NETWORK OPTIMIZATION PROBLEMS AND ALGORITHMS: 
AN ANNOTATED BIBLIOGRAPHY

Michael D. Grigoriadis

DCS-TR-99

December 1980

Department of Computer Science
Hill Center for Mathematical Sciences
Rutgers University - Busch Campus
New Brunswick, NJ, 08903

Revised February 1981
# Table of Contents

## I. INTRODUCTION

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

## II. NETWORK OPTIMIZATION PROBLEMS AND ALGORITHMS

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest Path Problems</td>
<td>2</td>
</tr>
<tr>
<td>SP from node s to node t</td>
<td>2</td>
</tr>
<tr>
<td>Label-Setting Methods</td>
<td>2</td>
</tr>
<tr>
<td>Label-Correcting Methods</td>
<td>4</td>
</tr>
<tr>
<td>Simplex-Based Methods</td>
<td>4</td>
</tr>
<tr>
<td>Shortest Paths Between all Pairs of Nodes</td>
<td>4</td>
</tr>
<tr>
<td>The K Best SPs</td>
<td>6</td>
</tr>
<tr>
<td>Shortest Path Update Algorithms</td>
<td>7</td>
</tr>
<tr>
<td>Minimum Spanning Tree (MST) Problems</td>
<td>7</td>
</tr>
<tr>
<td>Maximum Flow Problems (MFP)</td>
<td>8</td>
</tr>
<tr>
<td>The Network Simplex Method</td>
<td>10</td>
</tr>
<tr>
<td>Minimum Cost Network Flow (MCNF) Problems</td>
<td>11</td>
</tr>
<tr>
<td>The Network Simplex Method</td>
<td>12</td>
</tr>
<tr>
<td>Generalized MCNF Problems</td>
<td>13</td>
</tr>
<tr>
<td>An Implementation - RNET</td>
<td>13</td>
</tr>
<tr>
<td>Matching Problems</td>
<td>15</td>
</tr>
<tr>
<td>Bipartite Matching Problems</td>
<td>16</td>
</tr>
<tr>
<td>Computational Experience</td>
<td>17</td>
</tr>
<tr>
<td>Nonbipartite Matching Problems</td>
<td>17</td>
</tr>
<tr>
<td>Tour Problems</td>
<td>18</td>
</tr>
<tr>
<td>Linear Programming Problems</td>
<td>19</td>
</tr>
<tr>
<td>Other Problems</td>
<td>20</td>
</tr>
<tr>
<td>Partitioning Methods</td>
<td>20</td>
</tr>
<tr>
<td>Multicommodity Maximum Flow and Minimum Cost Problems</td>
<td>22</td>
</tr>
<tr>
<td>References</td>
<td>24</td>
</tr>
</tbody>
</table>
NETWORK OPTIMIZATION PROBLEMS AND ALGORITHMS:
AN ANNOTATED BIBLIOGRAPHY

ABSTRACT

This annotated bibliography is intended as a repository of notes and reminders of contributions to the field of network optimization and it is by no means historically or topically complete. It represents the current state of a computerized bibliography database and the author's attempt to track the large number of technical papers in specific areas of network optimization. While it may never reach perfect completeness, future versions of this report will be released with updates and additional references.
1. INTRODUCTION

During the last twenty years, major theoretical and computational advances were made in the solution of important classes of combinatorial and network optimization problems [Kle80]. Computational complexity theory provided an understanding of what a "good" algorithm is [Edm65A], what is computable within a reasonable time, what is not, and what may not be [Kar71] [Kar72], [Gar79]. Substantial advances in programming methods and implementation technology have also improved the running time and space requirements of combinatorial and network optimization algorithms [Knu68] [Knu69] [Knu73].

The observation that algorithms operate much faster (on almost all problems that they solve) than indicated by their worst-case time complexity, has resulted in the recent trend toward the analysis of approximation algorithms [Gar76] [Sah76] [Chr76] [Sah78], probabilistically good algorithms [Kar75] [Kar76A] [Kar77], or expected time complexity analysis (e.g. [Spi73], [Kar80]), which will continue well into the eighties [Law75]. More than in any other field, progress has been due to genuine algorithmic advances rather than to improvements in computer speed and memory sizes.

Surveys and bibliographies on combinatorial and network optimization have been compiled by Bradley [Bra75], Pierce [Pie75], Golden and Magnanti [Gol77A], Hu [Hu 71], Thomas [Tho76], Chvatal [Chv76], Klee [Kle80], Hoffman et al. [Hof81] and others. Golden and Magnanti list 1000 publications for deterministic network optimization, most of them since 1963.

Books that are relevant to the area include:

Ford and Fulkerson [For62]
Berge [Ber62]
Dantzig [Dan63]
Busacker and Saaty [Bus64]
Berge and Gouila-Houri [Ber65]
Knuth [Knu68] [Knu69] [Knu73]
Hu [Hu 69A]
Harary [Har69]
Iri [Iri69]
Elmaghraby [Elm70]
Anderson [And70]
Boesch [Boe71]
Frank and Frisch [Fra72]
Potts and Oliver [Pot72]
Gefenkeil and Nemhauser [Gar72]
Geoffrion [Geo72]

Read [Read72]
Even [Eve73]
Aho, Hopcroft and Ullman [Aho74]
Erdos and Spencer [Erd74]
Christofides [Chr76]
Roberts [Rob76]
Wirth [Wir76]
Lawler [Law76]
Horowitz and Sahni [Hor76]
Bazaraa and Jarvis [Baz77]
Reingold, Nievergelt and Deo [Rei77]
Elmaghraby and Molder [Elm78]
Minieka [Min78]
Horowitz and Sahni [Hor78]
Garey and Johnson [Gar79]
Kennington and Helgason [Ken80].
II. NETWORK OPTIMIZATION PROBLEMS AND ALGORITHMS

Comments on the bibliography are organized in eight major sections: shortest paths, minimum spanning trees, maximum flows, minimum cost network flows, matching, tour and "other" problems. Emphasis has been placed on algorithms and algorithmic development. When appropriate, discussion and references have been included about the role of the simplex method in solving some classes of network optimization problems.

1.0 SHORTEST PATH PROBLEMS:

Shortest Path (SP) Problems on directed networks G(N,E), with |N|=n nodes and |E|=e edges, each edge having a given "weight". The first computational methods for shortest path (SP) problems were proposed during the fifties by Ford [For56A], Dantzig [Dan57], [Dan60], Minty [Min57] and Bellman [Bel58]. In addition to the algorithm given by Ford and Fulkerson in [For62], the literature contains well over 100 contributions and work continues to the present time. Four classes of such problems will be addressed.

1.1 SP FROM NODE S TO NODE T:

The problem is to find the (directed) path from s to t, s,tEN, s\neq t, for which the sum of the edge weights along the path is minimum over all such paths, if they exist. Most methods achieve this by producing a tree of SPs, rooted at s, so that each (directed, unique) path from node s to some other node j has this property, and so that this tree covers the maximal subset N'\subseteq N of nodes reachable from s. We refer to these problems as SP(s,t) and SPT(s), respectively. Thus SPT(s) solves SP(s,t) if and only if t is reachable from s. A finite solution to SPT(s) exists if there is no negative cycle (a directed cycle of negative total edge weight) with all of its nodes in N'. Clearly, SPT(s) has a finite solution if all edges have nonnegative weights.

Algorithms for SPT(s) are often referred to as labeling algorithms. A label is a list function defined on N, e.g. Predecessor, Shortest Path Value, node status ("permanent" or "temporary"), etc. There are two classes of methods for solving SPT(s): Label-Setting and Label-Correcting. Detailed surveys and comparisons of known algorithms are given by Witzgall [Wit68], Dreyfus [Dre69], Gillissin and Witzgall [Gill73], Golden [Gol76], Dial et al. [Dial79], Denardo and Fox [Den79], and others. An excellent unified, theoretical exposition of SP algorithms is given by Lawler [Law76].

1.1.1 Label-Setting Methods:

These start with a tree consisting of the single node s and augment the tree by one node at each iteration and by fixing its labels. While the method is attributed to C. Minty (see [Pol60]), two separate suggestions, related to data structures, by Dijkstra [Dij59] and Dantzig [Dan60] (also see Whiting and Hillier [Whi60]) improved their performance. Dantzig's improvement requires that network data are presorted in forward stars, and in increasing edge weight
within each forward star (which requires \( n \log n \) operations). Dijkstra's
requires that nodes are designated into two sets (permanent = "in-tree", and
temporary), and that a minimization over temporary labels is carried out.
Procedures which proceed simultaneously from \( s \) and \( t \) have been proposed by
Nicholson [Nic66] and Murchland [Mur67A], but they are less efficient than
Dijkstra's (see e.g. Dreyfus [Dre69]).

