Algorithms for the Single-Source Uncapacitated Minimum Concave-Cost Network Flow Problem

G. M. GUISEWITE[1] and P. M. PARDALOS[2]
[1]*HRB Systems, State College, PA, U.S.A.*;
[2]*Pennsylvania State University, University Park, PA 16802, U.S.A.*

(Received: 16 May 1991)

Abstract. We investigate algorithms, applications, and complexity issues for the single-source uncapacitated (SSU) version of the minimum concave-cost network flow problem (MCNFP). We present applications arising from production planning, and prove complexity results for both global and local search. We formally state the local search algorithm of Gallo and Sodini [5], and present alternative local search algorithms. Computational results are provided to compare the various local search algorithms proposed and the effects of initial solution techniques.

Key words. Concave-cost network flow, uncapacitated, single, source, global optimization, local optimization, complexity theory, *NP*-hard.

1. Introduction

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 $G = (N, A)$ consisting of a set N of n nodes and a set A 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

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

subject to

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

and

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

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 [3]. Extreme points of the polyhedron correspond to basic feasible solutions in the simplex tableau [1]. The concave case differs from the convex 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 an arborescence [21]. 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 [9]. Unless stated otherwise, we consider in this paper cases of the SSU MCNFP with arc flow costs that are non-negative, nondecreasing 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. For example, in a production setting decreasing concave arc cost functions would exclude the influence of demand on production.

The SSU MCNFP arises naturally in numerous application areas. For production and inventory planning models, the single source can indicate a starting point in time for the model. The overall flow indicates the total production for the time window of the model, and the sinks represent intermediate production requirements for the time window. This approach is reflected in the following three production and inventory planning models.

The basic model in this area is the Wagner–Whitin model for a production and inventory system with no backlogging of demand [16], [17], [21], [22]. The variables for the model are:

1. $P_i(x_i)$ – cost of producing x_i units in period i,
2. $H_i(I_i)$ – the inventory holding costs in period i,
3. n – number of time periods,
4. r_i – market requirements for time period i,
5. x_i – amount produced in period i,
6. I_i – inventory in period i, $I_i = \Sigma_{k=1}^{i} (x_k - r_k)$,
7. Production nonnegative corresponds to $x_i \geqslant 0$,
8. No backlogging unsatisfied demand corresponds to $I_i \geqslant 0$.

The resulting model becomes

$$\min \sum_{i=1}^{n} P_i(x_i) + \sum_{i=1}^{n} H_i(I_i)$$

subject to

$$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} r_i$$

$$-x_i - I_{i-1} + I_i = -r_i \quad i = 1, \ldots, n$$

$$x_i \geq 0, \quad I_i \geq 0, \quad I_0 = 0, \quad I_n = 0.$$

The network is depicted in Figure 1 for a three time period model. It is an acyclic, single source, multiple sink, uncapacitated network. The flows from S to i correspond to production (x_i) in time period i, while the flows from i to j, $i < j$ correspond to inventory (I_i).

The effect of allowing backlogging of demand on the previous model is explored in [20], [22]. Here the inventory constraints can become negative ($I_i \geq -a$, $a \geq 0$), indicating demands in a time period can be satisfied in a later period. The resulting extended network is in Figure 2. Flow on a backward arc corresponds to backlogged demand.

Both of the previous models assumed a single production facility. Generalizing the model to a product which requires a series of processes, each process performed in a separate facility, results in a multi-echelon economic lot size model [22]. The network for a three facility, three time period model (without backlogged demand) is shown in Figure 3. Facility 0 is the origin of the material required for processing. Again, flow from $(l, i) \rightarrow (l, j)$ corresponds to production in time period l at facility j and $(i, l) \rightarrow (j, l)$ corresponds to inventory in period i at facility l.

Concave cost functions for the previous models arise from start-up costs, such as facilities and equipment, and economies of scale, reflected in storage requirements, shipping, and material purchases. Other areas given rise to this class of

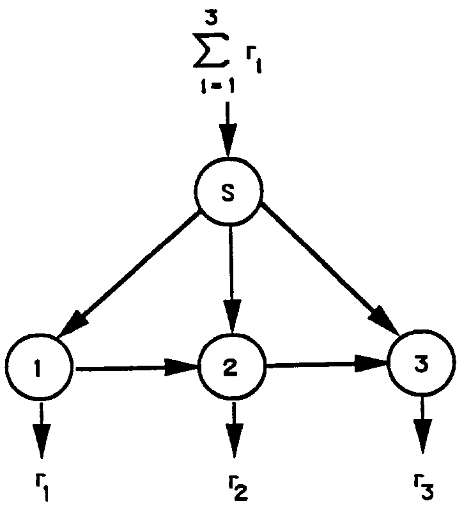

Fig. 1. A simple production model.

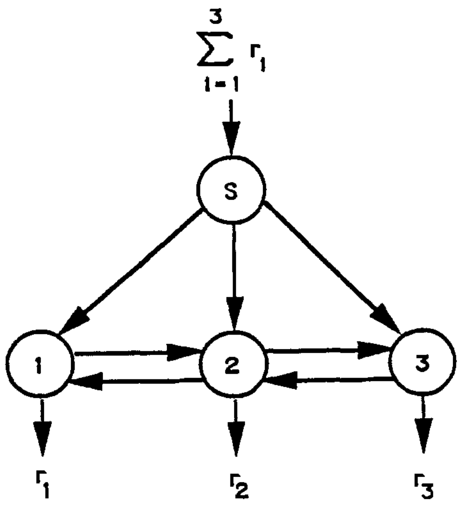

Fig. 2. Production model with backlogged demand.

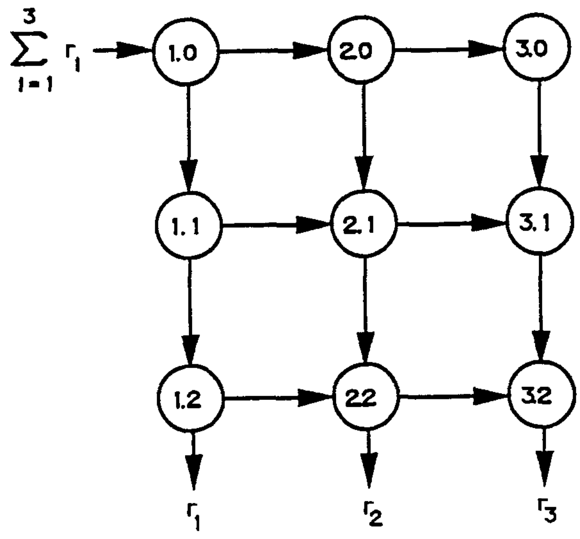

Fig. 3. Multi-echelon production model.

problems include transportation and shipping and communications. Also, many known NP-complete problems, such as 3-SAT, Minimum Cover, Vertex Cover, etc., can be formulated as SSU MCNFP.

