Global Search Algorithms for Minimum Concave-Cost Network Flow Problems

G. M. GUISEWITE[1] and P. M. PARDALOS[2]
[1]*HRB Systems, State College, PA, U.S.A.*; [2]*University of Florida, Gainesville, FL 32611, U.S.A.*

(Accepted: 16 October 1991)

Abstract. We present algorithms for the single-source uncapacitated version of the minimum concave cost network flow problem. Each algorithm exploits the fact that an extreme feasible solution corresponds to a sub-tree of the original network. A global search heuristic based on random extreme feasible initial solutions and local search is developed. The algorithm is used to evaluate the complexity of the randomly generated test problems. An exact global search algorithm is developed, based on enumerative search of rooted subtrees. This exact technique is extended to bound the search based on cost properties and linear underestimation. The technique is accelerated by exploiting the network structure.

Key words. Concave-cost network flow, uncapacitated, single-source, global optimization, random search algorithms.

1. Introduction

1.1. GENERAL PROBLEM

The single-source uncapacitated (SSU) version of the minimum concave-cost network flow problem (MCNFP) requires establishing a minimum cost flow from a single generating source to a set of sinks, through a directed network. All arcs are uncapacitated, indicating that the entire source flow can pass through any arc. The SSU MCNFP can be stated formally as follows:

Given a directed graph $G = (N_G, A_G)$ consisting of a set N_G of n nodes and a set A_G of m ordered pairs of distinct nodes called arcs, coupled with an n-vector (demand vector) $d = (d_i)$ with $d_1 < 0$ and $d_i \geq 0$, $i = 2, \ldots, n$, and a concave cost function for each arc, $c_{ij}(x_{ij})$, then solve

$$\text{global min} \sum_{(i,j) \in A_G} c_{ij}(x_{ij})$$

subject to

$$\sum_{(k,i) \in A_G} x_{ki} - \sum_{(i,k) \in A_G} x_{ik} = d_i\,, \quad \forall i \in N_G \tag{1}$$

and

$$0 \leq x_{ij}\,, \quad \forall (i,j) \in A_G\,. \tag{2}$$

All constraints and demands are assumed to be integral. The requirement that only $d_1 < 0$ corresponds to the single-source case. The lack of an upper bound for the x_{ij} gives rise to the uncapacitated case.

The SSU MCNFP is a concave optimization problem over a convex polyhedron. This indicates that if a finite optimal solution exists, then there exists an extreme point of the feasible domain that is optimal [2]. The concave case differs from the linear case in that a local optimum need not be a global optimum. For the SSU case an extreme flow (corresponding to an extreme point) is a tree [23]. The leaves of the solution tree correspond to a subset of the sink nodes. The integral constraints and demands give rise to extreme flows of integral value.

A SSU MCNFP has a finite optimal solution if it contains no negative cost cycles, and all sinks are reachable from the source (i.e., there exists a directed path from the source to each sink). The latter requirement is necessary for the existence of a feasible flow. The presence of a negative cost cycle would imply an unbounded negative cost solution; the absence of such a cycle guarantees a finite solution [13]. Applications that give rise to SSU MCNFP are discussed in [10].

1.2. ADDITIONAL CONSTRAINTS

We consider in this paper cases of the SSU MCNFP with arc flow costs that are non-negative, non-decreasing and concave. This property of objective functions accurately reflects cost functions for models of real world problems in areas such as production planning and transportation analysis. Concave costs in these settings correspond to economies of scale. In a production setting, decreasing concave arc cost functions would exclude the influence of demand on production.

This limitation on arc costs facilitates the computation of local optima for SSU MCNFP. Checking for a local optimum, for the constrained costs, can be achieved by solving a series of single-source shortest weighted path problems. In addition, if a solution exists for a problem, then a bounded solution exists. For general costs, linear network programming problems would have to be solved, and unbounded solutions could result. This entails more processing, and complicates the generation of test problems.

1.3. LOCAL SEARCH ALGORITHMS

In Section 2 we introduce a global search heuristic based on random extreme feasible solutions and local search. A solution X to a SSU MCNFP is locally optimal if no better solution exists in a specified neighborhood of X. Varying the definition of neighborhood results in different conditions for a local optimum. The standard marginal definition of local optimality defines a neighborhood of X to be

$$N_\varepsilon(X) = \{X' | X' \text{ satisfies (1) and (2) and } \|X - X'\| < \varepsilon\}$$

for a specified vector norm and $\varepsilon > 0$. Local search based on N_ε for concave optimization is explored by Minoux [14] and Yaged [22]. For the single commodity case with fixed-charge arc costs, all extreme points are local optima. This led to the development of the following generalized definition of neighborhood by Gallo and Sodini [7]:

$$N_{AEF}(X) = \{X' | X' \text{ satisfies (1) and (2) and } X' \text{ is an adjacent extreme flow to } X\}\,.$$

Here, X' is an adjacent extreme flow to an extreme solution X if X' is an extreme flow, and $X \cup X'$ contains a single undirected cycle.

An even more relaxed definition of neighborhood is the following:

$$N_{AF}(X) = \{X' | X' \text{ satisfies (1) and (2) and } X' \text{ is an adjacent flow to } X\}$$

Here, X' is an adjacent flow to an extreme solution X if X' results from rerouting a single sub-path of flow within X. This concept of neighborhood was developed by Guisewite and Pardalos [9] for single-source problems, and independently by Plasil and Chlebnican [19] for the multi-commodity case. We employ both N_{AEF} and N_{AF} neighborhoods in our global heuristic. Results in [9] indicate that the choice of neighborhood for local search affects the processing time for convergence. In Section 2 we demonstrate that the choice of neighborhood also affects the processing accuracy of the global search heuristic.

1.4. TEST PROBLEMS

Test networks are generated in a random fashion. Arcs are generated by computing two random integers uniformly distributed in $[1, 2, \ldots, n]$, where n is the number of nodes in the network. Duplicate arcs and arcs of the form (i, i) are discarded. After the specified number of arcs is successfully generated, the resulting network is tested for connectivity by solving a single source shortest path problem. If the connectivity is suitably high, then cost functions are generated for each arc. Unless stated otherwise, each cost function is of the form $\alpha_{ij} x_{ij}^{\beta_{ij}}$, where the a_{ij} are uniformly distributed in $[1, 2, \ldots, 100]$ and the β_{ij} are uniformly distributed in $[.1, .2, \ldots, 1]$. The test generator is implemented so that if two problems contain the same number of nodes, and the same random number seed, but have a different number of arcs, then the smaller network generated is a subset of the larger network.

