Journal of Global Optimization (2005) 33: 123–156
DOI 10.1007/s10898-004-3133-5

Global and Local Search Algorithms for Concave Cost Transshipment Problems

SHANGYAO YAN, DER-SHIN JUANG, CHIEN-RONG CHEN and WEI-SHEN LAI
Department of Civil Engineering, National Central University. Taiwan 32054, R.O.C. (e-mail: t320002@cc.ncu.edu.tw)

(Received 15 July 2003; accepted in revised form September 2004)

Abstract. Traditionally, the minimum cost transshipment problems have been simplified as linear cost problems, which are not practical in real applications. Recently, some advanced local search algorithms have been developed that can directly solve concave cost bipartite network problems. However, they are not applicable to general transshipment problems. Moreover, the effectiveness of these modified local search algorithms for solving general concave cost transshipment problems is doubtful. In this research, we propose a global search algorithm for solving concave cost transshipment problems. Efficient methods for encoding, generating initial populations, selection, crossover and mutation are proposed, according to the problem characteristics. To evaluate the effectiveness of the proposed global search algorithm, four advanced local search algorithms based on the threshold accepting algorithm, the great deluge algorithm, and the tabu search algorithm, are also developed and are used for comparison purpose. To assist with the comparison of the proposed algorithms, a randomized network generator is designed to produce test problems. All the tests are performed on a personal computer. The results indicate that the proposed global search algorithm is more effective than the four advanced local algorithms, for solving concave cost transshipment problems.

Key words: Concave cost, Genetic algorithm, Global search, Local search, Transshipment problems

1. Introduction

Transshipment problems are usually formulated as minimum cost network flow problems. Traditionally, in a minimum cost network flow problem the unit transit cost of each arc is assumed to have a fixed value. That is, the arc cost function is approximately linear. Although a mathematical model with a linear arc cost function is easier to solve, it may not reflect the actual transportation cost in real operations. In practice, the unit cost for transporting freight usually decreases as the amount of freight increases. The cargo transportation cost in particular is mainly influenced by the cargo type, the loading/unloading activities, the transportation distance, and the amount. In general, each transportation unit cost decreases as the amount of cargo increases, due to economy of scale in practice. Hence, in

actual operations the transportation cost function can usually be formulated as a concave cost function. Several cost function types that have arisen in concave cost network flow problems can be found in the literature, such as concave cost functions, piecewise concave cost functions, and linear fixed charge cost functions (Yan and Luo, 1998, 1999).

A minimum concave cost network flow problem is characterized as NP-hard (Garey and Johnson, 1979). It is difficult to find an optimal solution for a large-scale problem in a limited time. Traditionally, minimum concave cost network flow problems have usually been solved in the following three ways: (1) by solving a series of linear programming approximations to the original problem; (2) by local search methods designed to send an extreme flow by searching among candidate solutions; and (3) by meta-heuristic methods, such as the tabu search (TS).

Recently, Yan and Luo (1998, 1999) have employed the TS, threshold accepting (TA), and simulated annealing (SA) methods to develop several advanced local search algorithms to solve bipartite transportation network problems. It has been shown that these algorithms are more effective than the traditional linear approximation methods and local search methods. However, these advanced local search algorithms are not applicable to general transshipment problems. As well, they may lead to degeneracy problems when they are modified and applied to minimum concave cost network flow problems, which would lead to low efficiency in the search for an optimal solution. However, a genetic algorithm (GA), which is designed for global optimal solution searching, has led to good results in many applications (Goldberg, 1989). Therefore, here we utilize the GA to develop a global search algorithm for solving concave cost transshipment problems. Efficient encoding, initial population generation, as well as crossover and mutation strategies have been developed according to the problem's characteristics, to enhance the algorithm efficiency. To efficiently search for feasible solutions, a network flow algorithm that can efficiently modify an infeasible solution to become a feasible one has also been developed. Four efficient local search algorithms, based on the TA, the great deluge algorithm (GDA) and the TS, are also developed to evaluate the proposed global search algorithm.

To simplify the development of the algorithms, we will focus on pure (single commodity) network flow problem without arc flow upper bounds. Note that a network flow problem with arc flow upper bounds can be modified to become an equivalent network flow problem without arc flow upper bounds (Ahuja et al., 1993). Multi-commodity network flow problems with concave arc costs are expected to be more complicated than pure network flow problems and can be researched in the future.

The rest of this paper is organized as follows: First, we review the literature regarding the solution of concave cost network flow problems. Then

we formulate a mathematical model for a concave cost transship problem. Thereafter, an efficient global search algorithm and several advanced local search algorithms for solving the problem are developed. Many problem instances are generated and tested to evaluate the algorithm's performances. Finally, we give some conclusions and suggestions for future research.

2. Literature Review

In this section, previous research on the traditional concave cost transportation problems is reviewed first. Various meta-heuristics for solving combinatorial optimization problems are then discussed. Finally, the concepts to be used in a global search algorithm and four advanced local search algorithms are presented.

2.1. CONCAVE COST TRANSPORTATION PROBLEMS

Concave cost transportation problems are characterized as NP-hard (Larsson et al., 1994). In general, it is computationally expensive to find the optimal solution for a large-scale NP-hard problem. Many algorithms have been developed for solving these problems. Most approaches have employed the traditional mathematical programming techniques, such as linear approximation, Lagrangian relaxation, the sub-gradient method, the branch-and-bound method, and dynamic programming, to solve the problems (e.g. Zangwill, 1968; Rech and Barton, 1970; Yaged, 1971; Gallo and Sodini, 1979; Gallo et al., 1980; Hall, 1983; Blumenfeld et al., 1985; Jordan, 1986; Balakrishnan and Graves 1989; Thach, 1992; Guisewite and Pardalos, 1993; Larsson et al. 1994; Nourie and Guder 1994; Amiri and Pirkul, 1997). Note that Zangwill (1968) has pointed out the fact that the major difficulty in solving concave cost network flow problems comes from the existence of the enormous number of local optima in the search space. Consequently, the traditional approaches may be inefficient in terms of enumerating all the local optima to find the global optimum. Some algorithms based on the characteristics of concave cost network flows have been developed to improve the solution efficiency (e.g. Kuhn and Baumol, 1962; Suwan and Sawased, 1999; Yan and Luo 1999; Kim and Pardalos, 2000).