Disregarding sparsity and the use of efficient data structures, such algorithms
have time complexity of \( O(n^2) \) for nonnegative networks. The mere existence of a
negative edge weight may produce the wrong (undetected) result, whether negative
cycles exist or not.

Edmonds and Karp [Edm72] have proposed a modification to Dijkstra's algorithm
which runs in \( O(n^2) \) time for networks with all nonnegative edges, and in \( O(n^3) \)
for time for networks with any weights but with no negative cycles.
Unfortunately, the existence of a negative cycle may induce an exponential time
for their algorithm, as shown by example in [Joh73]. Nemhauser defines and uses
a labeling function which essentially allows the use of Dijkstra's method for
networks with all negative cycles [Nem72]. This amounts to requiring a dual
feasible solution. Bazaar and Langley give a procedure for constructing such a
dual feasible solution or to detect the existence of a negative cycle [Baz74].

E.L. Johnson [Joh72] introduced specific data structures for managing the
candidate list in Dijkstra's method and analyzed the complexity of the algorithm
for a special class of networks. Sparsity is exploited by D.B. Johnson [Joh77]
who uses a priority queue (managed as a heap) within Dijkstra's method, and
provides asymptotic results depending on the degree of sparsity and the depth of
the heap. For the appropriate problems and the corresponding "best" choice of
the heap depth, these results can be substantially better than the \( O(n) \) for
dense problems. We emphasize the attractive feature of adjusting the heap depth
based on information derived from the properties of the problem instance.
Another proposal for sparse graphs whose complexity depends on the problem data
(magnitude of the maximum edge weight) is by Wagner [Wag76].

Recent interest has focused on handling large, sparse networks and on
empirically studying the average performance of SP algorithms. Gillsin and
Witzgall [Gil73] report that Dial's modification [Dia69] of Dijkstra's method is
the fastest in practice. This uses an address calculation sort but requires
integer edge weights. It has a time and space complexity which depends on the
maximum edge weight, and thus complexity is affected by scaling the data. The
subsequent study by Dial et al. [Dia79] finds a (a la Dantzig) forward-star
modification of Dial's method attractive only on relatively dense problems, but
not decidedly so. Glover and Klingman [Glo80] report preliminary results with a
"threshold/partitioning" scheme for candidate list management indicating much
faster execution times than all other methods. Gallo and Pallotino [Gai79A]
report computational results for large practical problems.
1.1.2 Label-Correcting Methods:

These methods treat problems with negative edge weights and detect the existence of negative cycles in time $O(n^3)$, disregarding advantages of sparsity and data structures. They typically start with any tree covering all nodes in $N$, and a set of node labels, and monotonically improve the tree and labels in successive iterations, until no further improvement is possible. The general approach is due to Moore [Moo57], Ford [For62] and Bellman [Bel58], and is best understood in terms of Bellman's recursion. Yen [Yen70] has provided improvements which reduce the time and space complexity by half.

D.B. Johnson observes that while implementation of the algorithm in $O(n^3)$ is possible when the network data are given as a list, it is also possible to cause exponential time complexity by implementing imprudent search strategies for selecting among candidates. D'Esopo suggests ordering the nodes according to their (cardinality) distance from $s$ [Pol60], and Pape [Pap74] improves further by processing the candidate list from both ends. Gallo and Pallotino [Gal79], report that Pape's modification results in the best execution times over all other implementations of label-correcting methods, and Dial et al. [Dia79] claim this to be true over all labeling methods (with the mild exception for Dial's algorithm).

These results emphasize the importance of label correcting methods on two counts: no a priori knowledge of negative cycles is needed, and the experience for networks with all nonnegative edges is competitive with the best label setting implementations. However, it is necessary to have the network data presorted by forward stars.

1.1.3 Simplex-Based Methods:

The relationship of the above SP algorithms to the simplex method has been recognized since the inception of the problem (see e.g. [Law76]). A straightforward specialization of the network simplex method, albeit of exponential worst-case bound, handles negative edge weights and detects negative cycles. The implementation can be arranged so that the data need not be ordered in any way. This can be accomplished by considering a random sample of candidates and by selecting the best among this sample. This seemingly unsophisticated approach has been very successful for minimum-cost network flow problems [Gri73]. Recently, Cunningham [Cun79] has shown that a particular implementation of the simplex method (with a round-robin pricing strategy) is polynomially bounded for the SPT(s) problem.

1.2 Shortest Paths Between All Pairs Of Nodes:

The problem is to find shortest paths between all pairs of nodes (or all pairs in a given subset of nodes) of $G$. There are many practical applications of this problem, notably in steady-state traffic assignment models of transportation systems (e.g. [Yag71]), and in designing communication networks (see e.g. [Fra72], [Gre73], [Fra75], [Ber77], [Ber78], [Ber79], [Ber79a], [Ber80]). In some instances, the SP problem is embedded within another algorithm (e.g. in [Fra73] within a steepest descent minimization scheme), which requires that the
SP problem is solved numerous times. Special cases of the problem are also useful in practice, for detecting negative cycles, determining connectivity of pairs of nodes, etc.

The first algorithm for this problem is due to Floyd [Flo62] and it is related to Warshall's $O(n^3)$ transitive closure algorithm [War62]. Floyd's algorithm has time complexity of $2n^3$ and space complexity of $n^2$. Hoffman and Winograd [Hof72] have improved the time to $n^3$ comparisons and $O(n^2 \cdot 5)$ additions and subtractions. As in the case of label-correcting methods, negative cycles can be detected by a simple test, and thus their existence does not have any catastrophic effects. Dantzig gave a similar matrix algorithm [Dan66] which was subsequently improved by Tabourier [Tab73]. Spira [Spi73] presents an algorithm for all positive, i.i.d edge weights with expected time of $O(n^2 \log^2 n)$.

The use of a matrix by the above methods precludes the solution of large networks. To avoid this deficiency, recent studies have focused on methods which essentially consist of repetitive applications (with appropriate algorithmic embellishments) of labeling algorithms for solving SPT(s) problems. Such a method, which has been extensively tested with label-setting and label-correcting methods (as its subroutines), has been proposed by Gallo and Pallottino [Gdl79A]. It requires storage of $5n+2e$. Another method by Florian et al. [Flo79] is based on the observation that SPT(i), for some root node i, is dual feasible for SPT(j), j#i. They present experimental results to support the claim that the sequence of (n-1) dual-simplex restarts give faster solution times than the repeated application of the best label-correcting algorithm of [Pap74] or of the "best" label-setting algorithm of [Dia69]. The sequence in which roots are selected is obviously important. Both methods make this selection heuristically. Kelton and Law [Kel78] report on computational comparisons of the Floyd, Dantzig-Tabourier, and the repetitive applications of label setting and label correcting algorithms for SPT(i); i=1,...,n. Their results indicate the superiority of the repetitive label correcting algorithm for sparse networks.

Johnson [Joh77] shows that his SP algorithm for sparse networks can be extended to solve the present problem with a time complexity of the order of \( \min(n+\sqrt[3]{n}/k), n^2 \log n + n \log \log n \) where k is the selected heap depth. Both Johnson, and Gallo and Pallottino use the labeling function proposed by Edmonds and Karp [Edm72] and Nemhauser [Nem72].

Sparsity in large networks has also been treated by taking advantage of problem structure via Partitioning or decomposition schemes. The first such method was proposed by Land and Stairs [Lan67] for large networks which "consist of several parts with limited interconnections". Their original idea was later improved by Hu [Hu 68], and Hu and Torres [Hu 69], which could run as fast as $O(n^2)$ for some classes of networks. Provided that the network data presented to these methods is accompanied by an attractive partitioning recipe, the approach could prove advantageous for large problems.
1.3 The K Best SPs:

In many applications, pragmatic modeling approaches omit important but complex constraints, e.g., see [Hor80]. It is thus useful to exhibit the best, the next best,..., the K-th best solution to the (simplified) problem, so that the analyst may select the one which is most "suitable". The generation of K (disjoint) paths is extensively used in reliability analysis of communication networks. Murty [Mur68], was the first to study a related problem, that of "ransom" all solutions to an assignment problem, in increasing order. Other early work is by Hoffman and Pavley [Hof59] and Bellman and Kalaba [Bel60]. Since then, such methods have been devised for SP(s,t) problems, undirected spanning trees and arborescences, and path trees. The ranking of solutions to SP(s,t) gives rise to two types of problems: For SPs with repeated nodes and without.