1.5. PROCESSING EQUIPMENT

The algorithms developed in subsequent sections were implemented on one or multiple microprocessors. Our processing system consists of one to twenty Transputer T800s. The Transputer is a microprocessor developed under the European Espirit project, and is designed to facilitate parallel processing. Each 20 MHz T800 consists of a 10 MIPS fixed point processor, a 1.5 MFLOPS floatingpoint co-processor, a 4 KByte cache, and 4 DMA I/O processors, all on a single chip. Our system includes 1 MByte of memory per processor. Experience indicates that 3 to 5 MIPS are achievable by each processor for large general processing applications. Four processors are configured in a pipeline on a single board, with the remaining DMA links connected to the board edge through a cross-bar switch.

1.6. OVERVIEW

Efficient algorithms for the SSU MCNFP have been found only for a small set of structured problems. This is not surprising as the general global search problem for the SSU MCNFP is known to be *NP*-hard [8], [9], [13]. An overview of existing techniques for general MCNFP can be found in [10]. Due to the complexity of global search, numerous techniques based on local search have been developed. In Section 2, a global search heuristic based on both random and local search is presented and analyzed. In Section 3 of this paper an exact global search technique for the SSU MCNFP is developed.

2. Random Search

Existing techniques to locate global optima for SSU MCNFP can, in the worst case, require time exponential in the size of the problem. This indicates that computing exact solutions to large problems in a reasonable amount of time may not be achievable. A global search technique should offer an increased probability that an improved solution is found, as elapsed processing time increases. Local search provides a means to improve upon a given extreme feasible solution, but offers no guarantee on the nearness to a global optimum. In [9] it is noted that the choice of initial solutions for SSU MCNFP has little effect, in the average case, on the value of the local optimum computed. This result was observed for a range of greedy and non-greedy approaches to generating basic feasible solutions for SSU MCNFP.

Exploiting the above observations results in a global search heuristic for SSU MCNFP. Applying local search to randomly generated extreme feasible solutions should generate a range of candidate solutions. As more initial solutions are processed, the probability of obtaining a global optimum increases. Even if the overall global solution is not located, the best local optimum obtained should provide a "good" approximation to the global optimum.

In the following sections a global search heuristic is developed and analyzed. In Section 2.1 an algorithm to generate random extreme feasible solutions is provided. In Section 2.2 this algorithm is incorporated into a global search heuristic. The probabilistic aspects of the heuristic are presented. In Section 2.3 processing results for randomly generated test problems are presented. In Section 2.4 the global search heuristic is used to analyze the complexity of test cases, based on the arc cost functions employed.

2.1. GENERATING RANDOM INITIAL SOLUTIONS

Any extreme feasible solution to a SSU MCNFP is a tree, rooted at the sink, with leaves that correspond to a subset of the sink nodes. Randomly selecting arcs for a solution tree is inefficient, as disconnected arcs would have to be detected and removed. A substantial improvement, in terms of processing time, can be achieved by randomly generating paths from the source to each sink. Starting the path generation from the source could result in a non-extreme solution (i.e., intersecting paths). Also, decisions as to where to start each path would be required. Generating paths backwards from each sink simplifies the process. A backward path terminates whenever a previously visited node, from a complete path to another sink, is encountered. The basic algorithm is presented in Figure 1.

Generating a backward path from t_i to a visited node presents some difficulties. Figure 2 depicts a situation where the process can dead-end due to a cycle. If the generation process generates the arc sequence $(4, t_i)$, $(3, 4)$, $(5, 3)$, $(4, 5)$, then a cycle (and a non-extreme solution) results. This can be avoided by labelling the current search path, *visited*(*node*) = 1, and not permitting a node to be visited twice. If the sequence generated is $(4, t_i)$, $(3, 4)$, $(5, 3)$, $(1, 5)$, $(1, 2)$ then the process (with labelling to avoid cycles) dead-ends at node 2. This case could be handled by maintaining a stack for the current path, popping the stack when a dead-end is encountered, and identifying the remaining candidate nodes reachable (backwards) from the current node by excluding those marked as dead-ends. Another approach would be to avoid dead-ends by continually identifying nodes currently reachable from the source node, in a network with nodes satisfying *visited*(*node*) = 1 removed. An alternate approach was chosen to avoid the overhead in the above techniques. In the event of a detected dead-end, the process restarts at the current sink with a different seed to the random number generator. Although the process could revisit dead-ends, in practice it has been found to be very efficient. The resulting algorithm is presented in Figure 3. The time to generate initial solutions for large test cases (1000 nodes, 5000 arcs, and 50 sinks) averages 0.25 seconds. This is significantly faster than the greedy techniques evaluated in [9].

Processing results for local search applied to random initial solutions and greedy initial solutions are summarized in Figure 4. The results correspond to 10

```
(1) Establish a sink labelling $t_1, t_2, \dots, t_k$
(2) Mark each node except the source as not visited
    $visited(j) = 0,\ \forall j \in A_G - \{S\}$
    $visited(S) = 2$
WHILE $\{\exists i \ni visited(t_i) = 0\}$
    {(3) Randomly generate a backward path $p_{n_0,t_i}$
         from $t_i$ to any node $n_0$ satisfying $visited(n_0) = 2$
     (4) Mark all nodes in $p_{n_0,t_i}$ visited
         $visited(j) = 2,\ \forall j \in p_{n_0,t_i}$}
```

Fig. 1. Generating a random extreme solution.

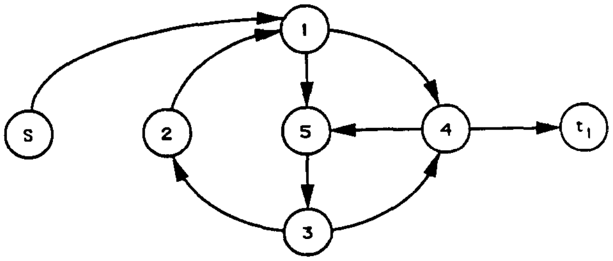


Fig. 2. Example network for random solution generation.

test cases, each with 50 nodes, 250 arcs, and 20 sinks. The results for random search are averaged over 200 random initial solutions for each test case. The rate of convergence for random initial solutions is significantly slower than for the greedy initial solutions. This is due to an increased number of iterations required for convergence. The objective value of the computed solutions generated from local search with random starting solutions is roughly the same (on average).

