CENTER FOR PARALLEL OPTIMIZATION

GENERALIZED NETWORKS:
PARALLEL ALGORITHMS AND AN EMPIRICAL ANALYSIS

by

R. H. Clark
J. L. Kennington
R. R. Meyer
M. Ramamurti

Computer Sciences Technical Report #904

December 1989

Generalized Networks: Parallel Algorithms and an Empirical Analysis*

R.H. Clark[1], J.L. Kennington[2], R.R. Meyer[1] and M. Ramamurti[2]

1 Center for Parallel Optimization and the Computer Sciences Department
The University of Wisconsin-Madison
Madison, Wisconsin 53706 USA

2 Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275 USA

Abstract

The objective of this research was to develop and empirically test simplex-based parallel algorithms for the generalized network optimization problem. Several parallel algorithms were developed that utilize the multitasking capabilities of the Sequent Symmetry S81 multiprocessor. The software implementations of these parallel algorithms were empirically tested on a variety of problems produced by two random problem generators and compared with two leading state-of-the-art serial codes. Speedups on fifteen processors ranged from 2.6 to 5.9 for a test set of fifteen randomly generated transshipment problems. A group of six generalized transportation problems yielded speedups of up to 11 using nineteen processors. An enormous generalized transportation problem having 30,000 nodes and 1.2 million arcs was optimized in approximately ten minutes by our parallel code. A speedup of 13 was achieved on this problem using fifteen processors.

* This research was supported in part by NSF grant CCR-8709952, the Air Force Office of Scientific Research under grants AFOSR-87-0199 and 89-0410 , the Department of Defense under contract number MDA 903-86-C0182, and the Office of Naval Research under contract number N00014-87-K-0223.

The *generalized network flow problem* (also called the *flow with gains model*) in its most general form is defined as follows:

$$\begin{aligned} \min_{x} \quad & cx \\ \text{s.t.} \quad & Gx = b \\ & 0 \leq x \leq u \end{aligned} \tag{GN}$$

where G is an $m \times n$ matrix having at most two nonzero entries in each column, c is a $1 \times n$ vector of costs, b is an $m \times 1$ vector of right-hand-sides, and u is an $n \times 1$ vector of upper bounds. Associated with each matrix G is a graph $[V, E]$, where V is a set of nodes and E is a set of pairs of nodes (edges). The nodes correspond to the rows of G and the edges correspond to columns of G. 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 a single rooted spanning 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 shows a forest of quasi-trees.

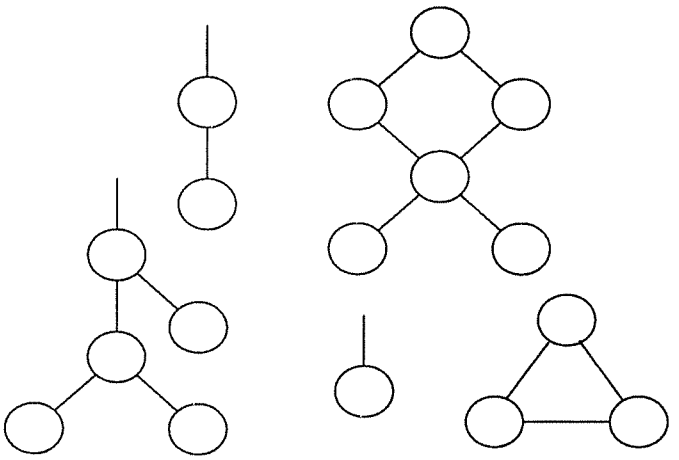

Figure 1 A Forest of Quasi-Trees

1. Survey and Overview

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 any arc may gain or lose flow at a linear 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]. The graphical structure of a basis for G allows the use of labeling procedures for basis representation. 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 code 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]. [Tomlin 84] developed the first assembly language code which is part of Ketron's MPS III system. Recently, other codes have been developed by Enquist and Chang (see [Enquist and Chang 85]), Mulvey and Zenios (see [Mulvey and Zenios 85]), and by Ali, Charnes, and Song (see [Ali, et al, 86]). The first parallel generalized code 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. 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, a modification of GENNET, and GRNET2 are discussed in [Muthukrishnan 88] and [Clark and Meyer 88] respectively. Computational results for these two codes will be given in Section 3. TPGRNET, a parallel code that assigns distinct tasks to different processes is discussed in [Clark and Meyer 88] and additional results for this code are given in Section 3. A summary of this prior software may be found in Table I.

In Section 2, a brief discussion of some strategies for parallelizing the primal simplex method will be given, along with detailed discussions of two codes PGRNET and TPGRNET. PGRNET executes pivots and computes reduced costs in parallel. TPGRNET computes reduced costs in parallel and overlaps this with the serial execution of pivots.

Table I Survey of generalized network codes

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 (parallel)	FORTRAN	Chang, M., Engquist, M. Finkel, R., Meyer, R.	1987
PGRNET (parallel)	FORTRAN	Clark, R., Meyer, R. Chang, M.	1987
GNO/PC	C	Nulty, W., Trick, M.	1988
GRNET-A	ASSEMBLER	Chang, M., Cheng, M. Chen, C.	1988
GENFLO	FORTRAN	Muthukrishnan, R.	1988
GRNET2 (serial)	FORTRAN	Clark, R., Meyer, R. Chang, M.	1989
TPGRNET (parallel)	FORTRAN	Clark, R., Meyer, R.	1989

The test problems discussed in Section 3 are generated by 1) NETGEN [Klingman, et al, 74], a pure network problem generator, 2) GNETGEN, a generalized network problem generator based on NETGEN, and 3) MAGEN [Clark and Meyer 88], a generalized network problem generator based on GTGEN [Chang and Engquist 86].

In Section 3, it is established that the two serial codes GRNET2 and GENFLO are competitive with GENNET, a state-of-the art generalized network code discussed in [Brown and McBride 84]. This comparison is made by giving results for NETGEN and GNETGEN problems. Next, results are given for TPGRNET for a group of transshipment problems

generated by GNETGEN with up to 6,000 nodes and up to 50,000 arcs. Speedups on the Sequent Symmetry S81 range from 2.6 to 5.9. Results are then given for PGRNET and TPGRNET for a group of transportation problems generated by MAGEN. All of these problems have 30,000 nodes and over 320,000 arcs. Speedups on the Sequent Symmetry S81 range from 3.8 to 11.1 for PGRNET and from 3.7 to 6.6 for TPGRNET.

2 SIMPLEX ALGORITHMS FOR GENERALIZED NETWORKS

2.1 Serial Primal Simplex Algorithms

In this section we briefly discuss the specialization of the primal simplex method for generalized networks. A more detailed discussion can be found in [Kennington and Helgason 80] and [Jensen and Barnes 80].