2.2. RECENT META-HEURISTICS FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Traditionally, the local search approach has been used to find the optimal solution for combinatorial optimization problems. However, there can be an enormous number of local optima in the solution set of this type of

optimization problem. A local search can easily fall into a local optimum, the objective of which is far away from the optimal one.

To overcome this weakness in the traditional local search method, several meta-heuristic algorithms have recently been developed, with a traditional local search as the core, combined with high-level meta-strategies for jumping out of the local optima found in neighborhood searches, thus finding a better solution (Osman and Kelly, 1996). The aforementioned high-level meta-strategies are mostly employed for the simulation of the behavior and logic of physics, chemistry, natural science, and human thinking.

Some examples of recent meta-heuristics include: the SA (Kirkpatrick et al., 1983), the TS method (Glover, 1989, 1990), the GA (Goldberg, 1989), the TA (Dueck and Scheuer, 1990), the GDA (Dueck, 1993), the record-to-record travel method (Dueck, 1993), the noising method (Charon and Hurdy, 1993), the neural network (Reeves, 1993), and the search space smoothing method (Gu and Huang, 1994). Note that the neural network algorithm and the GA differ from the other meta-heuristics which are classified as neighborhood search algorithms. In comparison with traditional local search algorithms, these algorithms can allow the search to escape from a local optimum and can usually lead to good results for many applications. Even more recently, some of these algorithms have been combined to produce more efficient hybrid algorithms. For example, Booker (1987) has incorporated a SA approach and a GA to develop an efficient hybrid algorithm.

In the past, these types of meta-heuristics were rarely applied to the solution of concave cost network flow problems. To the best of the authors' knowledge, only Yan and Luo (1998, 1999) have developed a family of algorithms combining TS, TA, and SA for solving concave cost bi-partite transportation network problems.

2.3. GENETIC ALGORITHMS

GAs, first proposed by Dr. John Hollard in 1975, have been applied widely in diverse fields such as engineering, economics, finance, manufacturing, and commerce. A detailed description of the method is given in Goldberg (1989). The most praised features of the GAs are their robustness, parallelism, flexibility, good problem-solving capability, and domain independent search. In general, their search range is wider than general neighborhood search algorithms (Reeves, 1997). Recently, there has being increasing amounts of research work adopting GAs for solving difficult combinatorial optimization problems (Gen and Cheng, 1997).

Simple GAs have been used to efficiently solve many well-known problems. However, these may not be suitable for application in other unknown fields (Kershenbaum, 1997). To improve the solution quality and efficiency

of the simple GA, extensive research has been devoted to the influence of the parameters, such as the population size, reproduction strategies, crossover, mutation, fitness function, and encoding schemes (e.g. see DeJong, 1975; Booker, 1987; Davis, 1989; Thierens and Goldberg, 1994; Rudolph, 1994; Reeves, 1997).

Although some strategies may speed up algorithm convergence, none of them can ensure that solutions will not be trapped into a local optimum. To improve this deficiency, and to increase the adaptability of the method, several hybrid GAs have been proposed. For example, Seiichi et al. (1995) has combined a GA and an SA to develop a GSA. Cheng et al. (2000) have combined Lagrangian relaxation and GAs to develop an LRGA. Jeffrey and Christopher (1994) has proposed a penalty method to handle infeasible solution, and Srinivas and Patnaik (1994) has used adaptive genetic algorithms (AGA) to automatically change the crossover and mutation probability.

Although GAs have been extensively used to solve engineering and management problems, their application to network problems is still limited. Palmer and Kershenbaum (1995) developed a GA using an Link and Node Biased (LNB) encoding method to solve an optimal communication spanning tree problem. Abuali et al. (1995) developed a GA that used a determinant encoding method to solve a probabilistic minimum spanning tree. Taguhi et al. (1998) presented a GA, with a non-uniform mutation and an arithmetic crossover, to solve an optimal flow assignment problem in handling computer networks.

2.4. ADVANCED LOCAL SEARCH ALGORITHMS

Advanced local search algorithms, such as SA, TA, GDA, and TS, can usually lead to good results in various applications (e.g. see Dueck and Scheuer, 1990; Dueck, 1993; Yan and Luo, 1998, 1999). In general, TA and GDA have been superior to SA in many tests. Therefore, we employ TA, GDA and TS to develop four efficient local search algorithms, which can be compared with the proposed global search algorithm. These advanced local search algorithms are as follows.

2.4.1. *Simulated Annealing (SA)*

SA is a stochastic optimization algorithm. The main difference between SA and the local search method is that, when finding a solution to a given problem, the latter technique searches for solutions in a downhill direction. As a result, if a local optimal solution has been reached using a local search method, there is no way to bring the search out of this local optimum. This drawback can be overcome in SA by allowing an occasional jump out of a possibly inferior region. In other words, the SA technique

does not always search for solutions in a downhill direction. To examine whether it is acceptable to move from the previous solution to a new solution, SA determines the difference between the objective value of the previous solution and that of the new one. If the difference is favorable, the previous solution is discarded and replaced by the new one, as is done in a local search technique. If the difference is not favorable, the new solution is accepted along with a certain probability, which depends on the amount of the difference between the two solutions. After being first introduced by Kirkpatrick et al. (1983) to solve VLSI layout and graph partitioning problems, SA has been applied to many other NP-complete problems.