2.2. RANDOM SEARCH ALGORITHM

Coupling the random initial solution algorithm with a local search algorithm results in the global search heuristic in Figure 5. The algorithm is a variation of one proposed in Pardalos [18] using gradient local search for general concave optimization. The following probabilistic analysis is generalized from a result found in [18]:

Suppose that $V = \{v_1, \ldots, v_l\}$ is the set of all local optima to the current problem. Without loss of generality, assume that the cost of the local optima, $C(v_i)$, satisfy

$$C(v_1) \leq C(v_2) \leq \cdots \leq C(v_l) . \tag{3}$$

```
(1) $current.node = t_i$, $p_{n_0,t_i} = \emptyset$
WHILE $\{visited(current.node) \neq 2\}$
    {(2) Identify candidates for previous node
         $CANDIDATES = \{j | (j, current.node) \in A_G$
                                  AND $visited(j) = 0$ or $2\}$
    (3) IF ($|CANDIDATES| = 0$) THEN
        {$current.node = t_i$
        $visited(j) = 0\ \forall j \in p_{n_0,t_i}$
        $p_{n_0,t_i} = \emptyset$}
        ELSE
        {randomly select $n_{prev} \in CANDIDATES$
         $visited(current.node) = 1$
         $p_{n_0,t_i} = (n_{prev}, current.node) + p_{n_0,t_i}$
         $current.node = n_{prev}$}}
```

Fig. 3. Algorithm to find a random path from t_i to a visited node.

	Processing time (secs)	Number of iterations	Average number of subproblems
Greedy	12.6	6.5	172.4
Random	31.7	16.8	413.6

Fig. 4. Comparison of local search techniques.

Define $S_i = \{v_1, \ldots, v_i\} \subseteq V$, for $i = 0, \ldots, I$. If N extreme feasible solutions are randomly generated, local search is applied to each solution, then the probability that no local optima in S_i is obtained is $P_i(0) = (1 - p_i)^N$, where p_i is the probability that a single randomly generated initial solution has associated local optimum in S_i. More generally, the probability that k local optima in S_i are located, given N random initial solutions is

$$P_i(\xi = k) = \binom{N}{k} p_i^k (1 - p_i)^{N-k}, \quad k = 0, \ldots, N. \tag{4}$$

Here, ξ is a binomially distributed random variable that arises from sampling with replacement from the elements in $S(i)$ and $V - S(i)$.

The value of the p_i depends upon several characteristics of the current problem:

- The structure of the network
- The arc cost functions
- The neighborhood and search algorithm used for local search
- The method of generating "random" initial solutions.

The first three characteristics account for the mapping of extreme feasible solutions to the associated local optima. The last characteristic accounts for the difficulty in generating truly random initial solutions. In fact, the random solution technique described in the previous section could be described as a locally random technique. The process generates random paths by making local decisions based on arcs adjacent to the current node. Alternate initial solution techniques could be based on randomly generating break points (forks in the paths), random selection of nodes, etc.

For local search with neighborhood N_F the number of local optima, I, can be the number of vertices of the constraint polytope. For local search based on neighborhood N_{AEF} or N_{AF}, the number of local optima is less for problems with

```
current.best.cost = BIG
WHILE (a specified time or solution count has not been exceeded)
  {generate a random extreme feasible solution X
  apply local search to X
  IF(new.solution.cost < current.best.cost)
    current.best.cost = new.solution.cost}
```

Fig. 5. Global search heuristic.

vertices of differing objective function value. This is a result of local search being defined in terms of neighboring vertices (extreme feasible solutions), as opposed to neighboring, possibly non-extreme feasible solutions. Still there can exist an exponential number of local optima that are not globally optimal, as is shown by the following example.

The network in Figure 6 corresponds to a network formulation of a 3-satisfiability problem. The 3-SAT problem is known to be NP-complete [8]. The network formulation of 3-SAT is developed in [10]. All arcs have zero cost, except for arcs (V_{ij}, T_{ij}) and (V_{ij}, F_{ij}). These arcs have cost of one if they carry non-zero flow. The following notation is used to establish the existence of 2^k local optima, where k is the number of clauses; recall that x_{ij} denotes the flow on arc (i, j):

$$(x_{V_{ij},T_{ij}} > 0) \wedge (x_{V_{ij},F_{ij}} = 0) \Rightarrow V_{ij} = \mathrm{T}\,(\mathrm{TRUE})$$

$$(x_{V_{ij},T_{ij}} = 0) \wedge (x_{V_{ij},F_{ij}} > 0) \Rightarrow V_{ij} = \mathrm{F}\,(\mathrm{FALSE})$$

$$(x_{V_{ij},T_{ij}} > 0) \wedge (x_{V_{ij},F_{ij}} > 0) \Rightarrow V_{ij} = \mathrm{U}\,(\mathrm{UNDETERMINED})\,.$$

Note that the assignment $V_{i1} = T$, $V_{i2} = T$, $V_{i3} = T$, $i = 1, \ldots, k$, is the unique assignment satisfying the clauses $(V_{i1} \wedge (V_{i2} \vee V_{i3}))$ and $(V_{i2} \wedge (\overline{V_{i1}} \vee V_{i3}))$, $i =$

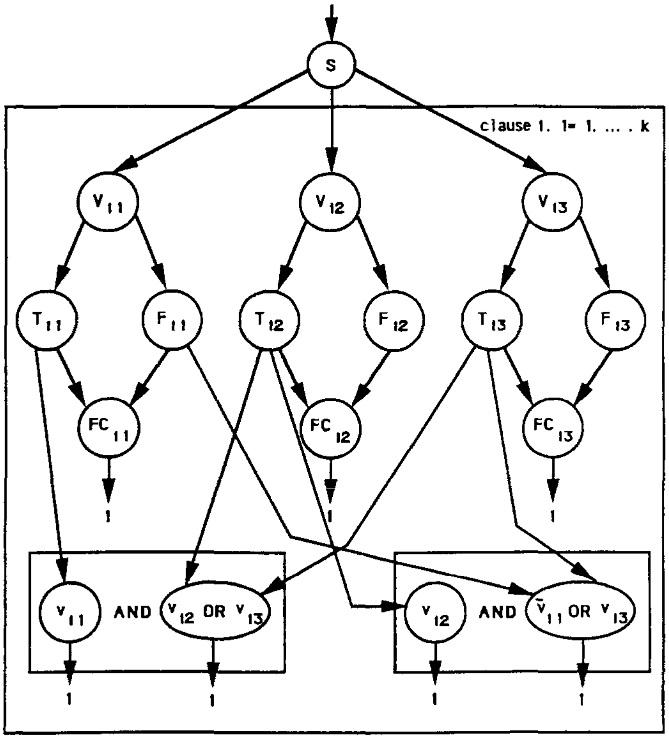


Fig. 6. Network with an exponential number of non-global local optima.

$1, \ldots, k$. Denote this assignment for clause i as $CL_i = 1$. This assignment determines the unique flow that is a global optimum, with cost $3 * k$. The cost per clause is 3. Note, also, that the assignment $V_{i1} = U$, $V_{i2} = T$, and $V_{i3} = F$ results in a valid flow at any clause i. Denote this assignment for clause i as $CL_i = 0$. Assignment $CL_i = 0$ results in a local optimum, as at least two flow paths must be rerouted to alter the flow cost (one at V_{i3} and one at V_{i1}). The cost per clause for this assignment is 4. Combining the above observations, there exist at least 2^k local optima. A local optimum exists for each of the assignments $CL_i = 0$ or 1, $i = 1, \ldots, k$. Only one of these local optima is a global optimum.