Dreyfus stated an O(Kn^2) algorithm (a modification of [Hof59]) for finding the K best SP(s,t). The method was later improved to O(Kn log n) by the use of priority queues by Knuth (see e.g., [Law76]). The space complexity is Kn^2.

Generalizing the notions of Murty [Mur68], Lawler [Law72], [Law76] has treated the ranking problem in the more general setting of a (0,1) optimization problem. He then specialized his approach to the present problem without repeated nodes, thus improving the method of Yen [Yen71] (also see [Yen72] and [Wil73]). His algorithm has time complexity of O(Kn^3) and space complexity of order O(Kn). The SP(s,t) computation dominates the running time, i.e. O(Kn) SP computations are needed, each requiring O(n^2) time. The method can handle negative edge weights, but only after a preprocessing step of O(n^3) operations, which transforms the problem into one with nonnegative weights. It is assumed that there are no negative cycles. The implementation of Lawler's algorithm presents a pronounced time-space tradeoff. The best implementation strategy depends on the data structure (e.g., priority queues), and on the amount of available storage. Storage requirements can be reduced by accepting more SP computations (see [Law76] and [Mar73]). As discussed earlier, each SP computation can also be made faster by the use of appropriate data structures (e.g., [Jon73]).

The K-best disjoint path problem has also been studied by Suurbale [Sus74], who uses K-1 shortest path computations with edge weight transformations (see [Edm72]) for each to allow the use of an O(n^2) SP computation for all nonnegative weights. More recently, Shier [She76], [She79] has performed extensive computational experiments on medium to large grid, complete and random networks. He indicates that for low-density networks, best implementations of label-setting algorithms perform the best.

We note that the node (or arc)-disjoint paths problem can also be formulated as a minimum-cost flow problem, as originally suggested by Ford and Fulkerson [Ford62]. Frisch provides a related specialized algorithm [Fris72]. Specialized applications of the network simplex method for treating this problem have not been investigated.
1.4 Shortest Path Update Algorithms:

Algorithms for updating existing SPT(s), or the shortest paths between all pairs of nodes, in order to reflect small modifications to the data have been proposed by several authors. Based on the original work of Murachland [Mur67], which considered the modification of one edge weight, modification algorithms were proposed by Halder [Hal70], Dionne [Dio74], and used by Boyce et al. [Boy73]. Other modification algorithms have been proposed by Pape [Pap69], Spira and Pan [Spi73A], Cheston [Che76B], and Goto et al. [Got78]. Presently, no update algorithm, which will reflect all cases of changes to the original problem and which has better worst-case bound than the methods available for re-solving the problem, is known. Similarly, the use of parametric programming, albeit of exponential time complexity for linear programs [Mur80], may also be effective in practice. Some of the proposals in these references appear to be computationally attractive in practice [Che79].

2.0 Minimum Spanning Tree (MST) Problems:

The problem is to find a spanning subtree of a given connected (undirected) network which has minimum weight. This problem, which has been studied since 1926 [Bor26], has applications in communication network design [Gre73], [Ker74], [Gril74A], [Gril77A], network reliability analysis [Van71], [Van71A], cluster analysis [Zah71], pattern recognition [Lut74], circuit layout [Lob57], multi-terminal network flows [Gom61], establishing the connected components of a given network, and others. The NP-complete optimum communication spanning tree problem is a variant of the MST problem which has important applications in network design [Gar79]. Other constrained spanning tree (ST) problems, all NP-complete, are also of practical importance. Such problems are degree constrained ST, capacitated ST [Pap78] and geometric capacitated ST, bounded component ST, Steiner tree in graphs, geometric Steiner tree [Gar79].

While it is possible to study "greedy" algorithms in terms of matroid theory [Lau76], algorithms for the MST problem originate from two basic approaches: the method of Kruskal [Kru56], and the methods of Prim [Pri57] and Dijkstra [Dij59]. Implementations of MST algorithms differ in their choice of data structures (see e.g. [Fox78]) and candidate search schemes within the framework of these two classes of methods.

The Kruskal method performs the following procedure n-1 times: Select an edge with the least weight from the list of candidate edges (initially all edges are candidates); if it does not cause a cycle, add it to the tree. In any case, remove this edge from the candidate list.

Implementations of this scheme naturally consider two computational chores: the selection of a candidate edge and the detection of a cycle. The first can be accomplished by searching over all candidate edges, by presorting the entire edge list by increasing weights, or by using a treesort [Flo64], [Wil64] for maintaining the candidate edges in a heap, as first proposed by E.L. Johnson (for the shortest path problem [Joh72]) and by Kershchenbaum and Van Slyke [Ker72]. The second chore has been extensively studied along approaches such as introducing list functions for identifying connected components, linked list structures for the same purpose, using the root of the subtrees for identifying
the connected components and by using the predecessor list function to trace to
the subtree root from any node, and using path compression and efficient "find"
and "set union" operations (see e.g. [Ker72], [Obr64], [Sep70], [Che76].)
Depending on the choice of the above, Kruskal's method can be implemented with
time complexity ranging from $O(n^2)$ to $O(n \log n)$. Computational experience with
various approaches is given in [Ker72], and also in [Hay80], where the best
Kruskal implementation is reported as the one which uses heapsort for
maintaining the candidate list and a linked list for maintaining connected
component identifications.

The Prim/Dijkstra method tries to successively satisfy the optimality criteria
for an MST (i.e. a tree is MST iff for every subset of nodes there is a tree
edge with least weight among all edges connecting a tree node to a non-tree
node) by appending one node and edge to the tree which starts from one node and
grows to the MST which spans the maximal connected component containing
the initial node. The best implementation of this method is due to Kershonbaum
and van Slyke [Ker72] which uses a heapsort to maintain the temporary labels
[Joh72]. Recently, Haymond et al. [Hay80] have proposed the use of Dial's
[Dia69] address calculation sort which appears to be faster than the heapsort
approach but requires that the edge weights are appropriately scaled for each
problem instance. In variance to the results reported by Kershonbaum and
van Slyke [Ker72], Haymond et al. report that the Prim/Dijkstra type of
algorithms outperformed the best Kruskal implementation. However, as pointed
out in [Ker72], the choice of a particular algorithm may depend on many other
factors which are not equally addressed by the two methods. For instance,
Prim/Dijkstra type algorithms require the edge information in node incidence
form, they cannot be used directly to obtain minimum spanning forests, but they
are especially suited when edge weights are implicitly defined (and thus not
stored) by simple functions, e.g. Euclidean distance or functions of it. On
the other hand, Kruskal-type algorithms can accept the edge list in any form,
can produce minimum spanning forests but they require more storage (edge list
plus component list for cycle detection).

Algorithms for updating existing MSTs, in order to reflect small modifications
to the data have been proposed by Roger and Carpenter [Rog71], Spira and Pan
[Spi73], Cheston [Che76b] [Che79], Chin and Hock [Chi78] and others.
Presently, no update algorithm, which will reflect all cases of changes to the
original problem and which has better worst-case bound than the methods
available for re-solving the problem, is known. In practice however, some of
the proposals in these references appear to be attractive [Che79].

3.0 MAXIMUM FLOW PROBLEMS (MFP):

The MFP is to find the maximum flow from some node s ("source") to another node
t ("sink") in a capacitated directed network of n nodes and e edges. The
central theorem in the field is the Max-flow Min-Cut theorem of Ford and
Fulkerson [For56] (later generalized by Hoffman [Hof73]). This classical
problem in network flow theory is a specially structured LP which admits special
algorithmic treatment, and has many applications in practice as well as in
solving other seemingly unrelated problems, such as the bipartite cardinality
matching problem.
Ford and Fulkerson gave an algorithm [For56] which has time complexity O(ev), where v is the maximum s-t flow. Finiteness and convergence to the correct solution requires integer data. The algorithm uses two "labels" on each node to repetitively identify a (s to t) "flow augmenting path", and to increase the flow along such a path. The information reflected by these labels is discarded at the end of each iteration. This pioneering, but non-polynomial algorithm, was improved by Edmonds and Karp [Hu 69], [Edm72], by simply imposing a breadth-first search discipline in identifying the next edge of the flow augmenting path, which reduced the time complexity to O(n²e), and does not require integer data. The so-modified Max flow algorithm deals with flow augmenting paths of minimum (cardinality) length. Zadeh has shown that this bound is tight [Zad72], [Zad73A]. An O(n²e) algorithm which uses "layers" (each layer is longer than the preceding layer by one) and flow augmenting paths was independently obtained by Dinic [Din70], and later improved by Karzanov [Kar74] to O(n³). It is fair to state that while the Edmonds-Karp procedure is of great theoretical value, the Dinic and Karzanov methods are computationally more attractive.