2. Complexity

The general SSU MCNFP is known to be *NP*-hard. This follows immediately from this class of problems containing the "Steiner Tree in Graphs" problem [6], [9]. This result is based on arc costs that correspond to fixed arc weights.

In this section we prove that the SSU MCNFP is *NP*-hard for cases

1. With other types of arc cost functions,
2. For acyclic networks where all nodes have bounded total degrees less than or equal to three.

In addition, we prove that the problem of finding a strict local optimum, i.e., a feasible solution that has a lower cost than all solutions in a specified neighborhood, for SSU MCNFP is *NP*-hard if 3-SAT with unique solution is *NP*-hard.

The following transformation demonstrates that SSU MCNFP is *NP*-hard for cases involving objective functions other than the fixed-charge case $(c_{ij}(x_{ij} > 0) = w_{ij} \geq 0)$.

Consider the 3 Dimensional Matching (3DM) problem that is known to be *NP*-complete [6]:

INSTANCE. Set $M \subseteq W \times X \times Y$, where W, X, and Y are disjoint sets having the same number q of elements.

QUESTION. Does M contain a matching, i.e. a subset $M' \subseteq M$ such that $|M'| = q$ and no two elements of M' agree in any coordinate?

We construct the following flow problem:

1. Create a single source vertex S with source flow $d_1 = -3 * q$.
2. Create $n = |M|$ transshipment nodes M_i and arcs (S, M_i). These correspond to the elements (W_j, X_k, Y_l) for some j, k, and $l \ni 1 \leq j, k, l \leq q$.
3. Create $3 * q$ sinks $W_1, \ldots, W_q, X_1, \ldots, X_q, Y_1, \ldots, Y_q$. Each sink has flow requirement 1 $(d_i = 1)$. For each M_i add arcs $(M_i, W_j), (M_i, X_k)$, and (M_i, Y_l), where $M_i = (W_j, X_k, Y_l)$.
4. All arcs are uncapacitated.
5. All arcs have cost zero, except the arcs originating at the source.

The resulting network is pictured in Figure 4. Consider the following cost functions:

1. Start-up costs: This extends the fixed-charge case: $c_{ij}(0) = 0$, $c_{ij}(x) = a + (b * x), a > 0$.
2. Cost functions satisfying: $c_{ij}(0) = 0$, $c_{ij}(x + y) < c_{ij}(x) + c_{ij}(y)$, $x, y > 0$. This case includes all strictly concave functions, and a number of nonconcave, nonconvex functions.

For case 1, consider when all arcs (S, M_i) have cost 1 for nonzero flow. Then the optimal cost of the network problem is q if and only if the 3DM instance has answer *yes*. In general, if the cost on each arc is $c_{ij}(x) = a + bx$, then the optimal cost of the network problem is $(q * a) + (b * 3 * q)$ if and only if the 3DM instance has answer *yes*. In both cases the result follows by noting that if flow is split across

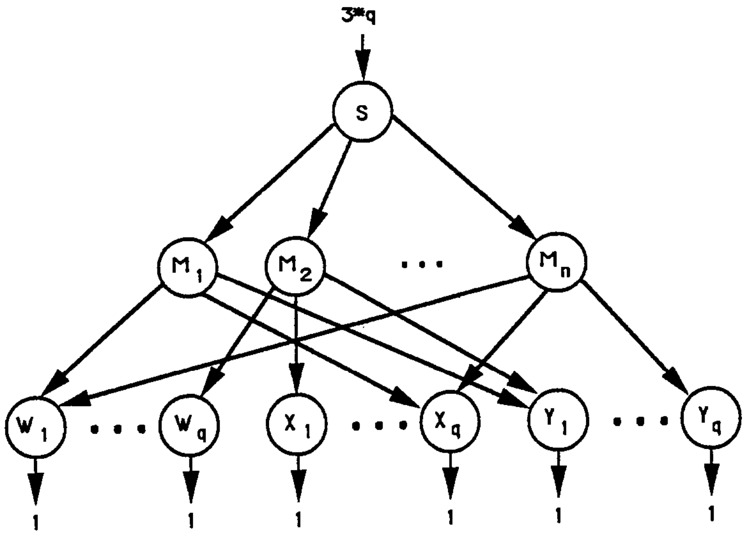

Fig. 4. Flow resulting from 3DM transformation.

more than q of the outgoing source arcs, then we incur additional startup costs. In addition, each M_i has a directed path to exactly 3 sinks (one in each of W, X, and Y), implying at least q of the (S, M_i) have nonzero flow.

For case 2, consider where the cost of all (S, M_i) are identical and satisfy the specified constraint. Here we find that the optimal cost of the network problem is $\Sigma_{i=1}^{q} c_{s,Mi}(3)$ if and only if the 3DM instance has answer *yes*. This follows from the observation that if no split of the flow occurs, the cost is as specified above and the arcs of the flow establish the solution to the 3DM problem. If any split occurs, then the cost increases.

For the case with no split:

$$\text{FLOWCOST} = \sum_{i=1}^{|M|} c_{S,M_i}(x_{S,M_i}) = \sum_{i=1}^{q} c_{s,M_i}(3) .$$

For the case with the flow split for a single arc (without loss of generality flow on (S, M_q) is split onto arc (S, M_{q+1})):

$$\begin{aligned}\text{FLOWCOST} &= \sum_{i=1}^{|M|} c_{S,M_i}(x_{S,M_i}) \\ &= \left(\sum_{i=1}^{q-1} c_{S,M_i}(3)\right) + c_{S,M_q}(x_{S,M_q}) + C_{S,M_{q+1}}(x_{S,M_{q+1}}) \\ &> \left(\sum_{i=1}^{q-1} c_{S,M_i}(3)\right) + c_{S,M_q}(3) = \sum_{i=1}^{q} c_{S,M_i}(3) .\end{aligned}$$

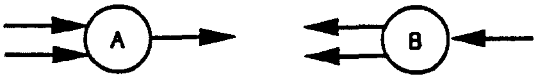

Fig. 5. Bounded degree constructors.

