Comput. & Ops Res., Vol. 2, pp. 71–81. Pergamon Press, 1975. Printed in Great Britain

PAST, PRESENT AND FUTURE OF LARGE SCALE TRANSSHIPMENT COMPUTER CODES AND APPLICATIONS

A. CHARNES*, DAVID KARNEY†, D. KLINGMAN‡ and JOEL STUTZ§
University of Texas at Austin

and

FRED GLOVER¶
University of Colorado

Scope and purpose—The transportation and transshipment problem was among the earliest to be attacked by the methods of operations research. The methods and algorithms used for optimizing the allocation of transportation capability are also used for many other seemingly unrelated problems of allocation and allotment. Because of this wide range of application, a very great amount of effort has been devoted to the development of algorithms for this problem. This article is primarily concerned with the past, present and future capability of such algorithms. It deals with the dependence of computer running time on problem complexity, and how that has changed as new computers and new algorithms were developed.

Abstract—Three generations of computers have elapsed since the first satisfactory method for solving transportation and transshipment problems was devised. During this time many computational advances have taken place in developing computer codes to solve these problems. For example, recent breakthroughs in the solution and human engineering aspects of transshipment problems have made it possible to solve problems in only a few minutes that require many hours of computing time with commercial LP packages. Additionally the computer memory requirements of new methods have enabled the solution of vastly larger problems than previously imagined possible (50,000 equations and 62 million variables). Enhancing the significance of these developments, new ways have been discovered for modelling broad classes of real world problems as transshipment or transshipment-related problems. The primary purpose of this paper is to summarize these events and to do some crystal ball gazing to provide what we believe to be "best estimates" of future trends.

PAST

Approximately 200 years have elapsed since the French Academy of Sciences posed the civil engineering problem of "cutting and filling." Their formulation of the problem was not the same as the transportation problem as we know it today, but was the equivalent of a transportation problem in continuous form. The current formulation of a transportation problem was due to Kantorovich [31], Hitchcock [29], and Koopmans [38].

Kantorovich showed in 1939 that a class of problems closely related to the transportation problem has a remarkable variety of applications. These were concerned typically with the allotment of tasks to machines whose costs and rates of production vary by task and machine type. Kantorovich gave a useful but incomplete algorithm for solving such problems. Later, in 1942, he studied both discrete and continuous versions of this problem and in 1948, along with Gavurin wrote an applicational study on the capacitated transportation problem.

*Dr. Abraham Charnes is a professor at the University of Texas. He is a Fellow of ORSA, AAAS, and the Econometric Society; Founding Corresponding Member of the National Academy of Engineering in Mexico; Founding Member and Past President of TIMS.

†David Karney is Executive Vice President of Analysis, Research, and Computation Inc. He holds a BA in Mathematics and an MA in Computer Sciences from the University of Texas at Austin. Mr. Karney is the co-author of several published papers and research notes on network analysis and co-author of a book on Testing Computational Algorithms.

‡Darwin Klingman is a professor in the departments of Mathematics, Operations Research, and Computer Sciences at the University of Texas at Austin, and is director of Computer Sciences Research for the Center for Cybernetic Studies. He is the author of more than twenty-five articles in such journals as *Management Science, Operations Research, and Transportation Science.* He is also co-author of books on linear programming and data processing.

§Joel Strutz is an Assistant Professor in Operations Research and Computer Sciences at the University of Texas. His current research is on generalized network problems and interfacing the human engineering aspects of data management information system with optimization programs.

¶Fred Glover is Professor of Management Science at the School of Business, University of Colorado, Boulder. His past fields of research have included integer programming and combinatorial or graph theory problems (such as scheduling, matching, and routing) relating to it.

Hitchcock developed an incomplete algorithm in 1941, which exploited special properties of the transportation problem to find starting solutions. Koopmans[38] independently arrived at the same problem in connection with his work as a member of the Combines Shipping Adjustment Board. He and Reiter discussed the problem from an economic efficiency analysis viewpoint and pointed out the analogy between it and the classical Maxwell–Kirchhoff electrical network problem. Because of their work, the problem is often referred to as the Hitchcock–Koopmans Transportation problems.

Early solution methods

The first generally satisfactory method for solving the general class of transportation and transshipment models was due to G. B. Dantzig in 1949. The method specializes the primal simplex method to exploit the network structure. Applied to transportation problems, this method is sometimes called the Row-Column Sum Method[9] or the MODI method[13]. Charnes and Cooper[8] later wrote an explanation (dubbed the Stepping-Stone Method) of the simplex steps involved in the Row-Column Sum Method. The Charnes–Cooper paper has become a standard reference in the field.

With the advent of a method for solving the transportation problem came numerous methods for securing starting bases. Two of the methods commonly referenced are the Northwest-Corner Rule[13] and the Vogel Approximation Method[49] (often referred to as VAM). Of all the start methods developed, VAM became the one most used for hand calculations due to the excellent start it provides. Thus in the folklore VAM is considered the best procedure for both computer and hand calculations.

