Journal of the Operational Research Society (1996) 47, 1150–1165 0160-5682/96 $12.00

Enhanced Adjacent Extreme-point Search and Tabu Search for the Minimum Concave-cost Uncapacitated Transshipment Problem

CÜNEYT F. BAZLAMACCI*[1] and KHALIL S. HINDI[2]
[1]UMIST, UK and [2]Brunel University, Middlesex, UK

Practicable methods for optimising concave-cost, uncapacitated transshipment networks are non exact. In this paper, one such effective method, that of adjacent extreme point search, is further developed to enhance its overall computational efficiency. The enhanced search algorithm is then imbedded in a tabu search scheme which proved capable of finding better solutions, by a wide margin in some instances. Another tabu search scheme, somewhat inferior in terms of solution quality but computationally more efficient, is also developed to provide an alternative solution vehicle for larger networks. Results of extensive computational testing are included.

Key words: concave-cost optimisation, heuristics, network programming, tabu search, transshipment problem

INTRODUCTION

Concave-cost, uncapacitated transshipment networks arise from, and have their applications in, *inter alia*, transportation and communication network design[1–4], production, distribution and inventory planning[5–8], facility location[9], waste water and water resource management and planning[10,11]. The non-linear concave costs arise from initial set-up costs, discounts and economies of scale.

Transshipment problems with concave objective functions are difficult to solve to global optimality. Exact methods can cope only with small single-source networks. On the other hand, non-exact solution methods, used for larger networks and multi-source flows, are local search procedures which can get trapped in a local optimum.

Successful applications of tabu search (see, for example, Glover *et al.*[12], Hertz and de Werra[13], Sun and McKeown[14]), a metaheuristic method specifically designed to help escape entrapment in local optima, argue for investigating its potential usefulness in this difficult area of optimisation. A suitable neighbourhood and a fast move evaluation are required as the main ingredients of tabu search. In this work, the computational efficiency of adjacent extreme point search[10,15] (vertex following algorithm) is enhanced considerably, making it feasible to imbed it in a tabu search scheme. A second tabu search scheme, with moves that resemble those employed in Network Simplex algorithms, is also developed.

The minimum, concave-cost uncapacitated transshipment problem

The difficulty of minimisation problems with non-convex functions stems from the fact that they may have many local optima with no local criteria to give information about global optimality. In particular, the class of problems

$$\min f(x)$$
$$\text{subject to } x \in D$$

where $f(x)$ is a real concave function and D is a convex compact set, to which the minimum concave-cost uncapacitated transshipment problem (MCCUTP) belongs, remains NP-hard[16,17].

Correspondence; K. S. Hindi, Department of Manufacturing and Engineering Systems, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
* *On leave from Electrical and Electronics Engineering Department, Middle East Technical University, Turkey*

In the MCCUTP, set D is determined by network constraints and the problem can be described as follows. Given a directed graph $G = [X, U]$ consisting of a set X of N nodes and a set U of M ordered pairs of nodes (arcs), coupled with N-vectors $d^k = (d_i^k)$ (demand vector) for every $k \in K$, where K is the set of commodities, and a concave-cost function $\Phi_{(i,j)}(\psi_{(i,j)})$ for each arc (i, j), solve

$$(P) \quad \begin{cases} \min \Phi(\psi) = \sum\limits_{(i,j) \in U} \Phi_{(i,j)}(\psi_{(i,j)}) & \\ \text{subject to} & \\ \psi_{(i,j)} = \sum\limits_{k} \phi_{(i,j)}^k & \forall (i,j) \in U \\ A\phi^k = d^k & \forall k \in K \\ 0 \leqslant \psi_{(i,j)} & \forall (i,j) \in U \end{cases}$$

assuming all constraints and demands are integral. $\phi_{(i,j)}^k$ denotes the kth individual flow component on arc (i, j) and A is the node-arc incidence matrix of G. Nodes with $d_i^k < 0$ are sinks, nodes with $d_i^k > 0$ are sources and nodes with $d_i^k = 0$ are transshipment nodes. The flow originating from a single source k is regarded as a single commodity flow. In a consistent system, total source supply is equal to total sink demand for each commodity, i.e. $\sum_{i=1}^{N} d_i^k = 0 \ \forall k$. Each concave cost $\Phi_{(i,j)}$ is a function of the total (multi-commodity) flow $\psi_{(i,j)}$ on arc (i, j).

Overview of existing methods

The literature on global concave minimisation and concave minimum-cost network flow problems is very large. Surveys can be found in Magnanti and Wong[16], Guisewite and Pardalos[17] and Pardalos and Rosen[18].

For single-commodity problems, there exist some extreme point ranking methods[19], branch and bound procedures[9,20,21] and dynamic programming methods[22]. However, these algorithms are capable of solving small or medium-size networks only.

For uncapacitated multi-commodity networks, approximate algorithms appear to be the only feasible option. Most effective among these are the algorithms suggested by Yaged[2], Gallo and Sodini[10,15] and Minoux[23,24].

Yaged's is a fixed-point algorithm based on successive approximation which converges in a finite number of steps to a local optimum (see Appendix). It is simple and computationally efficient. However, the local minimum obtained is highly dependent on the starting solution, i.e. the algorithm strongly tends to produce an extreme point which, in a sense, is 'the closest' to the starting point. Therefore, the result is good only if the starting solution is already a good approximation.

Although suggested for undirected networks, Minoux's greedy algorithm is also applicable to directed networks (see Appendix).

In the above linearisation approaches, the property that an optimal solution lies on an extreme point of the feasible set is not fully utilised. Thus at termination, neighbouring extreme points could provide improved solutions. Methods to improve solutions by considering such points have been suggested[3,10,15,25]. The method of Gallo and Sodini[10,15] will be discussed at some length in this work.

Overview of tabu search

Although the roots of tabu search date back to the late 1960s and early 1970s, it was proposed in its 'modern form' and publicised by Glover[26]. A good introduction can be found in Glover and Laguna[27]. An overview of the method is given below.

Tabu search is an iterative method for finding in a (finite or infinite) set X of feasible solutions, a solution s^* which minimises a real valued objective function f. A neighbourhood $N(s)$ is defined for each solution $s \in X$. Starting from an initial feasible solution, the neighbourhood $N(s^c)$ of the current solution s^c is examined and the solution s' with the best objective function is chosen as the next solution; i.e. $s' \mid f(s') < f(s''), s', s'' \in N(s^c)$. The process could be conceived of as one of steepest descent, mildest ascent. The fact that movement from an s^c to an $s' \in N(s^c)$ is allowed even if $f(s') \geqslant f(s^c)$ helps escape from local optima.

However, with the above scheme cycling is possible. To prevent cycling, a structure called *tabu list* T of length t (fixed or variable) is introduced to prevent going back to a solution visited in the last t iterations; other ingredients of Tabu search, such as aspiration criteria and intensification and diversification schemes, serve to enhance its efficiency and productivity.

The Tabu search process stops when the solution is close enough to a lower bound of f, if known; otherwise, it stops when no improvement occurs over the best solution for a certain number of iterations.

ENHANCED ADJACENT EXTREME-POINT SEARCH

In this section, the adjacent extreme point search method of Gallo and Sodini[10,15] is presented. An enhancement which considerably improves computational efficiency is also developed. This improvement makes it feasible to imbed the method within a tabu search scheme.

