(4)

A FINAL REPORT TO

AD-A210 722

MANPOWER RESEARCH AND DEVELOPMENT PROGRAM

ONE FILE COPY

by

Jeffery L. Kennington & Richard V. Helgason
Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275
(214)-692-3099

DTIC
ELECTE
JUL 18 1989
S D
CD

for

Optimization Algorithms for New Computer Architectures With Applications to Personnel Assignment Models

Approved for public release
Distribution Unlimited

June 29, 1989

ONR N00014-87-K-0223

SMU # 5-25105

89 08

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Southern Methodist University / 6b. OFFICE SYMBOL (If applicable): CSE	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code): Dallas, TX 75229	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Office of Naval Research / 8b. OFFICE SYMBOL (If applicable): ONR	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code): Office of Naval Research, 800 North Quincy Street, Arlington, VA 22217-5000	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification)	
12. PERSONAL AUTHOR(S): Jeffery L. Kennington and Richard V. Helgason	
13a. TYPE OF REPORT: Annual / 13b. TIME COVERED: FROM 88/05/01 TO 89/04/30	14. DATE OF REPORT (Yr., Mo., Day): 89/06/29 / 15. PAGE COUNT: 94
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	parallel programming, optimization, networks

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

One of the most important computer architecture innovations to appear in the market place during the last ten years is parallel processing on a shared memory multicomputer. This report presents our empirical results on a Sequent Symmetry S81 on four optimization models which are used in the area of personnel assignment. Both the detailed algorithms and computational results are presented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Jeffery L. Kennington	22b. TELEPHONE NUMBER (Include Area Code): (214)-692-3099 / 22c. OFFICE SYMBOL: CSE

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

Table of Contents

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist Avail and/or Special
A-1

DTIC COPY INSPECTED

STATEMENT OF WORK

Many Navy applications in the area of personnel assignment demand computer hardware several orders of magnitude faster than the fastest machines available. Computer designers (including Seymour Cray) have turned to parallelism as one of the more promising avenues for increased computational speed. Parallelism shifts some of the burden for increased speed from the hardware to the software engineer. Very powerful hardware (in terms of millions of floating point operations per second) can be built using many low cost standard chips, all designed to operate in parallel. Our research program objective is to develop and empirically test new parallel algorithms and software for a wide variety of optimization problems. The problems studied this past year include the the mimimal spanning tree problem, the shortest path problem, the dense assignment problem, and the generalized network model. Parallel algorithms were developed and implemented on a Sequent Symmetry S81 multicomputer having twenty computational processing units (cpu's). We also assisted the Navy Personnel Research and Development Center in developing and analyzing readiness models.

ACCOMPLISHMENTS AND PUBLICATIONS

The following is a list of the papers completed during the last year, an executive summary of the paper, and the publication status. These four papers also appear in this document.

Title

Minimal Spanning Trees: An Empirical Investigation of Parallel Algorithms, Technical Report 87-OR-02

Authors

R. S. Barr, R. V. Helgason, and J. L. Kennington

Executive Summary

This was our first empirical investigation using the Sequent Symmetry S81 multicomputer for parallel processing research. This problem was selected because the algorithms for sequential machines are straightforward and we

could concentrate our efforts on parallelization and learning the idiosyncrasies of the S81 system. We achieved speedups which ranged from a low of 2.79 to a high of 6.81 using ten processors.

Publication Status

This paper has been accepted for publication in Parallel Computing.

Title

Dijkstra's Two-tree Shortest Path Algorithm, Technical Report 88-OR-13

Authors

R. V. Helgason, J. L. Kennington, and B. D. Stewart

Executive Summary

The problem of finding the shortest path between a designated pair of nodes in a graph is a fundamental problem in operations research which also serves as a building block for other algorithms such as the out-of-kilter algorithm and the shortest augmenting path algorithm. The classical Dijkstra algorithm begins at one of the designated nodes and fans out from this node until the other designated node becomes a member of the labeled set. In this investigation we empirically demonstrate that a better algorithm is obtained by a procedure that begins at both designated nodes and fans out in both directions. This is accomplished mathematically by building trees rooted at the two designated nodes. The algorithm stops when any node appears in both trees.

Publication Status

Additional empirical tests have been performed on a wide variety of problems and the paper is being revised to reflect this analysis. We anticipate submitting the revised paper for publication by the end of August.

Title

An Empirical Analysis of the Dense Assignment Problem, Technical Report 88-OR-16

Authors

J. Kennington and Z. Wang

Executive Summary

There are three methods for comparing algorithms, (i) worst case analysis, (ii) average case analysis, and (iii) empirical analysis. Each technique has its strengths and each has its weaknesses. We performed a thorough empirical analysis comparing the auction algorithm with the shortest augmenting path algorithm (SAP) for the dense assignment problem. We found that the software implementation of the SAP algorithm was superior on serial machines. A parallel implementation of this software yielded speedups of four using ten processors. It appears to us that this problem is solved. We successfully solved problems having over one million arcs in less than 20 seconds on a Sequent Symmetry S81.

Publication Status

This paper has been submitted for publication and is currently under review.

Title

Solving Generalized Network Problems on a Shared Memory Multiprocessor, Technical Report 88-OR-21

Authors

J. Kennington and R. Muthukrishnan

Executive Summary

The generalized network problem in its most general form is a special case of a linear program in which every column of the constraint matrix has at most two nonzero elements. Special cases of this model include the flow with gains

problem, the minimal cost network flow problem, the transportation problem, the assignment problem, the maximal flow problem, and the shortest path problem. In this study we empirically demonstrate that specialized software for this model is an order of magnitude faster than MPSX. This algorithm was parallelized and speedups of from two to three were achieved on a Sequent Symmetry S81 multicomputer using eight processors.

Publication Status

Professor Robert Meyer and his Ph.D. student Robert Clark of the University of Wisconsin have independently and simultaneously been working on a parallelization of the simplex algorithm for the generalized network problem. We are combining forces with this group to combine our work presented in this paper with their work in an attempt to write the definitive paper on the subject. Professor Kennington visited the University of Wisconsin in June 1989, and we expect to complete this joint paper by the end of August 1989.

PERSONNEL

Faculty

Jeffery L. Kennington

Richard V. Helgason

Ph.D Students

Muthukrishnan Ramamurti
Ph.D. received May 1989
Dissertation Title: Parallel Algorithms for Generalized Networks

Levent Hatay,
D. Eng. received May 1989
Praxis Title: Sequential and Parallel Algorithms for the Shortest-Path Problem

Kumar Thiagarajan

Zhiming Wang

Betty Hickman

PRESENTATIONS

Denver ORSA/TIMS Fall 1988

An Asynchronous Algorithm to Solve Generalized Network Problems, J. Kennington & R. Muthukrishnan

Hierarchical Graph Partitioning for *MIMD Computers*, R. Barr, R. Helgason, D. Matula, & K. Thiagarajan

The Transportation Problem: A Shortest Augmenting Path Algorithm, Z. Wang

Vancouver CORS/TIMS/ORSA Spring 1989

Network Flow Problems: *Parallel Algorithms and Computational Experience*, J. Kennington

Identifying the Vertices of the Convex Hull of a Finite Set of Points, J. Dula, R. Helgason, & K. Thiagarajan

Solving Assignment Problems on a Shared Memory Multiprocessor, Z. Wang

Parallelization Strategies for the Network Simplex Algorithm, B. Hickman & R. Barr

INTERACTION WITH NAVY PERSONNEL RESEARCH AND DEVELOPMENT CENTER

The Navy Personnel Research and Development Center located at San Diego, California is a co-sponsor of this research. Professor Kennington met with

Dr. Iosef Krass at the Denver ORSA/TIMS Meeting and with Mr. Ted Thompson and Mr. Joe Blanco at the Vancouver CORS/ORSA/TIMS Meeting. Professor Kennington visited NPRDC during the summer of 1988 and gave a seminar on parallel processing to the operations research group. We have also had many phone conversations with both Dr. Timothy Liang, Mr. Ted Thompson, and Mr. Alan Whisman regarding several of their models. Our main goal in this work is basic research, but we also serve NPRDC as technical advisors and provide them with specialized network software that our group has developed over the years.

Technical Report 87-OR-02

MINIMAL SPANNING TREES: AN EMPIRICAL INVESTIGATION OF PARALLEL ALGORITHMS

by

R.S. Barr

R.V. Helgaon

and

J.L. Kennington

Department of Operations Research and Engineering Management
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275

revised March 1989

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

The objective of this investigation is to empirically test parallel algorithms for finding minimal spanning trees. Computational tests were conducted on three serial versions of Prim's algorithm, a serial version of Kruskal's algorithm, and a serial version of the Sollin's algorithm. For complete graphs, our implementation of the Prim algorithm is best. As the graph density is reduced, our implementation of Kruskal's algorithm is superior, and for very sparse graphs, the Sollin algorithm dominates. Parallel implementations of both the Prim algorithm and the Sollin algorithm were empirically tested on a Sequent Symmetry S81 multicomputer. In tests involving ten processors, the speedups ranged from a low of 2.79 to a high of 6.81.

ACKNOWLEDGMENT

This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C-0182, the Air Force Office of Scientific Research under contract Numbers AFOSR 83-0278 and AFOSR 87-0199, the Office of Naval Research under Contract Number N00014-87-K-0223, and ROME Air Development Center under Contract Number SCEEE PDP/86-75. The authors wish to express their appreciation to Professor Hossam Zaki of the University of Illinois and Professor Iqbal Ali of the University of Massachusetts at Amherst for their helpful comments.

I. INTRODUCTION

The new generation of computing hardware based on parallel processing technology requires that algorithm engineers redesign and reevaluate the standard methods. It may well be that algorithms which proved to be superior for single-processor machines may prove to be inferior in some of the new parallel processing environments. One of the more popular new parallel machines is Sequent Computer Systems' Symmetry S81. The objective of this investigation is to computationally test parallel algorithms for finding minimal spanning trees on a twelve-processor Symmetry S81.

An undirected graph G = [V,E] consists of a vertex set V and an edge set E. Without loss of generality we assume that the edges are distinct. If G' = [V, E'] is a subgraph of G, then G' is called a spanning subgraph for G. If, in addition, G' is a tree, then G' is called a spanning tree for G. A graph whose components are trees is called a forest, and a spanning subgraph for G, which is also a forest, is called a spanning forest for G. We will call $\{[\{u_i\}, \Phi] : u_i \in V\}$ the trivial spanning forest for G and each $[\{u_i\}, \Phi]$ a trivial tree. Associated with each edge (u,v) is a real-valued cost c(u,v). The minimum spanning tree problem may be stated as follows: Given a connected undirected graph each of whose edges has a real-value cost, find a spanning tree of the graph whose total edge cost is minimum.

Applications (see Christofides [4]) include the design of a distribution network in which the nodes represent cities or towns and the edges represent electrical power lines, water lines, natural gas lines, communication links, etc. The objective is to design a network which uses the least length of cable or pipe. The minimum spanning tree problem is also used as a subproblem for algorithms for the traveling salesman problem (see Held and Karp [6, 7] and Ali and Kennington [3]). Some vehicle routing algorithms require the solution of a traveling salesman problem on a subset of nodes. Hence, a wide variety of applications require the solution of minimal spanning trees. Some applications

require a single solution and some use the model as a subproblem within another algorithm.

II. THREE CLASSICAL ALGORITHMS

The algorithms in current use may be traced to ideas developed by Prim, Kruskal, and Sollin. These three classical algorithms all begin with the trivial spanning forest $G_0 = \{[V_i, T_i], i = 0, \ldots, |V| - 1\}$. A sequence of spanning forests is obtained by merging spanning forest components. Given spanning forest G_k, a nonforest edge (u,v) is selected and the components $[V_i, T_i]$ and $[V_j, T_j]$ with $u \in V_i$ and $v \in V_j$ are removed from G_k and replaced by $[V_\ell, T_\ell]$, where $\ell = k + |V|$, $V_\ell = V_i \bigcup V_j$, and $T_\ell = T_i \bigcup T_j \bigcup \{(u, v)\}$, yielding spanning forest G_{k+1}. After $m = |V|-1$ edges have been selected, $G_m = \{[V_{2m}, T_{2m}]\}$ is a minimal spanning tree for G.

Let $[V_i, T_i]$ and $[V_j, T_j]$ denote two disjoint subtrees of G and define the shortest distance between the trees by $d_{ij} = \min \{c(u,v): (u,v) \in E, u \in V_i, v \in V_j\}$. Let $\Gamma_{ij} = \{(u, v) : (u, v) \in E, u \in V_i, v \in V_j, c(u, v) = d_{ij}\}$. The nonforest edge selected must be an element of Γ_{ij} for some (i,j) pair. The three classical algorithms may be viewed as using different selection methods for the nonforest edge from among all Γ_{ij}.

In Prim's algorithm, the nonforest edge (u,v) for G_k is always selected so that $(u,v) \in V_i \times V_{j^*}$, where j^* is the largest index j such that $[V_j, T_j]$ is a component of G_k. Thus a single component continues to grow as trivial trees disappear. An excellent description of Prim's algorithm is given in Papadimitriou and Steiglitz [15, p. 273], along with its (serial) computational complexity of $O(|V|^2)$. It is believed that this algorithm is best suited for dense graphs.

In the Sollin algorithm, the nonforest edge (u,v) for G_k is always selected so that $(u,v) \in V_{i^*} \times V_j$, where i^* is the smallest index i such that $[V_j, T_j]$ is a component of G_k. Thus a variety of different-sized components may be produced as the algorithm proceeds. All trivial trees will be removed first in the early stages of this algorithm. A description of the Sollin algorithm is given in Papadimitriou and Steiglitz [15, p. 277], along with its

(serial) computational complexity of $O(|E| \log |V|)$. This algorithm appears to be best suited for sparse graphs.

Kruskal's method may be viewed as an application of the greedy algorithm. The minimum spanning tree is constructed by examining the edges in order of increasing cost. If an edge forms a cycle within a component of G_k, it is discarded. Otherwise, it is selected and yields G_{k+1}. Here also different-sized components may be produced. A description of Kruskal's algorithm is given in Sedgewick [18, pp. 412–413], along with its (serial) computational complexity of $O(|E| \log |E|)$.

III. COMPUTATIONAL RESULTS WITH SEQUENTIAL ALGORITHMS

Computer codes for the Sollin algorithm, Kruskal's algorithm, and three versions of Prim's algorithm were developed. For the code SPARSE PRIM, additional data lists are utilized so that the sets $F[u] = \{(u,v): (u,v) \in E\}$ and $B[v] = \{(u,v): (u,v) \in E\}$ can be quickly determined for all $u,v \in V$. This is called maintaining the edge data in both forward and backward star format. DENSE PRIM maintains the edge data in an $|V| \times |V|$ matrix. HEAP PRIM maintains the edge data in both forward and backward star format and makes use of a d-heap as described in Tarjan [19, p. 77]. KRUSKAL makes use of a partial quicksort as described in [1, 8] to produce the least-cost remaining edge. SOLLIN is a straightforward implementation of the algorithm presented in [15].

The five codes were tested on randomly generated graphs whose density varied from 100% (complete graphs) down to 0.5%. All costs were uniformly distributed on the interval [0, maxcost] for different values of maxcost. All codes are written in FORTRAN for the Sequent Symmetry S81 (Rev. A).