Noting that the above transformation from 3DM is a polynomial transformation indicates that the SSU MCNFP is *NP*-hard, even for cases with arc costs other than fixed-charge.

NP-hardness results can be extended for each of the cost function cases to networks of bounded arc degree (both in-degree and out-degree). This can be seen by noting that any network can be converted to a network with all nodes of total degree less than or equal to three simply by using the constructors depicted in Figure 5. Constructor *A* is used to reduce the in-degree of a node, while *B* reduces the out-degree. All new arcs added have associated a zero cost. The conversion process can be viewed iteratively as follows:
Suppose node *X* has total degree $n > 2$.

```
WHILE degree(X) > 2
  IF in.degree(X) > 1
    (1) Pair the incoming arcs, resulting in $(\lfloor \frac{in.degree(X)}{2} \rfloor)$ pairs $a_i$, $b_i$
    (2) Replace each arc pair with a type A constructor
  IF out.degree(X) > 1
    (1) Pair the outgoing arcs
    (2) Replace each arc pair with a type B constructor.
```

The process terminates in at most $\lceil log_2(n) \rceil$ steps. The number of added nodes is $O(n)$, the number of added arcs is, also, $O(n)$. A sample transformation is provided in Figure 6.

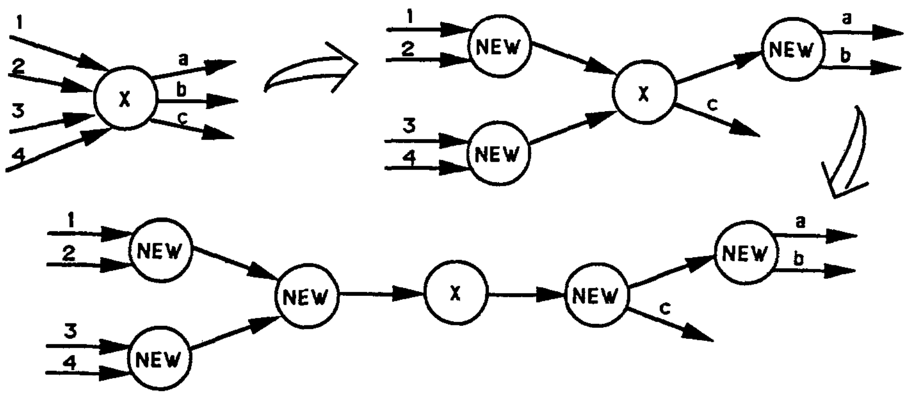

Fig. 6. Sample degree reduction transformation.

Each of the above results have addressed the complexity of locating a globally optimal solution for the SSU MCNFP. Here we investigate the complexity of checking if a solution is locally optimal, and of finding a local optimum for a SSU MCNFP. Before we can investigate these problems we must establish the criteria for a local optimum. For the MCNFP, the standard marginal definition of local optimality (i.e., rerouting a small portion of flow [19]) is not satisfactory, as some cost functions, e.g., fixed costs ($f(0)=0$, $f(x>0)=c>0$), result in all extreme flows being locally optimal. We use the definition of a local optimum as defined by Gallo and Sodini [5]. Here, a feasible solution is a local optimum if its objective value is less than or equal to all of its neighboring vertices. Gallo and Sodini also demonstrate that the problem of checking if a feasible solution for SSU MCNFP is a local optimum is in P. This result indicates that the problem of checking if a solution is a strict local optimum is in P. The complexity of finding a local optimum for SSU MCNFP is an open problem. Using the following 3-SAT transformation, originally developed in [8], we can establish a result for the problem of finding a strictly local optimum.

The transformation from 3-SAT is as follows:

INSTANCE. Collection $C=\{c_1, c_2, \ldots, c_m\}$ of clauses on a finite set U of variables such that $|c_i|=3$ for $1 \leq i \leq m$.

QUESTION. Is there a truth assignment for U that satisfies all the clauses in C?

We construct the following flow problem:

Let S denote the source vertex.

1. Add $(S, V_i), \forall i=1, \ldots, k$ where $k=|V|$ is the number of distinct variables in the 3-SAT instance.
2. Add $(V_i, T_i), (V_i, F_i), \forall i$. This corresponds to a TRUE or FALSE assignment for each variable.
3. Add $(T_i, FC_i), (F_i, FC_i), \forall i$. This forces a choice at each node.
4. The remaining arcs and nodes depend on the structure of the 3-SAT clauses. For example, if $C_i = c_x \wedge c_y \wedge c_z$ we add nodes t_{i1}, t_{i2}, t_{i3} and arcs (T_x, t_{i1}), (T_y, t_{i2}), (T_z, t_{i3}). If c_x were negated we would add arc (F_x, t_{i1}) in place of (T_x, t_{i1}). For the case where $C_i = c_x \wedge (c_y \vee c_z)$ we would add nodes t_{i1}, t_{i2} and arcs (T_x, t_{i1}), (T_y, t_{i2}), (T_z, t_{i2}). The addition of only two new nodes results from the choice of the conjunction in forcing a variable's value of TRUE or FALSE. All other cases are handled in a similar manner. Figure 7 demonstrates instances of this subset of the transformation.
5. Let $|T|$ denote the number of t nodes added at step 4. Let $|V|$ denote the number of distinct variables occurring in clauses. Then set the source flow to $|T| + |V|$. $|V|$ units of flow are used to force the assignment of TRUE or FALSE to each variable. $|T|$ units of flow are used to force the satisfiability

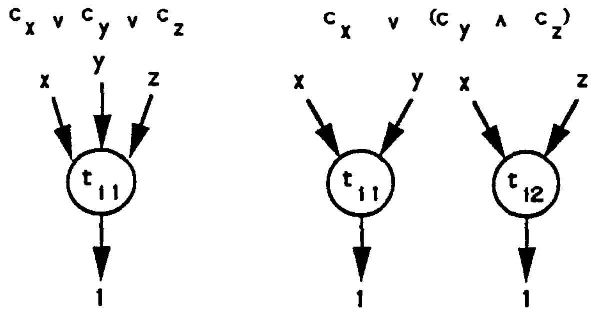

Fig. 7. Transformation of the clauses.

of each clause. This corresponds to each FC_i and each t_{ij} being a sink with requirement 1.

6. All arc flow costs are zero except for (V_i, T_i) and (V_i, F_i) which have cost zero if flow equals zero and one if flow is greater than zero.