In order to define a neighbourhood and permissible moves for a steepest descent adjacent extreme point search, relevant facts from the theory of extremal flows are summarised.

Let S_k denote the polyhedron (set of feasible points) defined by the constraints for the k-th commodity and let V_k denote the set of vertices of S_k. A flow $\phi^k \in V_k$ is called an *extremal flow* and a flow pattern $\xi = (\phi^1, \phi^2, \ldots, \phi^{|K|})$ with $\phi^k \in V_k \; \forall k$ is called an *extremal flow pattern.*

When not empty, the set of optimal solutions of problem (P) contains extremal flow patterns[5]. Therefore, to define a neighbourhood, it is sufficient to consider only such patterns. Let $Adj(\phi^k)$ denote the set of all extreme flows adjacent to a given extreme flow ϕ^k. Given an extremal flow pattern ξ, a neighbourhood $N_h(\xi)$ is the set of all the flow patterns $\bar{\xi} = (\bar{\phi}^1, \bar{\phi}^2, \ldots, \bar{\phi}^{|K|})$ such that $\bar{\phi}^h \in Adj(\bar{\phi}^h)$, and $\bar{\phi}^k = \phi^k$ for $k \in K \neq h$ for some $h \in K$.

The following theorems, stated without proof, characterise the extreme points and their neighbourhoods, $\forall U' \subseteq U$, let $g(U') = [N(U'), U']$, where $N(U') = \{i, j \in X \mid (i, j) \in U' \text{ or } (j, i) \in U'\}$, then

Theorem 1[5,28]

A flow ϕ^k is an extreme flow if and only if $g[U_1(\phi^k)]$ is a tree with the source as root and the sinks as terminal nodes, where $U_1(\phi^k) = \{(i, j) \in U \mid \phi^k_{(i, j)} > 0\}$.

Theorem 2[15]

Two extreme flows ϕ^k and $\bar{\phi}^k$ are adjacent vertices of S_k if and only if $g[U_1(\phi^k) \cup U_1(\bar{\phi}^k)]$ does contain a unique cycle.

An important implication of Theorem 2 is that if ϕ^k and $\bar{\phi}^k$ are adjacent vertices of S_k then the arcs of $U_1(\bar{\phi}^k)$, which do not belong to $U_1(\phi^k)$, constitute a path connecting two nodes of $N(U_1(\phi^k))$ without containing any other node of $N(U_1(\phi^k))$ as an intermediate node (desired path property). Thus to move from one vertex ϕ^k to an adjacent one $\bar{\phi}^k$, an existing arc can be removed and a path with the desired path property added such that the flow constraints are not violated and the supply-demand structure is not changed. For example, in Figure 1, removing the arc (r, v) and adding the path shown between p and v constitutes a move.

To calculate the total change in the objective function value resulting from a move, note that when $\phi^k \in V_k$ and $\bar{\phi}^k \in V_k$ are adjacent, the unique cycle in $g[U_1(\phi^k)] \cup g[U_1(\bar{\phi}^k)]$ can always be decomposed into three paths; π_{pv}, π_{qv} and π_{qp} as shown in Figure 1. The change due to the flow amount $\phi^k_{(r, v)}$ entering node v being sent to node p instead of node r from the join q is a constant independent of the path selected between nodes p and v. It can be calculated as:

$$c^k_v(p) = \sum_{(i, j) \in \pi_{qv}} [\Phi_{(i, j)}(\psi_{(i, j)} - \phi^k_{(r, v)}) - \Phi_{(i, j)}(\psi_{(i, j)})] + \sum_{(i, j) \in \pi_{qp}} [\Phi_{(i, j)}(\psi_{(i, j)} + \phi^k_{(r, v)}) - \Phi_{(i, j)}(\psi_{(i, j)})]$$

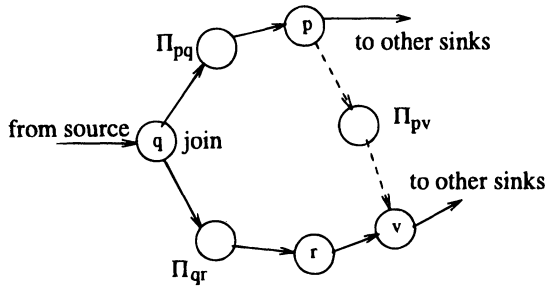

FIG. 1. *Different segments of a cycle.*

Also the change incurred when the new path π_{pv} is used by the flow amount $\phi^k_{(r,\,v)}$ is

$$c^k_v(\pi_{pv}) = \sum_{(i,\,j)\,\in\,\pi_{pv}} \Phi_{(i,\,j)}(\psi_{(i,\,j)} + \phi^k_{(r,\,v)}) - \Phi_{(i,\,j)}(\psi_{(i,\,j)}).$$

Thus the total change in the objective function is given by $c^k_v(p) + c^k_v(\pi_{pv})$.

When an arc (r, v) is to be deleted, there may be a number of alternative paths with the desired path property leading to v. The set of such paths, P_v, may also be empty for some v. The best path $\pi^*_{pv} \in P_v$, which results in the minimum cost adjacent extreme flow, can be obtained by solving the following problem:

$$\Delta^k_v = \min\{c^k_v(p) + c^k_v(\pi_{pv}) \mid \pi_{pv} \in P_v\},$$

which is, in fact, a shortest path problem from node 0 to node v, on a specially derived graph.

Gallo and Sodini[10] derive such a graph (see Figure 2) from the cotree by deleting all the arcs incident onto nodes $N_v = N(U_1(\phi^k)) - \{v\}$, and adding arcs from an artificial source node 0 to each node $p \in N(U_1(\phi^k))$, obtaining a graph G', and then assigning lengths in G' as:

$$C_{(i,\,j)} = \begin{cases} c^k_v(j) & \text{if } i = 0 \\ \Phi_{(i,\,j)}(\psi_{(i,\,j)} + \phi^k_{(r,\,v)}) - \Phi_{(i,\,j)}(\psi_{(i,\,j)}) & \text{otherwise} \end{cases}$$

The value of the shortest path from 0 to v then gives the total change in the objective function value which would result from redirecting the flow $\phi^k_{(r,\,v)}$ entering into node v, and

$$\Delta^k = \min\{\Delta^k_v \mid v \in N(U_1(\phi^k))\}$$

gives the maximum possible objective function value improvement that can be achieved by moving to an adjacent vertex after changing only the k-th commodity tree.