The computational results for high density graphs are presented in Tables 1 and 2. In both tables the times are the total seconds required to solve three problems (excluding input and output). The cost range for the problems in Table 1 is 0 to 10,000 while the cost range for those in Table 2 is 0 to 100,000. For both tables DENSE PRIM was best for problems having densities of 100% and 80%. However, as the density was reduced, KRUSKAL was the best followed closely by DENSE PRIM and SPARSE PRIM. None of the five codes were sensitive to the cost range with total times approximately the same for both tables. SOLLIN was the clear loser for all of these experiments which is consistent with its worst case computational complexity.

The computational results for low density (5% to 20%) random graphs is presented in Table 3. KRUSKAL remained the clear winner, but the ranking for second through fifth changed. SOLLIN performs much better as the density decreases, and DENSE PRIM

performs much worse. KRUSKAL is very robust, performing well over a wide range of problem densities.

Computational runs for very sparse problems may be found in Table 4. For these runs SOLLIN dominated with HEAP PRIM placing second. This is consistent with the worst-case complexity analysis of the Sollin algorithm.

Tables 1, 2, 3, 4 About Here

IV. PARALLEL ALGORITHMS

Parallel versions of the three classical algorithms have appeared in the literature (see [2, 5, 9, 10, 11, 12, 14, 16, 17]), however; no computation experience has been reported. The overhead required for coordinating the work of multiple processors can only be determined by actual implementation and empirical investigation on a parallel processing machine. The Sequent micro-tasking software facilitates the parallel execution of subroutines. The details regarding parallel processing on Sequent machines may be found in Osterhaug [1986].

Prim's Algorithm

Let dd[v,w] denote the distance from node v to node w and let d[v] ≠ v denote the node nearest node v. Prim's algorithm requires |V|-1 iterations. At iteration k, the DENSE PRIM code executes the two modules which follow.

Module MIN

0. $a \leftarrow \infty$

1. for i = 1 to |V|-k

2. v← status [i];

3. if dd[v,d[v]] < a , then

4. $a \leftarrow$ dd[v,d[v]];

5. m← i;

6. end if

7. end for

and

Module UPDATE

1. for $i = 1$ to $|V|-k$

2. v← status[i];

3. if dd[v, d[v]] > dd[v, status[m]], then d[v]← status[m];

4. end for.

Suppose there are p processes and id gives the identification number (0, 1, . . ., p–1) of each of the processes. The MIN module was executed simultaneously by the processes by modifying step 1 to be

$$\text{for } i = id+1 \text{ to } |V|-k \text{ step } p$$

followed by setting the global minimum to the smallest of the p local minima. The UPDATE module involves no dependencies and can be parallelized by the same modification to step 1. A similar parallel algorithm has been described by Deo and Yoo [5].

The computational experience for a parallel version of DENSE PRIM is presented in Table 5. The first five rows of Table 5 correspond to the five sequential codes. For test problem #1, KRUSKAL was best with a time of 4.05 seconds while for test problem #2 DENSE PRIM was best. The speedup is calculated by dividing the best sequential time by the parallel time. The sixth row, which gives the PARALLEL DENSE PRIM code run with a single processor, provides a measure of the overhead for running this algorithm in parallel. The overhead is approximately 22% for test problem #1, and 20% for test problem #2. For the ten processor runs the speedups were 2.79 and 4.39, respectively.

Table 5 About Here

Sollin's Algorithm

The Sollin algorithm performed best on sparse graphs, hence we studied a parallelization of the Sollin algorithm for grid graphs. The most time-consuming component of the Sollin's sequential algorithm may be described by the following procedure:

for all $(u,v) \in E$:

let i and j denote the subtrees containing u and v, respectively;

if $i \neq j$ then

if $c[u,v] < d[i]$, then $d[i] \leftarrow c[u,v]$;

if $c[u,v] < d[j]$, then $d[j] \leftarrow c[u,v]$;

end if

end for.

That is, all the edge costs must be examined and certain subtree data are updated. Our parallelization of this scan relies upon a partitioning of the grid into p components (one for each processor). A three-processor partitioning of a 7 x 7 grid network is illustrated in Figure 1.

Figure 1 About Here

The above edge scan is performed in two stages. The first stage performs a parallel scan over edges both of whose vertices lie within the same partition. The second stage performs a parallel scan over edges across cut sets. If each partition consists of at least two rows of the grid, then all subtree data updating can be performed independently without the requirement of a lock.

The second part of the Sollin algorithm is to merge two subtrees by appending a new edge. The merger of subtrees, both of which lie in the same partition can also be executed in parallel. A related algorithm may be found in Quinn [17].

The computational experience with the parallel version of the Sollin algorithm for grid graphs may be found in Table 6. For these two test problems, the overhead for parallel processing was only 4%. The speedups for the ten processor runs were 5.89 and 6.81.

Table 6 About Here

V. SUMMARY AND CONCLUSIONS

Five computer codes were developed to solve the minimum spanning tree problem on a sequential machine. These codes were computationally compared on random graphs whose densities varied from 0.5% to 100%. An implementation of Prim's algorithm was best for 100% dense problems and an implementation of the Sollin algorithm was best for sparse problems. None of the codes were sensitive to the cost range. Kruskal's algorithm using a modification of a quicksort was the most robust of all implementations, working very well on a wide variety of problems. Unfortunately, a quicksort is difficult to parallelize. Both the DENSE PRIM code and the SOLLIN code were parallelized by the method of data partitioning (homogeneous multitasking), yielding ten-processor speedups of 2.79 up to 6.81.

REFERENCES

1. Aho, A.V., J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts (1974).

2. Akl, S., "An Adaptive and Cost-Optimal Parallel Algorithm for Minimum Spanning Trees," Computing, 36 (1986) 271-277.

3. Ali, I., and J. Kennington, "The Asymmetric M-Traveling Salesman Problem: A Duality Based Branch-And-Bound Algorithm," Discrete Applied Mathematics, 13 (1986) 259-276.

4. Christofides, N., Graph Theory: An Algorithmic Approach, Academic Press, New York, NY (1975).

5. Deo, N., and Y. Yoo, "Parallel Algorithms for the Minimum Spanning Tree Problem," Proceedings of the 1981 International Conference on Parallel Processing, IEEE Computing Society Press, (1981) 188-189.

6. Held, M., and R. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees," Operations Research, 18 (1970) 1138-1162.

7. Held, M., and R. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II," Mathematical Programming, 1 (1970) 6-25.

8. Knuth, D.E., Sorting and Searching, Addison-Wesley, Reading, Massachusetts (1973).

9. Kwan, S., and W. Ruzzo, "Adaptive Parallel Algorithms for Finding Minimum Spanning Trees," Proceedings of the 1984 International Conference on Parallel Processing, IEEE Computing Society Press, (1984) 439-443.

10. Lavallee, I., and G. Roucairol, "A Fully Distributed (Minimal) Spanning Tree Algorithm," Information Processing Letters, 23 (1986) 55-62.

11. Lavallee, I., "An Efficient Parallel Algorithm for Computing a Minimum Spanning Tree," Parallel Computing 83, (1984) 259-262.

12. Nath, D., and S. Maheshwari, "Parallel Algorithms for the Connected Components and Minimal Spanning Tree Problem," Information Processing Letters, 14, 1 (1982) 7-11.

13. Osterhaug, A., Guide to Parallel Programming on Sequent Computer Systems, Sequent Computer Systems, Inc., Beaverton, Oregon (1986).

14. Parallel Computers and Computations, Editors J. van Leevwen and J.K. Lenstra, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, (1985).

15. Papadimitriou, C. and K. Steiglitz, Combinatorial Optimizaiton: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, New Jersey (1982).

16. Pawagi, S. and I. Ramakrishnan, "An O(log n) Algorithm for Parallel Update of Minimum Spanning Trees," Information Processing Letters, 22 (1986) 223-229.

17. Quinn, M.J., Designing Efficient Algorithms for Parallel Computers, McGraw-Hill, New York, New York (1987).

18. Sedgenwick, R., Algorithms, Addison-Wesley, Reading, Massachusetts (1983).

19. Tarjan, R.E., Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania (1983).

Figure 1. A Three Processor Partitioning of a 7 x 7 Grid Graph.

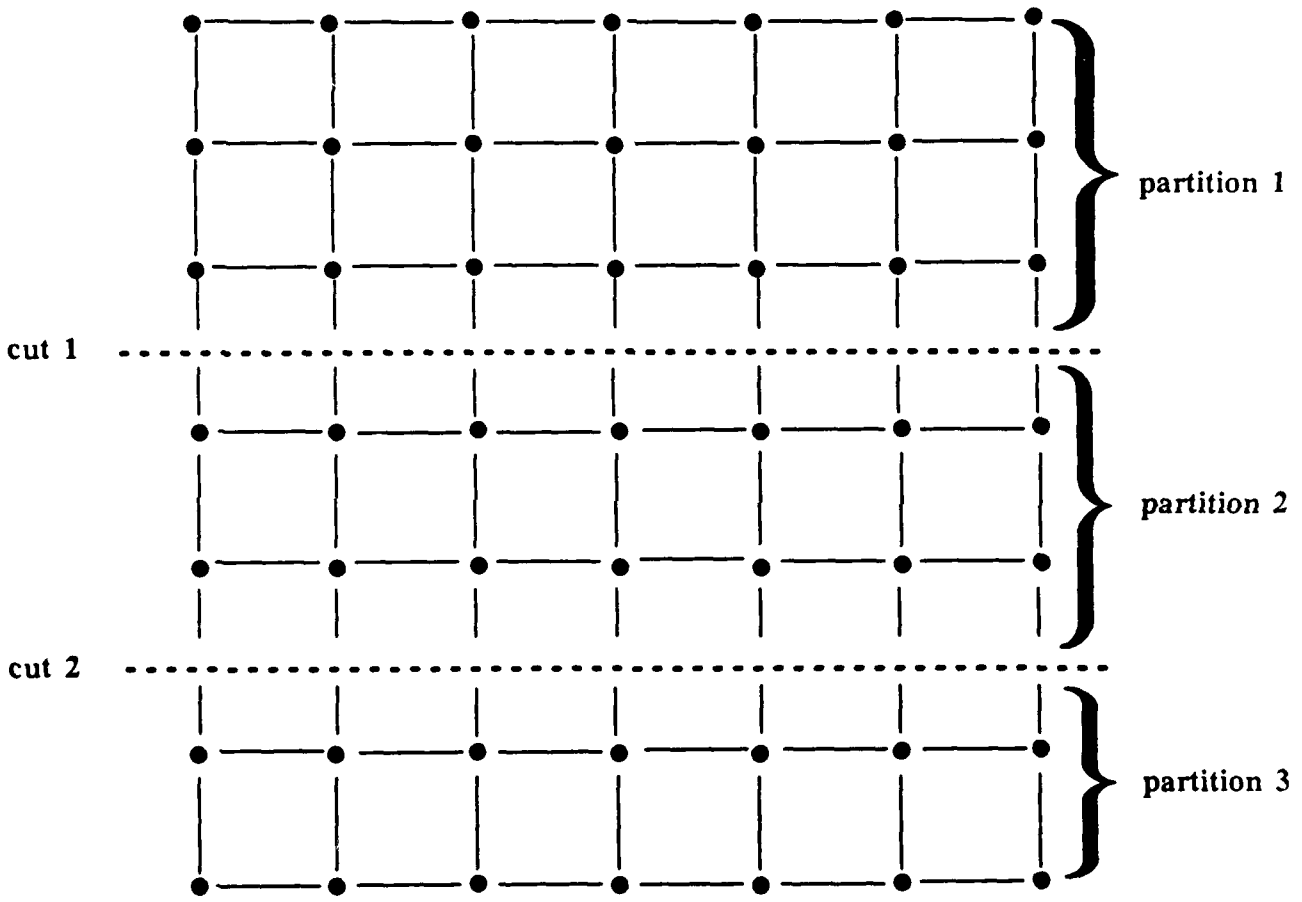


Table 1. Comparison of Sequential Codes on High Density Random Graphs Having a Cost Range of 0 to 10,000 (times are total seconds to solve three problems)

Vertices	Graph Density	Edges	SPARSE PRIM	DENSE PRIM	HEAP PRIM	SOLLIN	KRUSKAL
200	100%	19900	1.22	0.91	1.60	3.13	1.22
200	80%	15920	1.11	0.95	1.37	2.37	1.36
200	60%	11940	0.82	0.90	1.08	1.73	0.71
200	40%	7960	0.63	0.90	0.82	1.08	0.66
400	100%	79800	5.27	3.71	5.97	12.50	3.64
400	80%	63840	4.17	3.67	5.02	10.46	2.70
400	60%	47880	3.66	3.81	4.34	8.89	2.90
400	40%	31920	2.85	4.12	3.46	5.16	2.22
600	100%	179700	11.60	8.51	13.20	29.32	9.74
600	80%	143760	9.46	8.44	10.82	23.67	6.85
600	60%	107820	7.69	8.32	8.56	17.57	7.15
600	40%	71880	5.87	8.34	6.15	11.83	4.85
TOTAL TIME (sec)			54.35	52.58	62.39	127.71	44.00
RANK			3	2	4	5	1

Table 2. Comparison of Sequential Codes on High Density Random Graphs Having a Cost Range of 0 to 100,000 (times are total seconds to solve three problems)

Vertices	Graph Density	Edges	SPARSE PRIM	DENSE PRIM	HEAP PRIM	SOLLIN	KRUSKAL
200	100%	19900	1.19	0.90	1.55	3.04	1.17
200	80%	15920	1.00	0.90	1.31	2.29	0.96
200	60%	11940	0.81	0.90	1.07	1.72	0.73
200	40%	7960	0.62	0.90	0.81	1.07	0.67
400	100%	79800	4.91	3.66	5.88	11.37	4.38
400	80%	63840	4.14	3.67	4.93	10.36	2.70
400	60%	47880	3.36	3.66	3.92	7.89	2.69
400	40%	31920	2.58	3.67	2.87	4.64	1.79
600	100%	179700	11.89	8.66	13.58	29.37	9.57
600	80%	143760	9.54	8.67	11.23	23.71	8.42
600	60%	107820	7.71	8.32	8.62	17.66	6.37
600	40%	71880	5.86	8.34	6.14	11.85	4.37
TOTAL TIME (sec)			53.61	52.25	61.91	124.97	43.82
RANK			3	2	4	5	1

Table 3. Comparison of Sequential Codes on Low Density Random Graphs Having a Cost Range of 1 to 10,000 (times are total seconds to solve three problems)

Vertices	Graph Density	Edges	SPARSE PRIM	DENSE PRIM	HEAP PRIM	SOLLIN	KRUSKAL
200	20%	3980	0.45	0.90	0.54	0.54	0.42
200	10%	1990	0.37	0.89	0.41	0.30	0.33
200	5%	995	0.33	0.90	0.34	0.16	0.25
400	20%	15960	1.81	3.66	1.75	2.69	1.32
400	10%	7980	1.43	3.64	1.16	1.39	1.00
400	5%	3990	1.25	3.64	0.90	0.72	0.81
600	20%	35940	4.10	8.32	3.62	6.07	2.47
600	10%	17970	3.22	8.26	2.28	3.09	2.06
600	5%	8985	2.79	8.27	1.61	1.53	1.43
TOTAL TIME (sec)			15.75	38.48	12.61	16.49	10.09
RANK			3	5	2	4	1