The resulting network, presented in Figure 8, has optimal flow cost $|V|$ if and only if the 3-SAT instance has a satisfying assignment. This can be seen by noting that any feasible flow has cost greater than or equal to $|V|$ due to the FC_i sinks forcing a unit of flow through each V_i. If the additional flow necessary to satisfy the sinks

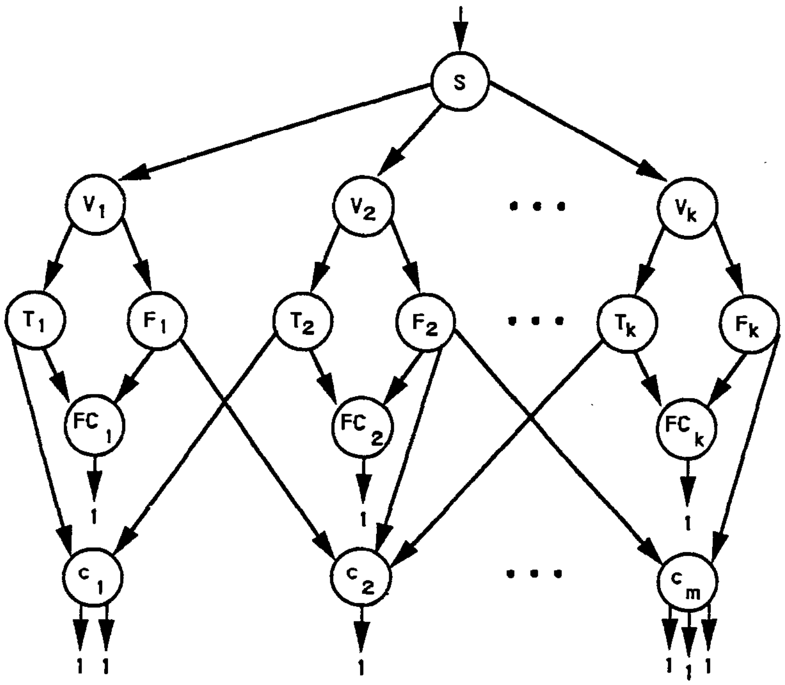

Fig. 8. Network resulting from 3-SAT transformation.

resulting from the clauses (t_{ij}) can be met without taking a path from V_i to T_i or F_i which currently has zero flow, then the cost of the flow remains at $|V|$. In this case, the assignment $v_i = \text{TRUE}$ if the flow on (V_i, T_i) is greater than zero, else FALSE, results in a satisfying assignment for the 3-SAT instance. If no satisfying assignment exists, we see it is necessary to have some i such that both (V_i, T_i) and (V_i, F_i) have nonzero flows. This implies the optimal network flow has cost greater than $|V|$.

We use the following properties of the 3-SAT transformation:

1. If a feasible flow is not a global optimum, then it is not a strict local optimum. This results from a nonoptimal flow having some variable with flow on both (V_i, T_i) and (V_i, F_i). An adjacent solution of equal cost can be obtained by altering the flow to the corresponding FC_i sink.
2. If the 3-SAT instance has a unique solution, then this global optimum is a strict local optimum.

These combined facts indicate that if 3-SAT with unique solution is *NP*-hard, then the problem of finding a strict local optimum for SSU MCNFP is also *NP*-hard. Although the complexity of 3-SAT with unique solution is an open problem, Valient and Vazirani [15] prove that SAT with unique solution is *NP*-hard under randomized polynomial-time reductions. The existence of a parsimonious transformation from SAT to 3-SAT [6] carries the randomized result over to 3-SAT. Pardalos and Schnitger [11] prove that checking strict local optimality for the indefinite case is *NP*-hard, indicating that finding a strict local optimum for this case is *NP*-hard.

Few problems in the class SSU MCNFP have been identified as solvable in polynomial time. For many of the cases that are, the problem can be reformulated and solved as a shortest weighted path problem. This includes the single sink case [21]. For this problem, using the previously stated definition of a local optimum, any extreme solution is adjacent to all other extreme solutions. This implies a single local search test based on a shortest weighted path problem (as discussed in the next section) will locate a global optimum. The following results present shortest path reformulations of the Wagner–Whiten model for production and inventory systems, and the extension to allow backlogged demands. Both of these problems were stated formally in the previous (introduction) section.

Figure 9 represents a state diagram for the Wagner–Whiten production model without a backlogged demand. Changing states in this model corresponds to forcing a specified flow on one of the (S, i) arcs of Figure 1. The candidate flow magnitudes correspond to $R_{ij} = \Sigma_{l=i}^{j} r_l$ where $1 \leq i \leq j \leq n$. The state diagram is acyclic, corresponding to the fact that the flow requirements are met in the specified order $r_1, r_2, \ldots, r_n$. Being in state i indicates that all requirements up to, and including, r_i have been met, and no other flows have been met. Moving from state i to state j, $j > i$, corresponds to moving $R_{i+1,j}$ units of flow across arc $(S, i+1)$, then forwarding the excess flow $(R_{i+1,j} - r_{i+1})$ along arc $(i+1, i+2)$,

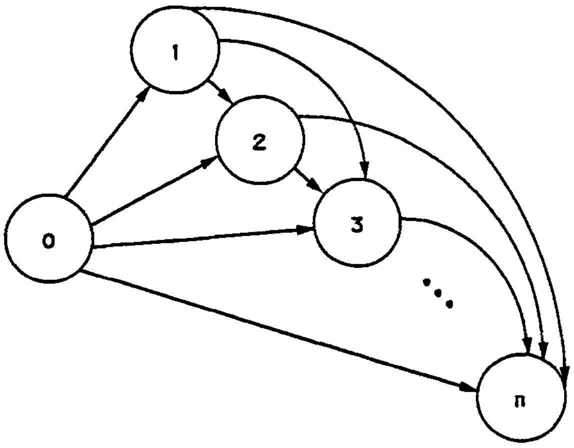

Fig. 9. State diagram for Wagner–Whiten problem.

$(R_{i+1,j} - r_{i+1} - r_{i+2})$ units along $(i+2, i+3)$, etc. This formulation of the problem indicates that there are 2^{n-1} basic feasible solutions. This follows from all solutions terminating in state (node) n, and passing through any subset of the remaining states. The cost assigned to each arc is as follows (c' = new costs, c = original arc costs):

$$c'(i, j) = c_{S,i+1}(R_{i+1,j}) + \sum_{l=i+1}^{j-1} c_{l,l+1}(R_{l+1,j}) .$$

The resulting shortest path problem can be formulated and solved in $O(n^2)$ operations using a single source shortest weighted path algorithm for dense networks (e.g., Dijkstra's algorithm).