Input:

1. A graph $[V, E]$.
2. A cost $c[e]$ and arc capacity $u[e]$ for each $e \in E$.
3. The generalized constraint matrix G.
4. A requirement $r[n]$ for all $n \in V$.

Output:

1. The termination type indicator β and flow array $\overline{x}[e]$. ($\beta = 1$ implies that the problem is unbounded, $\beta = 2$ implies that the problem has no feasible solution, and $\beta = 3$ implies that the optimal solution is given in $\overline{x}[e]$.)

The primal simplex algorithm for generalized networks can be divided into three subroutines, PRICE, RATIO and UPDATE. These correspond to the computation of reduced costs, the ratio test, and the basis update in the simplex method for general LP's. Each of the subroutines makes use of the block diagonal (and nearly triangular) nature of the bases for (GN), as discussed in [Adolphson 82] and [Barr, Glover and Klingman 79]. The primal simplex algorithm can be summarized as follows:

Procedure SIMPLEX

begin

1. $\beta \leftarrow 0$
2. initialize duals (π)
3. call module PRICE
4. if $\beta \neq 0$, then terminate
5. call module RATIO
6. if $\beta \neq 0$ terminate
7. call module UPDATE
8. goto 3.

end

In module PRICE, reduced costs are computed for arcs (variables) by using the formula $r_{ij} = (c - \pi G)_{ij}$, where r_{ij} is the reduced cost for arc (i, j). The expression $(c - \pi G)_{ij}$ has at most three terms, since G has at most two non-zero entries in each column. The heuristics employed by GRNET2, PGRNET and TPGRNET to determine which arcs to price and to use as pivots will be discussed later, and involve maintaining candidate lists of pivot eligible arcs and managing the candidate lists in different ways. In module RATIO, the ratio test for a given pivot-eligible arc is performed by identifying the incident quasi-trees (i.e. the incident basis components) and following the path from each end of the arc to the (generalized) root of the corresponding quasi-tree(s). As this traversal is made, the flow on the basic arcs in the path is checked, and the arc with the "minimum ratio" is selected as the outgoing arc. In module UPDATE, flows as well as other tree data structures are updated.

GENFLO and GRNET2 are implementations of this algorithm, and they will be shown to be comparable in speed. However, the two codes differ in a few ways. GENFLO is a modification of GENNET, described in [Brown and McBride 84] and it uses the GENNET pricing strategy. This strategy involves selecting a node and pricing out all of its incident arcs. GRNET2, on the other hand, scans its list of arcs linearly to locate pivot eligible arcs and doesn't attempt to focus on the arcs that are adjacent to some node. GRNET2 is specialized to solve problems for which at least one of the multipliers defined for each arc is equal to 1, while GENFLO allows each arc to have two arbitrary associated multipliers. The latter approach is desirable in integer programming applications since scaling variables to make one of the multipliers equal to 1 may destroy integrality. Also, reflecting an arc to convert a negative multiplier into a 1 requires that the arcs have upper bounds. Both GENNET and GRNET2 use the "little m" (or "gradual penalty") method [Grigoriadis 84] to find a feasible solution. Under this method, a moderate initial cost is given to the artificial arcs, and the resulting problem is approximately solved. Next, the cost on the artificial arcs is increased to create a new problem, and the optimal (or nearly optimal) basic feasible solution from the last problem is used as a warm start for the new one. This process of gradually increasing the cost on the artificial arcs and solving a sequence of "easy" problems can be shown empirically to yield a vast improvement over the "big M" method in terms of the total number of pivots required to solve a problem and in terms of the total CPU time. Some tests with GRNET2 have shown that the "little m" method is 29 times faster than the "big M" method for large problems having only one quasi tree in the optimal basis. GENFLO uses a simple closed-form expression to calculate the cost of the artificial arcs at each iteration. (The initial cost for the GENFLO artificial arcs

is about 200. The cost on the artificial arcs is then roughly doubled at each step in the gradual penalty method.) The initial cost for the GRNET2 artificial arcs is 20 (assuming that the cost range for the regular arcs is 1-100). GRNET2 adds 5 to the cost on the artificial arcs after each step until the cost reaches 130, then for the next two steps the increment is by 10, and for the last two steps the increment is by 20. Finally, the cost is increased to "big M" and the problem is solved to optimality. An empirical comparison of these two codes with each other and with MPSX, the IBM general LP code, is given below.

2.2 Parallel Simplex Algorithms

In [Clark and Meyer 87] an implementation of PGRNET is discussed. This code executes in parallel both pivots and pricing. Pivots are executed in parallel only if they involve updating separate quasi-trees (basis components). Even if the basis has only one component at optimality, this algorithm behaves quite efficiently during the beginning of the solution process, because the initial starting basis has as many components as nodes. PGRNET is used in [Clark and Meyer 87] to solve problems generated randomly by GTGEN [Chang and Engquist 86], and a code called MPGRNET is used to solve randomly generated multiperiod problems with a block diagonal structure generated by MPGEN [Chang 86]. MPGRNET reduces contention between processors by allocating specific quasi-trees to specific processors and allowing processors to execute pivots involving their quasi-trees (and only their quasi-trees) until they can find no "local" pivot eligible arcs. Optimality is then achieved by reverting to the PGRNET algorithm. Speedups for PGRNET in [Clark and Meyer 87] range from 4.8 to 8.8 on 7 processors, and speedups for MPGRNET algorithm ranged from 8.8 to 36.9 on 12 processors. The superlinear speedup for some problems led the authors to develop a serial program that was much more efficient for the block diagonal, or "multi-period" problems. The resulting serial timing results yielded speedup results for MPGRNET that were slightly sublinear. An improved version of PGRNET is discussed in [Clark and Meyer 88], and in Section 2.3 below.

A number of parallel algorithms for GN are discussed in [Muthukrishnan 88]. The "Chaotic Column Partitioning Algorithm" (CCP) is similar to PGRNET in that it allows pivots to be executed in parallel, provided that the pivots involve updating separate quasi-trees. Another algorithm, known as the "Column Partitioning Algorithm" (CP) is similar to MPGRNET. The (CCP) algorithm, the (CP) algorithm and other algorithms are applied in [Muthukrishnan 88] to problems generated randomly by GNETGEN, a modification of

NETGEN [Klingman, et al, 74]. Speedups for the (CCP) algorithm on 8 processors range from 1.27 to 1.73, and speedups for (CP) on 8 processors range from 1.33 to 2.48.

In Section 2.4 we discuss TPGRNET [Clark and Meyer 88], an algorithm that devotes one processor to the task of executing pivots, and devotes all other processors to the task of computing reduced costs and managing candidate lists. (TPGRNET denotes "Task Parallel GRNET".) An algorithm (the Data Partitioning Algorithm) that is similar to TPGRNET in its partitioning of tasks is discussed in [Muthukrishnan 88] and is applied to a set of generalized networks defined on grids. The advantage to having one processor do all pivoting is that there is no contention between processors for shared data structures, even when there is only one basis component in the optimal basis. This strategy yields an algorithm that is robust in the sense that it has a behavior that does not depend heavily on the nature of the optimal basis, and hence is applicable to general LP's.