Since these results, there has been renewed interest in MFP:

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ford-Fulkerson</td>
<td>O(ev)</td>
</tr>
<tr>
<td>Edmonds-Karp [Hu 69A] [Edm72]</td>
<td>O(ne²)</td>
</tr>
<tr>
<td>Dinic [Din70] [Eve76]</td>
<td>O(n²e)</td>
</tr>
<tr>
<td>Karzanov [Kar74] [Eve76]</td>
<td>O(n³)</td>
</tr>
<tr>
<td>Malhotra et al. [Mal78]</td>
<td>O(n³)</td>
</tr>
<tr>
<td>Cherkasky [Che76A]</td>
<td>O(n²e²0.5)</td>
</tr>
<tr>
<td>Galil [Gal78] [Gal78A]</td>
<td>O(n³/3e²/3)</td>
</tr>
<tr>
<td>Shiloach [Shi78]</td>
<td>O(n(n+e) log²(n+e))</td>
</tr>
<tr>
<td>Galil and Naamad [Gal79]</td>
<td>O(en log² n)</td>
</tr>
</tbody>
</table>

Depending on the density of the problems to be solved, the various algorithmic proposals above provide a "choice" of the method to be used. The methods of Karzanov, Malhotra et al., Cherkasky, Galil, and Galil-Naamad, etc., can be viewed as data structure-related improvements of the Dinic method, and require programming tasks of widely varying complexity. Implementation of the Dinic and Karzanov methods (with the modifications indicated in [Mal78]) appear to be easier than the others. This emphasizes the need for establishing the multiplicative constants of the polynomial expressions indicated above. A recent computational study by Cheung [Che80A] has addressed this question. Other computational experiments have been reported by Glover et al. [Glo80B]. The results of our experiments with the Edmonds-Karp, Dinic and specializations of the network simplex method are discussed below. The MFP with parametric capacities is discussed by Somers and Adler [Som79], and a cut search algorithm presented by Phillips and Dessouly [Phi79].
3.1 The Network Simplex Method:

When applied to MFP, the network simplex method has time complexity of $O(2^{n/2})$ and it suffers from highly degenerate basic solutions which cause slow convergence in practice. Computational experience [Gri78] tends to support previous arguments of its inapplicability to MFP (see e.g. [Hu 69]). We believe that in addition to these arguments, the poor performance of the network simplex method can be explained as follows.

Since all edge costs are zero (except for the circulation edge), the dual variable associated with each node that is connected to the source node on the basis tree is zero, and the dual variable for each other node is one. Therefore, the cost of a unit flow around a flow augmenting path can only be 0, 1, or -1. If 0, there is no flow augmentation, if -1, there is a flow augmentation (if nondegenerate), and if +1, the is "negative" flow augmentation (if nondegenerate). There is little information to make an intelligent choice for the incoming edge during the "pricing" step.

In order to remedy this deficiency, we examined the effectiveness of the steepest-edge simplex method proposed by Goldfarb and Reid [Go176] for general LPs, and further specialized by Goldfarb [Go177] to MCNF problems. The latter moves to an adjacent vertex down an edge of the polyhedron (defined by the problem constraints) that is most downhill with respect to the objective function. In terms of the network simplex method, this translates into selecting the edge which induces the shortest (least number of edges) negative-cost flow augmenting path.

While Goldfarb and Reid were able to show the usefulness of the steepest-edge criterion for general LPs, our experiments showed that the same advantage does not materialize for MCNF problems. Mulvey [Mul78], [Mul78A] reports similar experience. However, we did obtain substantial improvements for MFPs, both in number of iterations and in total running time [Gri78].

Our RNET code [Gri79A], [Gri79B], contained the steepest-edge modification in all of its versions which were made publicly available since 1979. We are not aware of any other implementations of this procedure. RNET does not include any other specialization for MFP, even though such specializations would improve running times for MFP by some multiplicative constant.

We have also implemented efficient versions of the Edmonds-Karp modification of the Ford-Fulkerson flow augmenting path algorithm, and of the Dinic algorithm. Our implementations of the latter two made use of data presorting, two ways: By heads and by tails of the directed edges. RNET does not require the data to be sorted in any way. We used the NETGEN program [Kii74] to generate several large, sparse networks with integer capacities. We generated twenty eight problems, ranging in size from 400 nodes to 8,000 nodes. In all instances, the steepest-edge network simplex method (RNET) was faster than the Dinic implementation which in turn was faster than the Edmonds-Karp implementation by a wide margin. Disregarding one sort time for the Dinic and Edmonds-Karp implementations, our results are as indicated in Table 1.
Table 1. Consolidated Test Results for Maximum-Flow Test Problems Derived from [Kl74].
(Times are IBM System 370, Model 168 seconds.)

<table>
<thead>
<tr>
<th>MaxFlow Problem Number</th>
<th>Number of Nodes</th>
<th>Number of Edges</th>
<th>Pct of Cap. Edges</th>
<th>Execution Times (CPU secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Kl74]</td>
<td></td>
<td></td>
<td></td>
<td>RNET (modif)</td>
</tr>
<tr>
<td>16,18</td>
<td>400</td>
<td>1306</td>
<td>20</td>
<td>1.1</td>
</tr>
<tr>
<td>17,19</td>
<td>400</td>
<td>2143</td>
<td>20</td>
<td>1.2</td>
</tr>
<tr>
<td>20,22</td>
<td>400</td>
<td>1416</td>
<td>40</td>
<td>0.9</td>
</tr>
<tr>
<td>21,23</td>
<td>400</td>
<td>2836</td>
<td>40</td>
<td>2.0</td>
</tr>
<tr>
<td>24</td>
<td>400</td>
<td>1382</td>
<td>80</td>
<td>1.3</td>
</tr>
<tr>
<td>25</td>
<td>400</td>
<td>2676</td>
<td>80</td>
<td>1.5</td>
</tr>
<tr>
<td>26</td>
<td>400</td>
<td>1382</td>
<td>80</td>
<td>1.1</td>
</tr>
<tr>
<td>27</td>
<td>400</td>
<td>2676</td>
<td>80</td>
<td>1.3</td>
</tr>
<tr>
<td>39</td>
<td>5000</td>
<td>15000</td>
<td>0.7</td>
<td>1.2</td>
</tr>
<tr>
<td>40</td>
<td>3000</td>
<td>23000</td>
<td>0.7</td>
<td>1.5</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td></td>
<td>14.1</td>
</tr>
</tbody>
</table>

4.0 Minimum Cost Network Flow (MCNF) Problems:

Given G(N,E) with exogenous flows at its nodes and a capacity and cost on each of its edges, find the minimum cost flow through the network which satisfies all exogenous flows and edge capacities. The MCNF problem is widely used in modeling many applications in industry [Glo77]. Its generality allows many other problems to be formulated as such, e.g., maximum flow, shortest paths SP(s,t), assignment (weighted matching), production scheduling and distribution, etc. Glover and Mulvey show that some (0,1) integer programs are reducible to generalized or pure MCNF problems [Glo80A].

The first algorithms that have been proposed for solving the MCNF problem are Ford and Fulkerson's Minimum Cost Flow Algorithm [For62] and Fulkerson's Out-of-Kilter Algorithm [Ful61], [For62]. The former requires that the initial flow is set to zero, lower bounds on edge flows are given as zeroes and edge costs are nonnegative. The latter (a primal-dual method) does not impose any of the above restrictions and it accepts and uses good approximations to the dual variables. The out of kilter algorithm has been extensively programmed (e.g. [Cla68A], [Bra64], [Bar74]), and experimental results compared to the primal and dual network simplex methods have been presented [Gri68], [Hat75], [Glo78A].