The corresponding formulation for the model with backlogged demand requires additional states. In this case, each state (j, k) reflects the maximum required flow currently satisfied, r_k, and the arc from the source used to convey this flow, (S, j). Now a transition from state (j_1, k_1) to state (j_2, k_2) corresponds to forcing $\Sigma_{l=k_1+1}^{k_2} r_l$ units of flow across arc (S, j_2), where $j_1 \leq k_1 < j_2 \leq k_2$. This flow represents R_{k_1+1,j_2-1} units of backlogged demand. The cost for arc flows in this state diagram are

$$c'((j_1, k_1), (j_2, k_2)) = c_{S,j_2}(R_{k_1+1,k_2}) + \sum_{l=j_2}^{k_2-1} c_{l,l+1}(R_{l+1,k_2})$$
$$+ \sum_{l=k_1+2}^{j_2} c_{l,l-1}(R_{k_1+1,l-1}) .$$

Nodes exist for the initial state $(0, 0)$ and all states $(i, j) \ni 1 \leq i \leq j \leq n$, resulting in $O(n^2)$ nodes. Arcs exist for all valid transformations, (j_1, k_1) to

(j_2, k_2) where $j_1 \leq k_1 < j_2 \leq k_2$, resulting in a dense network. The resulting shortest path formulation and solution requires $O(n^4)$ operations, with the optimal solution corresponding to the minimum shortest weight path from $(0, 0)$ to any (i, n), $i = 1, \ldots, n$.

3. Local Search Algorithms

Because the global search problem for the SSU MCNFP is *NP*-hard, efficient algorithms exist only for highly structured subcases, e.g., the production models presented in Section 1. Recent work by Thach [14] addressed the circuitless case by decomposing the problem into linear programs and developing successive approximations to the concave objective function. The send-and-split method [4] applies to the uncapacitated case in general. It employs dynamic programming, and the resulting processing time is exponential only in the number of sources and sinks. Numerous general concave enumeration techniques can be applied to this problem. A recent summary can be found in [10].

In order to avoid the excessive computations required by exact global search, we investigate the performance of local search for the SSU MCNFP. Work by Gallo and Sodini [5] indicates that even though the complexity of local search for this problem is unknown, preliminary computational results are encouraging. We investigate the effects of initial solution techniques and various search techniques on local search for the SSU MCNFP. The algorithms considered are stated here. Computational results are presented in the following section of this paper.

The initial work in this area employed the following local search algorithm for the uncapacitated single source case:

```
Algorithm 1 (Vertex Local Search):
  Find an initial extreme feasible solution $X$
  WHILE ($X$ is not a local optimum)
    move to the best adjacent vertex $X'$
    $X \leftarrow X'$.
```

Gallo and Sodini used Algorithm 1 to locate a locate optimum. They determined if X was a local optimum, or detected the best adjacent solution to X by solving a series of shortest path problems. Their approach required a modified network to be constructed and a shortest weighted path problem to be solved for each vertex in the current solution flow X. Each modified problem identified the optimal rerouting of flow from the source S to a vertex x in the current flow X. The modifications to the network prevented computing a nonextreme and/or nonadjacent solution. The problems solved had arc costs of the form $\alpha_{ij}x_{ij}^{\beta}$, where $\beta = 0.25$, 0.50 or 0.75. Initial solutions were generated by solving a shortest weighted path problem with arc weights α_{ij}. This approach is stated formally here:

Let $T = (N_T, A_T)$ denote the solution tree for the current solution X. Denote the root of this tree by S. Flow on arc (i, j) is of magnitude x_{ij}. Denote by $G = (N, A)$ the original network flow problem with concave arc costs c_{ij}, flow requirements d_i, and size $n = |N|$ and $m = |A|$. For node $j \in N_T$ in the current solution X, denote the flow into j by $IN(j)$. Let P_{jk} denote the path from j to k in T.

For each node i in the current solution tree T a shortest weighted path problem based on the network $G' = (N', A')$ with arc weights w_{jk} is constructed and solved. G' is constructed as follows:

1. Reduce flow on the path from S to i by $IN(i)$ units: $X' = X$, $\forall (l, m) \in P_{Sj}$, $x'_{l,m} = x'_{l,m} - IN(i)$.
2. Generate a weight for each arc in G' based on the augmenting flow $IN(i)$: $\forall (l, j) \in A'$ $w_{lj} = c_{lj}(x'_{lj} + IN(i)) - c_{lj}(x'_{lj})$.
3. Update G' to avoid nonextreme solutions: For each node $k \in N_T$
   (*a*) $(k = i) \Rightarrow$ remove the single arc $(l, k) \in A_T$,
   (b) $(k \in N_T)$ and (k is not a descendent of i in T) $\Rightarrow$ remove all incoming arcs to node k, add arc (S, k) with weight $w_{S,k} = \Sigma_{(l,j) \in P_{Sk}} w_{l,j}$,
   (c) $(k \in N_T)$ and (k is a descendent of i in T) $\Rightarrow$ remove node k and all associated arcs.

Steps (1) and (2) generate a weighted network that corresponds to the cost of rerouting $IN(i)$ units of flow from path P_{Sj} onto any valid path in G. Step (3) forces the new flow to be adjacent to flow X, and extreme. This corresponds to the fact that if X' is an adjacent flow to X, then $X + X'$ contains a single cycle [5]. A flow across arc (S, k) in G' represents the original flow across path P_{Sk} in solution X of G. This forces the new flow X' to follow the old flow until node k, then divert onto a new path until reaching node i. Flow cannot pass on arcs of solution X after the first diversion because of Steps 3(a) and 3(b), where incoming arcs are removed for nodes in solution X. This limits $X + X'$ to the single cycle requirement. Step 3(c) prevents the case where X' consists of X with a cycle appended at i. Figures 10 and 11 provide a sample transformation.

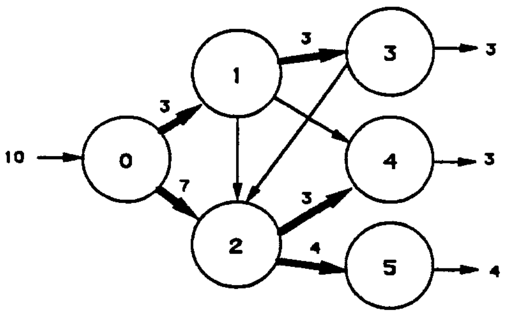

Fig. 10. Original graph and solution.