2.4.2. *Threshold Accepting (TA)*

TA, a method similar to SA, was first introduced by Dueck and Scheuer (1990). The essential difference between SA and TA is the different acceptance rules. To examine whether it is acceptable to move from the previous solution to a new solution, TA determines the difference between the objective value of the previous solution and that of the new solution. If the difference is favorable, the previous solution is discarded and the new solution takes its place, which is the same as in SA. When the difference is not favorable, the new solution is accepted if the difference falls within a given threshold. Intuitively, TA accepts every new solution which is "not much worse" than the previous one, while SA will only accept worse solutions with rather small probabilities. TA is simpler than SA, because with TA it is not necessary to compute probabilities or to make random decisions. Dueck and Scheuer (1990) in their studies found that TA performed better than SA.

2.4.3. *Great Deluge Algorithm (GDA)*

GDA, as proposed by Dueck (1993), is a deterministic algorithm similar to TA. The difference between GDA and TA is that TA considers whether the difference between the new and previous solutions falls within a threshold of acceptance, whereas GDA considers whether the objective value of the new solution for a minimization problem is below a standard level (water level). Whether the improvement for the new solution over the previous one is in a positive or negative direction, the solution will be accepted if it is below the standard level (for a minimum problem). Similar to TA, GDA also sets an initial water level, and then reduces this level gradually until reaching a designated convergent one.

2.4.4. *Tabu Search (TS)*

TS, proposed by Glover (1989, 1990), considers all moves, both "up" and "down," except for a certain prohibited or "tabu" set. The "tabu" moves

are kept as a list of length L, which effectively prevents the L most recent moves from being reversed. Each time a move is made, its "inverse" is added to the list, while the oldest move on the list is dropped. If the length, l, is too small, there is a chance that the method will simply cycle around the same sequences indefinitely, but this seems not to occur if the L is large enough. This can be thought of as simulating a form of "short-term memory", so that the procedure can recognize (and avoid) areas of the solution space that it has already encountered. Glover (1989, 1990) has also discussed ways of simulating "long-term memory" and procedures for overriding the basic algorithm by using "aspiration levels". The approach has been well documented by Glover.

2.5. DISCUSSION

From the above review, we found that concave cost transshipment problems have been traditionally solved using such algorithms as the branch-and-bound method, the dynamic programming method, the modified network simplex algorithms, the Lagrangian relaxation with sub-gradient methods, and other heuristic algorithms. Although some modern meta-heuristics, such as TS, TA, and SA, are useful for solving concave cost bi-partite networks (Yan and Luo, 1998, 1999), they have not been applied to concave cost transshipment problems. In other words, their performance in relation to general transshipment problems is unknown; possibly they will be easily confined to local optima. As well, most network problems discussed in the literature have belonged to particular networks (e.g. Hall, 1983; Yan and Luo, 1999; Suwan and Sawased, 1999). In this research, however, we aim to develop global and local search algorithms for solving minimum cost transshipment problems with concave arc cost functions.

3. Problem Formulation

Let a directed network, $G = (\bar{N}, \bar{A})$, be a transshipment network with concave arc cost functions, where $\bar{N}$ is the set of all nodes, and $\bar{A}$ is the set of all arcs. The problem can be formulated as follows:

$$\text{Minimize} \sum_{ij \in a} f_{ij}(x_{ij}) \tag{1}$$

$$\text{S.T.} \sum_{j} x_{ij} - \sum_{k} x_{ki} = b_i \quad \forall i \in \bar{N} \tag{2}$$

$$x_{ij} \geqslant 0 \quad \forall (i,j) \in \bar{A} \tag{3}$$

The objective is to minimize the total cost, as given in (1). The arc (i, j) cost function, $f_{ij}(x_{ij})$, is concave. Equation 2 represents the flow conservation constraint at every node. Note that: (1) if node i is a supply node,

then $b_i > 0$; (2) if node i is a demand node, then $b_i < 0$, and (3) if node i is a terminal, then $b_i = 0$. We assume that every arc flow is at least zero, as shown in Equation 3. Therefore, the problem is to transport commodities from all supply nodes to all demand nodes at a minimum transportation cost. For ease of algorithm development, the following assumptions are made:

(1) Each arc flow is between zero and infinity.
(2) Referring to Yan and Luo (1998, 1999), the arc cost function is assumed to be $f_{ij} = c_{ij}\sqrt{x_{ij}}$, where x_{ij} denotes the flow of arc (i, j), and c_{ij} is the associated arc constant. Note that the algorithms proposed in this research can be suitably modified and applied to solve the problems with other types of concave cost functions.
(3) The total node supply equals to the total node demand, so that the network flow conservation constraint is ensured.

This problem is a non-linear minimum concave cost network flow problem, which is characterized as NP-hard. In a minimum cost network flow problem, a feasible spanning tree, corresponding to a feasible solution, is designated in the simplex method a basic feasible solution. Usually, a feasible spanning tree in this type of problem can be expressed as an extreme point (Gallo et al., 1980). Since every arc cost function is concave, the objective function in (1) is also concave (Ahuja et al., 1993). For a concave cost function, there is at least one optimal solution located at an extreme point (Larsson et al., 1994). Furthermore, if all the parameters are integers, every extreme point is an integer solution, because of the integrality property. As a result, the optimal solution is also an integer. Similar to traditional integer programming problems, where integer solutions are spread throughout the solution set, the candidates of optimal solutions are spread throughout the linear solution set. Thus, traditional combinatorial optimization techniques, such as global and local search algorithms, can be applied to our problem. Hence, in this research we are attempting to develop a global search algorithm to solve the problem. We also develop four advanced local search algorithms for the evaluation of the global search algorithm.

We note that, in the simplex method, if one of the basic variables in a basic feasible solution (an extreme point) is zero, then the solution is degenerate. For the network problem, there always exist arcs with zero flow (Yan, 1996). Therefore, using local search algorithms to solve the problem may lead to degeneracy problems in neighborhood searches, which may be handled by the appropriate mechanisms (e.g. the TS technique). On the other hand, the crossover approach, which is a non-local search typically used in GA, will significantly reduce these degeneracy problems. In this respect, it may be more effective than local search algorithms in solving network flow problems.