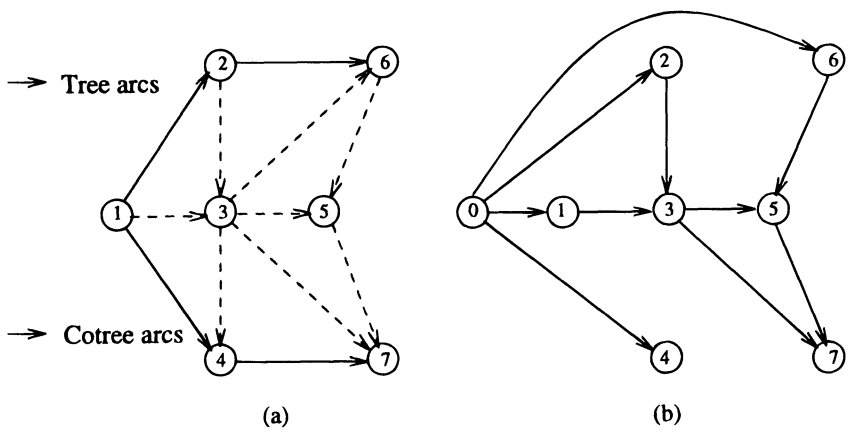

FIG. 2. *(a) An example network and solution (b) Special graph G' for Δ^1_7.*

Thus given an extremal flow pattern, the set of all possible next moves to an adjacent extremal flow pattern is given by

$$V = \{(k, \lambda, \delta^p, \pi^*_{pv})\ \forall k, \lambda, p\}$$

where the 4-tuple $(k, \lambda, \delta^p, \pi^*_{pv})$ represents a single move, with k representing a commodity, $\lambda = (., v)$ an arc and δ^p the objective function value change obtained by deleting λ and redirecting the flow through the best path π^*_{pv}.

To enhance the efficiency of the vertex following search method of Gallo and Sodini, described above, it is worth noting that when calculating $c^k_v(p)\ \forall p$, all nodes in $N(U_1)$ contained in the subtree rooted at v, if any, need not be included in graph G'.

Gallo and Sodini's method has a complexity of $O(|K|N^3)$ for one iteration[10]. However, to increase computational efficiency in practice, it is necessary to arrange computation such that shortest path calculations are carried out on positive-weight graphs. Fortunately, this is possible, as explained below.

To construct an alternative complementary graph G'' from the cotree (Figure 3), we start, as in Gallo and Sodini[10], by deleting the arc (r, v) and all arcs incident onto nodes $N_v = N(U_1(\phi^k)) - \{v\}$, as well as all nodes in $N(U_1)$ contained in the subtree rooted at v, if any. The direction of the remaining arcs is then reversed, while assigning them lengths:

$$C_{(i,j)} = \Phi_{(i,j)}(\psi_{(i,j)} + \phi^k_{(r,v)}) - \Phi_{(i,j)}(\psi_{(i,j)}) \tag{1}$$

At this point, shortest paths $c^k_v(\pi^*_{vp})$ from v to all nodes in the resulting positive-weight graph G'' are computed. Now, the value Δ^k_v is calculated by addition and comparison as:

$$\Delta^k_v = \min_{p\,|\,c_v{}^k(\pi_{vp}{}^*) < \infty} c^k_v(p) + c^k_v(\pi^*_{vp})$$

It is worth noting that $c^k_v(p)$ need be calculated only for nodes $p\,|\,c^k_v(\pi^*_{vp}) < \infty$, leading to further saving in comparison with Gallo and Sodini[10].

When the graph is sparse and is defined by the successors of each vertex, the binary-heap implementation of Dijkstra's algorithm for positive-weight shortest paths requires a maximum time of $O(M \log N)$[29], leading to an overall $O(|K|NM \log N)$ complexity for one iteration of the search. Here, the binary-heap implementation of Dijkstra's algorithm is used whenever a shortest path computation is required.

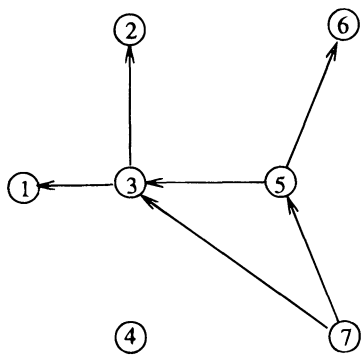

FIG. 3. *G″ for deleting arc (4, 7) for the network in Figure 2a.*

It is now possible to state the search scheme as the following algorithm:

Algorithm

Step 1. Given an extremal flow pattern ξ, initialise

$$c = 0$$

$$\psi^0_u = \sum_k \phi^{k,0}_u\ \forall u$$

Step 2.

$c = c + 1$

repeat $\forall k$

repeat $\forall u = (r, v) \in U_1(\phi^{k,c})$

Form the complementary graph G''

Assign costs $C_{(i,j)}$ as in equation (1)

Find the shortest path π^*_{vp} from node v to every other node $p \in G''$

$\forall p \in G''$ if $\pi^*_{vp} < \infty$ then compute $c^k_v(p)$

$\Delta^k_v = \min\{c^k_v(p) + c^k_v(\pi^*_{vp}) \mid p \in G''\}$

Compute Δ^k

Step 3. If $\Delta^k \geqslant 0$, stop else move to the state corresponding to the move associated with Δ^k generating ψ^c and go to step 2.

Most of the computational effort is expended on constructing G'' and the corresponding nonlinear function evaluations. Thus the sequence in which arcs $(r, v) \in U_1(\phi^{k,c})$ are selected in step 2 is crucial for overall computational efficiency. If arc (., r) is selected after arc (r, v), then minimal modification is required in G''; namely the following steps are carried out in the stated order:

1. delete arcs leaving v.
2. add, reverse the direction of, and compute $C_{(i,j)}$ for arcs incident onto r in G, except arc (., r) itself.
3. delete all nodes in the subtree rooted at r in $U_1(\phi^k)$ and the arcs incoming to such nodes.

Moreover, if $\phi^k_{(.,r)} = \phi^k_{(r,v)}$ then $C_{(i,j)}$ remains the same for all arcs existing in G'' prior to modification.

Therefore, in step 2, the following sequence is adopted: starting from a leaf v, climb upwards until the root of a previously encountered node is reached and repeat until all arcs are processed. For each leaf node, G'' and the corresponding weights $C_{(i,j)}$ are calculated from scratch; otherwise G'' is modified appropriately and the necessary $C_{(i,j)}$ calculations are performed.

A reachability matrix calculated beforehand also helps to isolate nodes that can never reach to a particular node v, effectively reducing the size of G'' leading to further efficiency.

The combination of the use of a smaller graph G'' with its positive-weight arcs, rather than graph G', the elimination of nodes of the subtree rooted at v in $N(U_1)$, the reduction in the number of $c^k_v(p)$ calculations, and appropriately sequencing the processing of arcs in step 2 of the algorithm, leads to a much enhanced vertex following steepest descent search. This is borne out by the results of computational testing reported in the last section.

TABU SEARCH SCHEMES

Two search schemes have been used; differing in the definitions of neighbourhood and move. Both, however, share the following characteristics.

A simple tabu list is used to implement a recency-based memory function. Each entry of the list corresponds to a move and the move attributes recorded are the commodity and the added arcs. Removal of the arcs from the tree corresponding to the commodity concerned remains tabu for as long as the move remains on the tabu list. When the tabu list is full, the oldest move is removed before adding a new move. Thus a move remains tabu for a number of iterations equalling the length of the tabu list t. The choice of t is crucial. Static rules choose a value that remains fixed throughout the search, while dynamic rules allow t to vary. Reportedly, the latter are more robust[30]. In the present work, both static and dynamic tabu lists are investigated. For the latter, a simple dynamic rule is adopted by choosing $t_{\min} = \lfloor N/8 \rfloor$ and $t_{\max} = N$ and varying t randomly

between t_{min} and t_{max} every t_{max} non-improving iterations. This choice may not be the best, but appears to be effective for the range of problems tested. Appropriate choice of the size of the tabu list is still an open question and choices different from those mentioned above can also be used.