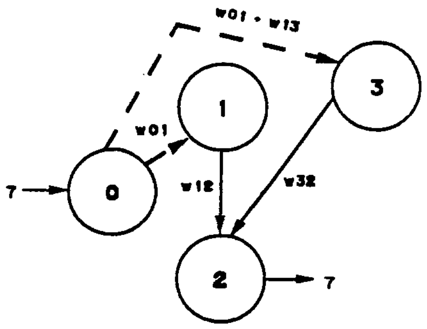

Fig. 11. Transformed sub-problem for node 2.

A simple variation of algorithm 1 is to move to the first better adjacent solution detected:

```
Algorithm 2:
  Find an initial extreme feasible solution $X$
  WHILE ($X$ is not a local optimum)
    move to the first detected better adjacent vertex $X'$
    $X \leftarrow X'$.
```

This algorithm provides additional options in terms of the order that adjacent solutions are evaluated.

An alternative to searching only adjacent vertices is to relax the search to a superset of the adjacent vertices of X that are easily computed. For the above case, this can be achieved by solving a series of shortest weighted path problems that correspond to rerouting flow to a vertex, but only for a subset of the vertices in X. In this case, we do not modify the graph to prevent nonextreme solutions and/or nonadjacent solution. The maximum number of shortest path problems solved is $(2 * (\text{number of sinks})) - 1$. This corresponds to branch points (vertices with degree greater than two) in the solution X, along with the sink vertices. In the event that a nonextreme solution is selected as the next candidate local optimum, cycles in the new solution are detected and flow is rerouted appropriately. This can result in additional improvements to the solution. The relaxed approach can be shown to converge to a local optimum. However, it is possible that it visits several local optima before terminating. This relaxed approach gives rise to two additional algorithms:

```
Algorithm 3 (Relaxed Local Search):
  Find an initial extreme feasible solution $X$
  WHILE ($X$ is not a "relaxed" local optimum)
    move to the best "relaxed" adjacent vertex $X'$
    force the solution to be extremal
    $X \leftarrow X'$.
```

```
Algorithm 4:
  Find an initial extreme feasible solution $X$
  WHILE ($X$ is not a "relaxed" local optimum)
    move to the first detected better "relaxed" adjacent vertex $X'$
    force the solution to be extremal
    $X \leftarrow X'$.
```

Formally the relaxed algorithm is as follows:

For each branch point and sink vertex solve a shortest weighted path problem on network G', where G' is constructed by applying steps (1) and (2) of the vertex local search algorithm. Nonextreme solutions are detected and forced to a neighboring extreme solution by

1. Performing breadth first search on the solution graph (not necessarily a tree in this case).
2. Identifying cycles as surplus arcs (arcs $(i, j) \ni (i, j)$ is not used in the breadth first tree).
3. Removing the cycles in order $i_1, i_2, \ldots, i_k \ni \text{depth}(i_l) \leq \text{depth}(i_{l+1})$, $l = 1, 2, \ldots, k-1$. Cycles can be removed in a fashion that improves the current solution cost. This is achieved by exploiting the fact that a cycle results from two paths terminating in the same node ($P_{S_1,i}$ and $P_{S_2,i}$). If f_{i1} denotes the flow into node i from path $P_{S_1,i}$ and f_{i2} denotes the flow from $P_{S_2,i}$, we can reroute the flow based on the cost of rerouting flow f_{i2} onto $P_{S_1,i}$ as compared to rerouting flow f_{i1} onto $P_{S_2,i}$.

To see that it is sufficient to solve problems only for the branch points and sinks, we note that

1. For any $x_{ij} \in X \ni x_{ij} > 0$ there exists a branch point or sink with incoming flow equal to x_{ij}.
2. For the relaxed case, G' permits any rerouting of the current flow $IN(i)$. This includes cases whee the flow diverts from P_{Sj}, returns at node j, and proceeds to i via P_{ji}.

These two facts combined indicate that the problem of diverting flow of size $IN(j)$ to node j, occurs as a subproblem for some case based on rerouting flow to a branch point or sink.

Additional variations for all the above algorithms can be obtained by varying the initial solution techniques. We considered a range of such techniques:

(a) Shortest paths based on arc weights α_{ij}.
(b) Shortest paths based on arc weights β_{ij}, where we address problems with a randomly generated β_{ij} for each arc.
(c) Solving a series of independent shortest path problems based on the flow requirement for each sink, and the actual concave costs of each arc. For each sink, a shortest weighted path problem is solved from the source to

the sink. The weights are computed for each arc as the actual cost of flow equal to the sink requirement.

(d) Solving a series of inter-related shortest path problems. The sinks are assigned an ordering $s_1, \ldots, s_k$. Again a shortest weighted path problem is solved for each sink. For this case, the arc weights for the i-th shortest weighted path problem are based on the cost of augmenting flow to the computed flows for sinks $s_1, \ldots, s_{i-1}$. This method allows for many variations, based on the order that we solve the shortest path problems. We consider the sinks in (1) input order, (2) largest required flow to smallest, (3) smallest required flow to largest, and (4) a two pass approach where (a) is used to compute the cost per unit flow for each sink, then $(d1)$ is executed with the sinks in order of smallest cost per flow.

Algorithms for local search in multicommodity networks are explored in [12]. Simplex based local search algorithms and global search heuristics for the fixed-charge case are presented in [2], [13], and [18].

4. Computational Results

For each combination of basic algorithm $(1, 2, 3, 4)$ and initial solution technique $(a, b, c, d1, d2, d3, d4)$ we implemented and tested the local search algorithms on randomly generated graphs of varying density. These results were originally reported in [7]. Here we extend the results to include computational tests on layered networks. A sample of our results is presented in Figures 12 through 15. Each row in these figures corresponds to 100 test cases on a network with 25 nodes and 10 sinks. The average number of shortest paths indicates the number of shortest path problems solved in the course of locating a local optimum. The maximum and average number of iterations corresponds to the number of vertices checked for local optimality during the search. The average initial solution is the objective function value for the solution generated by the initial solution technique. The average first solution corresponds to the result after one iteration of the local search algorithm. The average final solution corresponds to the objective function value of the detected local optimum.