4. Development of a Global Search Algorithm

In this section, we first propose a new encoding method to formulate a network flow solution, then an approach to adjust an infeasible flow to become a feasible one. Two methods for generating an initial population are proposed. Thereafter, the selection, reproduction, crossover and mutation operations are developed. Finally, we introduce the new population in the succeeding generation.

4.1. ENCODING

The encoding for an individual (a solution), which is the basis of GA, typically affects its efficiency. As the design of variables in a network flow problem is not natural for GA applications, the encoding in GA is generally not easy. Traditionally, the decision variables for a combinatorial optimization problem can be coded either directly or indirectly (Davis, 1987; Gen and Cheng, 1997). Take the spanning tree, which is often used in network applications, as an example. Direct encoding of every arc in the spanning tree is recorded. Although direct encoding requires no decoding process, it is usually longer. In contrast, the length of indirect encoding is usually shorter than direct encoding, but a decoding process is required to convert it to the original solution.

Abuali et al. (1995) and Palmer and Kershenbaum (1995) respectively have proposed two indirect encoding methods for solving undirected networks. These methods are however not applicable to directed networks. Taguhi et al. (1998) proposed using an indirect encoding method for directed networks with a flow conservation constraint at every node; however, the method may result in an infeasible solution, because the flow conservation condition may be destroyed after the crossover and mutation operations.

To meet our problem solution needs, we use a predecessor array to record the spanning tree, as is commonly employed in computer data structures. In this method, n elements in a one-dimensional array are used to store the values of the predecessors of nodes 1 to n, respectively, in the network. Therefore, the encoding length is equal to the number of nodes. Each node in the spanning tree corresponds to a unique predecessor. The root's predecessor is zero. Note that in a directed network, the encoding method does not ensure a feasible spanning tree after crossover or mutation operations. In this case, the flow augmentation algorithm, as proposed by Yan and Yang (1996), is employed to adjust an infeasible spanning tree to become a feasible one.

Compared with the encoding method proposed by Taguhi et al. (1998), the encoding method proposed in this paper can reduce the number of encoding and decoding steps leading to a shorter encoding length. Note that using the flow augmentation algorithm to adjust an infeasible solution to become a feasible one may cause the loss of some genes, but it does

offer the opportunity to search for different genes concurrently. Consequently, by incorporating the encoding method into crossover and mutation operations, the resultant global search algorithm can efficiently reach a near optimal solution. We also note that our encoding method can be applied to both directed and undirected networks. For example, as shown in Figure 1, the encoding of the undirected spanning tree B is {0, 1, 4, 1, 4, 5, 3, 3, 7}, and the encoding of the directed spanning tree C is {0, 1, −4, −1, 4, −5, −3, −3, −7}. In the directed spanning tree C, a positive sign indicates that the arc's direction moves from the associated node to its predecessor, and a negative sign indicates the arc's direction is from the node's predecessor to itself.

4.2. ADJUSTMENT OF INFEASIBLE SOLUTIONS

An infeasible solution generated by crossover or mutation may contain cycles or a set of disconnected sub-networks. Note that the cycles or sub-networks contain arcs that were previously in a spanning tree before crossover or mutation. In this research we use the least cost flow augmentation algorithm (Yan and Yang, 1996) to adjust infeasible solutions to become a feasible spanning tree. The algorithm is described as follows:

1. Connect all disconnected sub-networks between any two sub-networks by adding a number of arcs, which are not in the sub-networks, to make a connected network. Note that these arcs are found using a graph search technique. Let these arc flows be zero and their arc costs be unchanged. Solve the least cost flow augmentation cycle.
2. Employ the flow augmentation algorithm to augment the flow in the cycle until a reverse arc flow is reduced to zero. Delete this arc and eliminate the cycle.

Repeat the two steps until all cycles are eliminated to find the feasible spanning tree.

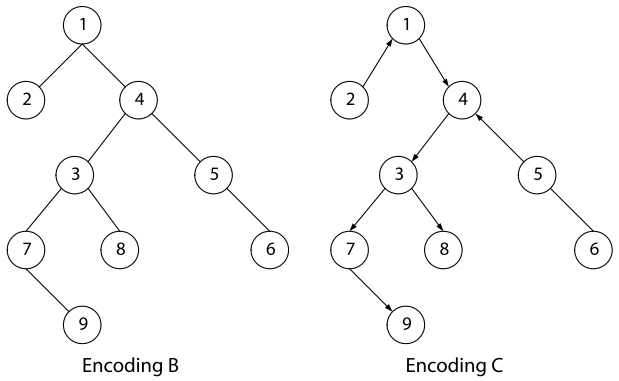

Figure 1. Encodings B and C.

4.3. GENERATION OF THE INITIAL POPULATION

Two heuristics for generating the initial population are proposed. The first is the linear cost initial solution algorithm, and the other is the concave cost initial solution algorithm.

4.3.1. *Linear Cost Initial Solution Algorithm (LCISA)*

The algorithm is outlined as follows:

Step 1: Set each arc cost to be the average unit transportation cost at the current flow (see Yan and Luo, 1999). That is, for an arc (i, j) with flow x_{ij}, its arc cost is set to be $c_{ij}/\sqrt{x_{ij}}$. Note that if arc (i, j) has no flow, then its average unit transportation cost (i.e. $c_{ij}/\sqrt{x_{ij}}$) at the zero flow is undefined. To facilitate problem solving, as done in Yan and Luo (1999), we set its flow to be one, for the approximation of the arc cost. As a result, the arc cost is set to be c_{ij}.

Step 2: Randomly select a supply node i and a demand node j and solve for the shortest path from node i to node j.

Step 3: Determine the maximum amount of goods, T_{ij}, that can be transported from supply node i to demand node j, that is, $T_{ij} = \min(s_i, d_j)$, where s_i is the remaining supply for node i, and d_j is the remaining demand for node j. Transport T_{ij} through the shortest path from node i to node j.

Step 4: Deduct T_{ij} from node i's supply and node j's demand, and update the arc flows along the path.

