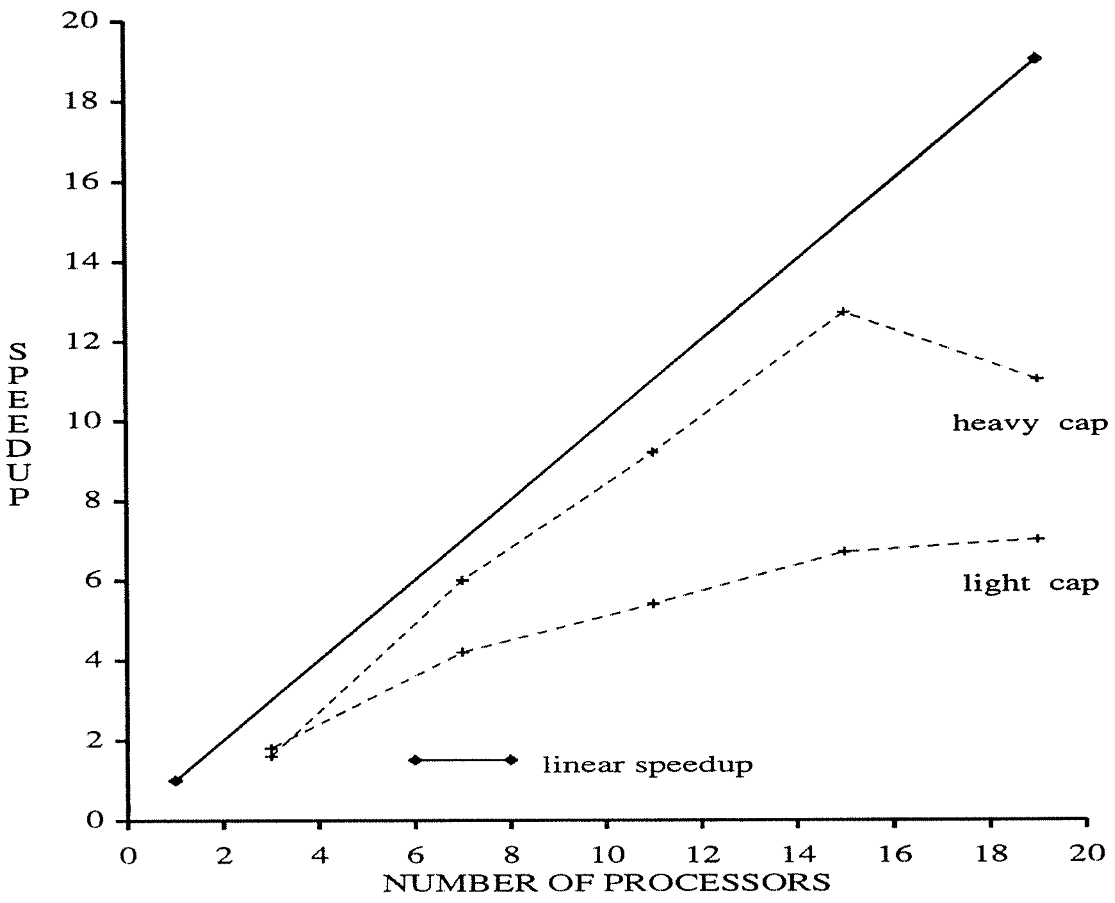

Figure 4.25 Speedups for PGRNET (Sequent)

program	# arcs	# nodes	# qtrees	time	pivots	maj page swap
serial	1,267,185	30,000	14,859	8,415	706,776	914,478
19 procs	1,267,185	30,000	14,859	760	716,218	100,614

Table 4.25 PGRNET Results for Problem 4.99.heavy (Sequent)

program	# arcs	# nodes	# qtrees	time	pivots	maj page swap
serial	1,267,185	30,000	14,859	3,305	184,379	363,755
19 procs	1,267,185	30,000	14,859	470	185,522	51,146

Table 4.26 PGRNET Results for Problem 4.99.light (Sequent)