Table 4. Comparison of Sequential Codes on Sparse Random Graphs Having a Cost Range of 0 to 10,000 (times are total seconds to solve three problems)

Vertices	Graph Density	Edges	SPARSE PRIM	DENSE PRIM	HEAP PRIM	SOLLIN	KRUSKAL
800	2.0%	6392	4.47	15.03	1.79	1.25	2.03
800	1.0%	3196	4.32	14.78	1.53	0.69	1.47
800	0.5%	1548	4.24	14.30	1.42	0.45	1.04
1000	2.0%	9990	6.97	23.95	2.46	1.91	3.04
1000	1.0%	4995	6.72	23.54	2.05	1.11	2.34
1000	0.5%	2498	6.60	22.79	1.80	0.65	1.51
1200	2.0%	14388	10.02	35.08	3.18	2.87	4.55
1200	1.0%	7194	9.66	34.40	2.65	1.57	3.76
1200	0.5%	3597	9.47	33.45	2.35	0.88	2.37
TOTAL TIME (sec)			62.47	217.32	19.23	11.38	22.11
RANK			4	5	2	1	3

Table 5. Parallel DENSE PRIM on G = [V,E] with |V| = 900 and |E| = 404,000 Having a Cost Range of 0 to 100,000 (all parallel times are the average for five runs)

		Problem 1		Problem 2	
cpu's	Algorithm	time (sec)	speedup	time (sec)	speedup
1	SPARSE PRIM	8.89	0.46	8.89	0.73
1	DENSE PRIM	6.39	0.63	6.45	1.00
1	HEAP PRIM	9.80	0.41	9.74	0.66
1	SOLLIN	22.22	0.18	26.31	0.25
1	KRUSKAL	4.05	1.00	7.35	0.88
1	PARALLEL DENSE PRIM	7.77	0.52	7.73	0.83
2	PARALLEL DENSE PRIM	4.15	0.98	4.14	1.56
3	PARALLEL DENSE PRIM	2.95	1.37	2.92	2.21
4	PARALLEL DENSE PRIM	2.37	1.71	2.36	2.73
5	PARALLEL DENSE PRIM	2.01	2.01	1.99	3.24
6	PARALLEL DENSE PRIM	1.79	2.26	1.79	3.60
7	PARALLEL DENSE PRIM	1.64	2.47	1.65	3.91
8	PARALLEL DENSE PRIM	1.55	2.61	1.56	4.13
9	PARALLEL DENSE PRIM	1.48	2.74	1.49	4.33
10	PARALLEL DENSE PRIM	1.45	2.79	1.47	4.39

Table 6. Parallel SOLLIN on 350 x 350 Grid Graphs With |V| = 122,500, and |E| = 244,300 Having a Cost Range of 0 to 10,000 (all times are the average for five runs)

	Problem 1		Problem 2	
cpu's	time (sec)	speedup	time (sec)	speedup
1†	34.33	1.00	36.99	1.00
1*	35.64	0.96	38.47	0.96
2	19.59	1.75	19.62	1.89
3	13.68	2.51	12.61	2.93
4	9.85	3.49	9.83	3.76
5	8.77	3.91	8.62	4.29
6	7.43	4.62	7.27	5.09
7	7.07	4.86	6.90	5.36
8	6.55	5.24	6.64	5.57
9	6.82	5.03	6.26	5.91
10	5.83	5.89	5.43	6.81

† best sequential SOLLIN code

* parallel code run with a single processor

Technical Report 88-OR-13

DIJKSTRA'S TWO-TREE SHORTEST PATH ALGORITHM

by

R.V. Helgason
J.L. Kennington
B.D. Stewart

Department of Operations Research and Engineering Management
School of Engineering and Applied Science
Southern Methodist Univeristy
Dallas, Texas 75275

Revised January 1989

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

The objective of this study is to computationally investigate a version of Dijkstra's algorithm for the problem of finding the shortest path between two nodes in a graph. The classical Dijkstra algorithm builds a shortest path tree rooted at one of the designated nodes. This method is computationally compared with a version that builds two shortest path trees rooted at each of the two designated nodes. Termination occurs when any node appears in both trees. Computationally we found that the classical Dijkstra method built trees containing approximately 50% of the nodes in the original graph while the two-tree method terminated with only 6% of the nodes in the two trees. In computational experiments involving over 480 test problems, the two-tree method produced a speedup of over four.

ACKNOWLEDGEMENT

This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C0182, the Air Force Office of Scientific Research under Contract Numbers AFOSR 83-0278 and AFOSR 87-0199, the Office of Naval Research under Contract Number N00014-87-K-0223, and ROME Air Development Center under Contract Number SCEEE PDP/87-95.

I. INTRODUCTION

Since the late fifties when the first methods were developed, the shortest path problem has become one of the fundamental problems in the areas of combinatorial optimization, computer science, and operations research. Algorithms and applications are commonly found in the important books in these areas (see for example [BeGa], [BeGh], [H], [JB], [L], [PS], [Q], and [T]). The study of this problem has been motivated by both its elegant mathematical structure and its many practical applications. Our recent interest in this problem was occasioned by the need to solve shortest path subproblems in several mathematical optimization procedures we are developing in an MIMD parallel computing environment.

Consider the network $G = [V, E]$ with node set V and arc set $E \subset (V \times V) \backslash \{(i,i) : i \in V\}$. Let $c(i, j)$ denote the length of edge (i, j). A path in G from $s \in V$ to $t \in V$ is a sequence of distinct edges $(v_1,v_2), (v_2,v_3), \ldots, (v_{k-1}, v_k)$ such that $v_1 = s$, $v_k = t$, and every edge $(v_j, v_{j+1}) \in E$. For brevity, we will also denote the path above simply as $s = v_1, v_2, \ldots, v_{k-1}, v_k = t$. The path length is given by $\sum_{j=1}^{(k-1)} c(v_j, v_{j+1})$. The (two node) shortest path problem is defined as follows: Given two distinct nodes s and t, find a directed path in G from s to t with minimum length.

Excellent surveys for the many variations of the shortest path problem may be found in Deo and Pang [DP] and Gallo and Pallottino [GP]. A survey of techniques and computational comparisons may be found in Dial, Glover, Karney, and Klingman [DGKKa], and [DGKKb], in Klingman, Mote, and Whitman [KMW], in Glover, Glover, and Klingman [GGK], in Desrochers [De], and in Divoky [Di]. In [DGKKa] all the methods are grouped into two general classes: label-setting algorithms and label-correcting algorithms. Dijkstra is credited with the first label-setting algorithm and any algorithm

which uses this approach has been considered a particular implementation of Dijkstra's original algorithm (see [GP]).

The Dijkstra algorithm is restricted to problems having non-negative edge lengths and it builds a shortest path tree rooted at s. At each iteration at least one new node and edge are appended to the shortest path tree. Hence, after at most |V| - 1 iterations t is appended to the tree and the shortest path from s to t is known. It occurred to us that in an MIMD parallel computing environment shortest path trees could be grown from both s and t using two independent processors and that when the trees meet we should have a solution. A simplistic argument led us to believe that on the average we should obtain a solution in half the time taken by a typical serial implementation. Early experimental results with a parallel implementation of this strategy applied to both the Dijkstra algorithm for the shortest path problem and the closely-related painted network algorithm for the painted problem (see [R]) were excellent. This led us to conjecture that a serial algorithm which mimics the action of the two parallel processors should be superior to the usual serial implementation. We subsequently developed such a serial implementation, and our conjecture was born out. A search of the literature revealed that a serial algorithm of this type, which we call a two-tree Dijkstra algorithm, had been anticipated.

In 1960 Dantzig [Da] suggested that a pair of trees be built with one rooted at s and the other rooted at t. No stopping criteria were given. This strategy also appears in the 1962 book by Berge and Ghouila-Houri [BeGh] with an incorrect stopping criterion. Nicholson [N] was the first to present a correct analysis of the Dijkstra two-tree algorithm. Additional discussion may be found in the 1969 survey by Dreyfus [Dr]. Unfortunately, the literature contains no computational studies and the excellent speedup possible with the two-tree Dijkstra algorithm has apparently not been previously observed.

The explanation for the excellent behavior of the two-tree Dijkstra algorithm lies in the small size of each rooted tree when the stopping criterion is satisfied. In our compu-

tational study, we found that the original Dijkstra algorithm generated a shortest path tree containing approximately 50% of the original nodes. However, both trees together in the two-tree method contain only 6% of the original nodes. This is an astonishing observation which can be used to improve numerous algorithms which depend upon the repeated solution of shortest path problems. It is the purpose of this paper to communicate our experimental results which indicate that the two-way Dijkstra should be used in preference to the ordinary Dijkstra algorithm, especially within algorithms which depend upon the repeated solution of shortest path problems. In addition, we present the algorithms in detail and a convergence proof with an appropriate stopping criterion. Our experience points up the lesson that examination of algorithms for the parallel computing environment may well lead to better design of algorithms for the serial environment.

II. THE CLASSICAL ALGORITHM

Dijkstra's classical algorithm begins at node s and builds a shortest path tree in which the shortest path from s to any node in the tree is known. When node t is placed in the tree we have a minimum length directed path from s to t. The algorithm may be stated as follows:

DIJKSTRA'S SHORTEST PATH ALGORITHM

Input:

1. A graph G = [V, E] with node set V and edge set E.
2. A length c(i, j), for each edge (i, j)$\in$ E .
3. Two nodes s and t, between which a shortest path is desired.

Working Entities:

1. A set Q of nodes which are to be scanned.
2. A set R of nodes whose labels are not permanent.
3. A set of labels {d(j)} for node distance from s.
4. A set of labels {p(j)} for node predecessor in tree.
5. A minimum distance value u for nodes whose labels are not permanent.

Output:

1. A shortest path in G from s to t implicit in labels {p(j)}. Explicitly, the path is

   $$s = p_w(t), p_{w-1}(t), \ldots, p_2(t), p(t), t,$$

   where $p_i(t)$ is defined recursively by $p_1(t) = p(t)$ and $p_i(t) = p(p_{i-1}(t))$ for $i \geq 2$.

2. The length u of a shortest path in G from s to t.

Assumptions:

All $c(i, j) \geq 0$, $s \neq t$, and there exists a directed path in G from s to t.

procedure DIJKSTRA1:

begin

1 $R \leftarrow V$, for all $k \in R, d(k) \leftarrow \infty$; $d(s) \leftarrow 0, p(s) \leftarrow s$;

2 $u \leftarrow \min\{d(k) : k \in R\}, Q \leftarrow \{k \in R : d(k) = u\}$, select $i \in Q, R \leftarrow R \backslash \{i\}$;

3 for all $(i, j) \in (\{i\} \times R) \cap E$,

4 if $d(j) > u + c(i, j)$, then

5 $d(j) \leftarrow u + c(i, j), p(j) \leftarrow i$;

6 end if

7 end for

8 if $i \neq t$, then go to 2;

end.

We define D(j) to be the length of a shortest path in G from node s to node j. We will say that an iteration has occurred each time that Steps 2 through 7 have been completed and at the end of an iteration the node i will be said to be permanently labeled and scanned. Proof for the following two propositions may be found in Even [E].

Proposition 1. During the execution of DIJKSTRA1, if $d(j) < \infty$, then there is a path in G from s to j of length d(j).

Proposition 2. If $j \in Q$ in Step 2 of DIJKSTRA1, then $d(j) = D(j)$.

The following four results are readily obtained from the previous propositions.

Proposition 3. During the execution of DIJKSTRA1, $d(j) \geq D(j)$.

Proposition 4. During the execution of DIJKSTRA1, if $j \notin R$, $d(j) = D(j)$.

Proposition 5. During the execution of DIJKSTRA1, if $j \in R$, $d(j) \geq D(q)$ for all $q \notin R$.

Proposition 6. During the execution of DIJKSTRA1, if $j \notin R$ and $j \neq s$, then $p(j) \notin R$.

Proposition 7. If $(q, r) \in E$, $D(q) \leq D(r) + c(q, r)$.

Proof. Let $s = v_1, v_2, \ldots, v_k = r$ is any shortest path in G from s to r.

Case 1. Assume $(q, r) \neq (v_{k-1}, v_k)$. Then edge (q,r) extends the path to a path in G from s to q of length $D(r) + c(q,r)$. Hence $D(q) \leq D(r) + c(q,r)$.

Case 2. Assume $(q, r) = (v_{k-1}, v_k)$. Then $D(r)=D(q)+c(q,r)$.

Hence, $D(q)=D(r)-c(q,r) \leq D(r)+c(q,r)$, since $c(q,r) \geq 0$. □

Proposition 8. After the first execution of Step 2 in DIJKSTRA1, if $i \in R$, there exists a shortest path $s = v_1, v_2, \ldots, v_k = i$ in G from s to i for which an integer j satisfying $1 < j \leq k$ exists such that $\{v_1, \ldots, v_{j-1}\} \cap R = \emptyset$ and $\{v_j, \ldots, v_k\} \subset R$.

Proof. After Step 1, R=V, d(s)=0, and $d(r) = \infty$ for all $r \in V$ where $r \neq s$. Thus the first time through Step 2 produces u=0, Q={s}, and R=V\{s}, so that $s \notin R$ from then on. Let $s = y_1, y_2, \ldots, y_q = i$ be any shortest path in G from s to i. Let r be the smallest integer such that $y_r \in R$. Then $\{y_{r+1}, \ldots, y_q\} \subset R$. If $r = 1$, $y_1, y_2, \ldots, y_q$ is the path we seek and j=2. Otherwise, reversing the path given by

$$y_r, p(y_r), \ldots, p_{w-1}(y_r), p_w(y_r) = s$$

and adjoining it to $y_{r+1}, \ldots, y_q$ produces the path we seek, with $j = r + 1$. By Prop. 6, all nodes in the reversed path are not in R. □

That an arbitrary shortest path in G from s to i may not exhibit the property of Prop. 8 is shown in Figure 1.

II. DIJKSTRA'S TWO-TREE ALGORITHM

The two-tree algorithm builds a pair of shortest path trees which we call the left tree and the right tree. The left tree contains s while the right tree contains t. The two trees are grown in alternate steps and termination occurs when a node appears in both trees. For the problem illustrated in Figure 2, termination occurred at iteration 4 when node 3 appeared in both trees. However, one should note that node 3 is not contained in the shortest path. To determine the shortest path, we add the left and right labels and select a node with smallest sum. For this example, the sum of the labels are as follows: (node 1, ∞), (node 2, 5), (node 3, 6), (node 4, 5), (node 5, ∞). Therefore, the shortest path has length 5 and is given by (1, 2), (2, 4), (4, 5). The two-tree Dijkstra algorithm may be stated as follows:

DIJKSTRA'S TWO-TREE SHORTEST PATH ALGORITHM

Input:

1. A graph G = [V, E] with node set V and edge set E.
2. A length c(i, j) for each edge (i, j)$\in$ E .
3. Two nodes s and t, between which a shortest path is desired.

Working Entities:

1. Sets Q^s and Q^t of nodes which are to be scanned with respect to the trees rooted at nodes s and t, respectively.
2. Sets R^s and R^t of nodes whose labels are not permanent with respect to the trees rooted at nodes s and t, respectively.
3. Sets of labels $\{d^s(j)\}$ and $\{d^t(j)\}$ for node distances from nodes s and t, respectively.