Aspiration criteria are normally introduced to determine when tabu restrictions can be overridden in order to enable the search to achieve its best performance. In the present work, aspiration by objective was used so that removal of an arc on the transshipment tabu list is permitted if it leads to an objective function better than that of the best solution found so far. The increase in the number of shortest path computation due to aspiration is fully justified by the improvement in the quality of solutions found.

The stopping criterion used was to stop calculation when no improvement occurs over the best solution for a number of iterations. Appropriate choice of this number is investigated in our computational experiments.

In many cases, repeating the tabu search process from a different starting point each time (multiple starts) leads to significant improvement in solution quality. In the present work, the tabu search is repeated twice for some problem instances as described in the last section.

Scheme I

The first tabu scheme is based on a neighbourhood definition and move calculations of the type employed in vertex following search. The latter always makes the best next move, continuing for as long as there is improvement in the objective function value (steepest descent). In contrast, in the tabu search scheme, the process is one of steepest descent, mildest ascent with the set of *admissible* moves consisting of those possible moves that are non-tabu or otherwise satisfy the aspiration criterion. Thus tabu search continues for much longer, making it essential for computational feasibility to implement moves in the manner of the enhanced vertex following search scheme developed here.

Scheme II

The definition of neighbourhood in this scheme is inspired by the type of moves normally employed in Network Simplex methods. As before, let $g[U_1(\phi^k)]$ be the tree associated with an extreme flow ϕ^k and let $X' = \{i, j \in X \notin N(U_1)$ and i, j reachable from the root$\}$. If the tree is extended to connect the nodes in X' by adding appropriate arcs with zero flow, then the result is a tree rooted at the source spanning all nodes reachable from it. An incoming cotree arc to a disconnected node is appropriate (for example arcs (q, c), (q, e), (b, d) and (e, f) in Figure 4) and selection of only appropriate arcs for the extension of the tree maintains reachability from the root for all nodes. An adjacent extreme flow is then defined by removing an arc with strictly positive flow (i, j) and replacing it by a cotree arc $(., j)$. For example, with reference to Figure 4, removing arc (r, v) and replacing it by the cotree arc (f, v) produces an adjacent extreme flow. Thus the neighbourhood of an extremal flow pattern is defined as consisting of the extremal flow patterns that could be reached by a move of the type just described, i.e. by replacing an extremal flow for

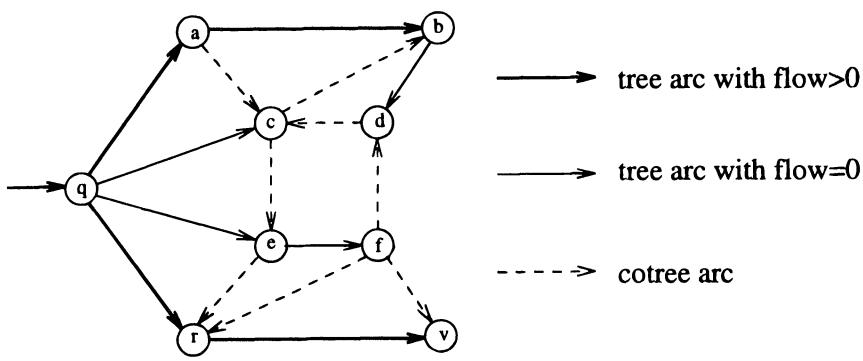

FIG. 4. *Spanning tree including arcs with zero flow.*

one commodity by an adjacent extremal flow maintaining a spanning tree structure for each commodity.

With this definition of neighbourhood, a move can be described as follows:

1. From the current tree arcs, choose one with a strictly positive flow, i.e. choose an arc (i, j) $\mid \psi^k_{(i,j)} > 0$.
2. Delete this arc and substitute it with the cotree arc $(., j)$ which would after redistribution of flow lead to the largest improvement of (or least deterioration in) the objective function value.

The change due to rerouting the flow amount $\phi^k_{(r,v)}$ using cotree arc (f, v) can be calculated as:

$$\Delta = \sum_{(i,j) \in \pi_{qv}} [\Phi_{(i,j)}(\psi_{(i,j)} - \phi^k_{(r,v)}) - \Phi_{(i,j)}(\psi_{(i,j)})] + \sum_{(i,j) \in \pi_{qf}} [\Phi_{(i,j)}(\psi_{(i,j)} + \phi^k_{(r,v)}) - \Phi_{(i,j)}(\psi_{(i,j)})] + \Phi_{(f,v)}(\psi_{(f,v)} + \phi^k_{(r,v)}) - \Phi_{(f,v)}(\psi_{(f,v)})$$

The neighbourhood thus defined is restricted. To see this, consider the example of Figure 4. The only path with the desired path property that will be considered in the neighbourhood of v is $q - e - f - v$, to the exclusion of three other paths; namely, $q - c - e - f - v$, $a - c - e - f - v$ and $b - d - c - e - f - v$. In fact, at most $|\Gamma_v^{-1}| - 1$ elements of the set of paths having the desired path property, P_v, can be considered, where Γ_v^{-1} is the set of predecessors of node v.

When there are few sinks and consequently, flow-carrying tree arcs are few, the number of alternative paths included in the restricted neighbourhood can be very small indeed. The converse is also true. Thus the restricted neighbourhood definition can be expected to be effective when the number of sinks is substantial for each commodity.

It is essential for computational efficiency to adopt appropriate data structures for representing trees. The data structures used in modern transshipment codes (parent-children, depth, preceding arc) are appropriate[31]. Backward star representation of the network was also adopted in order to access directly all arcs leading to a node; thus facilitating carrying out the second step of a move.

COMPUTATIONAL RESULTS

Three existing efficient methods found in the literature; namely, Yaged's, Minoux's greedy and Gallo and Sodini's were implemented. The enhanced vertex following search developed here was also implemented, as well as tabu search schemes I and II described above. All implementations were in Pascal and computations were carried out on an HP 735 workstation running under UNIX.

A total of 160 test problems were generated (see Appendix). These were organised in four classes (see Table 1) of 40 problems each. Each class, in turn, is divided into four equal groups, with the first involving 2 commodities; the second, 3; the third, 4 and the fourth, 5. In all experiments, the concave cost of each link is assigned as $\Phi_u = l_u \psi_u^{\alpha_u}$ $\forall u \in U$. In some experiments, all exponents α_u are equal; in others, they vary from link to link. The first case is indicated in the tables by quoting the value of the exponent ($\alpha_u = 0.2$, $\alpha_u = 0.4$, $\alpha_u = 0.6$, $\alpha_u = 0.8$), while in the second case, the range of variation is given ($0.2 \leqslant \alpha_u \leqslant 0.8$).