After the development of the Row-Column Sum Method, the transportation model, with integer parameters, rapidly became the chief "solvable" integer linear programming problem due to the integer extreme point property. Also, a survey by L. W. Smith, Jr. in 1956 indicated that at least half of the linear programming applications used this model.

Some of the early reasons for the large concentration on problems of this kind, particularly in applications, were:

(1) Business executives can understand the transportation model, leading to increased demand for its applications in practical settings.

(2) It is possible to approximate many linear programs by transportation problems.

(3) A number of seemingly unrelated linear programs have been found to be equivalent to transportation problems.

(4) Answers to "large" problems can be easily computed by hand, which is an impossible task for general linear programming problems of similar dimensions. Also, integer solutions were immediately attainable.

(5) Computer codes were developed as early as 1952 for solving transportation problems.

Investigations by the authors in the late 1960's strongly confirm these views. These investigations indicate that a very substantial proportion, perhaps as great as 70%, of the real world mathematical programming problems consist of—or can be transformed into—network and "network-related" problems. Specifically, the predominant number of practical mathematical programming applications appear to involve problems of the following types: assignment problems, transportation problems, transshipment problems, generalized transshipment problems, transshipment problems with extra linear constraints, integer problems whose relaxed problem is one of these, or a problem which is equivalent to one of these by a simple linear transformation.

Subsequent to the development of the simplex based Row-Column Sum Method, Ford and Fulkerson[17], developed a primal-dual method for solving transportation problems and Fulkerson[18] developed the out-of-kilter method, which is an extension of the primal-dual method, for solving transshipment problems. Somewhat earlier Munkres[44] and Gleyzal[28] also developed methods similar to the primal-dual method.

It is interesting to note that Dantzig, Ford and Fulkerson concluded on the basis of hand calculations that the primal-dual method was superior in efficiency to the Row-Column Sum Method. This conclusion was also supported by Flood[16] on computer codes. Consequently, this conclusion became part of the folklore. (However, the computer codes were tested on different problems and different computers.)

Another major early solution method was developed by Busacher and Gowen[7]. Their procedure successively saturates shortest paths in the network. An alternative method for solving transportation problems was developed by Balas and Hammer[2,3]. Later Charnes and Kirby[10] showed that this method may be viewed as a specialization of the dual simplex method.

Early computer codes

The first computer code for solving transportation problems was based on the Row-Column Sum Method and in terms of current jargon is called a primal simplex transportation code. In fact, the code was also an in-core out-of-core code, utilizing magnetic tape for peripheral storage. The code was developed in 1952 by the George Washington University Logistics Research Project in conjunction with the Computation Laboratory of the National Bureau of Standards[50]. Designed for use on the Bureau of Standards Eastern Automatic Computer, the code was further improved by the NBS Computation Laboratory. The improved version was capable of solving problems with at most 600 nodes and required more than 3 min per pivot. Current pivot times for problems of this size are 3–5 ms on the CDC 6600, UNIVAC 1108, and IBM 360/65 computers.

The in-core out-of-core primal simplex transportation code by Dennis[14] is one of the first codes to be described in detail in the literature. Dennis' paper is also one of the first to study different criteria for selecting pivot elements. Unfortunately, his study principally involved only one problem of size 30 origins by 260 destinations. The best solution time was 9·6 min on the Whirlwind computer. Current solution time for this size problem on a CDC 6600 is 1 s, or roughly 600 times faster on in-core codes.

Another in-core out-of-core primal simplex transportation code was developed in the late fifties by Glicksman *et al.*[20]. This code was developed for the UNIVAC I for solving "thin rectangular" problems. The code solved a 15 origin by 488 distination problem in 24 min. This is approximately 900 times slower than current in-core codes.

Also, during the late fifties in-core transportation codes were developed using the primal-dual method, implemented primarily on IBM computers. These codes include the one due to Flood[16] (using his proof of the König–Egervary Theorem) and the IB-TFL code (1958) developed by Rand Corporation. The code of Flood and the IB-TFL code were compared on a problem with 29 origins and 116 destinations on an IBM 704. Their times were 193 and 197 seconds, respectively. Current solution time would probably be 1 s.

Based on this testing Flood and others reinforced the earlier conclusion that the primal-dual method was computationally superior to the primal simplex method. Note that this conclusion was not well founded. In particular, it was based on solving *different* problems of different sizes on *different* computers. Additionally the primal simplex codes were in-core out-of-core codes using slow magnetic tape for peripheral storage, while the primal-dual codes were strictly in-core codes using only fast-access (central) memory for storage.

As far as we have been able to determine no computer codes based on the dual simplex method or the Busacher and Gowen method were developed prior to 1968. Additionally, no testing was conducted to determine best start procedures and pivot criteria to use with the primal simplex method and no primal simplex transshipment codes were developed.