4. Sets of labels $\{p^s(j)\}$ and $\{p^t(j)\}$ for node predecessors in trees rooted at nodes s and t, respectively.
5. Minimum distance values u^s and u^t for nodes whose labels are not permanent in the trees rooted at nodes s and t, respectively.

Output:

1. The length u of a shortest path in G from s to t.
2. A set of J nodes each lying on a shortest path in G from s to t.
3. A shortest path in G from s to t implicit in J and the predecesor labels $\{p_j\}$. Explicitly, the path is

   $s = p^s_u(r), p^s_{u-1}(r), \ldots, p^s(r), r, p^t(r), \ldots, p^t_{r-1}(r), p^t_v(r) = t,$

   where $p^s_i(\cdot)$ and $p^t_j(\cdot)$ are defined analogously to $p_i(\cdot)$ of Section I.

Assumptions:

All $c(i, j) \geq 0$, $s \neq t$, and there exists a directed path in G from s to t.

procedure DIJKSTRA2:

begin

1^s $R^s \leftarrow V$, for all $k \in R^s, d^s(k) \leftarrow \infty$; $d^s(s) \leftarrow 0, p^s(s) \leftarrow s$;

1^t $R^t \leftarrow V$, for all $k \in R^t, d^t(k) \leftarrow \infty$; $d^t(t) \leftarrow 0, p^t(t) \leftarrow t$;

2^s $u^s \leftarrow \min\{d^s(k) : k \in R^s\}$, $Q^s \leftarrow \{k \in R^s : d^s(k) = u^s\}$, select $i \in Q^s, R^s \leftarrow R^s \backslash \{i\}$;

3^s for all $(i, j) \in (\{i\} \times R^s) \cap E$,

4^s if $d^s(j) > u^s + c(i, j)$, then

5^s $d^s(j) \leftarrow u^s + c(i, j), p^s(j) \leftarrow i$;

6^s end if

7^s end for

8^s if $i \in R^t$, then go to 9;

2^t $u^t \leftarrow \min\{d^t(k) : k \in R^t\}$, $Q^t \leftarrow \{k \in R^t : d^t(k) = u^t\}$, select $j \in Q^t, R^t \leftarrow R^t \backslash \{j\}$;

3^t for all $(i,j) \in (R^t \times \{j\}) \cap E$,

4^t if $d^t(i) > u^t + c(i,j)$, then

5^t $d^t(i) \leftarrow u^t + c(i,j)$, $p^t(i) \leftarrow j$;

6^t end if

7^t end for

8^t if $j \notin R^s$, then go to 2^s;

9 $u \leftarrow \min \{d^s(j) + d^t(j) : j \in (V \backslash R^s) \bigcup (V \backslash R^t)\}$

10 $J \leftarrow \{j \in (V \backslash R^s) \bigcup (V \backslash R^t) : d^s(j) + d^t(j) = u\}$;

end

Note that with minor notational adjustments, Propositions 1 through 8 also apply to the tree-building steps 1^s through 8^s for the tree rooted at s as well as 1^t through 8^t for the tree rooted at t, with the roles of s and t reversed.

Proposition 9. (Nicholson [N]) When DIJKSTRA2 terminates, the length of the shortest path in G from s to t is given by

$$u = \min \{d^s(j) + d^t(j) : j \in (V \backslash R^s) \bigcup (V \backslash R^t)\}. \tag{1}$$

Proof. Let $n \in (V \backslash R^s) \bigcap (V \backslash R^t)$ and let r be any node of $(V \backslash R^s) \bigcup (V \backslash R^t)$ such that $d^s(r) + d^t(r) = u$. Let i be an arbitrary node of V. We will show that

$$u \leq D^s(i) + D^t(i) \tag{2}$$

Case 1. Assume $i \in (V \backslash R^s)$ and $i \in (V \backslash R^t)$. By Prop. 4, $D^s(i) = d^s(i)$ and $D^t(i) = d^t(i)$, so that $D^s(i) + D^t(i) = d^s(i) + d^t(i)$. Also, $i \in (V \backslash R^s) \bigcup (V \backslash R^t)$, so that $u \leq d^s(i) + d^t(i)$. Thus (2) holds.

Case 2. Assume $i \notin (V \backslash R^s)$ and $i \notin (V \backslash R^t)$. Then $i \in R^s$ and $i \in R^t$. Since $n \notin R^s$ and $n \notin R^t$, by Prop. 5, $D^s(i) \geq d^s(n)$ and $D^t(i) \geq d^t(n)$, so that

$D^s(i) + D^t(i) \geq d^s(n) + d^t(n)$. Since $n \in (V \backslash R^s) \bigcup (V \backslash R^t)$, $d^s(n) + d^t(n) \geq u$. Thus (2) holds.

Case 3. Assume $i \notin (V \backslash R^s)$ and $i \in (V \backslash R^t)$. By Prop. 8 there exists a shortest path $s = v_1, v_2, \ldots, v_k = i$ in G from s to i for which an integer w satisfying $1 < w \leq k$ exists such that $\{v_1, \ldots, v_{w-1}\} \cap R^s = \emptyset$ and $\{v_w, \ldots, v_k\} \subset R^s$. For any j such that $1 \leq j < k$, we must have that

$$D^s(v_{j+1}) - D^s(v_j) = c(v_j, v_{j+1}). \tag{3}$$

Also, for any j such that $1 \leq j < k$, we have from Prop. 6 that

$$D^t(v_{j+1}) - D^t(v_j) \geq -c(v_j, v_{j+1}). \tag{4}$$

Let p be any integer such that $1 \leq p \leq k-1$. Summing both (3) and (4) with j taking on all integral values from p to w–1, we obtain

$$D^s(v_k) - D^s(v_p) = \sum_{j=p}^{(k-1)} c(v_j, v_{j+1}) \tag{5}$$

and

$$D^t(v_k) - D^t(v_p) \geq \sum_{j=p}^{(k-1)} c(v_j, v_{j+1}). \tag{6}$$

Adding (5) and (6) and using $v_k = i$, we obtain for all p such that $1 \leq p \leq k-1$,

$$D^s(i) + D^t(i) \geq D^s(v_p) + D^t(v_p). \tag{7}$$

Subcase 3.1 Assume there is an integer h such that $1 \leq h \leq w-1$ and $v_h \notin R^t$. We also have that $v_h \notin R^s$, so that by Prop. 4, $d^t(v_h) = D^t(v_h)$ and $d^s(v_h) = D^s(v_h)$. By (7), $D^s(i) + D^t(i) \geq D^s(v_h) + D^t(v_h) = d^s(v_h) + d^t(v_h)$. Since $v_h \in (V \backslash R^s) \bigcup (V \backslash R^t)$, $d^s(v_h) + d^t(v_h) \geq u$. Thus (2) holds.

Subcase 3.2 Assume there is an integer l such that $w \leq l \leq k$ and $v_l \in R^t$. Since $l \geq w$, $v_l \in R^s$. By Case 2, $u \leq D^s(v_l) + D^t(v_l)$. From (7), $D^s(i) + D^t(i) \geq D^s(v_l) + D^t(v_l)$, so that (2) holds.

Subcase 3.3 Assume that $\{v_1, \ldots, v_{w-1}\} \subset R^t$ and $\{v_w, \ldots, v_k\} \cap R^t = \emptyset$. Since $v_{w-1} \notin R^s$ and $(v_{w-1}, v_w) \in (Q^s \times R^t) \cap E$ on the iteration v_{w-1} was scanned,

$$d^s(v_w) \leq D^s(v_{w-1}) + c(v_{w-1}, v_w). \tag{8}$$

From (3),

$$D^s(v_w) = D^s(v_{w-1}) + c(v_{w-1}, v_w). \tag{9}$$

From (8) and (9), $d^s(v_w) \leq D^s(v_w)$ and from Prop. 3, $d^s(v_w) \geq D^s(v_w)$, so that

$$D^s(v_w) = d^s(v_w). \tag{10}$$

Since $v_w \notin R^t$, from Prop. 4,

$$D^t(v_w) = d^t(v_w). \tag{11}$$

Adding (10) and (11), we have that

$$D^s(v_w) + D^t(v_w) = d^s(v_w) + d^t(v_w). \tag{12}$$

Since $v_w \in V \backslash R^t$, $v_w \in (V \backslash R^s) \bigcup (V \backslash R^t)$, so that

$$u \leq d^s(v_w) + d^t(v_w). \tag{13}$$

From (7),

$$D^s(i) + D^t(i) \geq D^s(v_w) + D^t(v_w). \tag{14}$$

From (12), (13), and (14), we have that (2) holds.

Case 4. Assume $i \in (V \backslash R^s)$ and $i \notin (V \backslash R^t)$. An argument similar to that of Case 3 shows that (2) holds. □

IV. COMPUTATIONAL EXPERIENCE

Both algorithms have been coded and run on dense mxn bipartite graphs having 2mn edges. The data structure used is identical to that used by Jonker and Volgenant [JV] in their highly successful assignment code. Both codes are written in FORTRAN and were run on a Sequent Symmetry S81 using a single Intel 80386 cpu. The computational experience is presented in Table 1.

Each data point in Table 1 is the sum for twenty problems, i.e. 240 problems were solved with three different codes. For each set of 20 problems, the cost structure was the same, and s was selected at random from among the left side nodes and t was selected at random from among the right side nodes.

The Dijkstra Left Side Only builds the left tree while the Dijkstra Right Side Only builds the right tree. The column entitled iter gives the number of nodes placed in the shortest path trees for all twenty problems. For the 1000 x 1000 problems with 2,000,000 edges, the Dijkstra Left Side Only algorithm builds a shortest path tree with 1057 nodes on the average. For this set of eighty problems, the Dijkstra two-tree algorithm builds shortest path trees with a total of 110 nodes on the average. This is an extraordinary reduction in the amount of computational work required to solve these eighty problems. However, there is overhead involved in the two-tree algorithm which accounts for a speedup of four while the reduction in the number of nodes placed in the shortest path trees is a factor of eight. The Dijkstra two-tree algorithm also takes almost double the storage of the classical Dijkstra method. The comparison of storage is given in Table 2.

Similar codes were developed and run on twelve of the (nonbipartite) NETGEN [KNS] problems. The computational experience is presented in Table 3. Each data point is the sum for twenty problems. From each NETGEN problem, twenty problems were generated by randomly selecting s to be a source and t to be a sink. Given randomly selected s and t nodes, the NETGEN problems may not contain a directed path from s to t. There-

fore, step 3 for the algorithms has been modified to terminate with no feasible solution when u, u^s, or u^t equal ∞. As with the bipartite structure only about 10% as many nodes need be scanned with the 2-tree algorithm as with the single tree algorithm.

V. SUMMARY AND CONCLUSIONS

Dijkstra's classical algorithm has proven to be extremely successful for the problem of finding the shortest path in a graph connecting two given nodes. A minor modification of this algorithm suggested by Nicholson [N] converts a good algorithm into an extraodinary method. While the classical algorithm required a scan over approximately 50% of the problem nodes, the two-tree algorithm only required a scan over 6% of the problem nodes. On 80 problems containing 2000 nodes and 2,000,000 edges, the new algorithm obtains the shortest path in approximately 1.7 seconds/problem compared to 7.5 seconds for the best classical algorithm.

REFERENCES

[BeGh] C. Berge and A. Ghouila-Houri, Programming, Games, and Transportation Networks, John Wiley and Sons, Inc., New York, NY (1962).

[BeGa] D. Bertsekas and R. Gallager, Data Networks, Prentice-Hall, Englewood Cliffs, New Jersey (1987).

[B] J. Boothroyd, "Algorithm 22: Shortest Path Between Start Node and End Node of a Network", The Computer Journal, 10 (1967) 306-307.

[Da] G. Dantzig, "On the Shortest Route Through a Network," Management Science, 6, (1966) 187-190.

[DP] N. Deo and C. Pang, "Shortest-Path Algorithms: Taxonomy and Annotation," Networks, 14, (1984) 275-323.

[De] M. Desrochers, "A Note on the Partitioning Shortest Path Algorithm," Operations Research Letters, 6, (1987) 183-187.

[DGKKa] R. Dial, F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," CCS Report 291, Center for Cybernetic Studies, The University of Texas, Austin, TX 78712 (1977).

[DGKKb] R. Dial, F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," Networks, 9 (1979) 215-250.

[Di] J. Divoky, "Improvements for the Thresh X2 Shortest Path Algorithm," Operations Research Letters, 6, (1987) 227-232.

[Dr] S. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," Operations Research, 17, (1969) 395-412.

[E] S. Even, Graphs Algorithms, Computer Science Press, Potomic, Maryland (1979).

[GGK] F. Glover, R. Glover, and D. Klingman, "Computational Study of an Improved Shortest Path Algorithm," Networks, 14, (1984) 25-36.

[GP] G. Gallo and S. Pallottino, "Shortest Path Methods: A Unifying Approach," Mathematical Programming Study, 26, (1986) 38-64.

[H] T. Hu, Combinatorial Algorithms, Addison-Wesley, Reading, MA (1982).

[JB] P. Jensen and J. Barnes, Network Flow Programming, John Wiley and Sons, Inc., New York, NY (1980).

[JV] R. Jonker and A. Volgenant, "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," Computing, 38, (1987) 325-340.

[KMW] D. Klingman, J. Mote, and D. Whitman, "Improving Flow Management and Control Via Improving Shortest Path Analysis," CCS Report 322, Center for Cybernetic Studies, The University of Texas, Austin, TX 78712, (1978).

[KNS] D. Klingman, A. Napier, and J. Stutz, "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimal Cost Flow Network Problems," Management Science, 20, (1974) 814-821.

[L] E. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart, and Winston, New York, NY (1976).

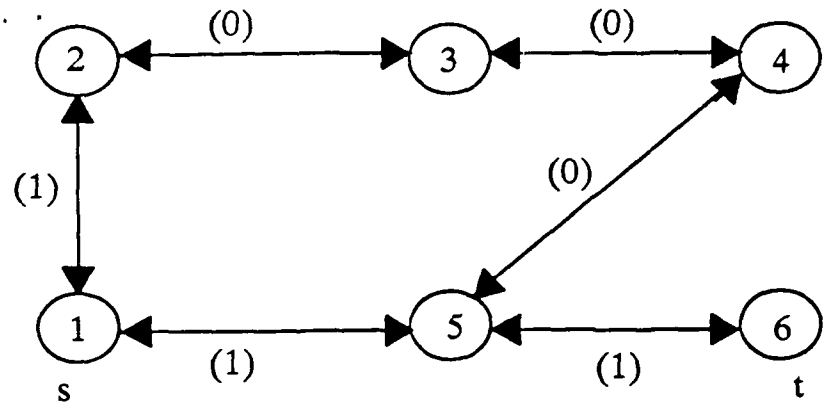
[N] T. Nicholson, "Finding the Shortest Route Between Two Points in a Network," The Computer Journal, 9, (1966) 275-280.

[PS] C. Papadimitriou and K. Steiglits, Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ (1987).

[Q] M. Quinn, Designing Efficient Algorithms for Parallel Computers, McGraw-Hill, New York, NY (1987).

[R] R. Rockafellar, Network Flows and Monotropic Optimization, John Wiley and Sons, Inc., New York, NY (1984).