Figure 5 and Table 2 present the timing results for the Gallo and Sodini algorithm as compared to the enhanced algorithm developed in this work. The averages are taken over all α_u values. The

TABLE 1. *Characteristics of test problems*

Class	Problem size ($N \times M$)	No. of Sinks	No. of Transshipment nodes
1	20 × 80	10	10
2	25 × 100	12	12
3	34 × 122	17	17
4	48 × 174	24	24

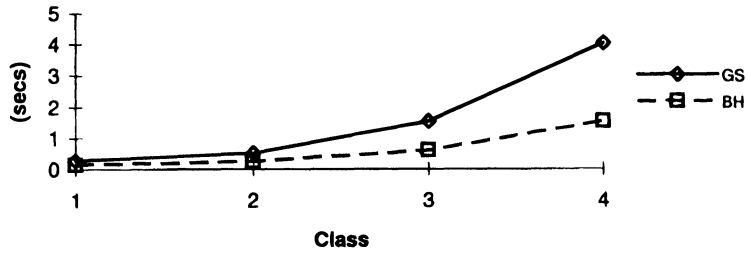

FIG. 5. *Average CPU times of vertex following algorithm over all* $|K|$ *and* α_u.

results show that computational speed is increased considerably with the ratio ranging from an average of 1.90 for the class of smallest networks to 2.63 for the class of largest networks with an average of 2.50 over all classes.

Starting from an initial solution, different combinations of heuristics can be used to reach a local minimum. The combinations chosen in the present work are depicted in Figure 6.

Figure 7 and Tables 3 and 4 show the average improvements achieved by the different heuristics over the starting solution. All percentage improvements are calculated as $100 * (S - H)/S$, where S is the cost of the starting solution and H is the cost of the solution achieved by applying the

TABLE 2. *Average CPU times of vertex following algorithm (s)*

Class	$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$		$\|K\| = 5$	
	GS	BH	GS	BH	GS	BH	GS	BH
1	0.12	0.06	0.20	0.10	0.33	0.17	0.40	0.22
2	0.22	0.11	0.42	0.21	0.64	0.32	0.76	0.40
3	0.57	0.21	1.28	0.48	1.87	0.76	2.42	0.99
4	2.06	0.69	3.44	1.27	4.91	1.87	5.73	2.29

GS: Gallo and Sodini; BH: Enhanced vertex following

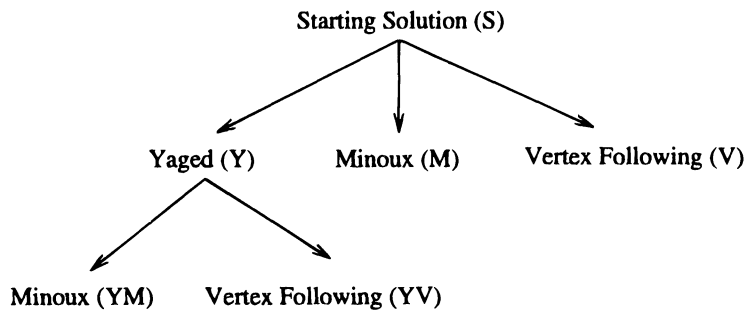

FIG. 6. *A subset of heuristic combinations.*

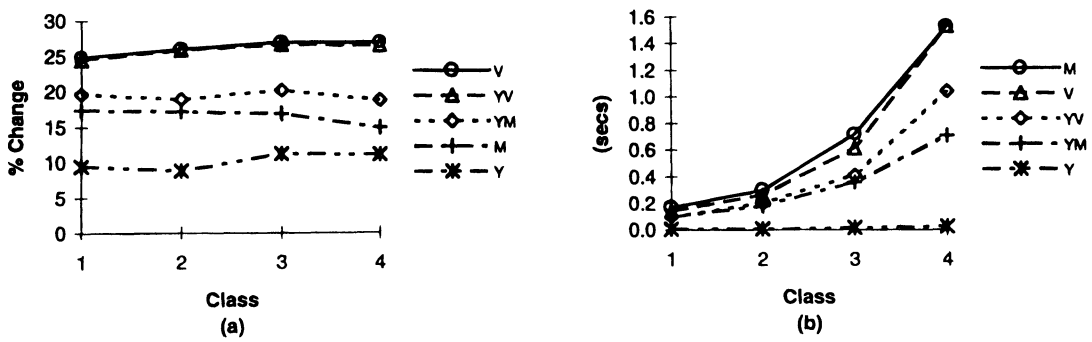

FIG. 7. *(a) Average improvements over starting MDR solution (b) Average CPU time.*

TABLE 3. *Average percentage improvements over starting solution*

Class	$\alpha_u = 0.2$					$\alpha_u = 0.4$				
	Y	M	YM	YV	V	Y	M	YM	YV	V
1	14.5	28.8	31.8	36.6	37.2	8.7	13.7	16.7	22.9	23.2
2	13.1	27.7	30.0	38.3	38.8	7.2	11.9	14.3	23.3	23.6
3	18.7	29.7	33.7	40.7	41.0	10.7	13.0	17.7	26.6	27.0
4	18.9	26.7	31.9	39.4	40.0	10.6	10.7	16.1	25.8	26.8

Class	$\alpha_u = 0.6$					$\alpha_u = 0.8$					$0.2 \leqslant \alpha_u \leqslant 0.8$				
	Y	M	YM	YV	V	Y	M	YM	YV	V	Y	M	YM	YV	V
1	3.6	4.5	6.2	10.5	10.7	0.8	0.7	1.3	2.5	2.5	19.5	38.7	41.8	49.8	50.3
2	2.9	3.6	5.1	10.4	10.2	0.7	0.6	1.1	2.3	2.3	19.9	41.8	44.0	55.0	55.2
3	4.4	3.9	6.8	12.7	12.6	1.1	0.6	1.5	3.0	3.0	20.6	36.9	40.9	49.4	50.7
4	4.7	3.0	6.4	12.6	12.9	1.1	0.4	1.3	3.0	3.0	19.9	33.6	37.6	51.3	51.6

TABLE 4. *Averages over all* α_u

Class	Improvements (%)					Timings (s)				
	Y	M	YM	YV	V	Y	M	YM	YV	V
1	9.4	17.3	19.6	24.5	24.8	0.00	0.17	0.10	0.10	0.14
2	8.8	17.1	18.9	25.8	26.0	0.01	0.30	0.18	0.20	0.26
3	11.1	16.8	20.1	26.5	26.9	0.01	0.72	0.36	0.40	0.61
4	11.0	14.9	18.7	26.4	26.9	0.02	1.53	0.71	1.04	1.53

relevant heuristic algorithm. A minimum distance routeing (MDR) for each source-sink pair computed on G with lengths assigned as l_u is used as the starting solution, where $\Phi_u = l_u \psi_u^{\alpha_u} \forall u \in U$.

Yaged's algorithm, with average linearisation until convergence, followed by linearisation using marginal cost, proved to be effective in obtaining better solutions from the starting solution (column Y in Tables 3 and 4). Table 3 presents the average improvements over the starting MDR solutions.

However, Minoux's greedy algorithm, which is based on a different neighbourhood definition and stronger optimality conditions, has the potential to improve considerably on solutions obtained by Yaged's (column YM in Tables 3 and 4). Application of Minoux's greedy algorithm directly from the starting solution does not give as good results (column M in Tables 3 and 4).

To the authors' knowledge, no report exists in the literature about the performance of Minoux's greedy algorithm relative to the vertex following algorithm. Comparison of columns YM and YV in Tables 3 and 4 reveals that the latter can find much better results for the class of transshipment problems under consideration. Only in 30 out of 800 problems solved, did Minoux's greedy algorithm outperform the vertex following algorithm and then only by a small margin.