Step 5: If there is no node with supply or demand, stop; else, go to Step 2.

We use LCISA to solve for a feasible solution. If the solution is not a spanning tree, then the least cost flow augmentation algorithm can be used to adjust it to become feasible. Note that each arc cost is set in Step 1 and does not change, which speeds up the algorithm. Finally, we can use the algorithm to generate enough feasible solutions to fit the designated population size, N_p.

4.3.2. *Concave Cost Initial Solution Algorithm (CCISA)*

Based on the characteristics of concave cost network flow problems (Kuhn and Baumol, 1962), if flows are assigned to arcs when the adjacent nodes have a greater supply or demand than other nodes, then the cost may be reduced. In addition, the total cost tends to be reduced if the arc flows tend to be zero or the allowed maximum amount. In other words, an all-or-nothing assignment rule should help to reduce transshipment costs. The heuristic based on this concept is outlined as follows:

Step 1: Set each arc cost to be the average unit transportation cost at the current flow. Note that, similar to that is done in LCISA, if an arc has no flow, then its flow is set to be one to approximate the arc cost.

Step 2: Choose a node (i) with the largest remaining node supply. Solve for the shortest paths from node i to all other demand nodes.

Step 3: Randomly choose a node (j) from among the three nodes with the three largest remaining node demands, where node j is reachable from node i.

Step 4: Determine the maximum amount of goods, T_{ij}, that can be transported from supply node i to demand node j, that is, $T_{ij} = \min(s_i, d_j)$, where s_i is the remaining supply for node i, and d_j is the remaining demand for node j. Transport T_{ij} through the shortest path from node i to node j.

Step 5: Deduct T_{ij} from node i's supply and node j's demand, and update the arc flows along the path.

Step 6: If there is no node with supply or demand, stop; else, go to Step 1.

We use CCISA to solve for a feasible solution. If the solution is not a spanning tree, then use the least cost flow augmentation algorithm to adjust it to become a feasible one. Finally, we can use the algorithm to generate enough feasible solutions to fit the designated population size, N_p. Note that there are two points where CCISA differs from LCISA. First, instead of using a constant arc cost, each arc cost in CCISA is updated once its flow has changed. Second, instead of randomly selecting a supply/demand node pair, CCISA chooses the node with the largest remaining node supply and a node randomly selected from among the three reachable nodes with the three largest remaining demands, to solve for the shortest path, and to transport the maximum allowable amount of goods along this path.

4.4. SELECTION AND REPRODUCTION

Selection is a process in which suitable individuals are selected for crossover. Each individual corresponds to a feasible spanning tree. In this research we employ the roulette wheel method (Goldberg, 1989) to select suitable individuals to perform reproduction and crossover. Before selection, the individuals are sorted in ascending order of their fitness values. Note that the fitness value associated with an individual is defined as its objective value, as has commonly been done in the past (Davis, 1991). The population is divided into K sets of individuals, so that each set contains the same number of individuals. In the roulette wheel method, the best set occupies K units, the second-best set $K-1$ units, the third-best set

$K - 2$ units, and so on. Consequently, the worst set occupies one unit. In general, a set with a larger area in the roulette wheel has a higher probability of being selected. Using the roulette wheel method, we first select a set, then randomly select an individual from the set. The process can be repeated until enough individuals have been selected. The determination of the K value necessary for obtaining good quality solutions will be tested in Section 6.

4.5. CROSSOVER

Crossover is useful for exchanging parts of genes for selected individuals to produce new individuals. Some commonly used crossover operators are one-point crossovers, two-point crossovers, and uniform crossovers (Gen and Cheng, 1997). After many tests, we found that, due to the characteristics of the spanning tree, both the one-point and the two-point crossover operations produced offspring that were widely dissimilar from the parent population. Since it is computationally expensive to adjust very infeasible offspring to become feasible, the uniform type of crossover is adopted in this research.

The uniform crossover method is performed as follows: Given two spanning trees, we first set a number, L, of genes to be exchanged, each gene being associated with an arc. Then we randomly select L arcs with positive flows from each of the two spanning trees. After this, we exchange these two sets of arcs, obtaining two new solutions, each possibly being infeasible. Finally, we employ the flow augmentation algorithm to adjust the infeasible spanning trees to become feasible ones. The proposed crossover method has the following three advantages. First, according to the characteristics of the concave cost function, the greater the flow that passes through the same arc, the better the objective value that can be obtained. Selecting arcs with positive flows implies the selection of superior genes. As a result, the crossover method helps evolve good solutions. Second, in conjunction with the mutation (to be addressed later), which selects arcs that are not in the spanning tree (their flows are 0) and is complementary to the crossover, good results can possibly be obtained. Third, any infeasible solutions can usually be efficiently adjusted to become feasible. Consequently, diverse solutions can be examined and good ones obtained.

4.6. MUTATION

The test results indicate that not many good individuals can be obtained after a crossover operation. Rudolph (1994) has mentioned that with a simple GA it is very difficult to converge to the optimal solution; however, if a mutation mechanism similar to a local search algorithm can be designed, then the solution quality can usually be improved. Thus, we

employ a traditional local search algorithm as the mutation operator, to improve the individuals evolved from crossover. Coupled with the crossover operator, which can help the search by jumping out of local optima, the mutation method is expected to assist in searching for the global optimum.

The mutation method is outlined as follows: First, for each feasible spanning tree produced following crossover, we use a mutation probability, P_m, to determine whether we need to perform a mutation (i.e. local improvement). If yes, we then perform a local search in the neighborhood of the spanning tree (called the search range), to find a better solution. In each local search, we randomly select an arc that is not in the spanning tree (whose flow is zero) and add this arc to the spanning tree, to form a non-spanning tree containing a cycle. Finally, we use the flow augmentation algorithm to adjust a non-spanning tree to become a spanning tree. The process is repeated until all feasible spanning trees have been examined.