Following these developments, there was a hiatus of half a dozen years during which little was visibly accomplished in the development of improved solution methods or computer implementations. From an algorithmic standpoint, it was widely believed that no significant refinements remained to be discovered. In retrospect, this attitude seems surprising, particularly in view of the paucity of experimentation to determine the computational strengths and weaknesses of alternative approaches. Then, in the later sixties and more particularly in the early seventies, a new surge of interest in network methods and applications came about, leading to a number of surprises for those steeped in the notions of a decade earlier.

It is to these more recent developments that we now turn.

PRESENT

Computational highlights to mid-1973

As already intimated, the early special purpose primal and primal-dual codes were capable of solving only small problems, were quite slow, and were not extensively tested. Beginning with the latter half of the sixties, several codes have been jointly developed by mathematical

programmers and systems analysts who have performed extensive experimentation on various algorithmic rules. The major code developments completed by mid-1973 are indicated in Table 1. These codes represent several "firsts" in computational and algorithmic development.

(1) The first implementations of dual simplex transportation and transshipment codes [21,24].

(2) The first implementation of a primal simplex transshipment code [21].

(3) The development of the negative cycle solution method for assignment problems by Klein [35], and its extension and implementation for transshipment problems by Bennington [6].

(4) The first primal simplex transportation code capable of solving capacitated problems and the first code to store only the existing costs rather than a full cost matrix [23].

Table 1 also shows that the computer memory requirements of non-simplex codes are substantially larger than those of simplex codes. The codes that make the most efficient use of computer memory are the primal simplex codes by Glover *et al.* [21,23] and Karney and Klingman [33]. It should be noted that all of the codes in Table 1 are coded in FORTRAN, and all except I/0 PNET are in-core codes. The I/0 PNET code, by Karney and Klingman, is an in-core out-of-core code designed for large-scale problems. Before this code was fully streamlined, it solved a problem for the Naval Personnel Laboratory in San Diego with 2400 nodes and 450,000 arcs on an IBM 360/65 and CDC 6600 in 26 and 23 min of central processing time, respectively.

All of the primal and dual simplex codes in Table 1 (except that of Graves and McBride) use the augmented predecessor list structure [22], which elaborates on Johnson's "triple-label

Table 1. Code development by mid-1973

	Developers	Name	Methodology	Core Memory Requirements	Date
1.	Barr, Glover, Klingman	SUPERK	Out-of-kilter	4N + 9A	1972
2.	Bennington	BENN	Negative Cycle Method	6N + 11A	1971–73
3.	Witzgall	Boeing	Out-of-kilter	6N + 8A	1966
4.	Clasen	SHARE	Out-of-kilter	6N + 7A	1966
5.	Control Data Corporation	NETFLOW	Out-of-kilter	6N + 7A	1970
6.	Gavish and Schweitzer		Primal Simplex Transportation	8N + 3A	1972
7.	General Motors	GM	Primal-dual Transportation	7N + 5A	1970–72
8.	Glover, Karney, Klingman	PTRANS	Primal Simplex Transportation	5N + 2A	1970–71
9.	Glover, Karney, Klingman	DTRANS	Dual Simplex Transportation	8N + 2A	1970–72
10.	Glover, Karney, Klingman	PNET	Primal Simplex Transshipment	6N + 2A	1972–73
11.	Glover, Karney, Klingman	DNET	Dual Simplex Transshipment	9N + 2A	1972–73
12.	Graves and McBride		Primal Simplex Transshipment	14N + 2A	1972–73
13.	Grigoriadis, et. al.		Rosen's Dual Method		1966–68
14.	Kennington and Langley		Primal Simplex Transportation	8N + cost matrix	1972–73
15.	Karney and Klingman	I/O PNET	Primal Simplex Transshipment	7N + Buffer	1973
16.	Srinivasan and Thompson	S-T	Primal Simplex Transportation	10N + cost matrix	1970–72
17.	Texas Water Development Board	TWB	Out-of-kilter	$4N + N^2 + 7A$	1968
18.	UNIVAC	UKILT	Out-of-kilter	4N + 9A 3N + 5A	1973

N – number of nodes
A – number of arcs

Note: All of these are in-core codes coded in FORTRAN.

method"[30] by providing an efficient method for characterizing successive basis trees with minimal relabeling. The augmented predecessor list structure has been a major contributor to the improvements in the computational efficiency of solution algorithms. With its use, the primal transportation code by Glover *et al.*[23] executes a pivot on a 600 node problem in 6 ms compared with the early breakthrough (1952) of reducing the time per pivot to 3 min. While this reduction is largely due to improvements in computers and the in-core nature of these codes, this is not the whole reason. For instance, the first accelerated primal transportation code developed by Srinivasan and Thompson[51] employed a list structure for proceeding in a forward direction through a spanning tree similar to Dennis' procedure. Upon comparing solution times of the Srinivasan and Thompson code with the Glover, Karney, and Klingman code on the same problems and machine, the efficiencies of the augmented index structure became apparent. Srinivasan and Thompson recoded using the augmented list structure and cut their solution times by more than half. Similarly, Gavish and Schweitzer[19] improved their solution times by a factor of 3 after adopting the predecessor list structure.