Primal methods for solving MCNF have been proposed by Busacker and Gowen [Bus61] and Klein [Kie67], and Lawler [Law76] who states a combination of the former two algorithms. These algorithms can be shown to have time complexity expressed as polynomials in the amount of flow and thus cannot be considered as polynomially bounded (in the length of the input). However, Edmonds and Karp [Edm72] have devised a "scaling" procedure for the out of kilter algorithm which renders a sequence of problems, approximating the given problem, polynomially solvable. Unfortunately, this improvement is not reflected in actual computational experience. Zadeh has constructed pathological instances which cause all MCNF
methods to have exponential time complexity, except for the latter. However, experimentation has shown that, in practice, a good implementation of the network simplex method is more efficient for solving large MCNF problems. MCNF problem data generators have been programmed by Klingman et al. [Kli74] and Ali et al. [Ali77].

4.1 The Network Simplex Method

The method is a straightforward specialization of Dantzig's revised simplex method [Dan63], enhanced by the use of powerful data structures. The major simplification results from the observation that any basis matrix of a node-edge incidence matrix (with one artificial column added to insure full rank) is triangular (after permuting rows and columns), so is its inverse and both are totally unimodular. Furthermore, the basis matrix corresponds to a rooted spanning tree, as observed by Dantzig [Dan63], and its inverse has columns \( j \) which correspond to the node-\( j \)-to-root path and rows \( i \) which correspond to the nodes of the subtree not containing the root node formed when edge \( j \) of the tree is removed. Thus the two basic operations required by the simplex method, premultiplication and postmultiplication of the inverse by a vector, can be implicitly performed [Gri71a] (also see Rardin [Rar79]). The third operation required by the simplex method, the implementation of a Gauss-Jordan pivot step (a Gauss step is not considered since the basis is triangular), is implemented the unique cycle created by the entering edge, identifying the edge in this cycle to be dropped so that feasibility is maintained, and reordering the basis (tree) so that it becomes triangular. Concrete proposals data data structures useful in implementing these ideas were by Scoins [Sco64], E.L. Johnson [Joh66] and Srinivasan and Thompson [Sri72]. Implementations are by Glover et al. [Glo74a], Bradley et al. [Bra77], Mulvey [Mul78a], [Mul78b], Grigoriadis [Gri71a], [Gri78], Grigoriadis and Hsu [Gri79a], Kennington and Helgason [Ken80]. An implementation of the dual simplex method is discussed in [Hel77]. Review of implementations and applications are found in [Glo74], [Cha75], [Lan74], [Ros75], [Kar76], [Glo77], [Glo80], [Ken80] and possibly many others.

The use of the simplex method for solving the transportation problem was proposed by Dantzig [Dan51], for the transshipment problem by Orden [Ord56], and for maximum flow problems by Fulkerson and Dantzig [Ful55] [Dan63]. Cycling (due to degeneracy) can theoretically occur in the application of the network simplex method to MCNF problems, particularly for those that have integer data. Gassner [Gas64] constructed an example for which the network simplex method suffers from cycling.

Recently, Cunningham has proposed a simple modification of the network simplex method which prevents cycling, even for integer data [Cun76]. Cunningham maintains "strongly feasible" bases (spanning trees) whose edges satisfy specific direction conditions. He accomplishes this by a specific choice of the edge leaving the tree. Bland has also proposed a method which prevents cycling by a specific choice of the entering and leaving edges. [Bla77]. Cunningham has further proposed a method which prevents both cycling and "stalling", an exponentially long sequence of consecutive degenerate pivots without cycling [Cun79]. As pointed out earlier, his method insures a polynomial number of steps for the shortest path problem when solved by the so-modified network simplex method. A polynomial bound has also been shown by Nemhauser and Thura for the vertex packing problem [Nem80]. Adolphson recently presented an
implementation of the network simplex method which performs only nondegenerate simplex steps, but it requires the computation of a shortest path matrix at each iteration [Ado79]. The computational efficacy of the method is not known. Finally, Zadeh has constructed pathological examples of networks for which the simplex method requires an exponential number of pivots [Zad73b].

4.2 Generalized MCNF Problems:

Applications of generalized MCNF problems are discussed by Maurras [Mau72] and Glover et al. [Glo78]. The revision of the above non-simplex algorithms to the generalized MCNF problem (with positive gains) is relatively straightforward, although it is no longer possible to show that their time complexity is polynomial in the amount of flow [Law76]. Such algorithms have been proposed by Balas and Ivanescu (Hammer) [Bal64], Eisenman [Eis64] and Balas [Bal66] for generalized transportation problems, and by Charnes and Raike [Cha66], Onaka [Ona67], Jarvis and Jezior [Jar72], Minieka [Min72], Grinold [Gri73] and others. The algorithm by Jewell [Jew62] which generalizes Ford and Fulkerson's algorithm, can handle negative gains. Simplex-based specializations, based on Dantzig's suggestions [Dan63] and E.L. Johnson's proposals [Joh66], can also handle this more general case and have been found to be very efficient in practice, as indicated by the work of Maurras [Mau72], Glover et al. [Glo72], [Glo73], Elam et al. [Elam79] and others. An excellent chapter on the use of the simplex method for such problems is given in [Ken80]. The application of the steepest edge criterion for the simplex method [Gol76a] as applied to the MCNF problem was proposed by Goldberg [Gol77] and Mulvey [Mul78a] and [Mul78a].

The possibility of transforming generalized MCNF to pure MCNF problems is attractive since the latter can be solved more efficiently. Glover and Klingman [Glo73a] have shown that a positive-gains problem can be transformed to a pure MCNF problem if the matrix of the original problem has rank deficiency of one, and have proposed a procedure which generates appropriate row and column scales to achieve this transformation, if one exists. Truemper [Tru76] has devised an efficient procedure for scaling problems with gains of arbitrary sign to obtain (whenever possible) an equivalent network with positive gains or with unit gains (pure case). This procedure may require modifications of the original graph in order to complete the transformation. On the other hand, Glover and Klingman's procedure scales as many edges as possible, thus reducing the overall number of edges of non-unit gains.

4.3 An Implementation - RNET:

The Rutgers Minimum Cost Network Flow Subroutines (RNET) is a set of portable and structured FORTRAN subroutines which can be called from user-written FORTRAN (or PL/1) programs or subroutines [Gri79], [Gri79a]. It is an efficient implementation of the network simplex method for solving MCNF problems. Assignment, transportation, transshipment, shortest paths, maximum flow and bipartite matching, can also be solved by RNET, provided that they are stated as MCNF problems. In addition to solving these problems, RNET provides a number of "services." These may be used to manipulate linear forms, to modify the original list data, to restart the optimization process under various options, to perform a specified simplex iteration, to prepare various types of output, etc. For
instance, the usual premultiplication or postmultiplication of vectors by the basis inverse can be performed without the explicit knowledge by the user of the basis tree, its list structures, the assumed order of traversal, and indeed without requiring the user to write list processing code for this task.

As outlined in [Gri78], RNET implements the network simplex method as a single-phase, multiple pass gradual penalty method (a form of Wolfe's composite phase approach [Wo165]). In the first pass, an all-artificial, one-level rooted tree (basis) is constructed and a mild penalty cost is assigned to all artificial arcs. Simplex iterations are performed until an optimal solution is obtained with respect to the current penalty cost. If there are artificial arcs with positive flow, the penalty cost is increased, the dual variables (node potentials) are recomputed, and simplex iterations are continued for the second pass, etc. The method terminates with an optimal solution when all artificial arcs have zero flow, or, provided that the penalty cost is sufficiently large, with an infeasible solution (if an artificial has positive flow), or with an infinite solution (caused by a directed negative cycle of infinite capacity.)

In performing a simplex iteration, arcs are selected to enter the tree by an almost random, intentionally unsophisticated partial pricing search strategy (using ordinary reduced costs), which does not require any prescribed order of the input arc list.

We have experimented with several implementations of Goldfarb's steepest edge criterion [Go177A] [Go177B]. We did not find these to be worthwhile. However, the special case of maximum flow showed a marked improvement when a hybrid strategy, composed of the simple "random" search and the steepest edge, was tested. Accordingly, this was included in the code, though no other space-saving or computational tricks arising from the structure of the maximum flow problem were included. Finally, we found that generating advanced starting basis trees ("crashing") was not worth the trouble mainly because such trees are multilevel and thus more complex than the single-level tree created by an all-artificial start.