Although Gallo and Sodini recommend that vertex following search is started from a good initial solution provided by Yaged's algorithm, our experience (compare columns YV and V in Tables 3 and 4) shows that applying vertex following search directly to the starting solution gives as good results on average and for some problem instances better results.

With increasing α_u, the problem converges to the linear case and the starting MDR solution chosen as described above approaches the optimum; consequently lowering the improvement levels obtained by applying the algorithms under consideration (Table 3). Figure 7 and Table 4 compare average improvements and timings over all α_u.

The results above show that the largest improvement is achieved by employing vertex following steepest descent either from the starting solution or from an initial solution provided by Yaged's algorithm. In assessing the proposed tabu search schemes in the sequel, the base for comparison is considered to be the better solution of the two.

The double start strategy for tabu search was implemented in the following way. An initial solution (V), with an objective function value $f(\mathrm{V})$, is provided first by vertex following from the starting solution. When the tabu search satisfies the stopping criterion, another start is attempted

by generating a solution (YV), with an objective function value f(YV), through applying Yaged's algorithm to the starting solution followed by vertex following. If (YV) turns out to be different from (V), tabu search is resumed from YV; otherwise the process is terminated.

In order to investigate the effect of using different problem characteristics (problem size, number of commodities and α_u) and different algorithm parameters (tabu list size and number of non-improving iterations allowed by the stopping criterion) systematically, an experiment to account for various combinations was designed. Given the large number of problems required to fully investigate all possible combinations, a two-phase analysis was performed. In the first phase, characteristics and parameters were analysed by varying them one at a time from nominal values. The nominal values were chosen to be Class 3 as problem size, 3 commodities, variable α_u ($0.2 \leqslant \alpha_u \leqslant 0.8$), a tabu list size of $\lfloor N/2 \rfloor$ and an allowed non-improving iteration count ($NICnt$) of 300.

Tabu search scheme I was able to find better solutions for 132 problem instances out of 160 for nominal values. Percentage improvements, calculated in the same way as for Tables 3 and 4, average less than 4%; significantly, however, there are problem instances where improvement levels are over 14% (Figure 8 and Table 5). In view of the large investments involved in some applications, the possibility of achieving savings of such magnitudes cannot be ignored.

Scheme II achieved improvements in 122 problem instances. The results produced are slightly worse than with scheme I on average (Figure 8 and Table 6), but there are instances where performance deteriorates appreciably. Computation time, however, is much shorter (Tables 7 and 8 and Figure 9).

Problem difficulty is found to be inversely related to α_u. In the case where α_u is the same for all links, the success rates of both tabu schemes decrease with increasing α_u as mentioned earlier (Table 9). However, starting from an arbitrary solution, the tabu schemes are likely to produce better solutions regardless of α_u.

The static tabu list size was parametrised in the experiments by varying l in the formula: $t = \lfloor (N \times l)/8 \rfloor$. Also the dynamic tabu list strategy was implemented by varying t randomly between

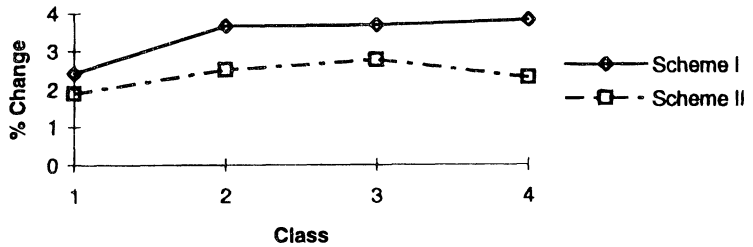

FIG. 8. *Average improvements of tabu schemes with nominal parameters over all* $|K|$.

TABLE 5. *Percentage improvement achieved by tabu scheme I*

Class	$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$		$\|K\| = 5$	
	Max	Ave.	Max	Ave.	Max	Ave.	Max	Ave.
1	8.49	2.37	11.85	4.47	8.10	1.72	5.34	1.09
2	14.25	5.22	12.36	4.51	9.49	2.18	7.92	2.75
3	8.21	3.40	6.40	3.90	11.88	4.08	10.85	3.35
4	12.33	5.38	9.17	3.73	10.15	2.50	8.66	3.62

TABLE 6. *Percentage improvement achieved by tabu scheme II*

Class	$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$		$\|K\| = 5$	
	Max	Ave.	Max	Ave.	Max	Ave.	Max	Ave.
1	6.06	2.26	8.38	3.01	8.10	1.40	5.34	0.88
2	12.29	3.31	10.04	3.17	9.49	2.15	4.78	1.40
3	8.21	3.28	6.32	2.87	4.62	1.82	10.69	3.06
4	12.33	2.50	7.50	3.14	10.03	1.97	6.55	1.60

TABLE 7. *Average CPU times of tabu scheme I (s)*

Class	$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$		$\|K\| = 5$	
	Max	Ave.	Max	Ave.	Max	Ave.	Max	Ave.
1	4.02	2.25	6.82	3.39	6.82	4.70	7.62	5.57
2	6.13	3.93	10.74	7.78	9.18	9.18	17.97	10.35
3	14.95	7.97	24.70	16.35	24.70	17.86	33.25	21.28
4	23.15	14.75	39.61	26.72	53.64	36.33	65.18	48.72

$t_{\min} = \lfloor N/8 \rfloor$ and $t_{\max} = N$ every $t_{\max}$ non-improving iterations. The success of tabu scheme I improved somewhat with increasing l but no such trend appeared for scheme II (Table 10).

The number of non-improving iterations allowed before termination affects the CPU time of both tabu schemes, but its effect on solution quality proved to be minor, provided it ranges between 200 and 300 (Table 11).

In the second phase of the analysis, a full factorial experiment was conducted using the same problem characteristics and algorithm parameters. However, these parameters were allowed to take at most four different values: small, medium, high and variable where appropriate. For example, Class 2 (small), Class 3 (medium) and Class 4 (high) problems were used. Other parameter values were 2 (small), 3 (medium) and 4 (high) for commodity size; $\alpha_u = 0.2$ (small), $\alpha_u = 0.4$

TABLE 8. *Average CPU times of tabu scheme II (s)*

Class	$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$		$\|K\| = 5$	
	Max	Ave.	Max	Ave.	Max	Ave.	Max	Ave.
1	1.49	0.80	2.06	1.29	2.53	1.85	2.78	2.04
2	2.17	1.43	3.36	2.64	3.22	3.22	4.93	3.17
3	3.28	2.26	7.27	4.68	7.37	5.56	9.48	7.12
4	6.62	4.55	12.89	8.33	11.75	10.07	15.99	12.40

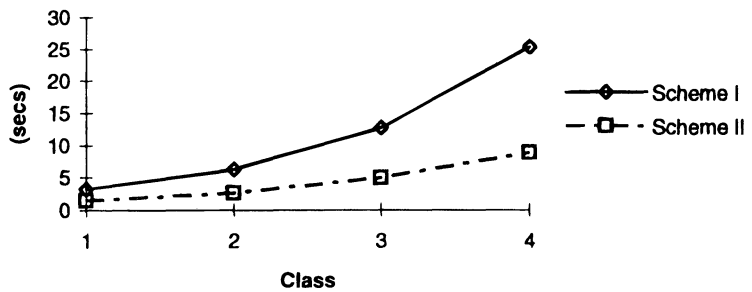

FIG. 9. *Average CPU times for tabu schemes with nominal parameters over all* $|K|$.