The code development and comparison of Srinivasan and Thompson[51] provides an important computational analysis of several primal start procedures and pivot criteria. The purpose of this study was to determine a design for an in-core uncapacitated primal transportation and assignment code which optimally combines start procedures and pivot criteria for maximum solution efficiency. The study disclosed that the best start method is the "modified row minimum start" procedure and the best pivot selection criterion is the "row most negative rule." This pivot rule was also found to be best by Dennis[14]. Maximum problem size solved was 350 nodes (origins plus destinations). This node limitation is due to the fact that it is an in-core code which stores a complete cost matrix. The average solution time on 175 origin by 175 destination transportation and assignment problems was 7·8 s.

The code development and comparison by Glover, Karney, Klingman, and Napier (1970–72) performed similar analyses on a broad profile of dense and nondense problems. The underlying code PTRANS was specially designed for solving both capacitated and uncapacitated problems with nondense cost matrices (i.e., transportation problems where some cells may not be allowable). This study also found the modified row minimum start and row most negative pivot rule to be best, thus casting doubt on the folklore of the superiority of VAM starts. In addition, using 100 problems, the study compared PTRANS to several other codes including Clasen's SHARE code (1966), the Glover, Karney, and Klingman dual code DTRANS (1970), and the state-of-the-art linear programming code OPHELIE/LP. As indicated in Table 2 this comparison revealed that the PTRANS Code was at least eight times faster than the SHARE and DTRANS codes, and 150 times faster than OPHELIE/LP. Thus the old folklore about the superiority of out-of-kilter methods, and a new folklore among computer service divisions about equivalence of general purpose and special purpose solution codes for transportation and transshipment problems were upended. (The times indicated in Table 2 for PTRANS in 1973 have been made three times faster in 1974. Thus the superiority of primal simplex codes appear even more pronounced than suspected.)

The largest problems solved in the study[23] were 1000 origin by 1000 destination problems with an average solution time of 17 s. This study also tested the primal code on four computers, IBM 360/65, UNIVAC 1108, CDC 6400, and CDC 6600 in order to provide insights into conclusions based on comparing times on different machines and compilers. It was discovered that standard guidelines concerning the relative efficiencies of different computers were completely misleading, since the primal code ran only 10–12% faster on the CDC 6600 than on the UNIVAC 1108 and the IBM 360/65 differing substantially from the estimates one would obtain by comparing instruction execution times of the machines.

Motivated by the fact that out-of-kilter codes were found to be substantially slower than the special primal code, Barr *et al.*[5] developed an improved version of the out-of-kilter method which was subsequently coded. This code was found to be only 40% slower (on the same problems and machine) than the primal transportation code of Glover, Karney, and Klingman on transportation problems. This code was also compared against Clasen's SHARE Code, Boeing's code, and the Texas Water Development Board code and found to be at least six times faster than the best of these (which differed from problem to problem). The study also examined a total of 215 capacitated and uncapacitated transhipment problems demonstrating the superiority of the improved version of the out-of-kilter code over the other out-of-kilter codes in all cases. The

Table 2. Solution times (sec) for Out-of-Kilter, OPHELIE/LP, primal and dual algorithms[2]

Problem Size	Mean Density	PTRANS Primal Solution Time[1]	OPHELIE/LP Solution Time	SHARE Out-of-Kilter Solution Time	DTRANS Solution Time
10 X 10	0.35	0.016	0.755	0.10	0.025
20 X 20	0.65	0.104	4.012	0.68	0.226
30 X 30	0.60	0.242		2.04	0.689
40 X 40	0.36	0.282	39.375	2.42	0.903
50 X 50	0.54	0.611		5.70	3.053
60 X 60	0.20	0.692		5.28	3.026
70 X 70	0.28	0.925		9.46	7.022
80 X 80	0.31	1.467		22.10	9.829
90 X 90	0.28	1.917		26.35	15.180
100 X 100	0.20	1.907	276.90	21.17	14.622
500 X 500	0.011	5.983			
1000 X 1000	0.005	17.081			

[1]These times have been improved by a factor of three in 1974.

[2]All times are median times with five problems per group.

largest problems solved were 1500 node transshipment problems. The mean solution time was 34 s.

Still more recently, Glover *et al.*[21] have developed a general primal transshipment code. Computational comparison of this code with the out-of-kilter code by Barr *et al.* reveals that the primal code is 30% faster on transshipment problems. This is rather startling since the Barr *et al.* code is probably the fastest out-of-kilter code in the world and conventional wisdom has it that labeling techniques are inherently more efficient than simplex techniques. The primal transshipment code was also tested against the negative cycle code by Bennington[6] and found to be ten times faster. This computational study also showed the superiority of the new primal transshipment code in terms of central memory requirements for storing network data. Specifically, the out-of-kilter codes discussed earlier require 6–10 arc-length arrays and 4–7 node-length arrays as compared to 3 arc-length arrays and 8 node-length arrays for the primal code. The substantially increased problem size that can be accomodated by the new primal code is illustrated in the study by the solution of an 8000 node problem.