[T] R. Tarjan, Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA (1983).

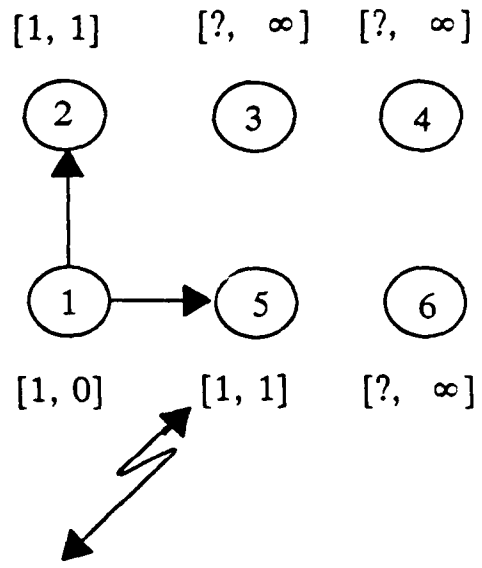


Iter. 1

Tree

Q = {1}

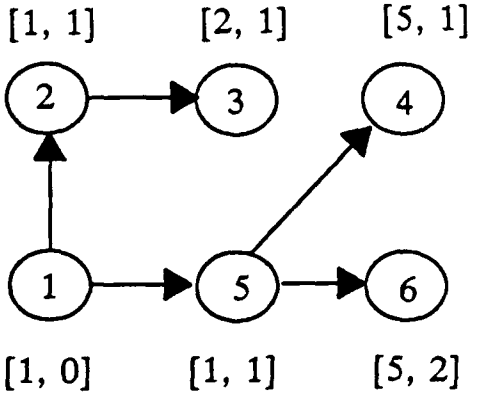
R = {2, 3, 4, 5, 6}



Iter. 2 i = 2

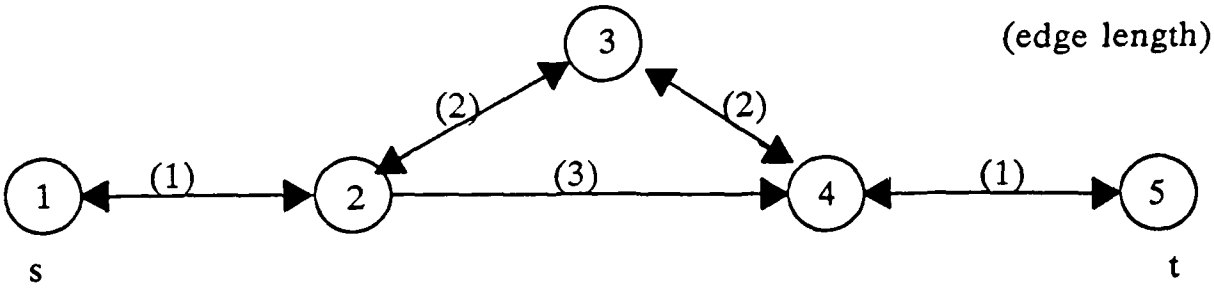
Iter. 3 i = 5

R = {3, 4, 6}



Shortest Path s=1, 2, 3, 4, 5, 6 = t has 1, $2 \notin R$, $3, 4 \in R$, $5 \notin R$, and $6 \in R$.

Figure 1. Shortest Path Without Prop. 8 Property



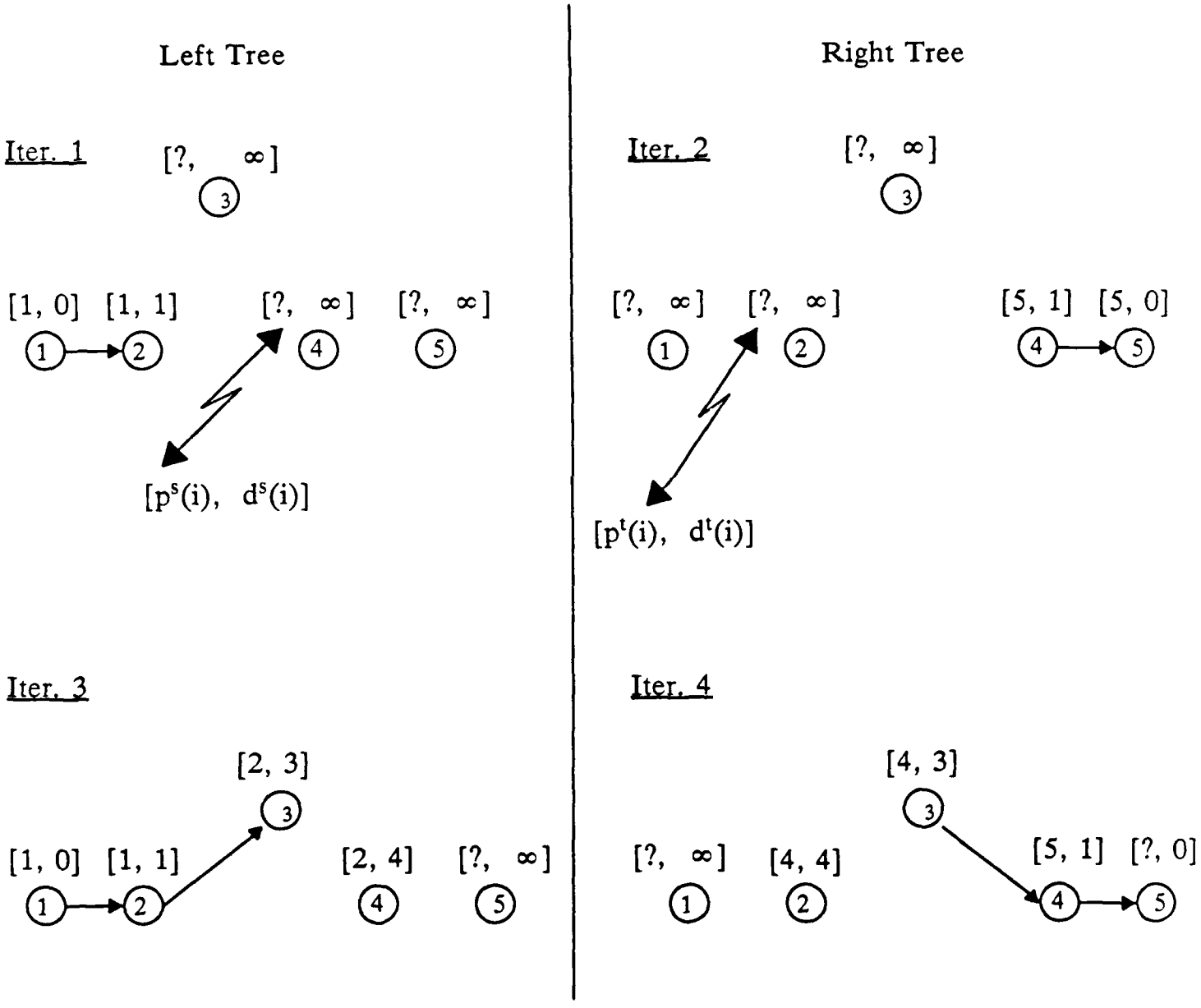


Figure 2. Example of Two-Tree Dijkstra (Shortest Path From 1 to 5 has Length 5)

Table 1. Comparison of Dijkstra Algorithms on Dense m x n Bipartite Graphs (all times are in seconds for 20 problems)

m (sources)	n (sinks)	cost range	Dijkstra Left Side Only		Dijkstra Right Side Only		Dijkstra 2-tree	
			iter	time(secs)	iter	time(secs)	iter	time(secs)
800	1000	0- 100	16698	99	16464	96	1910	21
800	1000	0- 1000	20108	114	12923	82	2252	26
800	1000	0- 10000	21173	132	12927	91	1859	28
800	1000	0-100000	21230	177	12809	120	1787	30
1000	800	0- 100	18280	96	17262	93	2086	19
1000	800	0- 1000	18240	108	18446	110	2580	28
1000	800	0- 10000	18073	117	18205	121	2253	31
1000	800	0-100000	18092	162	18143	161	2250	36
1000	1000	0- 100	22320	141	21325	136	2535	30
1000	1000	0- 1000	20532	129	17445	119	2315	31
1000	1000	0- 10000	22106	160	19712	144	1945	34
1000	1000	0-100000	22382	192	19641	204	1977	38
TOTALS			239,234	1627	205,302	1477	25,749	352

Table 2. Comparison of Storage for Classical and Two-Tree Dijkstra Codes

	Classical Dijkstra	Two-Tree Dijkstra
m x n length arrays	1	1
m-length arrays	3	8
n-length arrays	3	8

Table 3. Comparison of Dijkstra Algorithms on Random Networks (all times are in seconds for 20 problems)

NETGEN Problem #	Number of Nodes	Number of Arcs	Dijkstra Left Side Only		Dijkstra Right Side Only		Dijkstra 2-tree	
			iter	time (secs)	iter	time (secs)	iter	time (secs)
29	1000	3400	10,164	31.7	8300	26.6	1337	6.3
30	1000	4400	9057	24.9	8184	21.0	1138	5.0
31	1000	4800	8983	24.1	7539	19.0	1283	5.2
32	1500	4342	14,512	56.5	14,159	51.7	1559	10.6
33	1500	4385	12,228	53.9	10,899	44.7	1334	9.5
34	1500	5107	15,019	50.8	12,169	41.1	1650	10.4
35	1500	5730	14,479	48.0	15,720	47.7	1725	10.5
36	8000	15,000	68,514	778.6	76,442	816.7	3949	147.7
37	5000	23,000	52,210	181.8	41,693	180.1	3034	45.9
38	3000	35,000	20,001	44.7	19,797	40.1	1977	12.2
39	5000	15,000	47,491	255.3	48,267	276.1	2952	60.1
40	3000	23,000	26,603	69.6	24,718	58.1	2282	18.3
Totals			299,261	1620	287,887	1623	24220	341

Technical Report 88-OR-16

AN EMPIRICAL ANALYSIS OF THE DENSE ASSIGNMENT PROBLEM

by

J. Kennington
Z. Wang

Department of Computer Science & Engineering
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275

revised April 1989

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

The best algorithms for the dense assignment problem are acknowledged to be the auction algorithm and the shortest augmenting path algorithm. In this investigation we present an empirical analysis of the best software implementations of these two methods on three different serial machines. These software implementations were developed by Professor Bertsekas of Massachusetts Institute of Technology and by Professors Jonker and Volgenant of the University of Amsterdam. This was an independent evaluation of the software implementation of these two algorithms. For the sample of problems examined and the sample of hardware used (IBM 3081D, Sequent Symmetry S81, and VAX 750), we found that the shortest augmenting path algorithm was the best. We also report our empirical results with a parallel version of the shortest augmenting path algorithm. On 1200x1200 dense assignment problems, speedups of approximately four were achieved using ten processors. Million arc problems were routinely solved in less than fifteen seconds on a Sequent Symmetry S81 with the parallel shortest augmenting path algorithm.

ACKNOWLEDGMENT

The authors wish to express their appreciation to Dimitri P. Bertsekas and Paul Tseng of Massachusetts Institute of Technology for providing invaluable assistance during this study and to Roy Jonker and Ton Volgenant of the University of Amsterdam for sharing their software with us. This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C0182, the Air Force Office of Scientific Research under Contract Number AFOSR 87-0199, the Office of Naval Research under Contract Number N00014-87-K-0223, and ROME Air Development Center under Contract Number SCEEE PDP/87-95. This manuscript has benefited from a careful reading by Richard V. Helgason of Southern Methodist University.

I. INTRODUCTION

The classical assignment problem (also known as the weighted bipartite matching problem) is to assign n men to n distinct jobs so that the total cost of assignment is minimized. Mathematically, this may be formulated as the following special mathematical program:

$$\text{minimize} \quad \sum_{i,j} c_{ij}\, x_{ij}$$

$$\text{subject to:} \quad \sum_{i} x_{ij} = 1, \quad \text{(all j)}$$

$$\sum_{j} x_{ij} = 1, \quad \text{(all i)}$$

$$x_{ij} \in \{0, 1\}, \quad \text{(all i, j)}$$

where c_{ij} denotes the cost for assigning man i to job j and $x_{ij} = 1$ implies that man i is assigned to job j. Due to the total unimodularity of the constraint matrix, this problem can be solved by the simplex algorithm and every basic solution will have $x_{ij} \in \{0, 1\}$ (see Kennington and Helgason [1980]). Hence, classic linear programming duality theory and Kuhn–Tucker optimality conditions can be used in algorithm development for this problem.

Specialized algorithms for the assignment problem can be classified into five categories as follows:

(i) maximum flow (primal–dual),

(ii) primal simplex,

(iii) dual simplex,

(iv) auction algorithm, and

(v) shortest augmenting paths.

The first maximum flow algorithm was developed by Kuhn [1955] and is called the Hungarian method. Its name comes from the fact that the algorithm is developed from results of two Hungarian mathematicians. Variations were presented by Kuhn [1956]. Derigs

[1985] shows that the shortest augmenting path method can be viewed as a more economical implementation of the Hungarian method. Ahuja, Magnanti, and Orlin [1988] call the Hungarian method a primal-dual variant of the successive augmenting path algorithm.

A specialized primal simplex algorithm for the assignment problem was developed by Barr, Glover and Klingman [1977]. Their method, called the alternating basis algorithm, only considers a subset of the possible bases. This idea was further exploited by Hung [1983] in his development of a polynomial simplex algorithm for this problem. An extensive computational study comparing the Hungarian algorithm with primal simplex methods was performed by McGinnis [1983].

A dual polynomial simplex method known as the signature method has been developed by Balinski [1985, 1986]. Extensions for the sparse assignment problem were developed by Goldfarb [1985], and the relationship between the signature method and the shortest augmenting path method was presented by Derigs [1985]. Another variation of this algorithm has been developed by Akgul [1988].

Bertsekas [1979, 1981, 1987] and Bertsekas and Eckstein [1988] give a complete theoretical development of the auction algorithm. This algorithm critically depends on the ideas of ϵ-complementary slackness and adaptive scaling. The latest version of Bertsekas' auction code was completed in June 1988 and has been placed in the public domain. Barr and Christiansen [1989] have experimented with a parallel version of this algorithm written in C++ on the Sequent Symmetry S81, and Phillips and Zenios [1988] have experimented with this algorithm on the Connection Machine. Perry [1988] experimented with a parallel version of the auction algorithm on both the Alliant FX-8 and the Sequent Symmetry S81.

Hung and Rom [1980] presented a shortest augmenting path method which had both a polynomial bound and good computational results. Other variants of the shortest augmenting path method have been presented by Glover, Glover and Klingman [1986] and Jonker and Volgenant [1987]. An extensive computational study with the shortest augmenting path method may be found in Derigs [1985].

Recently, scaling based algorithms have been presented for the assignment problem (see Gabow [1985] and Orlin and Ahuja [1988]). These algorithms are derivatives of the Hungarian method and the auction algorithm, respectively, with the added feature of data scaling. Bertsekas is using a similar idea in his latest auction code.

There are three ways to analyze the performance of an algorithm: worst-case analysis, average case analysis, and empirical analysis. The worst-case analysis results for the assignment problem are presented in the excellent report by Ahuja, Magnanti, and Orlin [1988]. The objective of our study is to present an empirical analysis of the two top performing serial codes. These codes were obtained directly from the authors, and they represent the current best software implementation of the top competing algorithms. They were run on three different machines (IBM 3081D, Sequent Symmetry S81, and VAX 750) to allow for an analysis with respect to differences in machine architecture and compiler. The best code was then parallelized and the speedup achieved on a shared memory multiprocessor was reported.