Based on the fact that there can exist an exponential number of paths between two nodes in a general network, and the fact that N_{AF} search considers all such paths during one iteration, an exponential number of vertices may be fathomed in one iteration of N_{AF} based local search. In the above analysis, this would imply that a significantly smaller N would be required to find a "good" approximation to a global optimum.

Some qualification is necessary in defining "good". The desired optimal cost is $f(v_1)$. "Good" in the above sense indicates that a solution x is found in the range $f(v_1) \leq f(x) \leq f(v_j)$ for small j. Even for small j, $f(v_j) - f(v_1)$ could be large. It is, also, possible that a large number of vertices result in local search converging to a poor local optimum.

Regardless of the p_i for a specific problem, $P_i(0)$ decreases as the number of initial solutions, N, increases. This indicates that the algorithm expects to make progress as N increases. However, for large k, the size of N required to find the global optimum with large probability can be prohibitive. Results for moderately sized networks, as presented in the following sections, indicate that the heuristic performs well for randomly generated test cases.

2.3. PROCESSING RESULTS

The global search heuristic was implemented on a multiple micro-processor system. An asynchronous approach, in which each processor executes independently on a random starting solution, was used. The main routine grants permission to start a new problem based on the overall number of problems processed up to current point in time.

Empirical results for the global search heuristics, presented in Figures 7–9, are promising for the following reasons. In Figure 7, results for 100 random test cases for each problem size indicate that the heuristic, with 200 starting solutions, locates the optimal solution for problems with known solutions. In all cases, the heuristic finds the best known solution, as compared to local search with greedy initial solutions. In Figure 8, results indicate that multiple processors can be effectively utilized, even in the case of small N. Figure 9 demonstrates that the best overall solution is, in general, found early in the search, and that the local optima tend to be significantly better than the random initial solutions.

	Problem size (nodes/arcs/sinks)		
	10/50/5	25/125/10	50/250/20
Greedy	84%	71%	54%
Heuristic	100%	100%	100%
Enumeration	100%	X	X

Fig. 7. % of cases that the best known solution was located.

2.4. EVALUATING THE COMPLEXITY OF TEST PROBLEMS

The global search heuristic cannot determine if a global optimum is found. However, results for multiple initial random solutions can be used to evaluate the relative complexity of test problems for SSU MCNFP, and to compare alternate local search algorithms. The measures used in this process are

- Best solution detected.
- Frequency of the best solution detected (approximation of p_i).
- Number of distinct local optima detected.

The general idea is that a problem should be more difficult if it gives rise to a large number of distinct local optima, or if the probability of detecting a global optimum is low.

Examples of assumed easy and difficult problem characteristics are presented in Figures 10–11. Figure 10 represents a case with a large number of local optima, but a high probability of locating the (assumed) global optimum. Figure 11 corresponds to the worst case, a large number of computed local optima and a low frequency for the best detected local optimum. Each figure presents the relative frequency of computed objective values for 200 local optima.

The effects of varying the concavity of the arc cost functions on problem complexity are summarized in Figures 12–13. The test cases are generated with

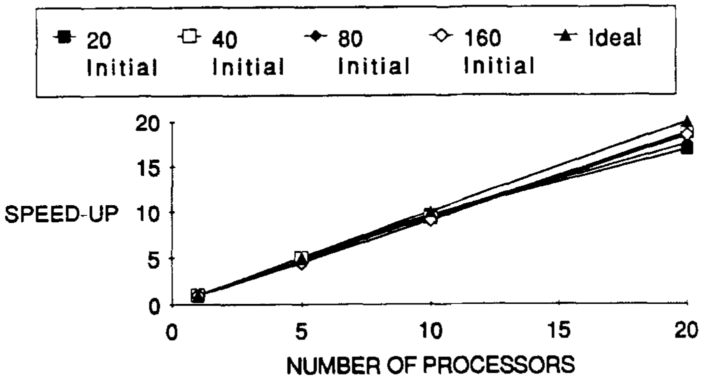
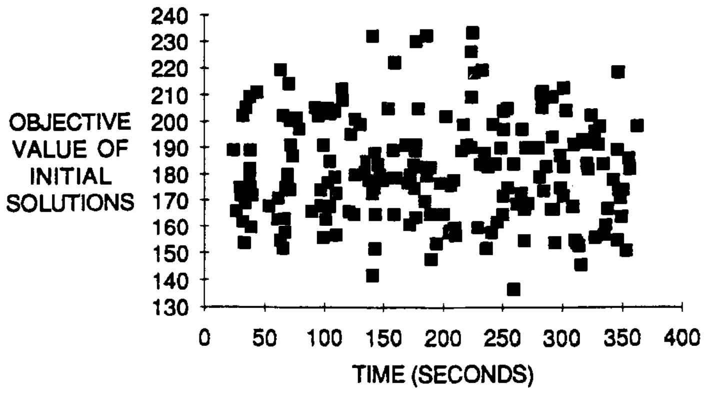


Fig. 8. Multiprocessor performance of the global search heuristic.



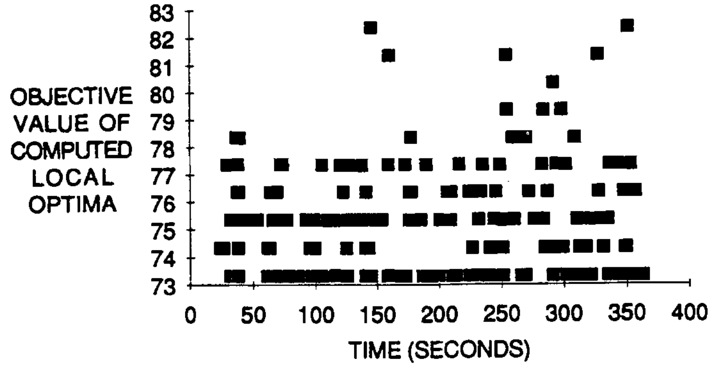


Fig. 9. Solutions over time, initial solutions and computed local optima.

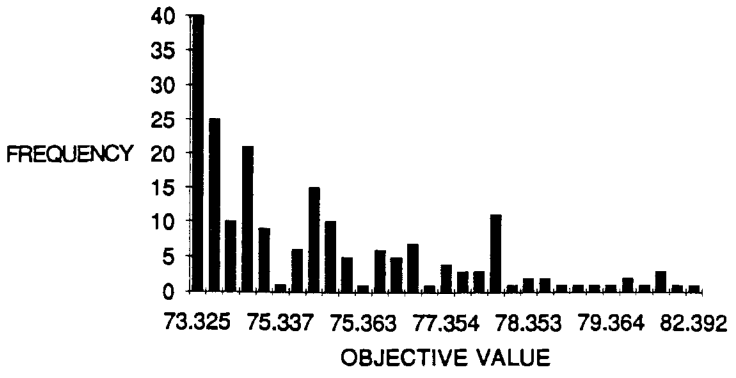


Fig. 10. Characteristics of an easy problem, computed local optima.