In addition to these code development efforts and computational comparisons, an extensive study of the effects of parameter values was conducted for transportation problems by Klingman *et al.*[36]. This study performed a detailed examination of the effects of problem dimensionality on solution times. The study included over 1000 randomly generated problems with 185 different combinations of number of origins, number of destinations and number of variables (not all cells being considered admissible). Every problem was solved using three starting procedures. Over 10,000 pieces of data were analyzed, providing numerous insights into the computational effects of the number of constraints, the degree of "rectangularity", and the number of variables.

Some of the conclusions drawn from this study are:

(1) For problems with a constant number of variables and a constant number of constraints, total solution time decreases as the problems become more rectangular. Additionally, the basis equivalent paths in a rectangular problem are shorter.

(2) For a constant density and a constant number of variables the total solution time increases as rectangularity increases.

(3) Total solution time increases slightly greater than in proportion to the change in the square of the number of constraints for problems with a fixed density and a fixed ratio of m/n.

(4) For a fixed number of origins and a fixed density, the total solution time increases in proportion to an increase in the number of destinations. (The paper contains least square solution time estimators.)

(5) For a fixed degree of rectangularity, increasing the number of constraints for a fixed number of variables has a greater effect on total solution time than increasing the number of variables for a fixed number of constraints.

(6) As variance increases in the cost distribution, solution time increases. Thus problems whose costs are randomly generated using a uniform probability distribution are hardest to solve for a fixed cost range.

These studies on the effect of parameter values and code comparisons repeatedly reinforced the conclusion that in order for researchers to compare their codes in a meaningful way it is necessary that they use exactly the same problems. This is due to the fact that, even if two randomly generated problems have the same parameter values, a generator inherently builds structure into the problems, particularly transshipment and transportation problems in which some arcs do not exist. This point is underscored in the computational study [5] and has been further demonstrated by the comparison of the codes due to Bennington; Glover, Klingman, and Karney; and Srinivasan and Thompson which yielded unexpected results when tested on the same problems and the same computer.

To enable researchers to meaningfully compare their solution codes, Klingman *et al.* [37] developed a code which generates assignment problems, and capacitated and uncapacitated transportation and transshipment problems. In addition to producing structurally different classes of transshipment problems, the code permits the user to vary structural characteristics within a class. By means of this code, researchers can generate identical transshipment problems (independent of the computer). Advantages of the transshipment generator, which is available with documentation to researchers for a nominal handling charge, are its ease of use (requiring only two data cards per transshipment problem) and the standardization of its output (generating problems for use by other codes in SHARE input format). In addition, the latter part of the documentation provides the user with the data on 40 assignment, transportation, and transshipment problems varying in size from 200 nodes to 8000 nodes and from 1300 arcs to 35,000 arcs. The objective function value and solution time for these problems are also provided for the SHARE, Boeing, TWB, GM, SUPERK, PNET, and BENN code.

Conclusions and limitations of mid-1973 testing

The code development and testing conducted between 1970 and mid-1973 produced implementations of all widely known transportation and transshipment solution methods except for the Busacker and Gowen method. Extensive computational comparisons of these methods have been made using the same problems, computer, and compiler [21, 23, 33, 37]. The results of this testing showed that the primal simplex transportation method embodied in PTRANS [23] was the most efficient transportation code in terms of both computer memory space and solution time. Similarly, the primal simplex transshipment code PNET [21] was the most efficient code for transshipment problems. PNET runs only 10% slower than PTRANS on transportation problems, for which the latter was especially designed. This raises the question "Is it worth developing both transportation and transshipment codes?"

Testing and code development after mid-1973

During the summer of 1973, Glover, Klingman, and Stutz developed a new list structure for storing and updating spanning trees called the Augmented Threaded Index Method (ATI) [26]. The ATI is the only list structure that uses only two pointers per node (in a non-binary tree) while providing the ability to traverse the tree both upward and downward efficiently. The ATI [26] is thus more efficient than the API [22] in terms of both computer memory requirements and solution speed. Following this development a number of new codes were developed (see Table 3). All of the codes in Table 3 use the ATI method except Edmonds and Karp, Graves, and Rao and Fong. Computational testing has shown all of these codes to be inferior to the ATI codes. (An interesting point which Jack Edmonds mentioned at the Combinatorial Conference at Versaille in September 1974 was that some computational testing on the streamlined version [15] of the Busacker and Gowen algorithm had been done and the results indicated that the algorithm in [15]