2.3 PGRNET (Parallel GRNET)

Figure 2 gives a flow chart for PGRNET. Parallel portions of the code are emphasized by parallel lines. "l.t." designates *list_threshold*, a candidate list parameter.

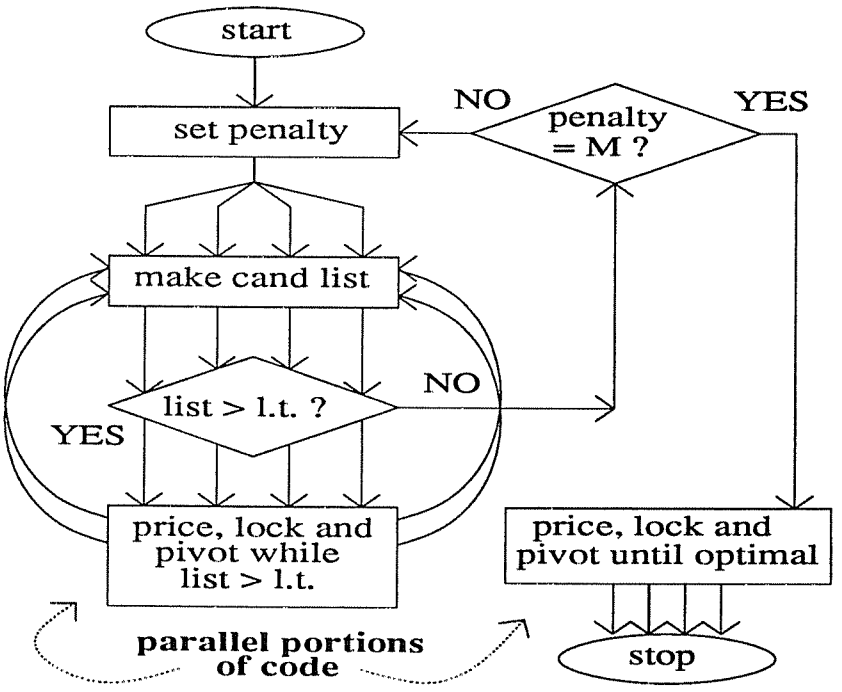

Figure 2. Flow Chart For Parallel Algorithm PGRNET

The (parallel) PGRNET algorithm can be summarized as follows:

PGRNET

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 selected 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, 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 roughly equally between segments. If there are n arcs and P segments, then processor 1 has arcs (1) through $\lceil n/P \rceil$, processor 2 gets arcs $\lceil n/P \rceil + 1$ through $\lceil 2n/P \rceil$ 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 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 serially, 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" . 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 no more than *list_threshold* arcs, p develops a new candidate list. If the new candidate list also has no 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 a *verification of optimality* stage. 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 , and continues pricing. If the processor prices 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.

2.4 TPGRNET (Task-Parallel GRNET)

This algorithm is divided into two main stages, but fewer than 1% of all pivots are executed in Stage 2. During Stage 1, different tasks are allocated to different processors. One processor executes all pivots, one processor selects pivot arcs for the pivoting processor, and all other processors do pricing and place pivot eligible arcs into a shared candidate list to be scanned by the selecting processor. If a pivot requires updating a large quasi-tree, the pivoting processor can request the help of the pricing processors by putting the root nodes of subtrees on a queue. When these root nodes are detected by the pricing processors, they take them off the queue and update the duals in the corresponding subtrees. In Stage 2, each processor scans a segment of the arc list belonging to that processor. When a processor finds a pivot eligible arc, it locks the quasi-trees at the ends of the arc, to temporarily exclude all other processors from modifying those quasi-trees, and executes the pivot. No candidate lists are developed in Stage 2. More details of the algorithm are given below. Figure 3 illustrates the flow of information during Stage 1, and Figure 4 gives a flow chart for TPGRNET. In both of these figures, dotted arrows indicate the direction of the flow of information.