V Directions for Future Research

A generalization of the parallel pricing approach that we wish to study is the utilization of some of the processing power for the simplex "ratio tests". That is, given a sufficiently large number of processors (say, more than 15 in the case of generalized networks), it may be more advantageous to utilize some of the additional processors to do ratio tests for pivot-eligible columns rather than to allocate them to do additional pricing. Further into the future, we would like to develop a heuristic for PGRNET that would allocate arcs to processors using information from a solved problem. The optimal basis of the solved problem (the warm start) might resemble the optimal basis of a problem with similar data. If so, then the warm start would indicate how to allocate groups of arcs to processors in such a way that processors would rarely collide with each other when selecting pivot arcs. Finally, we would like to test variants of the TPGRNET strategy on general linear programs. The TPGRNET algorithm clearly extends in a straightforward way to general linear programs, but implementation details will play an important role in determining efficiency.

A.1 Appendix 1 Derivation of Dual to GN′

Let G have no columns with more than two non-zero entries. Standard techniques can be used to show that all of the following problem formulations are equivalent:

$$\begin{array}{l} \min \quad cx \\ \text{s.t.} \quad Gx = b \\ 0 \le x_{ij} \le u_{ij} \quad \text{if} \quad i \ne j \\ 0 \le x_{ii} \end{array}$$

$$\begin{array}{l} \min \quad c_1 x_1 + c_2 x_2 \\ \text{s.t.} \quad G_1 x_1 + G_2 x_2 = b \\ \quad 0 \le x_2 \le u \\ \quad 0 \le x_1 \end{array}$$

(where x_1 is the vector of root arcs and x_2 is the vector of arcs having different "from" and "to" nodes and c and G are partitioned accordingly)

$$\begin{array}{l} \min \quad c_1 x_1 + c_2 x_2 \\ \text{s.t.} \quad G_1 x_1 + G_2 x_2 = b \\ \quad -x_2 \ge -u \\ \quad x_1, x_2 \ge 0 \end{array}$$

$$\begin{array}{l} \max \quad b\mu - uw \\ \text{s.t.} \quad \mu G_1 \qquad \le c_1 \\ \quad \mu G_2 - w \le c_2 \\ \quad w \ge 0 \\ \quad \mu \ \text{unres.} \end{array}$$

Since $u \ge 0$, the dual objective function is maximized as a function of w (subject to the constraints on w) by setting $w = (\mu G_2 - c_2)^+ = (c_2 - \mu G_2)^-$. This yields the following equivalent formulation:

Substituting ${\pi'}_k = ({\pi'}_k - \pi_k) + \pi_k$,

$$\begin{aligned}
b\pi' &- u(\rho - \pi'\mathcal{G})^- \\
&= \sum_i b_i\pi_i + b_k({\pi'}_k - \pi_k) - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
&\quad + \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}(\pi_k - {\pi'}_k) + \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j) \\
&= \sum_i b_i\pi_i - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
&\quad + ({\pi'}_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) + \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j).
\end{aligned}$$

The next expression splits the last sum by separating the positive and negative terms.

$$\begin{aligned}
b\pi' &- u(\rho - \pi'\mathcal{G})^- \\
&= \sum_i b_i\pi_i - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
&\quad + ({\pi'}_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) \\
&\quad + \sum_{(k,j)\in F} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j) + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j) \\
&= \sum_i b_i\pi_i - \sum_{(i,j)\in I} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
&\quad + ({\pi'}_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) \\
&\quad - \sum_{(k,j)} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j)^- + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j).
\end{aligned}$$

$$\begin{aligned}
b\pi' - &u(\rho - \pi'\mathcal{G})^- \\
= &\sum_i b_i \pi_i - \sum_{(i,j)} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
&+(\pi'_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j).
\end{aligned}$$