4.7. ELITISM

When using a traditional simple GA, the best individual in a population may fail to produce good offspring in the next generation through crossover and mutation. The elitist strategy (Davis, 1991) remedies this potential source of loss by copying the best individual in each generation into a succeeding generation. Consequently, for a population, which is dominated by a superior individual, the speed is increased and the performance of GA also appears to improve. The elitist strategy developed in this research keeps a certain percentage of the best individuals, termed the elite ratio P_e, in a population.

4.8. NEW POPULATIONS IN SUCCEEDING GENERATIONS

In each new generation, the population consists of three sets of individuals. Given a preset P_e and P_p, as well as the population size N_p, the first set contains a total of P_eN_p elite individuals, which are copied from the best P_eN_p individuals in the previous generation. The second set contains a total of P_pN_p individuals that are randomly generated using either of the two initial solution algorithms, LCISA and CCISA. Note that the purpose of generating new individuals (some call them immigrants) is to provide an opportunity for a more diverse search of the solution space. The third set contains a total of $(1-P_e-P_p)N_p$ individuals that are generated by employing the crossover and mutation operations on the previous generation. Note that two individuals in the previous generation are randomly selected to perform the crossover operation to obtain two new individuals. Each of them is then improved by performing the mutation operation. The

individuals in each of the three sets are generated until the preset number of individuals is met. Suitable parameter values will be set for P_e and P_p after the tests discussed in Section 6. Note that we have tried to use the roulette wheel method to select individuals from the population for the fourth set. We use a probability to determine whether to directly use the local search to improve the selected individuals, which is different from the local search (mutation) after crossover. Since the test results do not significantly improve, we discard the fourth set, to save computation time.

5. Local Search Algorithms

To test the performance of the proposed GA, several advanced local search algorithms, based on TA, GDA, and TS, are developed and compared. As referred to Yan and Luo (1999), these algorithms are designed as two-stage improvement algorithms. We first generate an initial feasible solution (a feasible spanning tree), then gradually improve the initial solution by exchanging arcs, as is done in mutation, until the criterion for stopping is met. Note that if an exchange of arcs results in an infeasible spanning tree, then we can use the least cost flow augmentation algorithm to adjust it to become a feasible spanning tree.

5.1. INITIAL SOLUTION

From Yan and Luo (1999), for local search algorithms the better the initial solution, the better the result that can usually be achieved. Therefore, we can use the aforementioned CCISA to generate several solutions, and then choose the best one as the initial solution.

5.2. THRESHOLD ACCEPTING (TA)

Our TA is similar to that in Yan and Luo (1999). We use the initial objective value multiplied by a fixed percentage as the initial threshold value. The difference between the new objective value and the current objective value divided by the current objective value determines whether the new solution can be accepted or not. To trade off efficiency and effectiveness, we use the EPOCH concept (Golden and Skiscim, 1986) to search the neighborhood solutions. In particular, we suggest examining a limited number (denoted as EPOCH) of arcs in every move. For simplicity, EPOCH is suggested to be a multiple of the total number of nodes in the test network, where the multiple is subjected to testing. We randomly choose an EPOCH of arcs from all feasible arcs and then determine the one that results in a solution that leads to maximum improvement among the EPCOH of neighborhood solutions. The solution is then evaluated as to whether it is acceptable as the next solution. Two convergence criteria

are proposed. One is controlled by the total number of iterations (iter). The other is controlled by the number of moves since the last update that yields no improvement (i.e. no_update). For details of the algorithm, please refer to Yan and Luo (1999). The parameters used in TA, such as the initial threshold value (T_0), the number of different thresholds (num), EPOCH, iter, and no_update, will be set after the tests.

5.3. GREAT DELUGE ALGORITHM (GDA)

GDA is similar to TA, except that GDA considers whether the objective value of the new solution is below an absolute deluge level. The initial deluge level is set based on the best solution among the initial solutions. A series of gradually decreasing ratios are set as the deluge levels. Note that the initial deluge level G_0 is 100%. If the objective value of a new solution is lower than the current deluge level, then the new solution is accepted. If the solution is better than the incumbent solution, then the incumbent solution is updated. Similar to TA, the parameters of the initial deluge level (G_0), the number of different deluge levels (num), EPOCH, iter, and no_update are set after the tests.

5.4. TA AND GDA WITH THE TABU SEARCH STRATEGY (TTA AND TGDA)

To improve the solution quality, and to avoid the inefficient degeneracy problem, the TS strategy is incorporated into TA and GDA. To efficiently and effectively record the tabu list, we refer to the auxiliary objective functions designed in Yan and Luo (1998), as follows:

$$\text{The } k\text{th objective function} \qquad \sum_{ij\in A} f_{ij}^k(x_{ij}) = \sum_{ij\in A} r_{ij}^k c_{ij}\sqrt{x_{ij}}, \tag{4}$$

in which r_{ij}^k is a random number (between 0 and 1) corresponding to arc (i, j) in the kth objective function. The various objective function values associated with a spanning tree can be used for the tabu reference. In particular, if the objective function values of two spanning trees are all the same, then the two spanning trees are much the same. After numerous tests were carried out, we found the five objective functions that yielded the most reliable results. The length of this tabu list is set after the tests.

Some spanning tree arc flows may be zero; this is called a degenerate spanning tree. Consequently, when we eliminate a cycle by exchanging arcs, if the leaving arc flow is 0, there is no change in the objective value. To avoid inefficient searches due to degeneracy problems, we create a tabu list that records recently entered arcs. If we select an arc to join the spanning tree that is in the tabu list, then it is discarded, thus avoiding moving back to a previously degenerate solution. The length of this tabu list is set after the tests. To conclude, two tabu lists are designed here. Tabu list 1

contains a number of the combinations of the five objective functions used to examine the same spanning tree. Tabu list 2 contains a number of the arcs used for examining the degenerate solutions. For convenience, we set the two tabu list lengths to be the same. Certainly, tabu lists of different lengths can be researched in the future.

In addition to the two aforementioned tabu lists, we also, from Glover (1990), setup an aspiration mechanism, which states that if a new solution found in tabu list 1, and its objective value is superior to that of the current solution, then the new solution will be accepted.