II. SEQUENTIAL CODES

Five algorithms for solving dense assignment problems have been implemented and computationally compared by various researchers. We are not aware of any computational studies involving the dual algorithms. Derigs [1985] shows that a shortest augmenting path implementation is superior to a Hungarian implementation. This has been con-

firmed by Jonker and Volgenant [1987] and by the authors in a comparison with the codes of Jonker and Volgenant [1987] and Rardin [1986]. Glover, Glover and Klingman [1977] found that their shortest augmenting path code was superior to the specialized simplex code of Barr, Glover and Klingman [1977]. The authors have confirmed this with a comparison of the Jonker and Volgenant [1987] and Barr, Glover and Klingman [1977] codes. Jonker and Volgenant [1987] also concluded that their dense shortest augmenting path code was superior to the auction code of Bertsekas [1981].

After many studies over a fifteen year period, it is acknowledged that the two best algorithms for dense assignment problems are the auction algorithm and the shortest augmenting path algorithm. We believe that the best software implementation of the auction algorithm is the code of Professor Bertsekas (Version 1.0, June 1988). Most of the other auction codes that we have seen are very sensitive to the cost structure and degrade as the cost range becomes larger. These other codes sometimes work very well for small cost ranges and fail miserably for cost ranges as small as [0,1000]. By the use of adaptive scaling, Professor Bertseklas' code works well for both a small cost range and a large cost range. This code scales all cost data by n+1 and solves a sequence of problems with decreasing values of the stopping criterion. All calculations are performed in integer arithmetic. We believe that the best implementation of the shortest augmenting path algorithm was developed by Jonker and Volgenant [1987]. Dijkstra's algorithm is used to obtain the shortest augmenting paths and only integer arithmetic is required. The code also incorporates an elaborate pre-processing stage which greatly reduces the total number of times that the Dijkstra algorithm is required. It also uses a clever data structure for updating the dual variables after a shortest augmenting path has been found. The code maintains dual variables and the reduced costs are calculated as required. Both codes are written in standard FORTRAN.

The empirical results of our experiment are presented in Table 1. For all test problems, all cost ranges, and all machines, the shortest augmenting path code dominated the auction code. The auction code had the greatest difficulty when the cost range was the smallest, i.e. [0,100]. When the cost range was at least [0,1000], the auction algorithm was affected very little by the cost range. The shortest augmenting path code was adversely affected by an increasing cost range. The machine type definitely affected the empirical analysis. On the Sequent, the shortest augmenting path code was 4.41 times faster than the auction code, on the IBM it was 3.87 times faster, while on the VAX it was 2.79 times faster. This confirms our belief that the comparative performance of two codes is intimately linked to the hardware, operating system, and compilers used. For our tests the IBM was running CMS and both the Sequent and the VAX were running UNIX™†. Our results contradict the widely held belief that the auction algorithm converges faster for lower cost ranges. Professor Bertsekas' latest version of the auction code for dense problems was not sensitive to changing the cost range from [0,1000] to [0,100 000]. In fact the shortest augmenting path code is much more sensitive to the larger cost ranges than the auction code is. We also observed the well-known phenomena of the auction algorithm that a significant amount of the computational time is spent attempting to complete the last few assignments. This is in contrast to the shortest augmenting path algorithm that achieves one more assignment with each application of Dijkstra's algorithm. Each application of the shortest path algorithm can be very expensive, but it is guaranteed to result in one more assignment. This feature along with the extensive preprocessing to obtain a good set of partial assignments makes this approach work extremely well.

---

† UNIX is a trade mark of AT&T Bell Laboratories.

III. A PARALLEL SHORTEST AUGMENTING PATH CODE

The algorithm of Jonker and Volgenant [1987] may be divided into three procedures as follows:

(i) column reduction,

(ii) augmenting row reduction, and

(iii) augmentation using a shortest path procedure.

The first two procedures require the fundamental operation of obtaining $\min\{d[i] : i \in K\}$ for a given vector $d[\cdot]$ and a given index set K. This operation is required twice during the column reduction and once during the row reduction. These three "minimum of a vector" operations have been parallelized by using prescheduled data partitioning. For a p processor run, K is partitioned into p subsets $K_1, \ldots, K_p$ having $K = \bigcup_{j=1,\ldots,p} K_j$ and $K_j \cap K_k = \Phi$ for all j, k. Processor j calculates $\min\{d[i] : i \in K_j\}$, and the global minimum is set to the smallest of the local minima. In the shortest path procedure using Dijkstra's algorithm, scanning a node requires comparing the current distance label with the distance label at the node being scanned plus the length of a given arc. The distance label is either updated with the new shortest path or remains unchanged. All of this work is independent and can also be distributed among p processors.

The Jonker-Volgenant algorithm is presented below:

THE SHORTEST AUGMENTING PATH ALGORITHM

Input:

1. The problem size, n.

2. The nxn cost matrix, c(i, j).

Output:

1. x[i] = j implies that man i is assigned to job j.

2. y[j] = i implies that job j is assigned to man i.

3. v[j] denotes the dual variable associated with job j.

procedure COLUMN REDUCTION

begin

1. x[i] ← 0, i = 1, ..., n;

2. for j = 1, ..., n

3. $\mu \leftarrow \min \{c[i, j] : i = 1, ..., n\}$;

4. v[j] ← μ and let $i^* \in \{i: c[i, j] = \mu\}$;

5. if $x[i^*] = 0$, then $x[i^*] \leftarrow j$, $y[j] \leftarrow i^*$;

6. end for

7. for i = 1, ..., n

8. if $x[i] \neq 0$, then

9. $\mu \leftarrow \min \{c[i, j] - v[j]: j = 1, ..., n \text{ and } j \neq x[i]\}$;

10. $v[x[i]] \leftarrow v[x[i]] - \mu$;

11. end if

12. end for

end

procedure AUGMENTING ROW REDUCTION

begin

13. l ← 0, t ← 0;

14. for i=1, ..., n

15. if x[i] = 0, then l ← l+1, f[l] ← i;

16. end for

17. if l = 0, then terminate with an optimum;

18. m ← l, k ← 1, l ← 0;

19. i ← f[k], k ← k+1;

20. $u_1 \leftarrow \min\{c[i, j] - v[j]: j=1, \ldots, n\}$, let $j_1 \in \{j: c[i, j] - v[j] = u_1\}$, $u_2 \leftarrow \min\{c[i, j] - v[j]: j=1, \ldots, n \text{ and } j \neq j_1\}$, let $j_2 \in \{j: c[i, j] - v[j] = u_2$ and $j \neq j_1\}$;

21. $i_1 \leftarrow y[j_1]$;

22. if $u_1 < u_2$, then $v[j_1] \leftarrow v[j_1] + u_1 - u_2$;

23. else if $i_1 = 0$, then go to 26, else $j_1 \leftarrow j_2$, $i_1 \leftarrow y[j_1]$;

24. if $i_1 = 0$, then go to 26;

25. if $u_1 < u_2$, then k ← k-1, f[k] ← i_1; else l ← l+1, f[l] ← i_1;

26. x[i] ← j_1, $y[j_1] \leftarrow i$;

27. if k ≤ m go to 19;

28. t ← t+1;

29. if l > 0 and t <2, then go to 18, else continue with procedure BUILD A TREE

end

procedure BUILD A TREE

begin

30. $m \leftarrow 1$

31. for $l = 1, \ldots, m$

32. $i^* \leftarrow f[l]$;

33. READY $\leftarrow \Phi$, TODO $\leftarrow \{1, \ldots, n\}$, RSINK $\leftarrow \{j: y[j] = 0\}$;

34. $d[j] \leftarrow c[i^*, j] - v[j]$, $pred[j] \leftarrow i^*$, $j=1, \ldots, n$;

35. $\mu \leftarrow \min\{d[j]: j \in TODO\}$, SCAN $\leftarrow \{j: d[j] = \mu, j \in TODO\}$, TODO $\leftarrow$ TODO\SCAN;

36. if SCAN $\cap$ RSINK $\neq \Phi$, then go to 45;

37. for all $j_1 \in$ SCAN

38. $i \leftarrow y[j_1]$, $h \leftarrow c[i, j_1] - v[j_1] - \mu$;

39. for all $j \in$ TODO

40. $p \leftarrow c[i, j] - v[j] - h$;

41. if $p < d[j]$, then $d[j] \leftarrow p$, $pred[j] \leftarrow i$;

42. end for

43. end for

44. go to 35

45. $v[j] \leftarrow v[j] + d[j] - \mu$, for all $j \in$ READY;

46. let j$\in$ SCAN $\cap$ RSINK

47. i $\leftarrow$ pred[j], y[j] $\leftarrow$ i, k $\leftarrow$ j, j $\leftarrow$ x[i], x[i] $\leftarrow$ k;

48. if i $\neq$ i* go to 47;

49. end for

end

Steps 3, 9, 20 and 39 are the most computationally expensive and it is precisely these steps which have been parallelized.

The dense assignment code of Jonker and Volgenant [1987] has been parallelized using this strategy and run on a Symmetry S81 from Sequent Computer Systems, Inc. This Symmetry S81 is a multiprocessor system with 32 Mbytes of shared memory and twenty Intel 80386 cpu's. For this study, both codes used only integer arithmetic and did not make use of math co-processors.

The empirical analysis with the parallel code is presented in Table 2 with the corresponding speedups given in Table 3. Each time is the average for five runs. Note that the speedup for the parallel code using a single processor ranged from a high of 0.87 to a low of 0.73. This implies that the overhead associated with parallel processing for this code ranged from 13% to 27%. As expected the speedups increased as the problem size increases. For the largest problems, speedups of approximately four were achieved using ten processors.

We have also experimented extensively with parallel versions of the auction algorithm, but our results were not competitive with those presented in Table 2. We also developed a modification of the shortest augmenting path code which used a Dijkstra two-tree shortest path algorithm (see Helgason, Kennington and Stewart [1988]), but that system was not competitive with the original shortest augmenting path implementation. At the termi-

nation of the classical Dijkstra shortest path algorithm, all the information required to update the duals is available and the dual update can be executed very efficiently. The two-tree Dijkstra method can obtain the shortest augmenting path faster than the classical Dijkstra shortest path method; however, additional work is required to discover which duals must be changed and by what amount. The overhead required for the dual variable updates exceeded the potential benefits of the two-tree Dijkstra method for finding the shortest augmenting path.

For the problem sizes and cost ranges analyzed, the times in Table 2 are the best times that we have seen. The parallel shortest augmenting path code is a powerful tool that can easily solve all 1,000,000 arc dense assignment problems in less than seventeen seconds using six processors. The 1,000,000 arc dense assignment problem with a cost range of [1,1000] required over five minutes for the parallel auction code of Barr and Christiansen [1989] using six processors.

IV. SUMMARY AND CONCLUSIONS

The empirical analysis presented in this study indicates that for dense assignment problems having a size up to 800x800, the shortest augmenting path software is faster than the auction algorithm software. This conclusion was based on test runs with sixteen randomly generated test problems with four different cost ranges and run on three different serial machines. Contrary to the widely held belief that the auction algorithm performs worse as the cost range increases, we found this not to be the case. We believe that Professor Bertsekas' latest implementation (Version 1.0, June 1988) has eliminated this difficulty. We did observe the difficulty with the "end game" in which an inordinate amount of time is required to complete the last few assignments. The shortest augmenting path method has the attractive feature that each time a shortest path is calculated, one new assignment is made. We found that the shortest augmenting path code was adversely

affected by an increasing cost range. As the cost range increases, larger trees must be developed by Dijkstra's algorithm to obtain the shortest path from an unassigned man to an unassigned job.

With only a moderate amount of code development, we parallelized the shortest augmenting path code of Jonker and Volgenant [1987] for the Sequent Symmetry S81. Speedups of approximately four were achieved on 1200x1200 dense problems using ten processors. Remarkably, 1,000,000 arc dense assignment problems were solved using this parallel code in less than fifteen seconds (wall clock time). Even though this code was developed for a particular multiprocessor system with shared memory, it can be used with any shared memory parallel processing system.

REFERENCES

Ahuja, R., T. Magnanti, and J. Orlin, [1988], "Network Flows," Sloan Working Paper No. 2059-88, Massachusetts Institute of Technology, Cambridge, MA 02139.

Akgul, M., [1988], "A Sequential Dual Simplex Algorithm for the Linear Assignment Problem," Operations Research Letters, 7, 155-158.

Balinski, M., [1985], "Signature Methods for the Assignment Problem," Operations Research, 33, 527-536.

Balinski, M., [1986], "A Competitive (Dual) Simplex Method for the Assignment Problem," Mathematical Programming, 34, 125-141.

Barr, R. and M. Christiansen, [1989], "A Parallel Auction Algorithm: A Case Study in the Use of Parallel Object-Oriented Programming," To appear in R. Sharada, et al., Impact of Recent Computer Advances on Operations Research, North-Holland Publishing Company, Amsterdam.

Barr, R., F. Glover and D. Klingman, [1977], "The Alternating Basis Algorithm for Assignment Problems," Mathematical Programming," 13, 1-13.

Bertsekas, D., [1979], "A Distributed Algorithm for the Assignment Problem," Laboratory for Information and Decision Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139.

Bertsekas, D., [1981], "A New Algorithm for the Assignment Problem," Mathematical Programming, 21, 152-171.

Bertsekas, D., [1987], "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," Technical Report LIDS-P-1653, Massachusetts Institute of Technology, Cambridge, M.A.

Bertsekas, D., and J. Eckstein, [1988], "Dual Coordinate Step Methods for Linear Network Flow Problems," to appear in Mathematical Programming, Series B.

Derigs, U., [1985], "The Shortest Augmenting Path Method for Solving Assignment Problems - Motivation and Computational Experience," Annals of Operations Research, 4, 57-102.

Gabow, H., [1985], "Scaling Algorithms for Network Problems," Journal of Computer and System Sciences, 31, 148-168.

Glover, F., R. Glover and D. Klingman, [1986], "Threshold Assignment Algorithm," Mathematical Programming Study, 26, 12-37.

Goldfarb, D., [1985], "Efficient Dual Simplex Algorithms for the Assignment Problem," Mathematical Programming, 33, 187-203.

Helgason, R., J. Kennington and D. Stewart, [1988], "Dijkstra's Two-Tree Shortest Path Algorithm," Technical Report 88-OR-13, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, Texas 75275.

Hung, M., [1983], "A Polynomial Simplex Method for the Assignment Problem," Operations Research, 31, 595-600.

Hung, M. and W. Rom, [1980], "Solving the Assignment Problem by Relaxation," Operations Research, 28, 969-982.

Jonker, R. and T. Volgenant, [1987], "A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems," Computing, 38, 325-340.

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

Kuhn, H., [1955], "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, 2, 83-97.

Kuhn, H., [1956], "Variants of the Hungarian Method for Assignment Problems," Naval Research Logistics Quarterly, 3, 253-258.

McGinnis, L., [1983], "Implementation and Testing of a Primal-Dual Algorithm for the Assignment Problem," Operations Research, 31, 277-291.