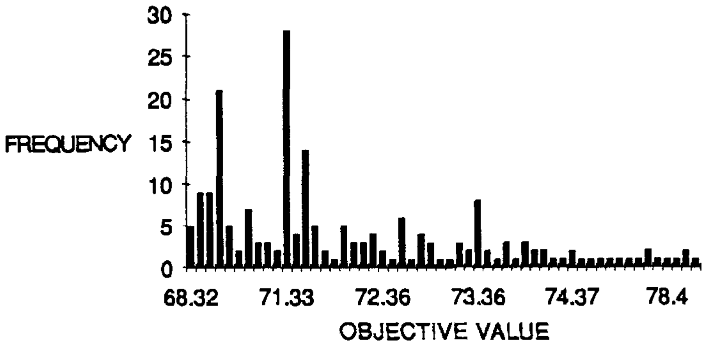


Fig. 11. Characteristics of a difficult problem, computed local optima.

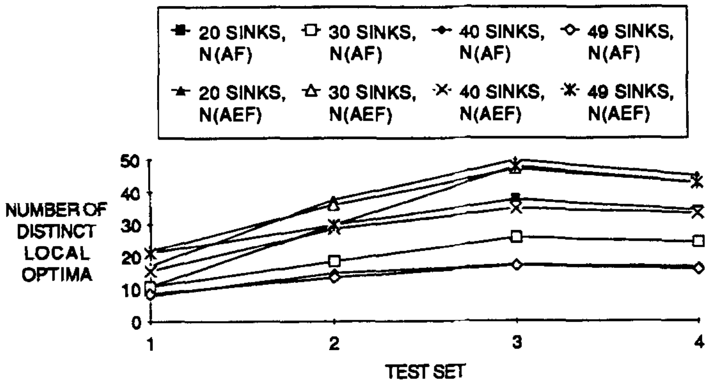


Fig. 12. Number of distinct local optima detected (varying β).

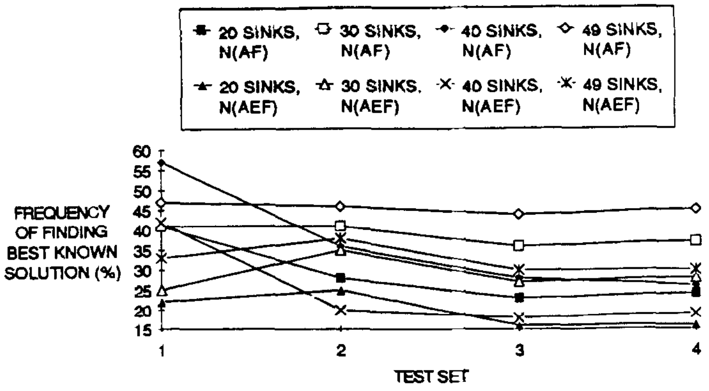


Fig. 13. Frequency of best known solutions obtained (varying β).

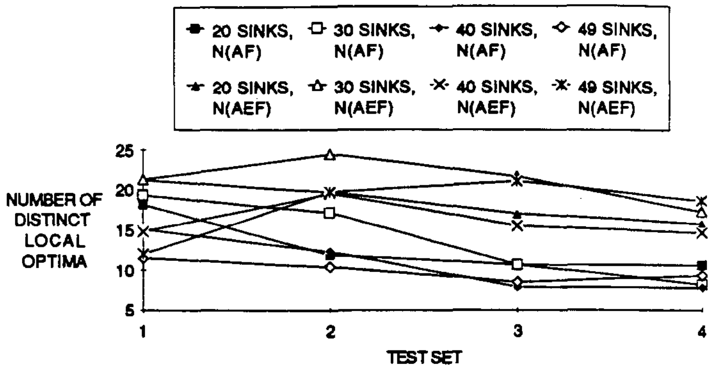


Fig. 14. Number of distinct local optima detected (varying α).

random arcs, and randomly generated arc costs of the form $\alpha_{ij} * x_{ij}^{\beta_{ij}}$. For each test set, α_{ij} is sampled uniformly from $\{1, 2, \ldots, 10\}$. Test set 1 samples the β_{ij} from $\{.1, .2, \ldots, 1\}$. Test set 2 assigns all β_{ij} to 0.1. Test set 3 assigns all β_{ij} to 0.01. Test set 4 assigns all β_{ij} to 0.001. Each test set corresponds to 10 distinct test case, each applied to 200 randomly generated starting solution. Figure 12 compares the average number of distinct local optima detected, for N_{AEF} based local search. Figure 13 compares the average number of times the best known solution was detected, for N_{AF} and N_{AEF} based search.

Results indicate that the number of distinct local optima increases as the β_{ij} decrease, for the majority of the cases. The frequency of detecting the best known solution remains nearly constant. Also, local search based on the N_{AF} neighborhood gives rise to fewer distinct local optima, and detects the best known solution a higher frequency of the time.

Figures 14–15 provide similar results for test cases in which the α_{ij} are varied. Test set 1 assigns all α_{ij} to 1. Test set 2 samples all α_{ij} from $\{1, \ldots, 10\}$. Test set 3 samples all α_{ij} from $\{1, \ldots, 100\}$. Test set 4 samples all α_{ij} from $\{1, \ldots, 1000\}$.

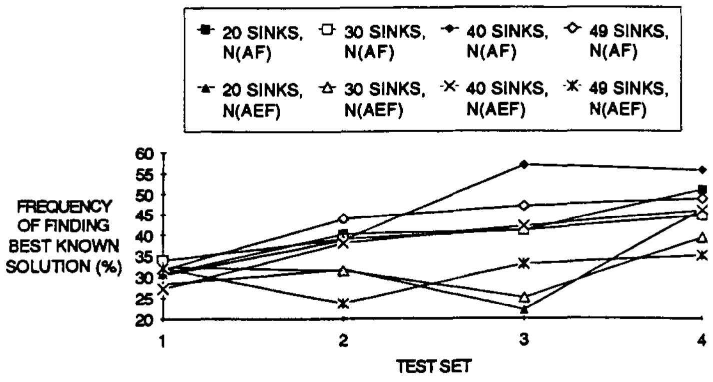


Fig. 15. Frequency of best known solutions obtained (varying α).

In the majority of the N_{AF} cases, the number of distinct solutions decreases, and the frequency that the best known solution is detected increases, as the variability of the α_{ij} increases. For the N_{AEF} search, no distinguishable pattern is evident. As in the βij test cases, the N_{AF} neighborhood gives rise to fewer distinct local optima, and detects the best known solution a higher frequency of the time.