The application of the two tabu lists and the aspiration mechanism are summarized as the following two cases: (1) When a new solution that is not the same as the current solution is found in the tabu list 1 (meaning that it has been previously searched), then it is rejected unless it satisfies the aspiration mechanism. (2) If the new solution is the same as the current solution (i.e. it is in the tabu list 1 and is also a degenerate solution), and it is also found in tabu list 2, then it is rejected. Note that in the second case, to prevent too much restriction of search direction, moves to degenerate solutions that are not recorded are allowed. For other cases, new moves are accepted unless they have been rejected by the TA or GDA acceptance rules.

Finally, similar to TA and GDA, the parameters of TTA and TGDA, such as the initial threshold value (T_0) (the initial deluge level, G_0), the number of different thresholds (deluge levels, num), EPOCH, iter, no_update, and the tabu list length (tabu_num) are set after the tests.

6. Numerical Tests

To evaluate the proposed algorithms, numerical testing is performed. To generate a sufficient number of problem instances for testing, we first design a network generator. The C++ computer language is used to code all the programs for the network generator and the proposed algorithms. Then suitable parameters for the algorithms are determined by testing. Finally, we compare the computational results of the global search algorithm and the four advanced local search algorithms. The tests are performed on a PC K7-700 in the environment of Microsoft Windows 2000.

6.1. NETWORK GENERATOR

To test the algorithms, we design a network generator that generates concave cost transportation network problems. Since the performance of the meta-heuristics may be influenced by the problem scale and the parameters, random numbers are used to generate randomized networks with various scales and parameters. The approach is as follows: For every node in a

given n-node network, we use a random number and a connected probability to determine if it points to the nodes in its neighborhood. This neighborhood is preset in order to prevent the generation of too many arcs in a network. If yes, we randomly generate a positive cost coefficient (i.e. c_{ij}) for the associated arc. After every node is examined for constructing arcs, we examine whether the network is connected using a graph search technique. If yes, the network is connected; otherwise, we have several disconnected sub-networks. We then choose several nodes, each randomly from a sub-network, and connect them to form a connected network.

As to the supply/demand nodes and their supplies/demands, we first randomly choose some node to be a supply node, and randomly move forward a number (Move_Num) of arcs to another node, which is used as the demand node. We then randomly generate some amount, the supply/demand for the node pair, meaning that this amount can flow through the path from the supply node to the demand node. By repeating this process several times (Ass_Num) (equal to the number of nodes times a control parameter D, which will be mentioned later) and then summing up the total supply/demand for each node, we generate a randomized network.

Note that we can find at least a feasible solution for any randomized network. Moreover, since arc cost function values are positive, the objective function for each network is bounded by zero. As a result, there is an optimal solution for every network.

To test the proposed algorithms, we design several types of networks with different scales, as shown in Table 1. For each network type, we use a parameter D, which represents the density of supply-demand nodes in all nodes, to generate various networks. For the network node sizes of 50, 100, 150, and 500, we set D to be 0.2, 0.5, and 0.8, respectively, which generates 12 networks. For the networks with node sizes of 10 and 20, that is their network scales are relatively small, we set D to be 0.5 to generate two networks. As a result, there are altogether 14 networks in this test.

For convenience, a network which has 20 nodes with D equal to 0.5 is denoted as 20_05; the other networks are similarly denoted. Table 1 shows the network types with different numbers of nodes, arcs, and network densities. Note that the network density is defined as the number of arcs

Table 1. Network types

No. of nodes	No. of arcs	Network density (%)
10	17	17.0
20	47	11.8
50	220	8.8
100	730	7.3
150	1117	5.0
500	4728	1.9

divided by the square of the number of nodes. The other parameters are set as follows: The arc cost coefficient is set uniformly between 50 and 300 and the node supply/demand is set uniformly between 1 and 150.

6.2. GLOBAL SEARCH ALGORITHM – GENETIC ALGORITHM (GA)

In GA, the solution quality can be greatly influenced by the setting of the parameters, such as the crossover probability, mutation probability, population size, and so on. In general, the best parameter values are problem-dependent. Thus, in this research we use 26 alternative parameter combinations (denoted as Scenarios *A–Z*), and perform many tests using different network sizes and different supply-demand node densities, to find the suitable parameter combinations leading to good quality solutions. Note that the parameters used are suitable for the problems in the research. More testing should be conducted when they are applied to other problems in the future.

For ease of comparison among different combinations of GA parameters, the objective value for each scenario is compared with the smallest objective value obtained from all parameter combinations for all algorithms. In particular, an error percentage, the relative difference from the best scenario, is calculated for each scenario. Thus, the smaller the error percentage, the better the parameter combination for a test network. Clearly, the smallest error percentage, that is the best scenario, would be zero.

After testing 14 networks and 26 scenarios, we found that Scenarios J, O, and W, on average, had better solutions than the others. Compared with the best solutions for medium- and large-scale networks, the average errors of the objective values for the three scenarios are 2.94, 2.86 and 3.19%, respectively. The most notable findings are introduced in the following.

6.2.1. *Population Size and the Number of Exchange Arcs in a Crossover*

Except for the number of arcs to be exchanged (exchange arcs) in a crossover on a spanning tree, all other parameters are related to the population size. Therefore, the influence of the population size and the number of exchange arcs on the solution quality will be discussed first. Three different population sizes, 50, 100, and 150, are considered. Three different numbers of exchange arcs, that is three exchange arcs, with a 0.5 arc exchange rate and a 0.9 arc exchange rate, are considered. Note that the three exchange arcs are applied so that only a few arcs in the spanning tree are exchanged. Also note that an arc exchange rate of 0.9 (meaning that 90% of the arcs are to be exchanged) is designed so that few arcs (i.e. 10%) are not exchanged, while an arc exchange rate of 0.5 is designed to be between of the above two cases.