The test cases in Figures 12 through 15 were generated in a random fashion. Arcs were 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) were discarded. After the specified number of arcs were successfully generated, the resulting network was tested for connectivity by solving a single source shortest path problem from the source to all nodes in the network. If the connectivity was suitably high, then cost functions were generated for each arc. Each cost function was of the form $\alpha_{ij}x_{ij}^{\beta_{ij}}$, where the α_{ij} were uniformly distributed in $[1, 2, \ldots, 100]$ and the β_{ij} were uniformly distributed in $[.1, .2, \ldots, 1]$. This randomness results in the objective value functions increasing for sparse networks. This follows directly from the fact that as network density

Method	# Arcs	Avg # of shortest paths	Max. # of Its	Avg. # of Its	Average Initial Solution	Average First Solution	Average Last Solution
1, a	500	100.88	12	6.92	1405	611	218
	250	95.26	12	6.50	2863	1119	439
	125	94.91	11	6.41	5053	1992	940
	62	72.58	9	4.76	8394	4892	3539
1, b	500	141.20	17	10.98	1013	839	224
	250	132.14	16	9.85	1156	974	429
	125	109.04	13	7.67	1588	1336	910
	62	62.42	9	4.12	4410	3809	3535
1, c	500	75.58	11	5.21	280	254	212
	250	75.89	11	5.26	549	498	418
	125	67.52	9	4.57	1085	1004	900
	62	47.62	6	3.14	3728	3608	3533
1, d1	500	41.22	7	2.92	234	222	214
	250	40.79	8	2.85	457	438	424
	125	41.15	6	2.85	977	935	906
	62	30.02	6	2.03	3608	3564	3542
1, d2	500	36.79	9	2.58	231	223	216
	250	38.63	7	2.70	460	443	430
	125	37.42	8	2.57	965	926	905
	62	31.11	6	2.07	3620	3565	3537

Fig. 12. Computational results for Algorithm 1.

Method	# Arcs	Avg # of shortest paths	Max. # of Its	Avg. # of Its	Average Initial Solution	Average First Solution	Average Last Solution
2, a	500	66.89	19	8.80	1405	1025	219
	250	59.47	16	7.88	2863	2085	442
	125	63.79	20	8.13	5053	3698	942
	62	49.75	15	5.69	8394	6781	3538
2, b	500	73.74	26	13.43	1013	934	231
	250	77.94	21	12.41	1156	1078	446
	125	71.65	18	9.62	1588	1489	930
	62	41.33	12	4.66	4410	4088	3535
2, c	500	44.22	14	5.94	280	263	215
	250	45.15	14	6.14	549	520	422
	125	41.69	15	5.30	1085	1044	900
	62	33.15	12	3.59	3728	3649	3536
2, d1	500	29.42	7	3.14	234	226	215
	250	27.75	10	3.01	457	442	424
	125	27.69	7	3.00	977	948	907
	62	23.07	8	2.10	3608	3573	3542
2, d2	500	26.55	9	2.65	231	225	216
	250	27.44	10	2.86	460	446	429
	125	26.37	8	2.66	965	934	904
	62	23.64	7	2.14	3620	3574	3538

Fig. 13. Computational results for Algorithm 2.

Method	# Arcs	Avg # of shortest paths	Max. # of Its	Avg. # of Its	Average Initial Solution	Average First Solution	Average Last Solution
3, a	500	73.50	12	6.34	1405	535	213
	250	69.46	11	6.02	2863	988	420
	125	69.66	11	5.98	5053	1811	903
	62	49.58	8	4.27	8394	4452	3534
3, b	500	108.05	17	9.87	1013	792	213
	250	104.43	16	9.22	1156	946	421
	125	84.83	12	7.33	1588	1319	905
	62	47.46	8	4.03	4410	3796	3532
3, c	500	58.76	11	5.04	280	253	211
	250	59.51	10	5.12	549	497	418
	125	51.99	9	4.41	1085	997	899
	62	36.58	6	3.10	3728	3604	3533
3, d1	500	35.00	9	3.03	234	221	212
	250	33.53	8	2.87	458	437	423
	125	33.12	6	2.85	977	931	903
	62	24.12	6	2.06	3608	3560	3536
3, d2	500	34.21	10	2.74	231	221	213
	250	35.05	7	2.76	460	438	425
	125	33.41	9	2.65	965	924	900
	62	26.88	5	2.11	3620	3561	3533

Fig. 14. Computational results for Algorithm 3.

Method	# Arcs	Avg # of shortest paths	Max. # of Its	Avg. # of Its	Average Initial Solution	Average First Solution	Average Last Solution
4, a	500	45.91	15	7.56	1405	1090	213
	250	42.73	13	7.26	2863	2140	419
	125	41.88	18	6.88	5053	3689	904
	62	29.45	9	4.71	8394	6341	3537
4, b	500	66.17	17	11.51	1013	914	212
	250	58.83	16	10.16	1156	1068	428
	125	50.79	14	8.31	1588	1478	902
	62	30.06	10	4.27	4410	4061	3537
4, c	500	36.37	12	5.66	280	265	213
	250	36.42	12	5.63	549	521	421
	125	32.81	10	4.98	1085	1040	897
	62	24.73	8	3.31	3728	3649	3536
4, d1	500	21.26	10	3.11	234	224	212
	250	20.28	7	2.92	457	440	423
	125	20.38	7	2.95	977	941	903
	62	16.83	7	2.10	3608	3569	3536
4, d2	500	21.73	10	2.80	231	222	213
	250	21.19	9	2.84	460	442	425
	125	21.12	12	2.75	965	930	900
	62	19.35	8	2.17	3620	3572	3533

Fig. 15. Computational results for Algorithm 4.

increases, the probability of a low cost solution existing between the source and any sink increases.

All of the algorithms we employed are based on solving shortest path problems. Our current implementation does not exploit network sparseness, indicating that processing time is not an accurate measure for our results on sparse networks. In order to present our results in an implementation independent manner, we use the number of shortest path problems solved to compare the alternate techniques. To provide an indication of performance in terms of processing time, we provide times for several complete networks in Figure 16. These times are for the various algorithms (with the number of sinks 10) executed on a single T800 Transputer. The T800 is an INTEL manufactured 32-bit microprocessor designed to facilitate parallel processing.