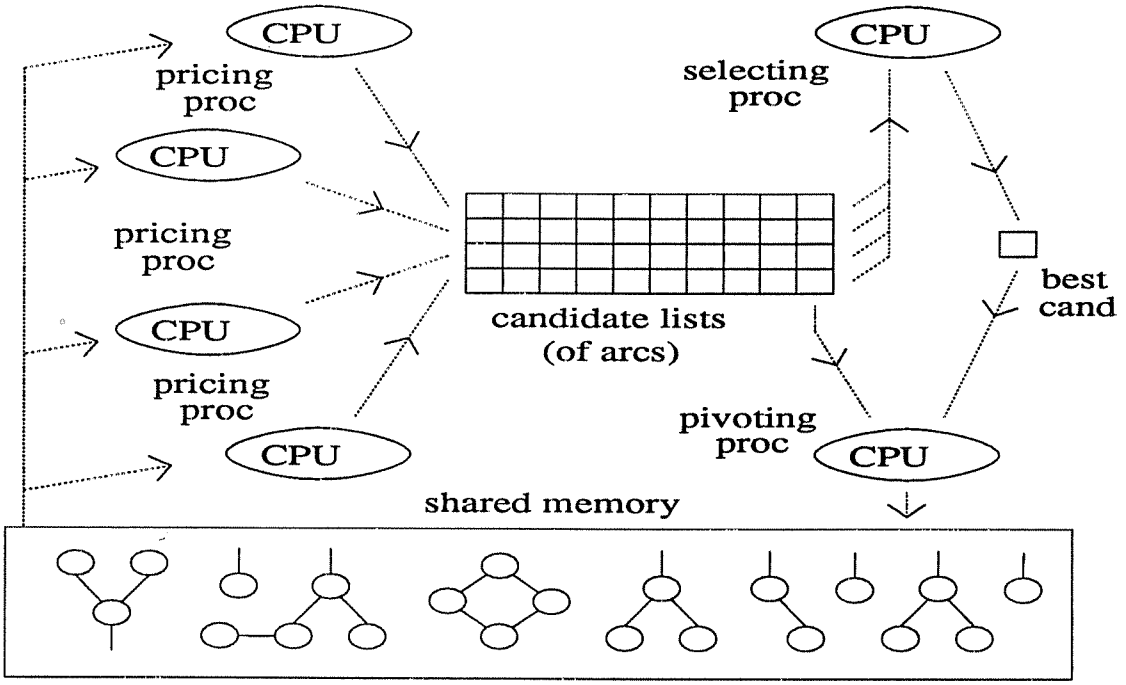

Figure 3 Flow of Information in TPGRNET, Stage 1

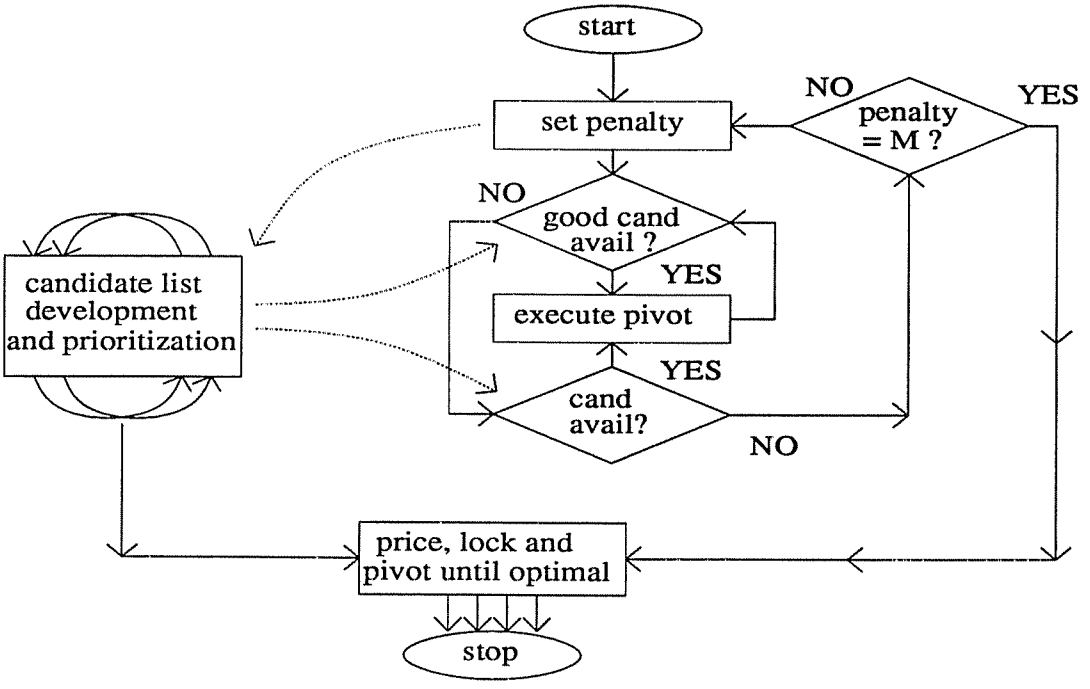

Figure 4 Flow Chart for Parallel Algorithm TPGRNET

The (parallel) TPGRNET algorithm is:

TPGRNET

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.

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.

We will now describe in detail 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 shared candidate lists of length 10. 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 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 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 each of the candidate lists 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 accepting 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 nodes of subtrees onto the dual update queue, and the pricing processors will then assume the tasks of updating the duals on those subtrees.

3 COMPUTATIONAL EXPERIENCE

3.1 Results for pure network problems

Pure networks are a special case of generalized networks (all multipliers have a magnitude of 1). In order to compare the efficiency of general versus specific codes, we consider the relative performance of a general simplex code (MPSX), two generalized network codes, and a pure network code on a class of pure network test problems. Table II gives results for a collection of problems generated by NETGEN [Klingman, et al, 74]. The problem numbers have the prefix "N", to indicate that they were generated by NETGEN, and a numerical suffix that indicates the standard NETGEN problem number [Klingman, et al, 74]. 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 is designed to solve pure network problems, and it utilizes the tree structure of bases and uses integer arithmetic. GENNET uses an improved pricing strategy that reduces the total 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 GENFLO allows for two arbitrary multipliers. These results clearly justify the utility of specialized generalized network software that is only slightly slower than a pure network code.

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

	Size		MPSX		GENFLO		GENNET		NETFLO	
Problem	nodes	arcs	pivots	secs.	pivots	secs.	pivots	secs.	pivots	secs.
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	20,467	20.23	21,902	15.75	47,919	12.92

3.2 Results for GNETGEN generalized network problems

NETGEN has been modified by D. Klingman to generate generalized network flow problems. The modified generator is called GNETGEN. Table III gives most of the GNETGEN input data for the small test problems G1 through G7. 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]. (The random seed for all of these problems is 13502460.)

Table III Input data for small GNETGEN problems

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
% Capacitated	0	100	100	0	0	100	100
Bound Range	—	1-2k	1-2k	—	—	1-2k	4-6k

Table IV 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. Note that relaxing the assumption that one of the multipliers is unity results in an increase in computing time of only about 10%.

Table IV Serial results for small GNETGEN problems (IBM 3081-D24)

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

Tables V through VII give the input data for the larger GNETGEN problems G8 through G22. 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 problems are grouped according to rectangularity (arcs/nodes).

Table V GNETGEN problems G8-G13

	Problems					
Characteristics	**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
% Capacitated	100	50	0	100	50	0
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100
Bound Range	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 VI GNETGEN problems G14-G16

	Problems		
Characteristics	**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
% Capacitated	100	50	0
Cost Range	1-100	1-100	1-100
Bound Range	1k-2k	1k-2k	—
Bound Range	0.5-1.5	0.5-1.5	0.5-1.5

Table VII GNETGEN problems G17-G22

	Problems					
Characteristics	**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
% Capacitated	100	50	0	100	50	0
Cost Range	1-100	1-100	1-100	1-100	1-100	1-100
Bound Range	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 VIII gives a comparison of GENFLO and GRNET2 for problems G8 through G22. The two programs give nearly the same performance for these test problems, despite the fact that the two codes have very different pricing strategies. The number of pivots for GRNET2 is about 15% less than for GENFLO, and the total time for GRNET2 on the test problem set is about 4% less.

Table VIII Serial results for large GNETGEN problems (Sequent S81)

	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 IX 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 given is always taken from the faster of the two codes. The time totals from the bottom of Table IX are graphed in Figure 5.