TABLE 9. *Average percentage improvements of tabu schemes for different* α_u

	$\alpha_u = 0.2$	$\alpha_u = 0.4$	$\alpha_u = 0.6$	$\alpha_u = 0.8$	$0.2 \leqslant \alpha_u \leqslant 0.8$
Scheme I	7.22	3.36	0.85	0.09	3.90
Scheme II	6.56	2.31	0.62	0.07	2.87

TABLE 10. *Average percentage improvements of tabu schemes for different tabu list sizes*

	l								
	1	2	3	4	5	6	7	8	variable
Scheme I	2.29	3.05	3.55	3.90	3.96	3.82	3.90	3.59	3.92
Scheme II	1.03	2.68	3.11	2.87	2.92	3.18	2.43	2.61	3.22

tabu list size is determined by the formula $\lfloor (N \times l)/8 \rfloor$.

TABLE 11. *Average percentage improvements of tabu schemes for different NICnt*

	NICnt							
	50	100	150	200	250	300	350	400
Scheme I	3.70	3.79	3.82	3.87	3.88	3.90	3.90	3.90
Scheme II	2.29	2.68	2.87	2.87	2.87	2.87	2.87	2.90

(medium) and $0.2 \leqslant \alpha_u \leqslant 0.8$ (variable) for cost function exponent; 2 (small), 4 (medium) and 6 (high) for l in the formula for the tabu list size, in addition to dynamic length tabu list size (variable); and finally 100 (small), 200 (medium) and 300 (high) for allowed non-improving solutions (*NICnt*). The experiment required $3 \times 3 \times 3 \times 4 \times 3$ different combinations with ten repetitions on a total of 90 different problems.

The purpose of the experiment is to evaluate the main and combined effect of the experimental variables on both tabu schemes. A randomised block design, in which the first two experimental variables represented a direction of blocking was used. For each Class $\times$ $|K|$ combination, a set of 10 randomly generated problems was used to test the remaining factors. An analysis of variance confirmed that all the experimental variables have a highly significant effect on the performance of the tabu schemes. Due to blocking, the interaction effects between the Class $\times$ $|K|$ term and other factors are treated as part of the error in the analysis. For tabu scheme I, there are significant interactions between Class and $|K|$, Class and α_u, Class and l and finally $|K|$ and l. *NICnt* is the only factor independent to any interaction.

The presence of interactions indicates that the performance of tabu scheme I depends on the particular combinations of the different levels of the above factors. However, it is still possible to choose the optimum algorithm parameters from the significant interaction and main effect plots presented in Figures 10 and 11. A variable l, i.e. a dynamic tabu list and a large *NICnt* (maximum number of allowed non-improving solutions) lead to the best performance in general. However, it is likely that the marginal gain resulting from increasing *NICnt* will decrease progressively, although this has not been pursued by our experiment.

The analysis for tabu scheme II was similar, except that there is an insignificant interaction effect between $|K|$ and l and an insignificant main effect *NICnt* at 10% significance level. The mean improvement values in tabu scheme II were smaller than the corresponding ones in tabu

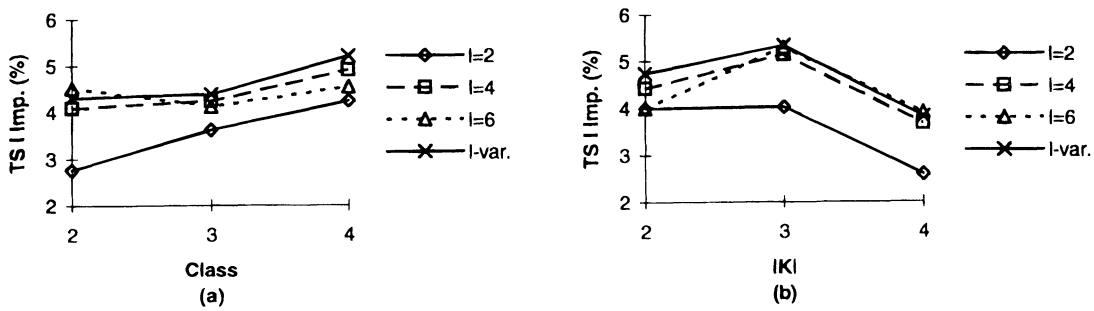

FIG. 10. *Average percentage improvement of tabu scheme I for different tabu list sizes.*

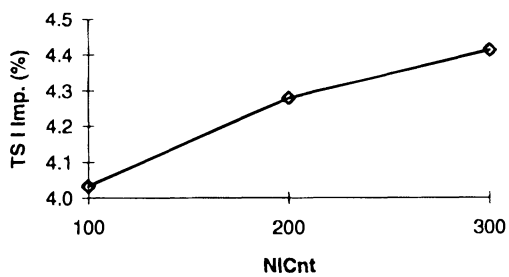

FIG. 11. *Average percentage improvement of tabu scheme I for different NICnt.*

scheme I. A variable tabu list size was also the best, but any of the values tested for *NICnt* seems to be equally valid.

SUMMARY AND CONCLUSIONS

Concave-cost, uncapacitated transshipment network problems arise in many applications. Such problems are difficult to solve to global optimality. Exact methods can cope only with small, single-source networks, while non-exact solution methods for larger networks with multi-source flows are local search procedures which can get trapped in a local optimum.

In this work, three of the most successful non-exact methods; namely Yaged's linearisation method, Minoux's greedy algorithm and Gallo and Sodini's method have been evaluated empirically. To the best of our knowledge, the comparison of these algorithms for the minimisation of directed concave-cost networks have not been reported previously. The latter two methods have been found to be superior to the first, with the last one clearly the most effective overall. According to the computational results, additional time devoted to obtaining a good initial solution for vertex following search (V) is not justified.

The effective method of Gallo and Sodini has been further developed in this work to enhance its computational efficiency considerably. This has been achieved through rationalising implementation details. The overall result has been approximately two and a half fold reduction in total computation time, making it possible to tackle larger networks.

The enhanced search algorithm has also been imbedded in a tabu search scheme which approved capable of finding better solutions, by the wide margin of 14% in some instances. Another tabu search scheme, somewhat inferior in terms of solution quality but computationally more efficient, is also developed to provide an alternative solution vehicle for larger networks. Apart from adopting a double start, the tabu search schemes employed are basic; it may be possible to achieve even better solutions using a tabu search with more sophisticated features[27].

APPENDIX

Problem generator

Test problems were generated using NETGEN[32], a software developed to generate standard random networks. The input parameters were selected so as to produce structurally different networks. The 'minimum and maximum costs', 'number of transshipment sources', 'percentage of high costs', 'percentage of capacitated arcs' and 'minimum and maximum upper bounds' parameters were set to zero. In each group of 10 problems, the 'NETGEN Random Number Seed' was started from 13502460 and incremented by one. The output of NETGEN was then modified to achieve feasible multi-commodity problems where the total supply is 1000 and the cost function Φ_u is in the form $\Phi_u = l_u \psi_u^{\alpha_u}$ with $10 \leqslant l_u \leqslant 100$ and $0 < \alpha_u < 1$.