Orlin, J. and R. Ahuja, [1988], "New Scaling Algorithms for the Assignment and Minimun Cycle Mean Problems," Technical Report, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139

Perry, E., [1988], "Programming Assignment Algorithms on Parallel and Vector Machines," Technical Report Ford Aerospace Corp., Colorado Springs, CO 80908.

Phillips, C. and S. Zenios, [1988], "Experiences with Large Scale Network Optimization on the Connection Machine," To appear in R. Sharada, et al., Impact of Recent Computer Advances on Operations Research, North-Holland Publishing Company, Amsterdam.

Rardin, R., [1986], "Private Communication."

Table 1. Comparison of Sequential Algorithms on Dense nxn Assignment Problems (all times are in seconds)

n	cost range	auction version 1.0 June 1988			shortest augmenting path algorithm		
		IBM 3081D	Symmetry S81	VAX 750	IBM 3081D	Symmetry S81	VAX 750
200	0 - 100	2.36	12.78	42.4	0.67	1.19	5.7
	0 - 1000	2.13	5.19	15.8	0.89	1.56	6.9
	0 - 10000	2.08	5.98	20.8	1.02	1.83	8.2
	0 - 100000	2.14	6.61	22.5	1.70	3.06	12.6
400	0 - 100	16.48	33.23	111.4	2.98	5.03	27.3
	0 - 1000	9.60	13.99	51.6	3.40	5.83	26.4
	0 - 10000	9.55	17.45	62.3	3.50	6.09	38.7
	0 - 100000	10.23	19.54	69.0	3.91	6.90	41.9
600	0 - 100	46.32	228.13	741.6	5.55	9.42	59.7
	0 - 1000	29.58	42.73	158.8	7.30	12.58	75.0
	0 - 10000	27.73	43.08	154.5	8.15	14.26	78.6
	0 - 100000	30.66	52.08	187.8	9.56	16.82	93.6
800	0 - 100	83.41	101.38	364.2	8.43	14.25	79.3
	0 - 1000	42.69	67.33	251.2	11.22	19.22	130.4
	0 - 10000	48.32	73.54	272.4	16.62	28.72	144.0
	0 - 100000	43.43	80.31	288.4	20.23	35.57	182.2
TOTAL		406.71	803.35	2814.7	105.13	182.33	1010.5

Table 2. The Parallel Shortest Augmenting Path Code (all times are in seconds for dense n x n assignment problems)

n	cost range	Sequential sap code (secs.)	cpu's used with parallel shortest augmenting path code									
			1	2	3	4	5	6	7	8	9	10
1000	0 - 100	20.02	27.56	13.56	9.51	8.53	7.28	6.45	6.11	5.84	5.38	5.22
	0 - 1000	29.39	35.35	24.05	16.76	14.36	12.06	11.65	9.90	9.65	8.99	8.27
	0 - 10000	41.56	51.87	29.01	21.01	18.03	15.21	13.62	13.19	11.96	11.81	11.34
	0 - 100000	46.73	59.45	33.44	25.94	20.97	18.68	16.80	16.52	14.46	14.19	13.61
1100	0 - 100	23.29	27.53	15.66	11.74	9.58	8.34	7.09	6.73	6.46	6.18	5.79
	0 - 1000	40.31	50.73	30.59	21.97	20.13	17.30	14.44	13.51	12.96	12.88	11.76
	0 - 10000	63.26	77.67	47.60	35.29	30.06	26.27	24.61	22.16	21.31	19.80	18.63
	0 - 100000	70.99	88.10	53.10	38.47	33.83	30.52	26.21	24.94	22.49	21.68	21.12
1200	0 - 100	27.15	35.41	18.13	12.98	11.30	9.57	8.70	7.82	7.68	7.29	6.64
	0 - 1000	59.82	69.00	37.12	30.58	24.85	20.78	18.28	16.51	16.08	14.86	13.73
	0 - 10000	75.25	93.86	52.97	37.13	33.19	27.49	26.18	24.46	22.90	21.70	19.34
	0 - 100000	86.01	106.72	52.71	42.96	37.16	30.32	29.19	26.59	24.49	24.24	21.82

Table 3. Speedup for Dense n x n Assignment Problems

n	cost range	Sequential sap code	cpu's used with the parallel shortest augmenting path code									
			1	2	3	4	5	6	7	8	9	10
1000	0 - 100	1.00	0.73	1.48	2.10	2.35	2.75	3.10	3.27	3.43	3.72	3.83
	0 - 1000	1.00	0.83	1.22	1.75	2.05	2.44	2.52	2.97	3.04	3.27	3.55
	0 - 10000	1.00	0.80	1.43	1.98	2.31	2.73	3.05	3.15	3.48	3.52	3.67
	0 - 100000	1.00	0.79	1.40	1.80	2.23	2.50	2.78	2.83	3.23	3.29	3.43
1100	0 - 100	1.00	0.85	1.49	1.98	2.43	2.79	3.29	3.46	3.51	3.77	4.02
	0 - 1000	1.00	0.79	1.32	1.84	2.00	2.33	2.79	2.98	3.11	3.13	3.43
	0 - 10000	1.00	0.81	1.33	1.79	2.10	2.41	2.57	2.85	2.97	3.20	3.39
	0 - 100000	1.00	0.81	1.34	1.85	2.10	2.33	2.71	2.85	3.16	3.28	3.36
1200	0 - 100	1.00	0.77	1.50	2.09	2.40	2.84	3.12	3.47	3.53	3.72	4.09
	0 - 1000	1.00	0.87	1.61	1.96	2.41	2.88	3.27	3.62	3.72	4.03	4.36
	0 - 10000	1.00	0.80	1.42	2.03	2.27	2.74	2.87	3.08	3.29	3.47	3.89
	0 - 100000	1.00	0.81	1.63	2.00	2.31	2.84	2.95	3.24	3.51	3.55	3.94

Technical Report 88-OR-21

SOLVING GENERALIZED NETWORK PROBLEMS ON A SHARED MEMORY MULTIPROCESSOR

by

J. Kennington
R. Muthukrishnan

Department of Operations Research and Engineering Management
School of Engineering and Applied Science
Southern Methodist Univeristy
Dallas, Texas 75275

November 1988

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

The objective of this investigation was to analyze simplex based algorithms for the generalized network problem in both the sequential and parallel computational environment. In comparisons with MPSX, it was shown that a generalized network code is ten times faster than this general linear programming system. 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 ten percent. A parallel asychronous primal simplex algorithm was developed and tested on a Sequent Symmetry S81. Test problems having two thousand nodes and fifty thousand arcs were solved in from three to four minutes using a single cpu and in less than two minutes using eight cpu's.

ACKNOWLEDGMENT

This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C0182, the Air Force Office of Scientific Research under Contract Number AFOSR 87-0199, and the Office of Naval Research under Contract Number N00014-87-K-0223

I. INTRODUCTION

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

$$\text{minimize } cx \tag{1}$$

$$\text{s.t.} \quad Gx = r \tag{2}$$

$$0 \leq x \leq u, \tag{3}$$

where G is an $\bar{m}x\bar{n}$ matrix having at most two nonzero entries in each column, c is a $1x\bar{n}$ vector of costs, r is an $\bar{m}x1$ vector of right-hand-sides, and u is an $\bar{n}x1$ vector of upper bounds. Many authors place the additional restriction on (1)-(3) that at least one of the nonzero entries in each column of G be a one (minus one). Not all instances of (1)-(3) can be scaled to meet this restriction and the new code developed for this investigation does not require this restriction. 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. If the k^{th} column of G, G(k), has a single nonzero entry in row i, then the corresponding edge is denoted by (i,i). If G(k) has two nonzero entries in rows i and j, then the corresponding edge is denoted by (i,j).

1.1. Survey of Literature

The graphical structure of a basis for G allows the use of labeling procedures for basis representation. Glover, Klingman, and Stutz [1973] 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, Hultz, Klingman, and Stutz [1978] and Elam, Glover, and Klingman [1979]). A similar implementation was also developed by Langley [1973]. Adolphson and Heum [1981] presented computational results with their generalized code which used an extension of the threaded index method of Glover, Klingman, and Stutz [1974]. Brown and

McBride [1984] presented the details of their generalized network code (GENNET) and offered the source code to our university for a nominal price. Tomlin [1984] 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 [1985], Mulvey and Zenios [1985], and by Ali, Charnes, and Song [1986]. The first parallel generalized code was developed by Chang, Enquist, Finkel, and Meyer [1987] for the Wisconsin Crystal multicomputer and the second by Clark and Meyer [1987]. The first C language code was developed by Nulty and Trick [1988]. Another assembly language code has been developed by Chang, Chen, and Chen [1988]. The code discussed in this paper is one of eight developed by Muthukrishnan [1988]. A summary of the available software may be found in Table 1.

1.2. Parallel Computing Machines

The UNIVAC 1, which appeared in 1951, was the first commercially produced computer. Four generations of computer development have since passed with each generation exhibiting major improvements over the previous one. These improvements were made possible in part by the introduction of greater parallelism at all levels of the computer architecture. The first generation machines were built with vacuum tubes and electromechanical relays. The second generation machines were build with discrete diodes, transistors, and printed circuit boards. Integrated circuits were introduced in the third generation machines and large scale integration is used in the fourth generation machines.

The first significant work in the area of parallel processing occurred in the 1970's with the development of the C.mmp at Carnegie Mellon University. In early digital computers, the memory and the central processing unit were made of different materials. Memory was cheaper than processing and the early parallel machines consisted primarily of minicomputers with small address spaces and limited computing power. Hence, they were

fairly expensive slow machines compared to a single processor mainframe. A technological breakthrough made possible the fabrication of both the memory and the cpu from the same material. The cost of the processors was reduced and it became possible in the 1980's to build inexpensive multiprocessor machines. Now over twenty different parallel computers are commercially available.

Parallel machines are classified according to the operational characteristics of both instruction and data streams. The stream characteristics are categorized as follows: single instruction stream (SI), multiple instruction stream (MI), single data stream (SD) and multiple data stream (MD). In SISD machines, a single stream of instructions operates on a single stream of data. This is the classical von Neuman architecture which is used in the following machines: VAX 11/780, IBM 3081, and Cray-1. Machines which use array processors, such as the CDC Cyber 205 and IBM 3090-600, and The Connection Machine CM-1 with 64,000 one bit processors are called SIMD machines. From an algorithm designers point of view, the most interesting machines are MIMD machines. In these machines, each processor can execute different instructions on different data segments simultaneously. Among others, the Sequent Symmetry S81, Encore Multimax, Intel iPSC hypercube, FPS T-20 hypercube, Butterfly, Alliant FX/8 and AT&T KORBX belong to this category.

MIMD machines can be further classified as either multiprocessors or multicomputers. Multiprocessors have both private and shared memory while multicomputers have only private memory. Hence, communication among cpu's on a multicomputer is through the slower procedure of message passing. Within the class of multiprocessors further distinction can be made depending on the processor interconnection pattern as tightly coupled and loosely coupled machines. The new hardware developed by Alliant, Cray, Encore, and Sequent are tightly coupled shared memory MIMD machines.

1.3. Objective of the Investigation

The primary objective of this investigation was to develop and computationally test a parallel primal simplex based algorithm for the generalized network problem. This model was selected for investigation due to the fact that the basis for a generalized network problem decomposes naturally into a block diagonal structure which can be easily maintained and updated using graphical data structures (labeling procedures). This study begins with the excellent generalized network code, GENNET, developed by Brown and McBride [1984].

II. PARALLEL CONSTRUCTS

Parallel algorithms use modules (subroutines) which may be executed in parallel. Suppose there are τ processors available for use. The set procs parallel programming construct generates $\tau - 1$ clones of the running process and places them in a wait state. The parallel operations are initiated by the main program using statements of the form:

for processors = 1 to τ , fork module WORK.

The main program and the $\tau - 1$ clones each execute module WORK. Processing in the main program continues only after all processors complete execution of WORK and the clones return to a wait state until the next fork is executed. In order to allow for mutual exclusion of certain sections of code, variables can be designated as locks. Variables so designated can assume two states: locked (1) or unlocked (0). Locked sections of code appear as follows:

lock [s]

.

.

.

unlock [s].

If a process reaches a lock statement and s=0, then it sets s to 1 and continues. Otherwise, the process spins until s=0, then it sets s to 1 and continues. When a process reaches an unlock statement it sets s to 0. Two processors can never execute the code between the lock and unlock statements simultaneously.

III. PARALLEL SIMPLEX FOR GENERALIZED NETWORKS

The best algorithm for solving the generalized network problem is a specialization of the primal simplex method which exploits the underlying graphical structure of the model. By a rearrangement of rows and columns, every basis for (1)–(3) can be placed in the block diagonal form

$$\begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_p \end{bmatrix}$$

where each B_i , i=1, ..., p, is either lower triangular or nearly lower triangular with only one element above the diagonal. Furthermore, each component B_i corresponds to a connected graph and all the simplex operations can be carried out directly on this graph. The simplex algorithm for this model in its most general form is presented below:

Primal Simplex Algorithm for Generalized Networks

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 $\bar{x}[e]$ for all $e \in E$. ($\beta = 1$ 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 $\bar{x}[e]$ for all $e \in E$.)

Working Entities:

1. An array π [n], the dual variable for each $n \in V$.
2. An array y[n], the component of the updated column for each $n \in V$.
3. A set E_B of basic arcs.
4. A matrix B of columns of G corresponding to E_B.

Procedure SIMPLEX

begin

1. let $E_B \subset E$ denote the set of basic arcs with corresponding basis B, and let $\bar{x}$ [e] denote the flows for all $e \in E$;
2. $\beta \leftarrow 0$;
3. $\pi \leftarrow c_B B^{-1}$, where c_B denotes the costs associated with E_B;
4. call module PRICE;
5. if $\beta \neq 0$, then terminate;
6. call module RATIO;
7. if $\beta \neq 0$, then terminate;
8. call module UPDATE;
9. go to 4.

end.

Procedure PRICE

begin

1. $P_L = \{j : \pi G(j) - c[j] > 0, \bar{x}[j] = 0, j \in E \backslash E_B\}$;
2. $P_U = \{j : \pi G(j) - c[j] < 0, \bar{x}[j] = u[j], j \in E \backslash E_B\}$;
3. if $P_L \bigcup P_U = \phi$, then
4. if E_B contains an artificial variable having flow > 0, then $\beta \leftarrow 2$; otherwise,

$\beta \leftarrow 3$;

else

5. select $k \in P_L \bigcup P_U$;

6. if $k \in P_L$, then $\delta \leftarrow 1$; otherwise, $\delta \leftarrow -1$;

end if

end.

Procedure RATIO

begin

1. $y \leftarrow B^{-1}G(k)$

2. $\Delta_1 \leftarrow \min\{\frac{\bar{x}[j]}{|y[i]|}, \infty : \sigma(y[i]) = \delta, G(j) = B(i), \ i = 1, ..., |V|\}$;

3. $\Delta_2 \leftarrow \min\{\frac{u[j] - \bar{x}[j]}{|y[i]|}, \infty : -\sigma(y[i]) = \delta, G(j) = B(i), \ i = 1, ..., |V|\}$;

4. $\Delta \leftarrow \min\{\Delta_1, \Delta_2, u[k]\}$;

5. if $\Delta = \infty$, then $\beta \leftarrow 1$;

end.