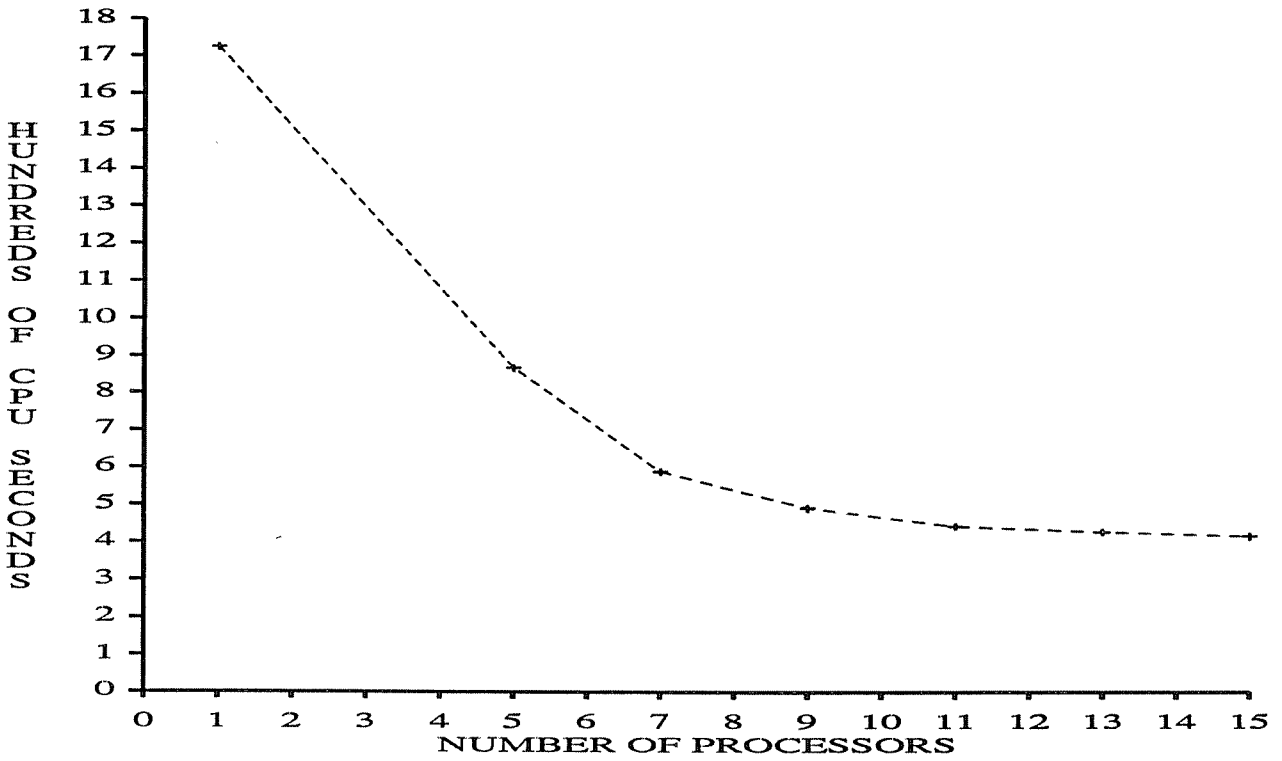

Figure 5 Composite results for TPGRNET

Table IX TPGRNET timings for problems G8 through G22

					Number of Processors (Sequent)						
Prob	nds	arcs	cap	qtree	1	5	7	9	11	13	15
G8	2k	13k	100	4	T51.9	29.3	19.8	16.2	16.3	16.9	15.8
G9	2k	13k	50	1	O44.2	24.6	16.7	14.0	12.6	13.1	13.8
G10	2k	13k	0	2	T48.5	25.2	17.5	14.2	12.9	13.2	13.3
G11	4k	26k	100	3	T121.9	58.3	43.8	34.3	35.6	33.8	31.8
G12	4k	26k	50	1	T115.1	61.2	41.1	36.4	33.5	31.6	31.4
G13	4k	26k	0	3	O99.3	56.8	38.7	34.0	29.0	29.9	29.6
G14	6k	39k	100	5	T142.2	87.5	58.4	50.3	44.3	45.1	43.8
G15	6k	39k	50	7	O126.3	87.0	65.5	51.6	48.1	46.5	47.0
G16	6k	39k	0	2	O141.0	89.5	61.3	51.3	45.8	40.8	40.8
G17	2k	25k	100	9	O89.2	39.0	25.2	20.2	19.2	18.1	16.8
G18	2k	25k	50	3	T78.4	41.3	29.8	22.5	22.5	21.2	19.0
G19	2k	25k	0	3	T81.3	41.8	25.7	21.9	18.4	18.1	18.2
G20	2k	50k	100	2	T194.8	77.5	49.3	44.3	35.0	34.8	33.0
G21	2k	50k	50	3	T194.9	83.4	50.8	42.2	36.5	31.7	33.5
G22	2k	50k	0	1	T133.8	64.4	44.5	37.9	32.1	32.0	27.9
Totals					1723.5	866.8	588.1	491.3	441.6	426.8	415.7

Table X gives speedup results for problems G8 through G22, and the results are graphed for problems G14, G15, G16 and problems G20 ,G21, and G22. Problems G14 , G15, and 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 , G21 and G22. These are the problems for which the arcs/nodes ratio is the largest. Since TPGRNET gets most of its parallelism from pricing 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.

Table X TPGRNET speedups for problems G8 through G22

					Number of Processors (Sequent)						
Prob	nds	arcs	cap	qtree	1	5	7	9	11	13	15
G8	2k	13k	100	4	1.0	1.7	2.6	3.2	3.1	3.0	3.2
G9	2k	13k	50	1	1.0	1.7	2.6	3.1	3.5	3.3	3.2
G10	2k	13k	0	2	1.0	1.9	2.7	3.4	3.7	3.6	3.6
G11	4k	26k	100	3	1.0	2.0	2.7	3.5	3.4	3.6	3.8
G12	4k	26k	50	1	1.0	1.8	2.8	3.1	3.4	3.6	3.6
G13	4k	26k	0	3	1.0	1.7	2.5	2.9	3.4	3.3	3.3
G14	6k	39k	100	5	1.0	1.6	2.4	2.8	3.2	3.1	3.2
G15	6k	39k	50	7	1.0	1.4	1.9	2.4	2.6	2.7	2.6
G16	6k	39k	0	2	1.0	1.5	2.3	2.7	3.0	3.4	3.4
G17	2k	25k	100	9	1.0	2.2	3.5	4.4	4.6	4.9	5.3
G18	2k	25k	50	3	1.0	1.8	2.6	3.4	3.4	3.6	4.1
G19	2k	25k	0	3	1.0	1.9	3.1	3.7	4.4	4.4	4.4
G20	2k	50k	100	2	1.0	2.5	3.9	4.3	5.5	5.5	5.9
G21	2k	50k	50	3	1.0	2.3	3.8	4.6	5.3	6.1	5.8
G22	2k	50k	0	1	1.0	2.0	3.0	3.5	4.1	4.1	4.7
Averages					1.0	1.9	2.9	3.5	3.8	4.0	4.1