3. Exact Global Search

In this section, an exact global search algorithm is presented for SSU MCNFP. The full algorithm is based on branch-and-bound enumeration of extreme feasible solutions. The basic enumeration subset of the algorithm exploits the fact that extreme feasible solutions correspond to sub-trees in the network. These sub-trees are constructed by establishing a path from each sink to the source node. Bounding the search is achieved using linear underestimation after a path is constructed. This allows the underestimating function to gain efficiency as the search progresses. The basic approach is similar to that of Gallo, Sandi, and Sodini [6]. The approach presented in this section differs in the method of enumerating extreme feasible flows. Also, the effects of initial solution techniques on the branch-and-bound search, and the exploitation of network structure are evaluated here.

Numerous exact algorithms have been proposed for the general MCNFP [10]. Extreme point ranking methods [15], [17] rely on linear underestimation applied to the initial problem. Results in this section demonstrate that the initial underestimation can be quite poor for this class of problems. Some branch-and-bound algorithms for MCNFP perform branching on the arc cost functions to improve the linear underestimation [1], [4], [12], [20], [21]. For these approaches, worst case processing time can exceed a brute-force enumerative approach. Other branch-and-bound approaches, such as in [5], suffer severe performance penalties when degenerate problems are encountered [10]. Dynamic programming solutions to MCNFP require excessive storage requirements [3]. The proposed algorithm can incorporate dynamic programming to efficiently solve subproblems, while limiting the storage requirements.

3.1. THE BASIC ALGORITHM

The basis of the exact global search algorithm is a procedure that systematically enumerates the extreme feasible solutions of the current SSU MCNFP. As noted earlier, these correspond to sub-trees of $G = (N_G, A_G)$ that establish a path from the source to each sink. An efficient approach to enumerating the desired sub-trees is a variation of the technique presented in Section 2.1 to generate random extreme solutions. A path is constructed from each sink to the source node over reversed arcs. As in the random search algorithm, non-extreme solutions (cycles) are avoided by labelling nodes as follows:

$$visited(i) = \begin{cases} 0 \text{ if node } i \text{ is unused} \\ 1 \text{ if node } i \text{ is in the current sink path} \\ 2 \text{ if node } i \text{ is in a previous path .} \end{cases} \tag{5}$$

This labelling defines a partition of the nodes of G

$$N_G = N_G^0 \cup N_G^1 \cup N_G^2 \tag{6}$$

where $n_j \in N_G^i$ implies $visited(n_j) = i$.

The current state of the enumeration process is summarized by a collection of stacks, one for each completed path from a sink to the source, and one for the current active path. Each entry within a stack contains a node in the path defined by the stack. For each node, additional information is maintained, including the *visit* value, a pointer to the next element in the path, and a pointer to the current in-coming arc under consideration for the node. Processing within a stack is similar to any branch-and-bound procedure. However, additional processing is required to handle cases where a stack becomes empty, or a path is completed.

If a stack becomes empty during the enumerative search, augmenting paths to the previous stack have been fathomed. The current stack is popped, and the previous stack becomes active again at its top node. This implies that the entries in the previous stack have their *visit* value reset to 1. When a path is completed, all entries within the current stack have their *visit* values set to 2. When the last stack is popped, the search is complete.

3.2. IMPROVED SEARCH

The algorithm presented in Section 3.1 does not have processing time proportional to the number of extreme feasible solutions for the current problem. This is a result of the dead-ends described in Section 2.1 for the random initial solution technique. A worst case example of the effects of a dead-end is presented in Figure 16. The current search node is 1, and the current path is depicted by the thicker arcs. For this case, the dead-end contains a clique of size k. The enumerative search process would enumerate all paths through the clique before establishing that no feasible path exists to the source or to any node in a previous path.

Fortunately, this can be avoided with some additional processing. When a new

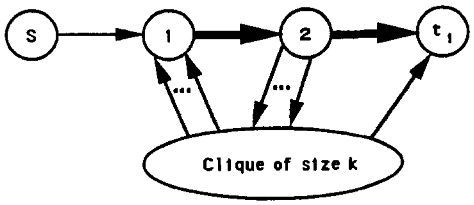


Fig. 16. Worst case dead end for enumerative search.

node is added to the current search path, all nodes reachable on the network $G' = (N_G^0 \cup N_G^2, A_G')$ are computed, where A_G' is the set of arcs in A_G connecting nodes in $N_G^0 \cup N_G^2$. The reachable nodes can be identified by solving a single-source shortest weighted path problem. For the example, the entire clique would be removed from consideration when node 2 is added to the current search path.

3.3. BOUNDING THE SEARCH

The global search algorithm can be extended to exploit the cost functions assigned to the arcs. For the case with non-decreasing cost functions, the cost incurred by an arc at an intermediate point in the algorithm cannot decrease with the addition of other paths. This can be exploited by maintaining a running cost of the current search at each position of the stack. This corresponds to the cost of the current flow up to the active node being processed. If the current flow cost exceeds the current best solution cost, then the search can be terminated for the current path, and the next path in the search can be considered. This approach is referred to as cost bounding.

The arc cost functions can be exploited further by using linear underestimation to project a lower bound on the cost of extending the current path. This is achieved by

- Computing the maximum possible flow through each node i, based on the sinks reachable from node i, *node.max*(i).
- Maintaining the current flow through each node i, *node.current*(i).
- Maintaining the current flow through each arc (i, j), *arc.current*(i, j).

Initially, the maximum flow possible for any arc (i, j) is *node.max*(j). The minimum flow through any arc, initially, is 0. During the search process the bounds on arc (i, j) can be tightened:

$$\begin{aligned} &\mathit{upper.bound}(i, j) = \mathit{node.max}(j) - (\mathit{node.current}(j) - \mathit{arc.current}(i, j)) \\ &\mathit{lower.bound}(i, j) = \mathit{arc.current}(i, j). \end{aligned} \tag{7}$$

The improvement in the bounding process is pictured in Figure 17.

The linear underestimation process uses the refined bounds and the currently unsatisfied sink nodes to project a lower bound on the final cost of the current search. This is achieved by computing new arc costs corresponding to the linear underestimating cost:

$$\mathit{linear.cost}(i, j) = \frac{c_{ij}(\mathit{upper.bound}(i, j)) - c_{ij}(\mathit{lower.bound}(i, j))}{\mathit{upper.bound}(i, j) - \mathit{lower.bound}(i, j)} . \tag{8}$$

Then the cost of an augmenting flow on arc (i, j) incurs cost

$$c'_{ij}(x_{ij}) = \mathit{linear.cost}(i, j) * x_{ij} . \tag{9}$$

A single-source shortest weighted path problem can be solved to obtain the

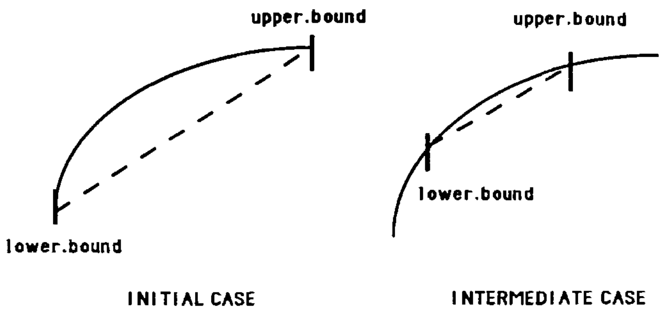


Fig. 17. Linear underestimation improvement.

shortest path from the source to each currently unsatisfied sink, using the *linear.cost* vector as the arc weights.