Procedure UPDATE

begin

1. $\bar{x}[k] \leftarrow \bar{x}[k] + \Delta\delta$

2. $\bar{x}[j] \leftarrow \bar{x}[j] - \Delta\delta y[i]$ for $i = 1, ..., |V|$ and $G(j) = B(i)$;

3. if $\Delta \neq u[k]$, then

4. $U_L = \{j : \bar{x}[j] = 0, \sigma(y[i]) = \delta, G(j) = B(i)\}$;

5. $U_U = \{j : \bar{x}[j] = u[j], -\sigma(y[i]) = \delta, G(j) = B(:)\}$;

6. select $r \in U_L \bigcup U_U$

7. $E_B \leftarrow (E_B \cup \{k\})\backslash\{r\}$ and update B so that it corresponds to E_B.

8. $\pi \leftarrow c_B B^{-1}$, where c_B denotes the costs associated with E_B;

9. end if

end

Since the components of the partitioned basis

$$\begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_p \end{bmatrix}$$

correspond to connected components of a graph, the partitioning is easily maintained after a pivot is performed. Since each column of G has at most two nonzero entries, a simplex pivot can affect at most two of the p partitions. Therefore, multiple processors can be performing pivot operations simultaneously on different partitions of the basis. In this parallel implementation of the simplex algorithm, all processors execute the simplex method asynchronously and a description of the method requires only an explanation of the operation of a single processor.

For a processor, the pricing module is called and some edge which prices favorably is selected for flow change. If the component(s) associated with this edge are locked, then the processor returns to the pricing module. Otherwise, the component(s) are locked and the candidate edge is repriced. If the candidate edge prices unfavorably during the repricing stage, the component(s) are unlocked and the processor returns to the pricing module. Pivoting is executed only if the candidate edge prices favorably when repriced. After returning from the update module, the affected components are unlocked. The parallel simplex algorithm specialization in its most general form is presented below:

Parallel Primal Simplex Algorithm for Generalized Networks

Input:

1. A graph [V,E].
2. A cost c[e], arc capacity u[e], from (tail) node f[e], and to (head) node t[e] for all $e \in E$.
3. The generalized constraint matrix G.
4. A requirement r[n] for all $n \in V$.
5. The number of processors, τ , available for use.

Output:

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

Working Entities:

1. An array $\pi[n]$, the dual variable for each $n \in V$.
2. An array y[n], the component of the updated column for each $n \in V$.
3. A set E_B of basic arcs.
4. A matrix B of columns of G corresponding to E_B.
5. A variable s to be used as a lock.
6. An array comp[n] which gives the component number in which $n \in V$ is a member.
7. An array lockc[m] which has the value of one if component m is busy and zero, otherwise.

Procedure PARALLEL SIMPLEX

begin

1. let $E_B \subset E$ denote the set of basic arcs with corresponding basis B, and let $\bar{x}$ [e] be the flows for all $e \in E$;
2. lockc[n]$\leftarrow 0$ for all $n \in V$;
3. let B be partitioned into

$$\begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_p \end{bmatrix}$$

corresponding to the connected components of $[V, E_B]$ and set comp[n] equal the partition number for which node (row) n is a member;

4. set procs τ
5. for processors = 1 to τ , fork module SOLVER;

end.

Procedure SOLVER

local data: β, k, δ, Δ, $\bar{f}$, $\bar{t}$, U_L, U_u, r

begin

1. $\beta \leftarrow 0$;
2. $\pi \leftarrow c_B B^{-1}$, where c_B denotes the costs associated with E_B;
3. call module PRICE;
4. if $\beta \neq 0$, then return;
5. lock s
6. if lockc[comp[f[k]]] = 0 and lockc[comp[t[k]]] = 0, then;

```
7.        (comment: lock components associated with the entering arc)
8.        lockc[comp[f[k]]] ← 1;
9.        lockc[comp[t[k]]] ← 1;
              (comment: reprice entering arc)
10.           if (x̄[k] = 0 and πG(j) - c[j] ≤ 0) or (x̄[k] = u[k] and πG(j) - c[j] ≥ 0)
              then
              (comment: entering arc does not price favorable, some other processor
              modified dual after pricing)
11.               lockc[comp[f[k]]] ← 0;
12.               lockc[comp[t[k]]]← 0;
13.               unlock [s];
14.               go to 3;
15.           end if
16.           unlock s;
17.    else
18.           unlock s;
19.           go to 3;
20.    end if
21.    call module RATIO;
22.    if β ≠ 0, then return;
23.    f̄ ← comp[f[k]]], t̄ ← comp[t[k]]];
24.    call module UPDATE;
25.    update comp[n] for all n ∈ V to correspond to E_B;
26.    lockc[f̄ ]← 0, lockc[t̄ ]← 0;
27.    go to 3;

   end
```

IV. COMPUTATIONAL EXPERIENCE

This investigation began with the excellent generalized network code of Brown and McBride [1984] known as GENNET. This code requires that the problem be scaled so that at least one of the nonzero entries in every column of G must be one. This restriction has two disadvantages, (i) it means that not all instances of (1)-(3) can be solved with this code and (ii) it is awkward to use this code as the continuous relaxation solver within a branch-and-bound framework for integer generalized networks. That is, scaling an integer variable prior to application of an integer solver requires special handling of that integer variable. The modified code which allows for arbitrary nonzero entries in each column of G is called GENFLO.

The computational experience comparing MPSX (the IBM proprietary mathematical programming system), NETFLO (Kennington and Helgason [1980]), GENNET (Brown and McBride [1984]), and GENFLO on pure network flow problems may be found in Table 2. The problem number refers to the NETGEN numbers (see Klingman, Napier, and Stutz [1974]). These problems were all run on an IBM 3081-D24. The three specialized codes are written in FORTRAN and used the FORTVS compiler with OPT=2. All times are in cpu seconds and exclude input and output.

NETFLO assumes that every column of G has two nonzero entries which are one and minus one. GENNET assumes that every column has at most two nonzero entries and one must be one. GENFLO assumes that every column has at most two nonzero entries and MPSX makes no assumptions about the entries in G. For these fourteen test problems, NETFLO is sixty times faster than MPSX, GENNET is fifty times faster, and GENFLO is forty times faster.

The computational experience comparing MPSX, GENNET, and GENFLO on generalized network problems may be found in Table 3. The test problems were generated by a

modification of NETGEN called GNETGEN developed by Professor Klingman and colleagues at the University of Texas at Austin and loaned to us by Professor Glover and colleagues at the University of Colorado in Boulder. Both specialized codes are ten times faster than MPSX. The arbitrary second multiplier in GENFLO results in a computational expense of approximately ten percent.

GENFLO has been parallelized using the asychronous parallel simplex method presented in Section 3 and run on the two parallel multiprocessor systems produced by Sequent Computer Corporation. The computational times are presented in Table 4 and the speedups are presented in Table 5. Test problems G1 through G4 are grid problems and R1 through R4 are random problems generated by GNETGEN. Problem C1 was generated with a modification of GNETGEN designed to produce problems with a large number of components at optimality. The change involved modifying the cost range so that the cost for random arcs dominated the cost for the arcs in the skeleton network. Problems generated using this cost structure yielded a large number of components (partitions) in the optimal basis. The column entitled "components at optimality" gives the number of partitions in the optimal basis. All runs began with an all artificial start. Since the asychronous simplex algorithm follows a different path in each parallel run, the times (and number of iterations) for solving a given problem may vary substantially. To account for variability, each problem was solved three times and the average is reported. The speedups varied from approximately two to three using eight processors.

V. SUMMARY AND CONCLUSIONS

The development 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. In this study, an asychronous simplex method for the generalized network problem has been implemented and computationally tested on a Sequent Balance 21000 and a Sequent Symmetry S81. A speedup of approximately three was achieved on a 90x90 grid problem having 8100 nodes and 16,020 arcs using eight cpus. Random problems having 2000 nodes and 50,000 arcs yielded speedups of approximately two using eight cpus.

REFERENCES

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

Ali, I., A. Charnes, and T. Song, [1986], "Design and Implementation of Data Structures for Generalized Networks," Journal of Information and Optimization Sciences, 7, 81–104.

Brown, G.G. and R.D. McBride, [1984], "Solving Generalized Networks," Management Science, 30, 1497–1523.

Chang, M., 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, ND.

Chang, M., M. Enquist, R. Finkel, and R. Meyer, [1987], "A Parallel Algorithm for Generalized Networks," Computer Science Technical Report #642, Department of Computer Science, University of Wisconsin–Madison, Madison, Wisconsin.

Clark, R. and R. Meyer, [1987], "Multiprocessor Algorithms for Generalized Networks," Computer Science Technical Report #739, Department of Computer Science, University of Wisconsin–Madison, Madison, Wisconsin.

Elam, J., F. Glover, and D. Klingman, [1979], "A Strongly Convergent Primal Simplex Algorithm for Generalized Networks," Mathematics of Operations Research, 4, 39–59.

Enquist, M., and M. Chang, [1985], "New Labeling Procedures for the Basis Graph in Generalized Networks," Operations Research Letters, 4, 151–155.

Glover, F., J. Hultz, D. Klingman, and J. Stutz, [1978], "Generalized Networks: A Fundamental Computer Based Planning Tool," Management Science, 24, 1209–1220.

Glover, F., D. Klingman, and J. Stutz, [1973], "Extension of the Augmented Predecessor Index Method to Generalized Network Problems," Transportation Science, 7, 377–384.

Glover, F., D. Klingman, and J. Stutz, [1974], "The Augmented Threaded Index Method for Network Optimization," INFOR, 12, 293–298.

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

Klingman, D., 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.

Langley, W., [1973], "Continuous and Integer Generalized Flow Problems," unpublished dissertation, Department of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

Mulvey, J. and S. Zenios, [1985], "Solving Large Scale Generalized Networks," Journal of Information and Optimization Sciences, 6, 95–112.

Muthukrishnan, R., [1988], "Parallel Algorithms for Generalized Networks," unpublished dissertation, Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, Texas.

Nulty, W. and M. Trick, [1988], "GNO/PC Generalized Network Optimization System," Operations Research Letters, 7, 101–102.

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.

Table 1. 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	Enquist, M. Chang, M.	1985
LPNETG	FORTRAN	Mulvey, J. Zenios, S.	1985
	FORTRAN	Ali, I. Charnes, A. Song, T.	1986
	FORTRAN	Chang, M. Enquist, M. Finkel, R. Meyer, R.	1987
PGRNET	FORTRAN	Clark, R. Meyer, R.	1987
GNO/PC	C	Nulty, W. Trick, M.	1988
GRNET-A	ASSEMBLER	Chang, M. Chen, M. Chen C.	1988
GENFLO	FORTRAN	Muthukrishnan, R.	1988

Table 2. Solution Times for Pure Network Problems (all times are in seconds on an IBM 3081D)

Problem Number	Size		MPSX		NETFLO		GENNET		GENFLO	
	nodes	arcs	pivots	time	pivots	time	pivots	time	pivots	time
15	400	4500	2818	30.60	2073	0.47	1307	1.19	1288	1.41
18	400	1306	2077	12.00	1079	0.24	578	0.39	593	0.49
19	400	24 3	4229	29.40	1305	0.23	765	0.53	688	0.71
22	400	141ɔ	3052	18.00	1284	0.29	504	0.33	613	0.52
23	400	2836	7073	57.60	1156	0.22	604	0.45	492	0.47
26	400	1382	4286	24.60	917	0.14	500	0.27	511	0.42
27	400	2676	11829	95.40	1730	0.28	826	0.46	628	0.55
28	1000	2900	3313	38.40	3524	0.93	1732	1.24	1487	1.39
29	1000	3400	3744	43.80	4570	1.12	1996	1.18	1889	1.59
30	1000	4400	4954	60.00	4346	1.04	1969	1.31	1947	1.87
31	1000	4800	6232	81.00	4798	1.13	2347	1.47	2171	2.13
33	1500	4385	5836	103.20	6113	2.16	2521	2.01	2645	2.83
34	1500	5107	6503	110.40	7640	2.37	2943	2.10	2498	2.50
35	1500	5730	7026	115.80	7384	2.30	3310	2.82	3017	3.35
TOTALS			72972	820.20	47919	12.92	21902	15.75	20467	20.23

Table 3. Solution Times for Generalized Networks (all times are in seconds on an IBM 3081D)

Problem Number	Size		MPSX		GENNET		GENFLO	
	nodes	arcs	pivots	time	pivots	time	pivots	time
1	200	1500	1151	7.80	590	0.62	533	0.95
2	200	4000	550	3.00	443	0.22	358	0.23
3	200	6000	2058	18.60	1448	2.07	954	1.53
4	300	4000	4112	47.40	2703	3.50	2106	4.23
5	400	5000	1870	26.20	1229	2.06	897	2.23
6	400	7000	1408	16.80	1591	1.59	1171	1.68
7	1000	6000	2811	40.20	3160	3.30	2352	3.60
TOTAL			13960	160.00	11164	13.36	8371	14.45

Table 4. Solution Times* for the Parallel Simplex Solver (times are in seconds)

Problems	nodes	arcs	components at optimality	Sequential code	Processors 1	2	3	4	5	6	8
G1	2500	4900	35	70.01	76.71	54.10	45.60	41.60	37.30	36.01	31.95
G2	4900	9660	43	167.01	183.48	117.50	90.50	78.00	68.51	64.50	59.90
G3	6400	12640	68	208.70	225.96	161.10	130.80	101.92	97.73	87.70	79.80
G4	8100	16020	89	268.33	296.75	194.90	140.90	117.60	110.60	95.50	90.30
R1	2000	25000	3	100.50	100.43	82.34	70.78	62.55	58.88	60.05	51.75
R2	2000	50000	2	220.81	219.52	169.27	150.50	135.66	125.11	118.45	110.62
R3	2000	50000	1	186.53	188.06	128.05	114.09	101.37	109.66	101.13	92.34
R4	6000	39000	14	157.06	163.58	127.12	115.03	102.75	100.07	95.29	85.27
C1	5000	100000	240	194.39	186.87	94.14	80.62	74.03	67.20	64.53	75.44

*Problems G1–G4 were run on a Sequent Balance 21000, all other problems were run on a Sequent Symmetry S81.

Table 5. Speedups for the Parallel Simplex Solver

Problems	nodes	arcs	components at optimality	Sequential code	Processors 1	2	3	4	5	6	8
G1	2500	4900	35	1.00	0.91	1.29	1.54	1.68	1.88	1.94	2.19
G2	4900	9660	43	1.00	0.91	1.42	1.85	2.14	2.44	2.59	2.79
G3	6400	12640	68	1.00	0.92	1.30	1.60	2.05	2.14	2.38	2.62
G4	8100	16020	89	1.00	0.90	1.38	1.91	2.28	2.42	2.81	2.97
R1	2000	25000	3	1.00	1.00	1.22	1.42	1.61	1.71	1.67	1.94
R2	2000	50000	2	1.00	1.01	1.30	1.47	1.63	1.76	1.86	2.00
R3	2000	50000	1	1.00	1.46	1.64	1.84	1.70	1.84	1.82	2.02
R4	6000	39000	14	1.00	0.96	1.24	1.37	1.53	1.57	1.65	1.84
C1	5000	100000	240	1.00	1.04	2.06	2.41	2.63	2.89	3.01	2.58