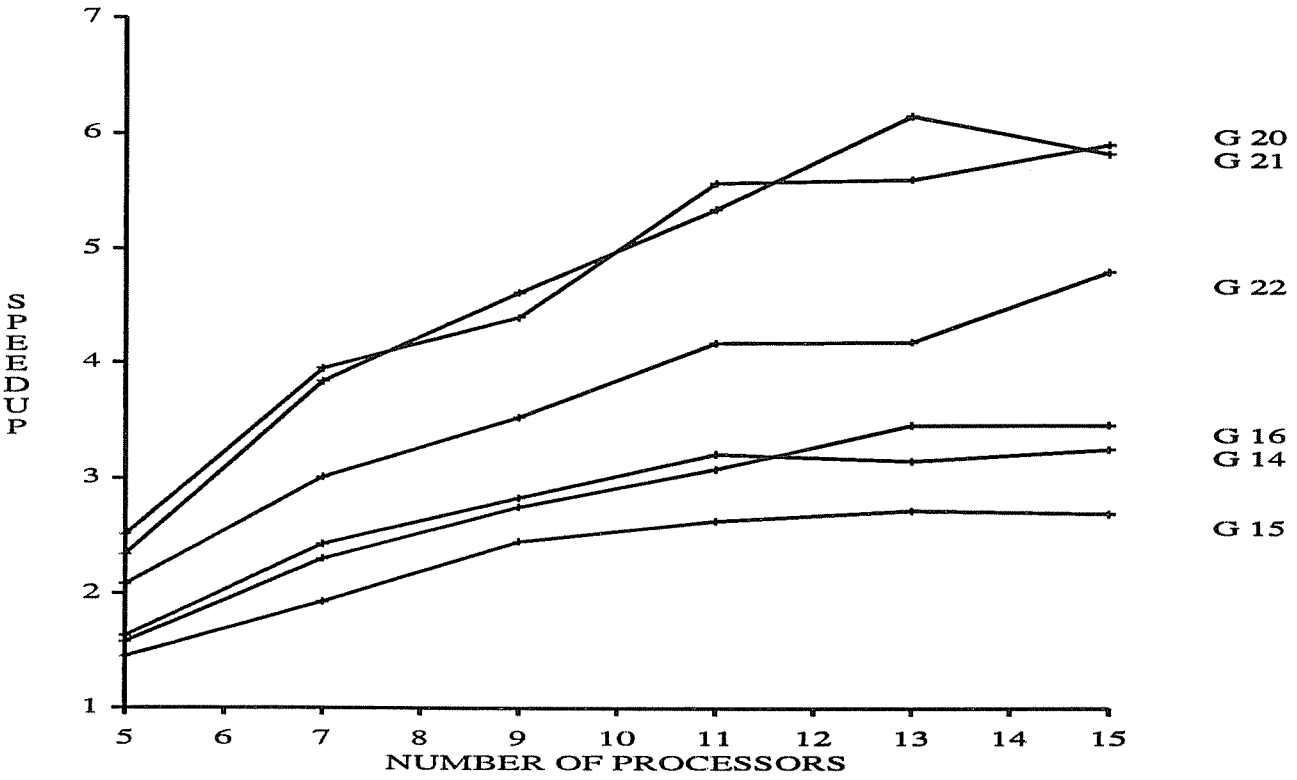

Figure 6 Speedups for TPGRNET

3.3 Results for MAGEN Problems

Table XI gives results for a group of large problems generated by MAGEN, the generator used in [Clark and Meyer 88]. This is a modification of GTGEN, a generator described in [Chang and Engquist 86]. All problems have 30,000 nodes and more than 300,000 arcs. The precise MAGEN input data for these problems, as well as optimal objective function values are given in [Clark 89]. MAGEN generates bipartite generalized network problems randomly, but allows the user to specify, roughly, the granularity of the generated problem, (i.e. the user may roughly specify the number of quasi-trees in the optimal basis). Since problems 4.00 through 4.50 have different granularities, the effect of granularity on the efficiency of TPGRNET and PGRNET can be studied by solving these problems. Problem 4.50 is by far the easiest problem in terms of CPU time. GRNET2 solves 4.50 sixteen times faster than 4.00, even though it does only 30% fewer pivots. This means that the pivots in 4.50 are relatively fast , probably due to the fact that quasi-trees are relatively small. 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). 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. Looking at problem 4.00, one sees that the fastest serial algorithm is GRNET2, and the fastest parallel algorithm is TPGRNET. In all of the other problems, PGRNET outperforms TPGRNET in terms of CPU time because it is able to execute pivots in parallel. The shared candidate list and parallel pricing strategy of TPGRNET makes TPGRNET outperform PGRNET in terms of the number of pivots for all problems, but for problem 4.00, the absence of lock contention makes TPGRNET 40% faster than PGRNET.

Table XI Results for PGRNET, TPGRNET, GENFLO, and GRNET2

Problem #	**4.00**	**4.01**	**4.03**	**4.05**	**4.10**	**4.50**
# qtrees at optimality	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 GENFLO	***	411,720	337,907	313,982	289,937	244,635
# pivots GRNET2	328,711	347,420	320,937	308,284	288,856	228,819
# pivs 19 procs PGRNET	365,633	383,483	345,325	326,323	300,496	231,507
# pivs 19 procs TPGRNET	265,774	305,979	288,279	272,322	263,411	221,544
CPU secs. GENFLO	***	36,523	10,524	5,121	2,436	1,038
CPU secs. GRNET2	22,434	9,679	4,525	3,096	2,340	1,388
CPU: 19 procs PGRNET	5,829	1,527	589	364	223	124
CPU: 19 procs TPGRNET	3,390	2,571	1,177	721	470	211
Speedup PGRNET	3.8	6.3	7.6	5.2	10.4	11.1
Speedup TPGRNET	6.6	3.7	3.8	4.2	4.9	6.5