Table 3. Code development completed after mid-1973

	Developer	Name	Type	Array Size	Year
1.	Analysis, Research, and Computation	ARC PPN	In-Core Primal Simplex Transshipment	6N + 2A	1974
2.	Barr	SUPERT	In-Core Primal Simplex Transportation	4(m + n) + 2d(mn)	1973
3.	Glover, Karney, Klingman	PNET-I	In-Core Primal Simplex Transshipment	5N + 2A	1973
4.	Graves		In-Core Primal Simplex Transshipment	7N + 2A	1974
5.	Karney and Klingman	I/O PNET-I	In-Core Out-of-Core Primal Simplex Transshipment	5N + Buffer	1974
6.	Mulvey	LP-NET	In-Core Primal Simplex Transshipment	6N + 2A	1974
7.	Rue and Fong		Primal-Dual In-Core Transportation	6(m + n) + 3mn	1973

For transshipment problems N is the number of nodes and A is the number of arcs.
For transportation problems m is the number of origins, n is the number of destinations and d is the density of the admissible cells.

is inferior to SUPERK[5].) Thus, after testing all of the basic network solution algorithms, it has been established that primal simplex based network codes are the fastest. Further, computational testing indicates that the fastest primal simplex transshipment code is the one by Analysis, Research, and Computation, Inc.[1] and the fastest transportation code is the one by Barr[4]. Both codes use the ATI method[26] augmented by a depth factor degeneracy check[42, 51] and a candidate list pivot selection procedure[42].

A significant recent finding by Mulvey[42] is that the best pivoting procedure determined in [14, 21, 23, 51] for small and medium size problems is not the most efficient for large problems. Mulvey's finding is that by using an appropriate candidate list pivot selection procedure (i.e., a multi-pricing procedure) when solving large problems, solution time can be cut in half over the pivot criteria used in [14, 21, 23, 51]. Another recent computational conclusion [19, 42] is that avoiding and/or exploiting degenerate privots can significantly enhance simplex based codes since degeneracy is as high as 99% in large problems.

Due to these important computational aspects of large problems, Glover and Klingman[27] very recently developed another list structure which extends the ATI method to directly incorporate information needed to exploit both degeneracy and depth. This new list structure has the disadvantage of requiring one more computer memory array than the ATI method augmented by the depth factor but offers substantial computational advantage. In particular, the new list structure requires significantly fewer computer operations (e.g., array references and arithmetic operations) to update the basis tree than the latter. In addition, the new list structure minimizes the efforts required to update dual information. Thus, it is quite likely that this list structure will produce another breed of primal simplex codes similar to the developments following the API and ATI developments [22, 26].

FUTURE

In spite of the major recent gains in the development and testing of network codes, significant avenues remain to be explored. In particular, most of the codes currently in vogue are in-core codes, all are coded in FORTRAN, most have not fully exploited problem size capability of third generation computers. Thus, we shall probably see codes developed in other languages (e.g., ALGOL and APL) in order to rigorously determine which language is best in network applications. Assembly language codes are also quite conceivably in the offing, but of course will be machine specific and sacrifice portability.

In the near future, we will undoubtedly see super large scale in-core out-of-core network codes developed which will be capable of solving network problems of almost unlimited size. For example, the Analysis, Research and Computation code, ARC-PPN,[1] is being extended and highly specialized to solve a 50,000 node, 62 million arc transportation problem on a UNIVAC 1108 for the U.S. Treasury Department. This Extended Transportation System (which is to include such special features as primal generated percent optimality bounds and dynamic candidate list selection pivot procedures) will probably determine future algorithm needs and provide important insights on current solution bounds.

While the transshipment generator[37] is a start towards helping benchmark these codes, we believe that a bureau needs to be established to enable standardized comparisons. We have been informed by Richard Jackson (at the National Bureau of Standards) that the Mathematical Programming Society in conjunction with various researchers, is considering the feasibility of establishing a service facility to accomplish this. The benefit of establishing such a service is apparent; however, numerous problems must be overcome. For example, Input–Output formats of codes and methods for timing codes must be standardized. While such matters may appear to be quite simple, they are not. To illustrate, every code benchmarked by the authors, (e.g., those due to Clasen, Boeing, the Texas Water Development Board, General Motors, Bennington, and Srinivasan and Thompson), used a different input format, and considerable effort was required to accommodate these differences. Also, while a valid criterion for the timing of in-core codes is quite easy to define (namely central processing time exclusive of data input and output), establishing an acceptable measure for the timing of in-core out-of-core codes in a multi-processing environment is far more complex. In any case, we believe some criteria need to be developed.

Other short range future developments which we foresee include:

(a) development of network computer systems similar to general linear programming systems. These systems will include such things as a command language (which allows the user to add, delete, and modify arcs), matrix generators, report generators, user subroutine control of individual components of the system, and interactive coupling with data base management information systems. The Control Data Corporation NETFLOW[45] system and the UNIVAC UKILT-1100[52] system are forerunners of such systems.