Portability cannot be assured for FORTRAN and PL/I programs employing integer arithmetic, by simply adhering to ANSI standard [ANS66] statements (see e.g. [Ryd74], [Ryd79], [Smi74], [Smi77].) It is not widely known that most FORTRAN, (under more specific circumstances, PL/I), and "C" programs may produce the wrong, undetected, answer when integer addition and multiplication of sufficiently large integers is involved in an all-integer arithmetic statement. For this reason, we included in RNET elaborate mechanisms to avoid such undetected computational errors due to overflow conditions arising from large user-specified costs and flow capacities. For each installation, the portable RNET code requires the specification of the wordlength for that machine, in bits. Polynomial arithmetic is performed whenever necessary. This observation has motivated the development of portable integer arithmetic programs which will not fail to produce the correct result for any computer [Gri80B]. Note that the facilities offered by the latter cannot be found in multiple precision integer arithmetic packages (such as [Go166]) since these packages are not portable. On the other hand multiple precision arithmetic packages are much more efficient in their detection of overflow conditions, precisely because that portion of the code is normally written in assembly language.
Table 2. Consolidated RNET Test Results for the Forty
Test Problems of [Kl74].
(Times are IBM System 370, Model 168 seconds.)

We have reported [Gri78] the computer execution times which were achieved for
all forty test problems in the literature [Kl74]. Table 2 presents a summary
of these results. We presented [Gri79] a computational study which dealt with
the sensitivity of the gradual penalty factors, the sample size of the edges to
be searched before one is considered for entry to the basis, and with the choice
of the steepest edge criterion within the framework of this pricing strategy.
Our results indicate that for maximum flow problems, a particular combination of
the latter two is substantially more efficient than our implementations of the
best family of maximum flow algorithms, [Edm72], [Din70], [Kar74], [Min78].
Prior to its public availability, RNET compared extremely favorably with other
implementations of the network simplex method [Gri78]. This experience is
confirmed in the recent book by Kennington and Helgason [Ken80].

5.0 MATCHING PROBLEMS:

A matching in a graph is a combinatorial optimization problem which requires
that some set of edges is found such that each node of the graph is incident to
at most one edge. There are several classes of such problems for bipartite and
nonbipartite graphs and networks. Mainly due to the work of Edmonds [Edm65a],
[Edm65b] [Mor69], these problems are solvable by algorithms of polynomial time
complexity. Nevertheless, good implementations are complicated and require the
use of sophisticated data structures, particularly for solving large, sparse
problems, and the recent book by Burkard and Derigs gives some FORTRAN
implementations [Bur80]. The books by Lawler [Law76] and Minieka [Min78]
contain excellent reviews and give practical examples for such problems.
Maximum matching problems are related to minimum covering problems (see e.g.
Balinski [Bal70], Minieka [Min78]).
5.1 Bipartite Matching Problems:

One of the simplest linear programs, that of finding the "best" assignment of, say n men to n jobs, is also a member of a larger class of problems, known as matching problems. The assignment problem has been extensively studied since the mid-fifties when the (primal-dual) "Hungarian" method was first proposed by Kuhn [Kuh55]. The development of primal-dual methods for the minimum cost network flow problem (by Fulkerson [Ful61] and Dantzig, Ford and Fulkerson [Dan65]), and for LPs by Dantzig, Ford and Fulkerson [Dan65], were motivated by the Hungarian method for assignment problems.

To define the problem, consider a bipartite graph with m sources, n sinks and e edges, where a given weight is associated with each edge. A matching is a subset of the edges of the graph such that no two arcs in this set are incident to the same node. A complete matching matches each and every node in this manner. Three classes of bipartite matching problems are of interest: (Maximum) Cardinality Matching [Ber57] (find matching with most edges), the Bottleneck Matching (Max-Min Matching) problem (find cardinality matching so that the smallest edge weight in the matching is maximized), and the Weighted Matching (find matching with maximum or minimum total edge weight.)

The cardinality matching problem is linearly reducible to the maximum flow problem, and the weighted matching problem is equivalent to the assignment problem. The cardinality matching problem also solves the (minimum) cardinality covering problem, and conversely. Furthermore, the weighted matching problem can be cast as a minimum cost network flow problem, which in turn, is easily expressed as a capacitated transshipment problem. The latter can be formulated (by doubling the number of nodes and by adding as many directed edges as there are nodes) to the capacitated transportation problem.

The bottleneck assignment problem is discussed by Gross [Gro60] and by Edmonds and Fulkerson [Edm70A]. The relationship between the bottleneck and weighted assignment problems is discussed by Burkard and Zimmermann [Bur80A]. In a computationally irrelevant manner, the capacitated transportation problem is reducible to a pure transportation problem, which, in turn, can be expressed (again in a computationally irrelevant manner) to the assignment problem. Therefore, the weighted matching problem and the minimum cost problem are mathematically equivalent. Finally, the SP(s,t) problem (for networks with no negative cycles) can be cast in terms of an assignment problem.

It is known that there can be no \( O(n) \) algorithm for any of the three matching problems stated above, except in the special case of cardinality matching for "convex" graphs, as defined by Glover [Glo69]. The best known algorithm for cardinality matching is due to Hopcroft and Karp [Hop73], which has time complexity of \( O(n^{2.5}) \). An \( O(m^{2.5}) \) algorithm is given by Lawler [Law76] and is attributed to Ford and Fulkerson, and Hall.

The best algorithms for max-min matching are the "threshold" method based on the proposals of Fulkerson, Glikberg and Gross [Ful73], and another method based on augmenting paths (see [Law76]). Both of these have time complexity of \( O(e^{2.5}) \).
Assignment (weighted matching) problems can be solved by the Hungarian method, or by a primal method due to Klein and Buscaker and Bowen (see [Law76]), in \(O(e^2 n)\) time. Balinski and Gomory proposed a different primal method [Bal64A]. Recently, Karp [Kar80] has shown that a modification flow augmenting path algorithm (using priority queues) has expected time complexity of \(O(ne \log n)\). We are not aware of any experimental results with this method. Bertsekas [Ber79B] has devised another algorithm which appears to be well suited for very dense problems. Its time complexity is \(O(n^3)\) for \(m=n^2\) problems, but experimental results indicate average running times one order of magnitude faster than the "best" implementation of the Hungarian method.

5.2 Computational Experience:

Large assignment problems arising in practice are solved by the network simplex method (such as RNET [Gri79]), by specializations of the network simplex method (such as the code by Barr et al. [Bar77]), by the out-of-kilter method, or by various implementations of the Hungarian method. Computational results have been reported by Hatch [Hat75] who compares the Hungarian method to a specialized version of the out-of-kilter method. These comparisons indicate that, for the problems tested, the out-of-kilter method was faster than the Hungarian method. Glover and Klingman [Glo80] report that their specialized network simplex code is slightly faster than the specialized out-of-kilter code of Hatch. Finally, Bertsekas indicates that his assignment code (which switches to the Hungarian method after a threshold value is reached) is faster than the Barr et al. code on dense problems. Specializations of Tomizawa's transportation algorithm [Tom72] to the assignment and bottleneck assignment problem were proposed by Derigs and Zimmermann [Der78] and FORTRAN programs given in [Bur80].

Our computational experience with assignment problems of up to 400 nodes indicates that the network simplex code RNET [Gri79] runs at least as fast as any other unspecialized network simplex code. We suspect that specialization of RNET along the lines of [Bar77] would improve its performance further.

It is almost certain that further comparative testing of the most promising algorithms will reveal not only the computational superiorities of each algorithm, but will indicate which classes of problems could most benefit by the application of specific data structures and specialized algorithms. It will thus be possible to construct "hybrid" methods, by switching from one method to another, at appropriately chosen instances in the computation. The algorithm reported by Bertsekas is a case in point. Other examples of hybrid implementations (for the traveling salesman problem) are given by Golden et al. [Gol80].