*** Did not finish after 14 hours.

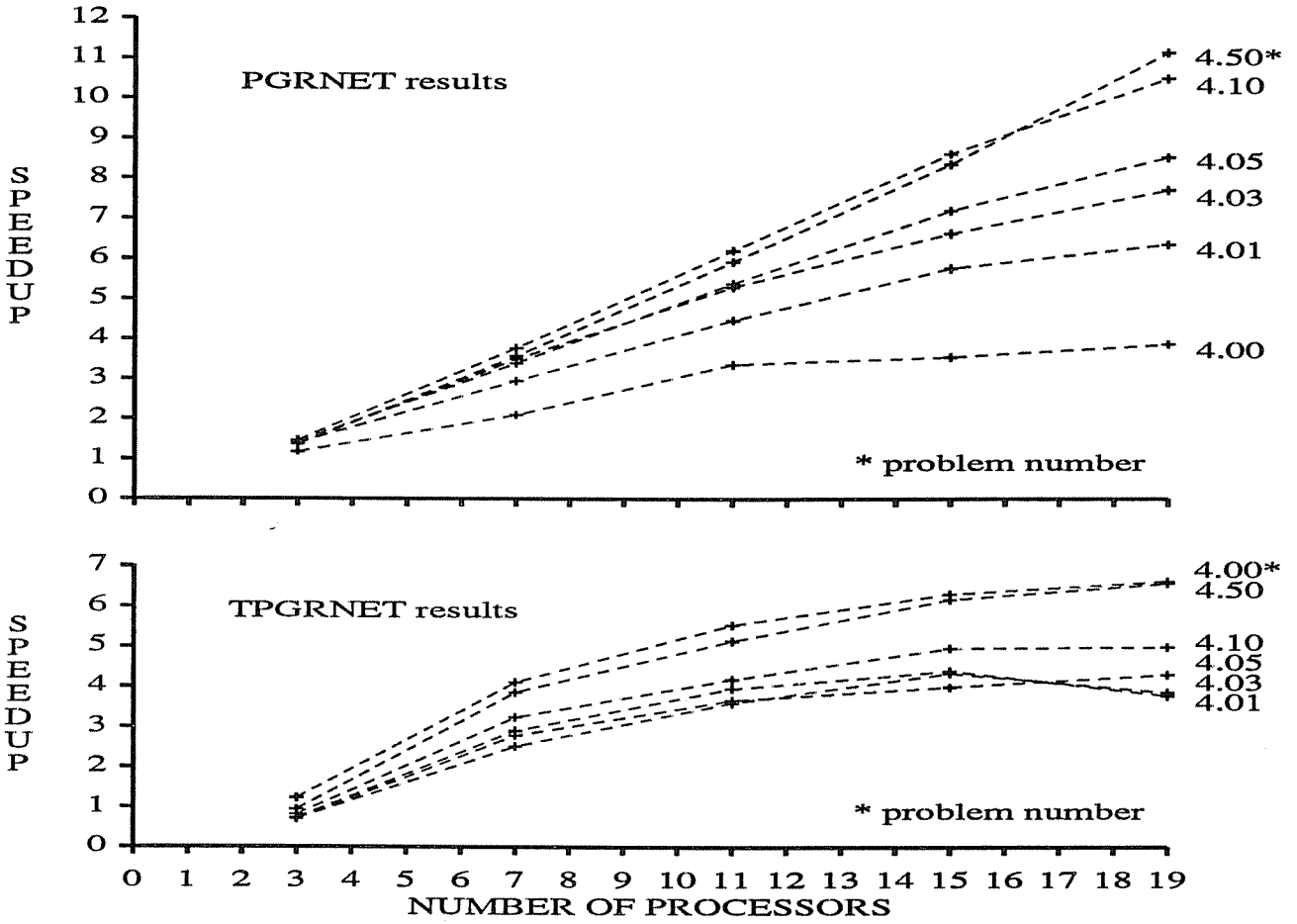

Figure 7 Speedups for PGRNET and TPGRNET for MAGEN problems

Figure 8 and Tables XII and XIII show results for two problems with more than a million variables. The problem reported in Table XII has tighter capacity constraints than does the Table XIII problem. Both of these problems are small grained, so they can be solved quite efficiently by PGRNET. The best speedup was achieved on the tightly constrained problem for which a speedup of 13 was achieved using 15 processors. Note that the tightly constrained problem was considerably more difficult to solve with the serial version of the code, so that there was more potential for improvement with a parallel approach.

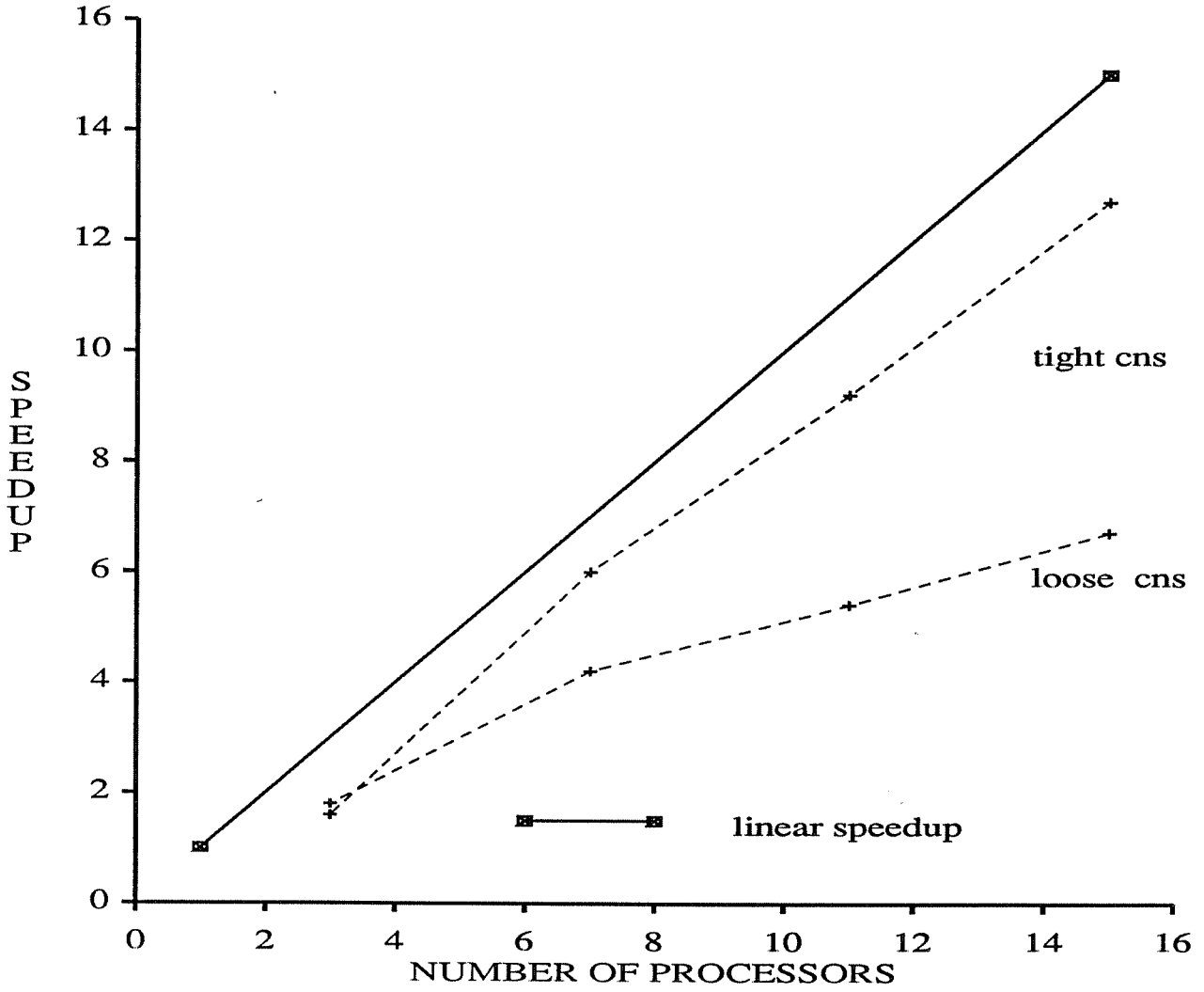

Figure 8 Speedups for PGRNET (# arcs > 1,000,000)

Table XII PGRNET results for tightly constrained problem

program	# arcs	# nodes	# qtrees	time	pivots	maj page swap
serial	1,267,185	30,000	14,859	8,415	706,776	914,478
15 procs	1,267,185	30,000	14,859	660	715,168	101,866

Table XIII PGRNET results for loosely constrained problem

program	# arcs	# nodes	# qtrees	time	pivots	maj page swap
serial	1,267,185	30,000	14,859	3,305	184,379	363,755
15 procs	1,267,185	30,000	14,859	490	186,969	45,672

4. SUMMARY AND CONCLUSIONS

The availability of relatively inexpensive parallel computers has generated widespread interest in the development of new optimization algorithms for such machines. Although parallel computers come in a variety of architectures, the popularity of multiple-instruction multiple-data (MIMD) machines can be attributed, in part, to the ease with which codes intended for sequential computers can be ported to these machines. The algorithms and software reported in this investigation were developed for a particular multiprocessor system with shared memory (Sequent Symmetry S81), but they can be used with any shared memory parallel processing systems.