By the same technique used in the proof of Lemma 1, the remaining steps of the proof relate the dual objective function to the primal objective function (evaluated at f). Since

$$b_t = \begin{cases} f_{tt} + \sum_{tj} f_{tj} + \sum_{jt} m_{jt} f_{jt}, & \text{if node } t \text{ has an attached root arc, or} \\ \sum_{tj} f_{tj} + \sum_{jt} m_{jt} f_{jt}, & \text{otherwise,} \end{cases}$$

$$\begin{aligned}
b\pi' - &u(\rho - \pi'\mathcal{G})^- \\
= &\sum_{i\in R} f_{ii}\pi_i + \sum_{(i,j)} f_{ij}\pi_i + \sum_{(i,j)} m_{ij} f_{ij}\pi_j \\
&- \sum_{(i,j)} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
&+(\pi'_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j) \\
= &\sum_{i\in R} f_{ii}\pi_i + \sum_{(i,j)} f_{ij}c_{ij} - \sum_{(i,j)} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j) \\
&- \sum_{(i,j)} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
&+(\pi'_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j).
\end{aligned}$$

By splitting the sum $\sum_{(i,j)} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)$ into its negative and positive terms, the dual objective becomes:

$$\begin{aligned}
b\pi' - & u(\rho - \pi'\mathcal{G})^- \\
= \; & \sum_{i\in R} f_{ii}\pi_i + \sum_{(i,j)} f_{ij}c_{ij} \\
& + \sum_{(i,j)} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- - \sum_{(i,j)} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^+ \\
& - \sum_{(i,j)} u_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
& + (\pi'_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j) \\
= \; & \sum_{i\in R} f_{ii}\pi_i + \sum_{(i,j)} f_{ij}c_{ij} - \sum_{(i,j)} f_{ij}(c_{ij} - \pi_i - m_{ij}\pi_j)^+ \\
& - \sum_{(i,j)} [u_{ij} - f_{ij}](c_{ij} - \pi_i - m_{ij}\pi_j)^- \\
& + (\pi'_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}) + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j).
\end{aligned}$$

By assumption, either $\pi_i = c_{ii}$, or $f_{ii} = 0 \;\forall\; i \in R$, so

$$\sum_{i\in R} f_{ii}\pi_i = \sum_{i\in R} f_{ii}c_{ii}.$$

Making this final substitution relates the value of the dual objective function for P' to the value of the primal objective function, given by $\sum_{i\in R} f_{ii}c_{ii} + \sum_{(i,j)} f_{ij}c_{ij}$.

$$\begin{aligned}
b\pi' - & u(\rho - \pi'\mathcal{G})^- \\
= \; & \sum_{i\in R} f_{ii}\pi_i + \sum_{(i,j)} f_{ij}c_{ij} - DG(f,\pi) \\
& + \sum_{(k,j)\in\overline{F}_k} u_{kj}(c_{kj} - \pi_k - m_{kj}\pi_j) + (\pi'_k - \pi_k)(b_k - \sum_{(k,j)\in F\cup\overline{F}_k} u_{kj}).
\end{aligned}$$

By the last hypothesis of Lemma 2, the dual objective value of P' is strictly greater than the the objective value of the primal feasible point f of P. This means that arc (k, k) must have a non-zero flow in any optimal solution of P, because the above analysis shows that removing the arc perturbs the optimal primal objective value of P. QED.

A.3 Appendix 3 Optimal Objective Values and (some) Random Seeds

Problem #	**1.00**	**1.01**	**1.03**
opt objective	770349279.42	793465562.21	847770723.41
Problem #	**1.05**	**1.10**	**1.50**
opt objective	906991800.97	1034729102.47	2083300638.99
Problem #	**2.00**	**2.01**	**2.03**
opt objective	1450349860.39	1498665913.44	1590072949.22
Problem #	**2.05**	**2.10**	**2.50**
opt objective	1695140388.56	1984306496.03	4076079297.43
Problem #	**3.00**	**3.01**	**3.03**
opt objective	2779725783.30	2863255942.83	3073159640.91
Problem #	**3.05**	**3.10**	**3.50**
opt objective	3269288033.85	3793989055.26	8120663463.16
Problem #	**4.00**	**4.01**	**4.03**
opt objective	2321940170.63	2394779386.82	2565668031.38
Problem #	**4.05**	**4.10**	**4.50**
opt objective	2735574013.80	3115868092.04	6249240332.12