5.3 Nonbipartite Matching Problems:

Two problems are of interest. The Cardinality Matching problem seeks a maximum number of pairwise disjoint edges of a given loopless graph. The weighted matching problem seeks a matching of total minimum weight for a given network with edge weights [Edm73], [Law76]. The perfect (or complete) versions of these problems (edges in the matching cover all nodes of the given network) are also of interest.
The cardinality matching problem was first studied by Berge [Ber57] (also see Norman and Rabin [Nor59]) who proved that a matching has maximum cardinality if and only if it does not admit an augmenting path (connecting two nodes not incident to any matching edges, and consisting alternately of a matching edge and a non-matching edge.) Edmonds gave an $O(n^4)$ algorithm (referred to as the blossom algorithms), later improved by the use of a "labeling procedure" (which allows backtracking through nested "blossoms") by Lawler [Law76] to $O(n^3)$ and implemented by Gabow [Gab76]. Gabow also generated a class of "worst-case" graphs for this problem. A FORTRAN implementation which uses a heuristically obtained initial matching is found in [Bur80]. This program has space complexity of $8n^2e$. Other labeling procedures have been given by Witzgall and Zahn [Wit65], Brilliski [Bal69], and Even and Kariv [Eve75A].

The weighted matching problem was first solved by Edmonds [Edm65A] and elaborated by E.L. Johnson [Joh65], L.L. White [Whi67], and Edmonds and Johnson [Edm70]. Edmond's original algorithm is of $O(n^4)$ complexity although it can be improved to $O(n^3)$ (see e.g. Lawler [Law76]). Derigs [Der79] proposed and programmed a shortest augmenting path algorithm which is a modification of the labeling and blossom shrinking procedure of Edmonds and Johnson [Edm73], and which uses a heuristic procedure to obtain an initial matching. A primal method is given by Cunningham and Harsh [Cun78]. Tobin [Tob75] develops conditions which relate the existence of negative and nonpositive simple cycles in an undirected network with edge weights to weighted perfect matchings on a derived network [Tob75].

Derigs [Der78A] shows that the bottleneck matching problem is equivalent to the symmetric bottleneck assignment problem, and has specialized the above weighted matching algorithm to the bottleneck case [Bur80].

6.0 TOUR PROBLEMS:

Of practical importance are the construction of Euler tours, Chinese postman tours, Hamiltonian circuits, and the traveling salesman problem.

The Chinese postman [Mei62] [Edm65] and the Euler tour problems are intimately related. Edmonds and Johnson [Edm73] discuss this relationship in terms of matching theory. A postman tour, in a given connected network with nonnegative edge weights, traverses every edge at least once. An Euler tour traverses every edge exactly once. The Chinese postman problem seeks a postman tour which minimizes its total weight. The problem was first introduced and solved by Mei Ko Kwan [Mei62]. Clearly, if one can construct an Euler tour for the given problem, then the tour also solves the Chinese postman problem. However, as Euler proved, an Euler tour exists if and only if each node of the graph is incident to an even number of edges. If this is not the case, one can appropriately add edges to the graph so that the resulting multigraph attains this property. Thus, the problem may be viewed as determining an efficient procedure for creating such a multigraph. The first polynomial-time algorithm of $O(n^3)$ proposed by Edmonds and Johnson [Edm73] obtains the set of edges which must be duplicated by a labeling technique which performs a shortest path computation over the nodes of odd degree and a weighted matching computation. The solution to the problem is then obtained by constructing an Euler tour in the multigraph, and such a method ("next-node algorithm") is given in [Edm73].
A Hamiltonian circuit is a circuit which passes through each node of a given graph exactly once. The traveling salesman problem seeks to find a Hamiltonian circuit of minimum total weight in a given network with edge weights. These problems remain NP-complete [Gar79]. Applications of the traveling salesman problem are abundant as are exact and approximate methods proposed for its solution. In its present form, this bibliography does not list all of these references. Bellmore and Nemhauser survey methods of solution for the traveling salesmen problem until 1968 [Bel68]. Christofides reviews Hamiltonian circuits and the traveling salesman problem, and applications (chapter 10 in [Chr76], also see [Chr76A]). Golden et al. [Gol80] provide a unified study of heuristics and computational results for the traveling salesman problem. The most recent survey for the traveling salesman problem is Hoffman et al. [Hof81]. A very small sample of other papers in this field is: Dantzig et al. [Dan54] [Dan59], Little et al. [Lit73], Held and Karp [Hei70] [Hei71], Lin and Kernighan [Lin73], Bellmore and Hong [Bel74], Lin [Lin75], Lenstra and Rinnoy Kan [Len75], Miliotis [Mil76], Grotschel and Padberg [Gro79], Crowder and Padberg [Cro80].

7.0 LINEAR PROGRAMMING PROBLEMS:

Many of the problems discussed in this section can be formulated as linear programs. Simplex-based methods for more complicated network problems (see below) require the pivoting services of an LP code. It is thus necessary to provide such facilities to allow the study, experimentation and implementation of these algorithms. The discussion here is an insignificant sample of the literature on linear programming. Among the many books which discuss the subject are Dantzig [Dan63], Gass [Gas75], Bradley et al. [Bra77A], Bazarra and Jarvis [Baz77].

Although not a "good" algorithm (in the sense of Edmonds [Edm65B], also see the examples by Klee and Minty [Kle70] (also modified by Chvatil), and Goldfarb and Sit [Gol79A]), the wide acceptance of the simplex method is generally attributed to its computational efficiency in solving large, practical linear programming (LP) problems. Nevertheless, straightforward implementations of the standard simplex method are impractical outside the classroom. In addition to its efficiency, factors which contributed to the success of the method include: sophisticated basis handling techniques which take advantage of sparsity (e.g. Dantzig et al. [Dan53], [Dan53A], Hellerman and Barick [Hei72], Forrest and Tomlin [For72]), and Land and Powell [Lan73], Powell [Pow75] [Pow80]), and which perform computations in a numerically stable fashion (e.g. Bartels and Golub [Bar69], Saunders [Sau70], Gustavson [Gus72]), the use of scaling algorithms (e.g. Tomlin [Tom75]), the provision of better pricing schemes (Harris [Har73], Goldfarb and Reid [Gol77B]), computer speed and storage tradeoffs (e.g. supersparsity, Kalan [Kal71]), the provision of a "convenient" environment for problem solving and problem oriented languages (e.g. Eisenstadt [Eis70], Slate and Spielberg [Sla78]), and extensive data management services (see e.g. Orchard-Hays [Orc68], Beale [Bee75], White [Bee75], Whi72, Benichou et al. [Ben76], Elam et al. [Elam79], Crowder et al. [Cro80], Suhl [Suh80], and others). Such advances have proved indispensable in the evolution of large mathematical programming systems.
The primal, dual, and primal-dual simplex methods are not the only algorithms which have been proposed for solving linear programs. Recently, great attention has been paid to a space dilatation method due to Schorr and Khachiyan (referred to as Khachiyan’s method, the ellipsoid method, the Russian method, etc.) which can be shown to have polynomial time complexity. A bibliography on the subject is given by P. Wolfe [Wol80], and a survey by Bland, Gelfarib and Todd [Bia80]. Computational results include with modified versions of the method include those given in [Ros79], [Wii80] and [Gri80]. Computational experience indicates that the ellipsoid method is not competitive with the simplex method for practical problems. An entirely different approach, which employs an SOR type computation has been proposed by Mangasarian [Man79].

8.0 OTHER PROBLEMS:

There are many other combinatorial and network optimization problems which are of practical interest. Such problems include spanning arborescences, capacitated MTSs, the optimal communication spanning tree problem, degree constrained subgraphs, shortest paths in undirected networks with edge weights of arbitrary sign, optimum routing problems, Steiner trees, reliability of flow networks, etc. See e.g. Lawler [Law76], Garey and Johnson [Gar79], Camerini et al. [Cam79] [Cam80], Gilbert and Pollack [Gil68], Cockayne [Coc70], Hakimi [Hak71], Carra [Car71], Dreyfus and Wagner [Dre72], Zadeh [Zad72A] [Zad73], Soukup [Sou73], Yaged [Yag73], Murchland [Mur73], Boyce et al. [Boy73], Ratcliff et al. [Rat75], Bird [Bir76], Lenstra and Rinnooy Kan [Len76], Aneja and Vemuganti [Ane80], Shiloach [Shi80], Wong [Won80], Denardo and Fox [Den80], and others.

Problem solving in network optimization always involves the use of data and graph manipulation algorithms such as sorting, data structure transformations, node and edge connectivitiy, traversal and search methods (Depth First, Breadth First), identification of connected components, simple paths, etc. (see books listed above and papers e.g. Tarjan [Tar72], Hopcroft and Tarjan [Hop73A], Even [Eve73A], Christofides [Chr75], Tarjan [Tar74], Read and Tarjan [Rea75], Even and Tarjan [Eve75], Fratta and Montanari [Fra75], Galil [Gal80], and others.)