(b) establishment of numerous special purpose integer programming codes using efficient network codes as the main computational vehicle, e.g. plant location codes, fixed charge network codes, integer generalized network codes, constrained network codes, multi-commodity (integer) network codes, constrained generalized network codes, and multi-commodity (integer) generalized network codes.

This integer programming development will be (and must be) integrated with the following analysis:

(1) find efficient ways to match data organization and manipulation schemes of network-related problems with integer programming information requirements.

(2) just as researchers have found in the past that there are different integer programming formulations for the same problem, researchers [25, 43] now are discovering that there are different types of network relaxations *within* formulations. Thus researchers need to test which of these relaxations are best along several dimensions. For example, studies must be conducted to determine the trades-offs between the strength of relaxation, solution time, and usability of special penalty calculations.

Looking farther into the future, we anticipate the following as possible developments:

(a) an efficient graph computer language which allows a user to write special purpose network codes in half a day. A forerunner is the GROPE language at the University of Texas.

(b) network and related optimization codes which modify themselves; for example, computer network codes which "learn" how to efficiently solve particular types of problems through experience in solving them. (Preliminary investigations of this type have now been going on for more than a decade.)

(c) multi-page linear programming codes which use special purpose codes (e.g., network codes) to solve pages (components) which have a special structure.

REFERENCES

1. Analysis, Research and Computation, Inc., Development and computational testing on a capacitated primal simplex transshipment code. ARC Technical Research Rept., P. O. Box 4067, Austin, Tex. 78765.
2. E. Balas and P. L. Hammer, On the transportation problem—part I, *Cabiers du Centre d'Etudes de Recherche Operationelle*, 4 (2), 98–116 (1962).
3. E. Balas and P. L. Hammer, On the transportation problem—part II, *Cabiers du Centre d'Etudes de Recherche Opertionelle*, 4 (3), 131–160 (1962).
4. R. S. Barr, Streamlining primal simplex transportation codes, Research Rept. to appear in Center for Cybernetic Studies, University of Texas, Austin.
5. R. S. Barr, F. Glover and D. Klingman, An improved version of the out-of-kilter method and a comparative study of computer codes, *Mathl Programming* 7 (1), 60–87 (1974).
6. G. E. Bennington, An efficient minimal cost flow algorithm, *Mgmt Sci*, 19 (9), 1021–1051 (1973).
7. R. G. Busacker and P. J. Gowen, A procedure for determining a family of minimal-cost network flow patterns, ORO Technical Rept. 15, Operational Research Office, Johns Hopkins University (1961).
8. A. Charnes and W. W. Cooper, The stepping stone method of explaining linear programming calculations in transportation problems, *Mgt Sci.* 7, 49–69 (1954).
9. A. Charnes and W. W. Cooper, *Management Models and Industrial Applications of Linear Programming*, 2 Vols. John Wiley (1961).
10. A. Charnes and M. Kirby, The dual method and the method of Balas and Ivanescu (Hammer) for the transportation model, *Cabiers du Centre d'Etudes de Recherche Operationelle*, 6 (1), 5–18 (1964).
11. R. J. Clasen, The numerical solution of network problems using the out-of-kilter algorithm, RAND Corporation Memorandum RM-5456-PR, Santa Monica, California, (1968).
12. G. Dantzig, *Activity Analysis of Production and Allocation*, Ch. XXIII. (Edited by T. Co. Koopmans), John Wiley (1951).
13. G. Dantzig, *Linear Programming and Extensions*. Princeton University Press (1963).
14. J. B. Dennis, A high-speed computer technique for the transportation problem, *J. Assoc. comput. Mach.* 8, 132–153 (1958).
15. J. Edmonds and R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, *J. Ass Comput. Mach.* 19, 248–264 (1972).
16. M. M. Flood, A transportation algorithm and code, *Nav. Res. Logist. Q.* 8, 257–276 (1961).
17. L. R. Ford and D. Fulkerson, A primal-dual algorithm for the capacitated Hitchcock problem, *Nav. Res. Logist. Q.* 4, 47–54 (1957).
18. D. R. Fulkerson, An out-of-kilter method for solving minimal cost flow problems, *J. Soc. Indust. appl. Math.* 9, 18–27 (1961).
19. B. Govish and P. Schweitzer, An algorithm for combining truck trips, *Transportation Sci.* 8, 13–24 (1974).
20. S. Glickman, L. Johnson and L. Eselson, Coding and Transportation problem, *Nav. Res. Logist. Q.* 7, 169–183 (1960).
21. F. Glover, D. Karney and D. Klingman, Implementation and computational study on start procedures and basis change criteria for a primal network code, *Networks* 4 191–212 (1974).
22. F. Glover, D. Karney and D. Klingman, The augmented predecessor index method for locating stepping stone paths and assigning dual prices in distribution problems, *Transportation Sci.* 6, 171–180 (1972).
23. F. Glover, D. Karney, D. Klingman and A. Napier, A computational study on start procedures, basis change criteria, and solution algorithms for transportation problems, *Mgmt Sci.* 20, 793–819 (1974).
24. F. Glover and D. Klingman, Double-pricing dual and feasible start algorithms for the capacitated transportation (distribution) problem, University of Texas at Austin (1970).
25. F. Glover and D. Klingman, Equivalence of mixed integer programs and mixed integer generalized networks, Research Rept. to appear Center For Cybernetics Studies, University of Texas, BEB-613, Austin 1974).
26. F. Glover, D. Klingman and J. Stutz, Augmented threaded index method for network optimization, to appear in *INFOR* October, (1974).
27. F. Glover and D. Klingman, Extensions of the augmented threaded index method. Research Rept. C. S. 190, Center For Cybernetic Studies, University of Texas, BEB-613, Austin (1974).
28. A. N. Gleyzal, An algorithm for solving the transportation problem, *J. Res. Nat. Bur. Stand.* 54, 123–216 (1955).
29. F. L. Hitchcock, The distribution of a product from several sources to numerous localities, *J. Math. Phys.* 20, 224–230 (1941).
30. E. Johnson, Networks and basic solutions, *Ops. Res.* 14, 619–623 (1966).
31. L. V. Kantorovich, On the translocation of masses, *Compt. rendu. Acad. Sci., U.S.S.R.* 37, 199–201 (1942).
32. L. V. Kantorovich and M. K. Gavurin, The application of mathematical methods to problems of freight flow analysis, *Akademia Nauk SSSR* (1949).
33. D. Karney and D. Klingman, Implementation and computational study on in-core out-of-core primal network code, to appear in *Ops. Res.*
34. J. L. Kennington, R. W. Langley and C. M. Shetty, Efficient computational devices for the capacitated transportation problem, to appear in *Now. Res. Logist. Q.* (1975).
35. M. Klein, A primal method for minimal cost flows, *Mgmt Sci.* 14, 205–220 (1967).
36. D. Klingman, A. Napier and G. Ross, A computational study on the effects of problem dimensions on solution time for transportation problems, to appear in *JACM.*
37. D. Klingman, A. Napier and J. Stutz, NETGEN—a program for generating large scale (un) capacitated assignment, transportation, and minimum cost flow network problems, *Mgmt Sci.* 20, 814–822 (1974).
38. T. C. Koopmans, *Activity Analysis of Production and Allocation*, Cowles Commission Monograph N. 13, John Wiley (1951).
39. T. C. Koopmans and S. Reiter, A model of transportation, *Activity analysis of Production and Allocation*, Cowles Commission Monograph 13, 222–259. Wiley. (1951).
40. H. W. Kuhn, The Hungarian method for the assignment problem, *Nav. Res. Logist. Q.* 2, 83–97 (1955).
41. S. Lee, An Experimental Study of the Transportation Algorithms, Master's Thesis, Graduate School of Business, University of California at Los Angeles (1968).
42. John Mulvey, Column weighting factors and other enhancements to the augmented threaded index method for network optimization, Joint ORSA/TIMS, San Juan De Puerto Rico, October (1974).