In our empirical study we found that our generalized network software when applied to pure network problems is at least forty times faster than MPSX and when applied to generalized network problems is at least an order of magnitude faster than MPSX. It was also shown that relaxing the restriction that at least one of the multipliers associated with an arc be one (minus one), results in an additional computational expense of only ten percent.

We believe that the current best sequential software for these problems is GENNET [Brown and McBride 1984] and GRNET2 [Clark and Meyer 1988] and we began our study of parallel algorithms with these codes. A two- multiplier version of GENNET called GENFLO and GRNET2 provided the best single processor times for the empirical analysis presented in this study. For most speedup calculations, both codes were run and the smaller of the two times was used in the numerator.

One general conclusion is that problems having only a few quasi-trees at optimality are more difficult for our parallel codes than those with many partitions of an optimal basis. Nevertheless, by exploiting the parallelism in the pricing operation, we still obtained speedups of over four on a set of problems having fewer than ten quasi-trees at optimality. For problems having hundreds of quasi-trees at optimality, the speedups ranged from five to eleven. The best speedup was achieved on a tightly constrained problem having 30,000 nodes and over 1.2 million arcs. For this problem a speedup of thirteen was achieved using only fifteen processors.

References

[1] D. ADOLPHSON and L. HEUM, 1981. *Computational Experiments on a Threaded Index Generalized Network Code*, presented at the ORSA/TIMS National Meeting in Houston, Texas.

[2] D. ADOLPHSON, 1982. *Design of Primal Simplex Generalized Network Codes Using a Preorder Thread Index*, Working Paper, School of Management, Brigham Young University, Utah, Provo.

[3] I. ALI, C. CHARNES and T. SONG, 1986. *Design and Implementation of Data Structures for Generalized Networks*, **Journal of Information and Optimization Sciences 7**, 81-104.

[4] R. BARR, F. GLOVER and D. KLINGMAN, 1979. *Enhancements of Spanning Tree Labeling Procedures for Network Optimization*, **INFOR 17**, 16-34.

[5] G. BROWN and R. MCBRIDE, 1984. *Solving Generalized Networks*, **Management Science 30**, 1497-1523.

[6] M. CHANG, 1986. *A Parallel Primal Simplex Variant for Generalized Networks*, Ph.D. thesis, University of Texas at Austin.

[7] M. CHANG, M. CHENG and C. CHEN, 1988. *Implementation of New Labeling Procedures for Generalized Networks*, Technical Report, Department of CS/OR, North Dakota State University, Fargo, North Dakota.

[8] M. CHANG and M. ENGQUIST, 1986. *On the Number of Quasi-Trees in an Optimal Generalized Network Basis*, **COAL Newsletter 14**, 5-9.

[9] M. CHANG, M. ENGQUIST, R. FINKEL and R. MEYER, 1988. *A Parallel Algorithm for Generalized Networks*, **Annals of Operations Research 14**, (1988) 125-145.

[10] R. CLARK and R. MEYER, 1987. *Multiprocessor Algorithms for Generalized Network Flows*, Technical Report #739, Department of Computer Sciences, The University of Wisconsin-Madison.

[11] R. CLARK AND R. MEYER, 1988. *Parallel Arc-Allocation Algorithms for Optimizing Generalized Networks*, to appear in *Proceedings of Workshop on Supercomputers and Large-Scale Optimization*, Minneapolis, 1988.

[12] CLARK, R., 1989. *The Efficient Parallel Solution of Generalized Network Flow Problems*, Ph.D. thesis, University of Wisconsin-Madison.

[13] J. ELAM, F. GLOVER and D. KLINGMAN, 1979. *A Strongly Convergent Primal Simplex Algorithm for Generalized Networks*, **Mathematics of Operations Research 4**, 39-59.

[14] M. ENGQUIST and M. CHANG, 1985. *New Labeling Procedures for the Basis Graph in Generalized Networks*, **Operations Research Letters** 4, 151-155.

[15] F. GLOVER, J. HULTZ, D. KLINGMAN and J. STUTZ, 1978. *Generalized Networks: A Fundamental Computer Based Planning Tool*, **Management Science 24**, 1209-1220.

[16] F. GLOVER, D. KLINGMAN and J. STUTZ, 1973. *Extension of the Augmented Predecessor Index Method to Generalized Network Problems*, **Transportation Science 7**, 377-384.

[17] F. GLOVER, D. KLINGMAN and J. STUTZ, 1974. *The Augmented Threaded Index Method for Network Optimization*, **INFOR 12**, 293-298.

[18] M. GRIGORIADIS, 1984. *An Efficient Implementation of the Network Simplex Method*, **Mathematical Programming Study 26**, 83-111.

[19] P. JENSEN and J. BARNES, 1980. *Network Flow Programming*, John Wiley and Sons, New York.

[20] J. KENNINGTON and R. HELGASON, 1980. *Algorithms for Network Flow Programming*, John Wiley and Sons, New York.

[21] D. KLINGMAN, A. NAPIER and J. STUTZ, 1974. *NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Problems*, **Management Science 20**, 814-821.

[22] W. LANGLEY, 1973. *Continuous and Integer Generalized Flow Problems*, Ph.D. thesis, Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

[23] J. MULVEY and S. ZENIOS, 1985. *Solving Large Scale Generalized Networks*, **Journal of Information and Optimization Sciences 6**, 95-112.

[24] W. NULTY and M. TRICK, 1988. *GNO/PC Generalized Network Optimization System*, **Operations Research Letters 7**, 101-102.

[25] M. RAMAMURTI, 1988. *Parallel Algorithms for Generalized Networks*, Ph.D. thesis, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, Texas.

[26] J. TOMLIN, 1984. *Solving Generalized Network Models in a General Purpose Mathematical Programming System*, presented at the Joint National Meeting of ORSA/TIMS in Dallas.