Table A3.1 MAGEN Groups 1-4

Problem #	**N101**	**N102**	**N103**
random seed	13502460	04281922	44820113
opt objective	6191726	72337144	218947553
Problem #	**N121**	**N122**	**N123**
random seed	72707401	93040771	70220611
opt objective	66366360	30997529	23388777
Problem #	**N201**	**N202**	**N203**
random seed	13502460	04281922	44820113
opt objective	5806427	61017205	158441376

Table A3.2 NETGEN Problems

Problem #	**G8**	**G9**	**G10**
opt objective	5903491.31	5512197.01	5408206.14
Problem #	**G11**	**G12**	**G13**
opt objective	6972287.27	7085296.58	6696965.30
Problem #	**G14**	**G15**	**G16**
opt objective	7107475.96	7646706.48	7467458.24
Problem #	**G18**	**G19**	**G20**
opt objective	3317323.00	3204520.69	3270061.77
Problem #	**G21**	**G22**	**G23**
opt objective	1727745.98	2085960.94	1847498.51

Table A3.3 GNETGEN Problems

	Problems	
Character.	**4.99.heavy**	**4.99.light**
Nodes	30,000	30,000
Sources	15,000	15,000
Sinks	15,000	15,000
Arcs per node	80-86	80-86
Cost range	1-100	1-100
Mult Range	.90-.98	.90-.98
Cap max	1k-2k	1k-2k
α	0.99	0.99
zfrac	0.50	0.01
Random Seed	0731246890	0731246890
Optimal Objective	8198100113.87	15830266.87

Table A3.4 Input Data MAGEN Million Variable Problems

A.4 Appendix 4 The Management of Candidate Lists

Candidate lists are managed in the same way by PGRNET and GRNET2. A candidate list is created by sweeping through a segment of the arc list and adding pivot-eligible arcs to the corresponding candidate list as they are found. When the sweep is completed or the list has *listsize* entries, the processor can choose a pivot arc from the list. The arc that is chosen is the pivot-eligible arc that has the largest reduced cost in absolute value. Any arcs in the list that are found to have zero reduced cost are left in the list. After locating the "best" arc, the processor executes the pivot, removes the arc from the list and looks for another pivot arc. In the case of GRNET2, the processor looks for the next pivot-eligible arc in the next candidate list. GRNET2 cycles through its collection of *num_pricers* lists and chooses a pivot arc from a different list each time. In the case of PGRNET, the processor always returns to the same candidate list when chosing pivot arcs. (In fact, the PGRNET candidate lists are not stored in the shared memory, so processors only have access to one candidate list). If a processor attempts to find a pivot-eligible arc in a candidate list and there are no pivot-eligible arcs, or the list has *list_threshold* or fewer entries (where *list_threshold* =(*listsize*/2)), then a new list is made. If, in the process of making a new candidate list, only *list_threshold* or fewer pivot-eligible arcs can be found, then the penalty on the artificial arcs is checked. If the penalty on the artificial arcs is smaller than *big M*, then the penalty is increased, and duals are recomputed, and this generally causes enough arcs to become pivot-eligible so that full new candidate lists can be created. If the penalty on the artificial arcs is equal to *big M*, then Stage 2 is begun.

References

Adolphson, D. and Heum, L. [1981]:
"Computational experiments on a threaded index generalized network code", Presented at the ORSA/TIMS National Meeting in Houston, Texas.

Adolphson, D. [1982]:
"Design of primal simplex generalized network codes using a preorder thread index", Working Paper, School of Management, Brigham Young University, Provo.

Ali, I., Charnes, A. and Song, T. [1986]:
"Design and implementation of data structures for generalized networks", *Journal of Information and Optimization Sciences* 7, 81-104.

Barr, R., Glover, F. and Klingman, D. [1979]:
"Enhancements of spanning tree labeling procedures for network optimization", *INFOR* 17, 16-34.

Brown, G.G. and McBride, R.D. [1984]:
"Solving generalized networks", *Management Science*, 30, 1497-1523.

Chang, M., Cheng, M. and Chen, C. [1988]:
"Implementation of new labeling procedures for generalized networks", Technical Report, Department of CS/OR, North Dakota State University, Fargo, North Dakota.