43. John Mulvey, Network relaxations for set covering, set partitioning and other integer programming problems, Research Rept. to appear Center for Cybernetic Studies University of Texas, BEB-613, Austin, Tex. 78765.
44. J. Munkres, Algorithms for the assignment and transportation problems, *J. SIAM*, **5**, 32–38 (1957).
45. *Network Flow Routine*, VIM/FOCUS Library Number : H3 CODA NETFLOW; Control Data Corporation, Software Mfg. and Distribution, 215 Moffett Park Drive, Sunnyvale, California.
46. *Out-of-Kilter Network Routine*, SHARE Distribution 3536, SHARE Distribution Agency, Hawthorne, New York (1967).
47. M. R. Rao and C. O. Fong, Accelerated labeling algorithms for the maximal flow problem with applications to transportation and assignment problems, W. P. no. 7222, University of Rochester, N.Y. (1972).
48. N. V. Reinfeld and W. R. Vogel, *Math. Programming*. Prentice-Hall (1958).
49. L. W. Smith, Jr., Current status of the industrial use of linear programming, *Mgmt Sci.* **2**, 156–158 (1956).
50. E. D. Stanley and L. Gainen, Linear programming in bid evaluation, *Nav. Res. Logist. Q.* I, 48–54 (1954).
51. V. Srinivasan and G. L. Thompson, Benefit-cost analysis of coding techniques for the primal transportation algorithm, *ACM*, **20**, 194–213 (1973).
52. *UKILT*-1100 *Programmer Reference Manual*, UNIVAC, Data Processing Division, Roseville, Minnesota.

(*Received July* 1973; *revised October* 1974)

Acknowledgement—This research was partly supported by Project No. NR 047–021, ONR Contracts N00014-67-A-0126-0008 and N00014-67-A-0126-0009 with the Center for Cybernetic Studies, The University of Texas. Reproduction in whole or in part is permitted for any purpose of the United States Government.