3.4. EXPLOITING NETWORK STRUCTURE

The enumeration process can be further improved by detecting subproblems that can be more efficiently solved using other techniques. For example, the network in Figure 18 has a subproblem to sink t_i that can be solved using a shortest weighted path algorithm. More general cases that are computationally expensive for enumeration, but more efficiently solvable by dynamic programming include sub-networks that have small total in and out-degree, and contain a large number of nodes.

Some simple subproblem cases can be detected by computing spanning trees for small perturbations of network G. For example, if subproblems with in-degree and out-degree of one are sought, then $O(n)$ spanning trees need to be computed. For each node $n_i \in N_G$, two problems are solved. The first identifies those nodes no longer reachable from the source when n_i is removed from network G, denoted by $SCUT_i$. The second identifies nodes that are disconnected from all sinks when n_i is removed, denoted by $TCUT_i$. The intersection of these sets for each node pair identifies a candidate subproblem, $SUB_{ij} = SCUT_i \cap TCUT_j$. The process can be applied once, and the maximal subproblems can be removed iteratively.

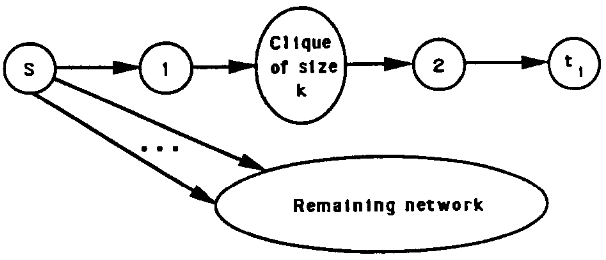


Fig. 18. Simple subproblem example.

In general, the problem of identifying the best subproblem can be *NP*-complete. For example, if we require that a subproblem have a specified lower bound on the number of nodes, and search for the node set with minimal total degree, then the problem corresponds to the Graph Partitioning problem [8]. However, heuristics, such as the Kernighan–Lin algorithm [16], have been successful identifying subgraphs with the desired property, or near to the desired optimum.

The enumeration process can avoid re-solving the detected subproblems by collapsing each subproblem into a single super-node. When a super-node is encountered during the search process, the sub-problem can be solved if it is new, or, its solution can be extracted using a look-up table if it was previously solved.

The overall process of exploiting subproblems can be viewed in relation to the send-and-split algorithm [3]. In send-and-split, the optimal flow for the entire network is solved using dynamic programming. This requires substantial storage, but avoids recomputing subproblems. The full enumeration algorithm requires little storage, but recomputes all subproblems. By exploiting a select subset of the subproblems, the resulting algorithm gains efficiency, while limiting the storage requirements based on available resources.

3.5. PROCESSING RESULTS

Variations of the enumerative search process were applied to randomly generated problems. The problems processed were of moderate density. Analysis of the problems indicated that dead-ends did not have a serious impact on the search. As a result, the search for unreachable nodes, as described in Section 3.2, was not performed. The networks were, also, preprocessed to identify subproblems with in-degree and out-degree equal to one. No significant subproblems of this type were detected. The algorithms used included

- Full enumeration (with no cost bounding)
- Cost bounding
- Cost bounding with linear underestimation
- Initial solutions provided.

In Figures 19 to 22, results are provided for four problem sets, each generated with a different random number seed. The effect of varying the number of sinks while maintaining the network structure is investigated. The results indicate that problems of moderate network size and density are solvable using the algorithm with cost bounding. Linear underestimation becomes efficient only for a few difficult problems. This is due to linear underestimation requiring a substantial amount of additional processing at each path completion. This could be avoided by selectively applying linear underestimation based on the current path cost, the previous path cost, and the amount of change between the current path and previous path.

TEST SET 1			
Nodes/Arcs/Sinks	Full enumeration	Cost bounding	Linear underestimation
10/40/5	4.64	0.13	0.36
15/60/5	753.08	0.22	0.52
15/60/10	7766.13	8.97	18.55
20/80/5	31820.84	1.15	2.50
20/80/10	X	25.84	41.77
20/80/15	X	624.96	1202.03
25/100/5	X	0.91	1.27
25/100/10	X	6.63	12.67
25/100/15	X	5224.91	12808.24
25/100/20	X	17908.40	44663.96
30/120/5	X	1.42	3.48
30/120/10	X	23.67	60.88
30/120/15	X	351.29	800.04

Fig. 19. Global search results – Test set 1.

In Figure 23, results indicate that the quality of the initial solution has a direct relation to the performance of the search, when cost bounding is employed. These results, also, provide a comparison of the performance of initial solutions computed using linear underestimation, random search, and greedy algorithms coupled with local search. The initial solutions obtained using random search were the best for all test cases. In fact, random search detected the global

TEST SET 2			
Nodes/Arcs/Sinks	Full enumeration	Cost bounding	Linear underestimation
10/40/5	7.57	0.37	0.53
15/60/5	311.89	0.42	0.58
15/60/10	3869.25	0.82	1.10
20/80/5	3525.02	12.25	13.69
20/80/10	X	21.36	26.96
20/80/15	X	32.70	47.60
25/100/5	X	0.24	0.54
25/100/10	X	32.28	61.64
25/100/15	X	138.83	247.56
25/100/20	X	717.58	1254.99
30/120/5	X	1.61	3.58
30/120/10	X	63.43	154.51
30/120/15	X	2942.59	7320.70

Fig. 20. Global search results – Test set 2.

TEST SET 3			
Nodes/Arcs/Sinks	Full enumeration	Cost bounding	Linear underestimation
10/40/5	2.50	0.08	0.13
15/60/5	195.34	0.51	1.34
15/60/10	697.81	0.66	1.55
20/80/5	1940.52	1.11	3.03
20/80/10	X	1.71	4.36
20/80/15	X	10.22	20.33
25/100/5	X	2.74	6.99
25/100/10	X	5.48	9.49
25/100/15	X	8.10	11.93
25/100/20	X	658.60	1292.08
30/120/5	X	6.01	8.47
30/120/10	X	65.40	119.60
30/120/15	X	101.29	180.51

Fig. 21. Global search results – Test set 3.

optimum for all test cases in Figures 19 to 22. The initial solutions computed using local search were, on the average, better than those obtained using linear underestimation. Figure 24 compares the number of solutions that were fully processed for enumerative search with cost bounding, and enumerative search with linear underestimation.