Generalizations of MCFN problems which have found wide applicability in practice are MCFN with (linear) side constraints, multicommodity MCFN problems [Gri72], MCFN problems with convex cost functions [Hu 66] [Mey80] [Pas80], concave cost functions [Gal180A], and others. Here, we discuss two of these areas, MCFN problems with additional linear constraints and multicommodity network flow problems, from the viewpoint of partitioning and basis factorization schemes.

8.1 Partitioning Methods:

The most promising methods for solving multicommodity network flow problems and MCFN problems with additional constraints are specializations of basis partitioning or basis factorization schemes. These are methods which take advantage of the commonly encountered "block diagonal" structure in large linear programs. Kron [Kro56] introduced an approach called "K-partitioning for linear systems and Dantzig [Dan55] [Dan62] and Dantzig et al. [Dan56] outlined the
main approach of basis factorization and partitioning for the simplex method.
(Also see [Gra73] [McB73]). Partitioning for the dual simplex method was first
proposed and implemented by Rosen [Ros64] who also developed a method for
structured nonlinear programs with coupling variables [Ros65]. The latter
specializes to a primal simplex-based method for linear problems.

Additional approaches for problems with only coupling rows are (primal)
simplex-based [Hee65], [Kau65], [Ben66], [Mul73], [GI80], dual simplex and
parametric [Ros64], [Geo68], [Rit67A], [Orc68], [Orc75], [Osh73], [Orc75], (also
see [Geo67] and [Las70]), primal-dual [Bal66], [Kno73], and other strategies such
as partial block pricing proposed by Markowitz (see [Dan63A] and [Win74]).
Methods proposed for treating problems with only coupling variables include
contributions by Rosen [Ros64], Baale [Bea63] [Ben77], and Gass [Gas66],
[Ho79], [Wol77], and others. Techniques for solving problems with both
coupling rows and coupling columns are Ritter's dual method [Rit67], [Gri69] and
the primal methods by Weber and White [Web68], Hartman and Lasdon [Har70] and
Winkler [Win74]. Methods for solving large problems with specialized blocks
include generalized upper bounding [Dan63A], [Dan67], [Gri73], [Bra73], [Mul73],
[Glo79] (the algorithm of the latter is a special case of [Gri73]),
transportation and network flow subproblems [Gri68], [Gri73], [Che77A], [Glo77],
and multi-item scheduling [Dzi63] [Gri66], [Las70], and others. We have not
included references for the important area arising from ramifications and
specializations of the Dantzig-Wolfe "decomposition principle" [Dan60A]
[Gom61A], and Bender's decomposition [Ben62] [Geo73].

An unattractive characteristic of the above partitioning schemes for general
diagonal blocks is the unusually high density of the "working basis", albeit of
smaller size than the overall problem. Nevertheless, experience indicates that
substantial savings can be obtained when the blocks exhibit further structure
and when the overall system is "weakly coupled" [Gri80C].

All known partitioning algorithms can be viewed as pricing strategies of the
primal, dual, or primal-dual-simplex methods [Gri73]. Performance estimates for
such algorithms can be obtained by simply exercising existing LP codes with
these pricing strategies, and by counting the types of pivot steps executed, and
by using derived or experimentally obtained computational bounds for each type
of pivot that might occur [Gri73A]. There are but three classes of pivoting
operations. When there are both coupling rows and columns, two additional
operations (increase/decrease the size of the working basis) are also required.
The pivot position depends on the type of simplex method that is adopted.

The control of the size of the working basis has been studied in [Gri69],
[Har70], [Gri72] and [Win74]. Experimental evidence suggests that when the
subproblems are "easy" to handle, it is preferable to maintain the working basis
as small as possible [Glo77A], [Har70]. Further reduction of the working basis
can be achieved in dual partitioning methods by instituting a relaxation
strategy which removes most of the coupling rows and retains a "working set"
(see [Gri72], [Pow80]).

The amount of work per pivot is related to the pivot position, which cannot be
controlled by the choice of the entering column in the (primal) simplex method,
except perhaps by simply avoiding the coupling columns for as long as possible.
In the dual simplex method, there is considerable flexibility in selecting the
more attractive rows during the early phases of the algorithm, thus allowing the
postponement of the more expensive pivots with the hope that only a few may have
to be performed [Geo68], [Gri73]. Rosen's technique [Ros64] can be interpreted in this light. The same comment applies to other proposals [Rit67], [Gri69], [Gri72], [Gri68], [Gri73], [Orc68], [Orc73], [Orc75].

When the diagonal blocks are networks, an initial set of subproblem bases is obtained quickly, and the block inverses can be inferred from the basis trees [Gri73] [Tru77]. Subsequently, each partitioning iteration requires that certain expressions involving the subproblem basis inverses be evaluated. This is normally done by using the basis tree list functions to implement the pre- and post-multiplication operations with the subproblem inverses. Major improvements in execution times result because of this specialization. Our implementation [GriB0C] which is based on RNET [Gri79], uses a dual pricing strategy to price from two subsets of rows. One of these sets contains all the rows (nodes) of the subproblem. For this set, primal feasibility is restored frequently. The other set corresponds to the rows of the working basis.

Several strategies have been tried to assess the number of dual working-basis pivots per subproblem pivot. For moderately large problems the strategy which returns to the subproblem every 0.5m working-basis pivots appears to be best [Hsu80].

8.2 Multicommodity Maximum Flow And Minimum Cost Problems:

Models involving multicommodity flows arise from numerous applications in such areas as urban traffic assignment, communication network design [Go171] [Fra72], [Whi72], school desegregation [Cla68], routing freight cars, tanker scheduling, international trade [Go75], sanitary engineering [Pan76], and others. Since the first algorithm by Ford and Fulkerson [For58] for the edge-chain formulation of the multicommodity maximum flow problem, numerous proposals have been made in three general categories: column generation (also referred to as price-directive or Dantzig-Wolfe [Dan60A] decomposition), resource-directive decomposition and basis partitioning.

In the first category are contributions by Ford and Fulkerson [For58], Tomlin [Tom66], Bradley [Bra66], Jarvis [Jar69], Wollmer [Wo169] [Wo172], Weigel and Cremeans [We172], Swoveland [Swo71] [Swo73], and others.

In the second category, contributions which describe the general price-directive decomposition approach are: Sakarovitch [Sac66] [Sac73], Geoffrion [Geo70], Grinold [Gri80] and Gratz and Steinlitz [Gra72]. Approaches for handling the "master" problem are discussed by Geoffrion [Geo70], and the "tangential approximation" procedure is used by Swoveland [Swo71] for the multicommodity flow problem. The subgradient optimization procedure proposed by Held, Wolfe and Crowder [Hel74], has been applied by Kennington and Shalaby [Ken77A] and Assad [Asa76].

In the third category are methods by Saligal [Sal67], Hartman and Lasdon [Har72], Kennington [Ken77] and McCallum [McC77] which apply the generalized upper bounding technique. The method by Maier [Mai74] uses LU decomposition along the same lines. A dual partitioning scheme (a la Rosen's Primal Partitioning Method) with a relaxation scheme was proposed by Grigoriadis and White [Gri72] [Gri74] and Powell [Pow75] [Pow80]. It is possible that the primal-dual method proposed by Jewell [Jew69], when modified to operate on basic solutions (as suggested by Johnson [Joh66]), can also be classified with these methods. All
basis partitioning methods for block angular systems differ in the manner in
which the pivot is selected (primal, dual or primal-dual simplex criteria) and
are identical in terms of the data structures and pivot implementation
procedures [Gri73].

Excellent reviews of most methods can be found in [Ass78], [Ken78], [Ken80], and
computational experience found in [Gri74], [Ken77A], [A1l80]. For a review of
special cases, such as single-terminal multicommodity flows and two-commodity
flows see [Rot68], [Rot66], [Rot67], [Hu 63], [Hu 69A]. The question of
integrality of multicommodity flows has been examined by Evans [Eva79].
REFERENCES


[Che79] Cheston, G.A., "Graph Update Algorithms and Their Application to Distance Matrix Correction", Report 79-1, Dept. of Computational Science, Univ. of Saskatchewan, Saskatoon, Canada, 1979.