Figures 12 through 15 indicate a subset of several thousand test cases executed to establish the effects of varying the initial solution technique and the local search technique. From these results we have observed the following:

1. Gallo and Sodini's results (Algorithm 1 and initial solution technique *a*) indicated a large improvement resulting from applying local search [5]. Our results demonstrate that this improvement is primarily due to the poor initial solutions resulting from the initial solution technique. In fact, technique *a* provided the poorest initial solutions when compared to the other six techniques considered.
2. In all the test cases, the maximum number of vertices examined during local search with initial solution techniques *c* and *d* was 20. This includes test cases with 100 nodes and 9900 arcs. This indicates a similarity between the performance of the Simplex method for linear programming and local search for the SSU MCNFP.
3. The initial solution techniques had little effect (on the average) on the objective function of the detected local optimum. This is surprising as the magnitude of the initial solutions varied significantly.
4. The initial solution techniques did have an effect on the convergence rate to a local optimum. This is especially evident when comparing techniques *a*, *b*, and *c* to the others.
5. The algorithms moving to the first better adjacent vertex (2 and 4) required on the average 25–40% fewer shortest path problems to be solved.

	Nodes/Arcs			
Method	12/132	25/600	50/2450	100/9900
1, d2	1.09	5.63	29.46	112.38
2, d2	0.76	4.28	21.48	88.10
3, d2	1.14	5.22	27.73	77.75
4, d2	0.81	3.55	16.09	53.07

Fig. 16. Timing results (seconds).

# Layers	Algorithm 1, d2	Algorithm 3, d2
5	25.8	14.3
10	40.3	16.7
20	70.4	21.7

Fig. 17. Processing comparison for layered graphs (average number of shortest weighted path problems solved).

6. The algorithms based on the relaxed search technique (3 and 4) required on the average 7–40% fewer shortest path problems to be solved. The cases with similar performance arise for initial solution technique *d*. The variation in performance will increase substantially for layered networks, as the random graphs for our test cases tend to have solutions with few transshipment nodes. This is demonstrated in Figure 17 where layered graphs with 5 nodes per layer and full interconnection between layers are solved using Algorithms 1 and 3. In these graphs the first layer consists solely of the source node, and each layer i is fully connected to layer $i+1$. It should be noted that the processing time for each iteration of relaxed search is larger. This is due to the additional processing required to convert nonextreme solutions to extreme solutions after the shortest path problems are solved.
7. Two of the initial solution techniques ($d2$ and $d4$) found local optima a significant portion of the time (20–40%) for sparse problems.
8. Network density had only a slight effect on the number of iterations required for local search, especially for initial solution techniques *c* and *d*.

These results indicate that local search techniques for the SSU MCNFP are computationally efficient. The surprising results are how efficient several initial solution techniques are at computing low cost solutions, and how little effect the initial solution technique has on the final solution's objective value. Another surprising result is the efficiency of local search (in terms of vertices visited) on dense networks. In addition, even though the relaxed local search algorithm computes a local optimum over a larger neighborhood than the Gallo and Sodini approach, the observed processing requirements for the relaxed approach are lower in most cases.

5. Summary

We have provided new complexity and computational results for the SSU MCNFP. We have shown that a wide range of these problems fall into the realm of *NP*-hard problems. We presented a formal description of the Gallo and Sodini [5] algorithm for local search. We also developed alternative local search algorithms, and implemented them for performance comparison. These results indicate that a substantial performance improvement can be obtained for local search by applying our relaxed algorithm, and by using greedy-based initial solution techniques. In addition, the presented algorithms are all amenable to a parallel solution. Each iteration involves the solution of multiple disjoint shortest weighted path problems that can be distributed across processors. These results indicate that the presented approach can be applied to the solution of large single-source uncapacitated minimum concave-cost network flow problems.

References

1. Dantzig, G. B. (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton, New Jersey.
2. Denzler, D. R. (1969), An Approximate Solution for the Fixed Charge Problem, *Naval Research Logistics Quarterly* **16**, 411–416.
3. Eggleston, H. G. (1963), *Convexity*, *Cambridge Tracts in Mathematics and Mathematical Physics No.* 47, Cambridge University Press, Cambridge, Mass.
4. 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.
5. Gallo, G. and Sodini, C. (1979), Adjacent Extreme Flows and Application to Min Concave-Cost Flow Problems, *Networks* **9**, 95–121.
6. 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.
7. Guisewite, G. M. and Pardalos, P. M. (1990), Uncapacitated Single-Source Minimum Concave-Cost Network Flow Problem, Working Paper, Department of Computer Science, Pennsylvania State University.
8. Guisewite, G. M. and Pardalos, P. M. (1990), Minimum Concave-Cost Network Flow Problems: Applications, Complexity, and Algorithms, *Annals of Operations Research* **28**, 75–100.
9. 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.
10. Pardalos, P. M. and Rosen, J. B. (1987), *Constrained Global Optimization*; *Algorithms and Applications*, Lecture Notes in Computer Science 268, Springer-Verlag, Berlin.
11. Pardalos, P. M. and Schnitger, G. (1988), Checking Local Optimality in Constrained Quadratic Programming is *NP*-Hard, *Operations Research Letters* **7**(1), 33–35.
12. 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.
13. Steinberg, D. I. (1970), The Fixed Charge Problem, *Naval Research Logistics Quarterly* **17**, 217–236.
14. Thach, P. T. (1989), An Efficient Method for Min Concave Cost Flow Problems Under Circuitless Single-Source Uncapacitated Networks, Technical Report, Technical University of Graz, Austria.
15. Valiant, L. G. and Vazirani, V. V. (1985), *NP* Is as Easy as Detecting Unique Solutions, *17'th STOC of the ACM*, 458–463.
16. Wagner, H.M. (1960), A Postscript to 'Dynamic Problems in the Theory of the Firm,' *Naval Research Logistics Quarterly* **7**, 7–12.
17. Wagner, H. M. and Whitin, T. M. (1958), Dynamic Version of the Economic Lot Size Model, *Management Science* **5**(1), 89–96.
18. Walker, W. E. (1976), A Heuristic Adjacent Extreme Point Algorithm for the Fixed Charge Problem, *Management Science* **22**(5) 587–596.
19. Yaged, Jr. B. (1971), Minimum Cost Routing for Static Network Models, *Networks* **1**, 139–172.
20. Zangwill, W. I. (1966), A Deterministic Multi-Period Production Scheduling Model with Backlogging, *Management Science* **13**(1) 105–119.
21. Zangwill, W. I. (1968), Minimum Concave-Cost Flows in Certain Networks, *Management Science* **14**(7) 429–450.
22. Zangwill, W. I. (1969), A Backlogging Model and a Multi-Echelon Model of a Economic Lot Size Production System – A Network Approach, *Management Science* **15**(9) 506–527.