Chang, M., Engquist, M., Finkel, R. and Meyer, R. [1987]:
"A parallel algorithm for generalized networks", *Annals of Operations Research*, 14(1988) 125-145.

Chang, M. and Engquist, M. [1986]:
"On the number of quasi-trees in an optimal generalized network basis", *COAL Newsletter* 14, 5-9.

Clark, R. and Meyer, R. [1987]:
"Multiprocessor algorithms for generalized network flows", Technical Report #739, Department of Computer Sciences, The University of Wisconsin-Madison.

Clark, R. and Meyer, R. [1989]:
"Parallel arc-allocation algorithms for optimizing generalized networks", Technical Report #862, Department of Computer Sciences, The University of Wisconsin-Madison.

Clark, R., Kennington, J., Meyer, R. and Muthukrishnan, R. [1989]:
"Parallel algorithms for generalized networks: computational experience.", to appear in 1989.

DeWitt, D., Finkel, R. and Solomon, M. [1984]:
"The CRYSTAL multicomputer: design and implementation experience" Technical Report #553, Department of Computer Sciences, The University of Wisconsin-Madison.

Elam, J., Glover, F. and Klingman, D. [1979]:
"A strongly convergent primal simplex algorithm for generalized networks", *Mathematics of Operations Research* 4, 39-59.

Engquist, M. and Chang, M. [1985]:
"New labeling procedures for the basis graph in generalized networks", *Operations Research Letters*, Vol. 4, No. 4, 151-155.

Glover, F., Hultz, J., Klingman, D. and Stutz, J. [1978]:
"Generalized networks: A fundamental computer based planning tool", *Management Science* 24, 1209-1220.

Glover, F., Klingman, D. and Stutz, J. [1974]:
"The augmented threaded index method for network optimization", *INFOR* 12, 293-298.

Glover, F., Klingman, D. and Stutz, J. [1973]:
"Extension of the augmented predecessor index method to generalized network problems", *Transportation Science* 7, 377-384.

Grigoriadis, M. [1984]:
"An efficient implementation of the network simplex method", *Mathematical Programming Study* 26, 83-111.

Jensen, P. and Barnes, J. W. [1980]:
Network Flow Programming, John Wiley and Sons, New York.

Kennington, J. and Helgason, R. [1980]:
Algorithms for Network Flow Programming, John Wiley and Sons, New York.

Klingman, D., Napier, A. and Stutz J. [1974]:
"NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow problems," *Management Science* 20, 814-821.

Langley, W. [1973]:
"Continuous and integer generalized flow problems", PhD thesis, Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

Mulvey, J. and Zenios S. [1985]:
"Solving large scale generalized networks", *Journal of Information and Optimization Sciences* 6, 95-112.

Murtagh, B. and Saunders, M. [1978]:
"Large-scale linearly constrained optimization", *Mathematical Programming* 14, 41-72.

Muthukrishnan, R. [1988]:
"Parallel algorithms for generalized networks", PhD thesis, M.S.O.R., Southern Methodist University, Dallas, Texas.

Nulty, W. and Trick M. [1988]:
"GNO/PC Generalized network optimization system", *Operations Research Letters* 7, 101-102.

Peters, J. [1988a]:
"A parallel algorithm for minimal cost network flow problems", Technical Report # 762, Department of Computer Sciences, The University of Wisconsin-Madison.

Peters, J. [1988b]:
"The network simplex method on a multi-processor", June 1988, to appear in *Networks*.

Tomlin, J. [1984]:
"Solving generalized network models in a general purpose mathematical programming system", Presented at the Joint National Meeting of ORSA/TIMS in Dallas.

Zenios, S. [1986]:
"Sequential and parallel algorithms for convex generalized network problems and related applications", PhD thesis, Civil Engineering Department, Princetion University, Princeton, N. Jersey.

Zenios, S. and Mulvey J. [1985]:
"A distributed algorithm for convex network optimization problems", Report EES-85-10, Engineering-Management Systems, Civil Engineering Department, Princeton University, Princeton, N. Jersey.