Yaged's linearisation algorithm[2]

The objective function is linearised at the current solution and the resulting linear program is then solved as a series of shortest path problems. The process is repeated until convergence. If the cost functions $\Phi_{(i,j)}$ are differentiable, then a Taylor first-order linearised approximation (marginal cost) can be used, i.e. $\Phi_{(i,j)}(\psi_{(i,j)}) \leftarrow d\Phi_{(i,j)}(\psi^c_{(i,j)})/d\psi_{(i,j)}\, \forall (i,j) \in U$, where ψ^c is the feasible solution at the current iteration. Alternatively, an estimate of the average arc flow cost can be used where the cost function $\Phi_{(i,j)}(\psi_{(i,j)})$ is approximated by $\Phi_{(i,j)}(\psi^c_{(i,j)})/\psi^c_{(i,j)}$. The latter approximation can be more accurate when some of the cost functions $\Phi_{(i,j)}$ contain discontinuities.

Minoux's Greedy Algorithm[24]

At every iteration, for every arc λ with strictly positive flow, the saving Δ_λ that would be achieved by deleting the arc and rerouteing the flow through it is calculated. The arc deletion

leading to the largest saving is carried out and the process is repeated until no improvement is possible. Δ_λ is calculated in the following way. Lengths $l_u > 0$ are assigned to the edges of G as:

$$\begin{cases} l_\lambda = +\infty & \text{for edge } \lambda \\ l_u = \Phi_u(\psi_u^c + \psi_\lambda^c) - \Phi_u(\psi_u^c) & \forall u \neq \lambda \end{cases},$$

where ψ^c is the current multi-commodity flow on G. A shortest path with length $L(\psi^c, \lambda)$ is then found between the end points of λ and the saving is calculated as:

$$\Delta_\lambda = L(\psi^c, \lambda) - \Phi_\lambda(\psi_\lambda^c)$$

Acknowledgements—The authors wish to thank anonymous referees for their constructive comments and suggestions. The first author acknowledges the support of Middle East Technical University of Turkey.

REFERENCES

1. M. Florian (1986) Nonlinear cost network models in transportation analysis. *Math. Prog. Study* **26**, 167–196.
2. B. Yaged, Jr. (1971) Minimum cost routing for static network models. *Networks* **1**, 139–172.
3. N. Zadeh (1973) On building minimum cost communication networks. *Networks* **3**, 315–331.
4. I. Baybars and R. H. Edahl (1988) A heuristic method for facility planning in telecommunications networks with multiple alternate routes. *Naval Res. Logist.* **35**, 503–528.
5. W. I. Zangwill (1968) Minimum concave cost flows in certain networks. *Mgmt Sci.* **14**, 429–450.
6. W. I. Zangwill (1969) A backlogging model and a multi-echelon model of an economic lot size production system—a network approach. *Mgmt Sci.* **15**, 506–527.
7. W. I. Zangwill (1966) A deterministic multi-period production scheduling model with backlogging. *Mgmt Sci.* **13**, 105–119.
8. H. Konno (1988) Minimum concave cost production system: multi-echelon model. *Math. Prog.* **41**, 185–193.
9. R. M. Soland (1974) Optimal facility location with concave costs. *Opns Res.* **22**, 373–382.
10. G. Gallo and C. Sodini (1979) Concave cost minimization on networks. *Eur. J. Opl Res.* **3**, 239–249.
11. J. J. Jarvis, V. E. Unger, R. L. Rardin and R. W. Moore (1978) Optimal design of regional wastewater systems: a fixed charge network flow model. *Opns Res.* **26**, 538–550.
12. F. Glover, M. Lee and J. Ryan (1991) Least-cost network topology design for a new service: an application of tabu search. *Ann. Opns Res.* **33**, 351–362.
13. A. Hertz and D. de Werra (1990) The tabu search metaheuristic: how we used it. *Ann. Math. and Artificial Intelligence* **1**, 111–121.
14. M. Sun and P. G. McKeown (1993) Tabu search applied to the general fixed charge problem. *Ann. Opns Res.* **41**, 405–420.
15. G. Gallo and C. Sodini (1979) Adjacent extreme flows and application to minimum concave cost flow problems. *Networks* **9**, 95–121.
16. T L. Magnanti and R. T. Wong (1984) Network design and transportation planning: models and algorithms. *Trans. Sci.* **18**, 1–55.
17. G. M. Guisewite and P. M. Pardalos (1990) Minimum concave cost network flow problems: applications, complexity and algorithms. *Ann. Opns Res.* **25**, 75–100.
18. P. M. Pardalos and J. B. Rosen (1986) Methods for global concave minimization: a bibliographic survey. *SIAM Review* **28**, 367–379.
19. P. M. Pardalos (1988) Enumerative techniques for solving some nonconvex global optimization problems. *OR Spectrum* **10**, 29–35.
20. P. Rech and L. G. Barton (1970) A non-convex transportation algorithm. In *Application of Mathematical Programming Techniques* (E. M. L. Beale, Ed.) pp 250–260. The English Universities Press, London.
21. G. Gallo, C. Sandi and C. Sodini (1980) An algorithm for the minimum concave cost flow problem. *Eur. J. Opl Res.* **4**, 248–255.
22. R. E. Erickson, C. L. Monma and A. F. Veinott (1987) Send-and-split method for minimum concave cost network flows. *Math. Oper. Res.* **12**, 634–664.
23. M. Minoux (1976) Multiflots de cout minimal avec fonctions de cout concaves. *Annales des Telecommunications* **31**, 77–92.
24. M. Minoux (1989) Network synthesis and optimum network design problems: models, solution methods and applications. *Networks* **19**, 313–360.
25. G. Daeninck and Y. Smeers (1977) Using shortest paths in some transshipment problems with concave costs. *Math. Prog.* **12**, 18–25.
26. F. Glover (1986) Future paths for integer programming and links to artificial intelligence. *Comps and Opns Res* **13**, 533–549.
27. F. Glover and M. Laguna (1993) Tabu search. In *Modern Heuristic Techniques in Combinatorial Problems* (C. Reeves, Ed.) pp 70–150. Blackwell Scientific Publications, London.
28. M. Florian, M. Arthiat and D. De Werra (1971) A property of minimum concave cost flows in capacitated networks. INFOR **9**, 293–304.
29. R. E. Tarjan (1983) Data structures and network algorithms. In *CBMS-NFS Regional Conference Series in Applied Mathematics* **44** (R. E. Tarjan) pp 90–91. Society for Industrial and Applied Mathematics, Philadelphia.
30. E. Taillard (1991) Robust taboo search for the quadratic assignment problem. *Parallel Comp.* **17**, 443–455.

31. K. S. HINDI (1993) Efficient Implementation of Network Simplex Algorithms. *DTG Report* **93**(3) (update of DTG report 87(7)), Department of Computation, UMIST.
32. D. KLINGMAN, A. NAPIER and J. STUTZ (1974) A program for generating large scale capacitated assignment, transportation and minimum cost flow network problems. *Mgmt Sci.* **20**, 814–821.

Received August 1994; accepted February 1996 after two revisions