TEST SET 4			
Nodes/Arcs/Sinks	Full enumeration	Cost bounding	Linear underestimation
10/40/5	9.51	0.33	0.41
15/60/5	90.47	0.16	0.35
15/60/10	2317.58	0.72	1.41
20/80/5	5318.94	0.96	1.88
20/80/10	X	3.23	6.60
20/80/15	X	2055.09	405.42
25/100/5	X	2.41	7.55
25/100/10	X	237.50	448.99
25/100/15	X	3300.59	7354.37
25/100/20	X	5134.41	11672.19
30/120/5	X	0.72	1.60
30/120/10	X	676.26	1747.11
30/120/15	X	(>60 hours)	51276.51

Fig. 22. Global search results – Test set 4.

Test Set (25/100/15)	Technique	Initial Solution	Time (seconds)
1	GREEDY	1589.76	5224.91
	LINEAR	1745.48	6351.81
	RANDOM	1506.38	3583.58
2	GREEDY	1007.78	138.83
	LINEAR	1133.64	152.96
	RANDOM	943.46	115.46
3	GREEDY	764.12	8.10
	LINEAR	1111.01	13.40
	RANDOM	764.12	8.10
4	GREEDY	1600.26	3300.59
	LINEAR	1869.02	3783.58
	RANDOM	1558.54	2949.78

Fig. 23. Global search results – Varying initial solutions.

Test set	Nodes/Arcs/Sinks	Cost bounding	Linear underestimation
1	20/80/15	971	598
4	20/80/15	59222	519
1	25/100/15	17512	8927
4	25/100/15	928	162
1	30/120/15	72	27
4	30/120/15	X	11392

Fig. 24. Global search results – Number of solutions examined.

4. Summary

Two algorithms for single-source, uncapacitated, concave-cost network flow problems were presented. The first, a random search algorithm, was demonstrated to be useful for obtaining good approximations to the global optimum. The processing time of the algorithm is dependent on the time required to perform local search, starting at an extreme feasible solution. In [11], empirical results indicate that local search processing time increases as a polynomial function of the number of nodes, network density, and the number of sinks. This indicates that the probabilistic algorithm can be used to solve large concave network flow problems.

The second algorithm presented, an exact global search algorithm, was demonstrated to be useful for problems of moderate size and density. This approach has the benefit of solving "easy" problems quickly, where an easy problem has few extreme feasible solutions, or has few solutions with cost near to the cost of the global optimum. The algorithm gains efficiency by bounding the search based on cost properties, and projected cost based on linear underestimation. Initial solutions obtained using random search and local search provide a good initial approximation to the global optimum in most cases.

References

1. Bornstein, C. T. and Rust, R. (1988), Minimizing a Sum of Staircase Functions Under Linear Constraints, *Optimization* **19** (2), 181–190.
2. Eggleston, H. G. (1963), *Convexity, Cambridge Tracts in Mathematics and Mathematical Physics No. 47*, Cambridge University Press, Cambridge, Mass.
3. Erickson, R. E., Monma, C. L., and Veinott, Jr., A. F. (1987), Send-and-Split Method for Minimum-Concave-Cost Network Flows, *Mathematics of Operations Research* **12** (4), 634–664.
4. Falk, J. E. and Soland, R. M. (1969), An Algorithm for Separable Nonconvex Programming Problems, *Management Science* **15** (9), 550–569.
5. Florian, M. and Robillard, P. (1971), An Implicit Enumeration Algorithm for the Concave Cost Network Flow Problem, *Management Science* **18** (3), 184–193.
6. Gallo, G., Sandi, C., and Sodini, C. (1980), An Algorithm for the Min Concave Cost Flow Problem, *European Journal of Operations Research* **4**, 249–255.
7. Gallo, G. and Sodini, C. (1979), Adjacent Extreme Flows and Application to Min Concave-Cost Flow Problems, *Networks* **9**, 95–121.
8. Garey, M. R. and Johnson, D. S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, CA.
9. Guisewite, G. M. and Pardalos, P. M. (1990), Algorithms for the Uncapacitated Single-Source Minimum Concave-Cost Network Flow Problem, Working Paper, Department of Computer Science, Pennsylvania State University.
10. Guisewite, G. M. and Pardalos, P. M. (1990), Minimum Concave-Cost Network Flow Problems: Applications, Complexity, and Algorithms, *Annals of Operations Research* **25**, 75–100.
11. Guisewite, G. M. and Pardalos, P. M. (1990), Performance of Local Search in Minimum Concave-Cost Network Flow Problems, Working Paper, Department of Computer Science, Pennsylvania State University.
12. Lamar, B. (1989), An Improved Branch and Bound Algorithm for Minimum Concave Cost Network Flow Problems, Working Paper, Graduate School of Management and Institute of Transportation Studies, University of California, Irvine.
13. Lozovanu, D. D. (1983), Properties of Optimal Solutions of a Grid Transport Problem with Concave Function of the Flows on the Arcs, *Engineering Cybernetics* **20**, 34–38.
14. Minoux, M. (1976), Multiflots de coût minimal avec fonctions de coût concaves, *Annals of Telecommunication* **31** (3–4), 77–92.
15. Murty, K. G. (1968), Solving the Fixed Charge Problem by Ranking the Extreme Points, *Operations Research* **16** (2), 268–279.
16. Papadimitriou, C. H. and Steiglitz, K. (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall Inc., Englewood Cliffs, NJ.
17. Pardalos, P. M. (1988), Enumerative Techniques for Solving Some Nonconvex Global Optimization Problems, *OR Spektrum* **10**, 29–35.
18. Pardalos, P. M. (1989), Parallel Search Algorithms in Global Optimization, *Applied Mathematics and Computation* **29**, 219–229.
19. Plasil, J. and Chlebnican, P. (1990), A New Algorithm for the Min Concave Cost Flow Problem, Working paper, Technical University of Transport and Communications, Czechoslovakia.
20. Rech P. and Barton L. G. (1970), A Non-Convex Transportation Algorithm, in E. M. L. Beale (ed.), *Applications of Mathematical Programming Techniques*, The English Universities Press LTD, London, 250–260.
21. Soland, R. M. (1974), Optimal Facility Location with Concave Costs, *Operations Research* **22** (2), 373–382.
22. Yaged, Jr., B. (1971), Minimum Cost Routing for Static Network Models, *Networks* **1**, 139–172.
23. Zangwill, W. I. (1968), Minimum Concave-Cost Flows in Certain Networks, *Management Science* **14** (7), 429–450.