With three network scales, of 100, 150 and 500 nodes and three supply-demand node density values (*D*), of 0.2, 0.5, and 0.8, there are in total 9

networks and 81 problem instances for testing. The results show that for a population size of 100 individuals, the use of three exchange arcs in a crossover would result in the best solution, with an average error of 3.38%. It should be mentioned that the test results for all scenarios show the best crossover probability to be 1.0. Therefore, we set the crossover probability to be 1.0 in all scenarios in later tests.

6.2.2. *Number of Groups for Roulette Wheel Selection*

Four values of K, 1, 5, 20 and 30, are tested. The other parameters are set as follows: the population size is 100, $P_e = 0.02$, $P_p = 0.01$, $P_m = 0.6$; the search range is 300 (i.e. investigating 300 solutions in the neighborhood of the current solution); and an arc exchange rate of 0.9. Note that when K equals 30, each group contains three or four individuals. For example, the first group contains three individuals, which equals the rounded-off value of 1*100/30. The second group contains four individuals, which equals the rounded-off value of 2*100/30−3. The others are calculated similarly. Tests are performed on three large-scale networks (100, 150, and 500), each with D values of 0.2, 0.5, and 0.8. Consequently, a total of 9 networks and 36 problem instances are tested. The results show that better solutions are yielded for K equal to 5 and 30, with an average error of 3.30 and 3.19%, respectively. The average errors for K equal to 1 and 20 are 8.07 and 4.76%, respectively. The average computation times for K equal to 5 and 30 are both less than those for K equal to 1 or 20. In conclusion, we can say that it is best for K to be 30. It should be mentioned that, before the evaluation, we performed preliminary tests on the K values of 1, 2, 3, 5, 10, 20, 30 and 50 for some of the nine networks. We found that the K values for 2, 3, 10, and 50 resulted in poorer solution quality than when K was equal to 5 or 30. Moreover, in the preliminary test results, we found that the solution quality was not correlated with the K value. Therefore, for simplicity, we chose the K values of 1, 5, 20 and 30 to complete the testing.

6.2.3. *Mutation Probability and Search Range*

In the literature the mutation probability has generally been set to be a small number, usually below 0.02. However, given the highly degenerate conditions in network flow problems, the algorithms may be inefficient for a low mutation probability. Therefore, we use three values of P_m, 0.3, 0.6, and 0.9, to represent the low, medium, and high mutation probability, respectively. The other parameters are set as follows: the population size = 100, $P_e = 0.02$, $P_p = 0.01$; the arc exchange rate is 0.9; the search range is 300; and the roulette wheel value K is 5. The tests are performed on three large-scale networks (100, 150, and 500), each with D values of 0.2, 0.5, and 0.8. Consequently, a total of 9 networks and 27 problem instances

are tested. The results show that the best solutions when the mutation probability is equal to 0.9, with an average error of 2.86%. A mutation probability of 0.6, with an average error rate of 3.30%, produces worse solutions than for 0.9, but better than that for 0.3, with an average error of 5.41%.

In addition, we use three exchange arcs to replace the 0.9 arc exchange rate, with the other parameters being unchanged. Additional tests of the mutation probability, $P_m = 0.3$, 0.6 and 0.9, respectively, are performed on three large-scale networks (100, 150, and 500), each with D values of 0.2, 0.5, and 0.8. Consequently, a total of 9 networks and 27 problem instances are tested. The results show that the mutation probability of 0.6 yield the best solutions, with an average error of 2.94%, 0.9 is the second, with an average error of 3.59% and 0.3 is the worst, with an average error of 5.32%.

These results imply that we can obtain relatively good solutions when fewer arcs are exchanged, with a mutation probability of 0.6, than when more arcs are exchanged with a mutation probability of 0.9. Although the solution qualities for the two scenarios are similar, the former requires less computation time than the latter, so is considered better than the latter.

It should be mentioned that we also evaluate the mutation search range. After many tests, we found that searching larger area in one move usually yields better solutions than searching a smaller area in more moves. Hereinafter, we test two strategies. Strategy one uses one move for local search, while strategy two uses more moves to reach a local optimum. Twenty-six different parameter combinations, with search ranges from 30 to 300, are tested for strategy one. Twenty-four different parameter combinations, with search ranges of 10, 30, 50, and 100, are tested for strategy two. The other parameters are set as follows: population size = 100, $P_e = 0.02$, $P_p = 0.01$, $P_m = 0.6$, three exchange arcs, and the roulette wheel value K is 5.

Three large-scale networks (100, 150, and 500), each with D values of 0.2, 0.5, and 0.8 are tested. Consequently, a total of 9 networks and 234 problem instances are tested for strategy one and a total of 9 networks and 216 problem instances are tested for strategy two. The results show that a search range of 300, with strategy one, efficiently yields the best solution quality, with an average error of 2.94%. We thus apply the search range of 300 and strategy one in later tests.

6.2.4. *Elite Ratio*

Three elite ratios of P_e, 0.01, 0.02, and 0.03, are tested, with the population size = 100, $P_p = 0.01$, $P_m = 0.6$, the number of exchange arcs = 3, the search range = 300, and the roulette wheel value $K = 5$. The tests are performed on three large-scale networks (100, 150, and 500), each with D values of 0.2, 0.5, and 0.8. Consequently, a total of 9 networks and 27 problem instances are tested. The results showed that when D is 0.8, the

solutions are not significantly different for the three elite ratios. For a D value equal to 0.2 and 0.5, the elite ratio 0.02 yields the best solutions, with an average error of 2.94%. The elite ratio of 0.03 yields the best second solutions, with an average error of 3.31%, followed by 0.01, with an average error of 3.83%. These results indicate that, for a population of 100 individuals in each generation, copying the best two individuals from the previous generation and incorporating them into the next generation would, on average, produce better solutions than other elite ratios. We also see that, as the elite ratio increases, the computation time decreases slightly; however, the difference is not significant.

6.2.5. *Immigrant ratio*

Three immigrant ratios P_p, 0, 0.01 and 0.1, are tested, with the population size = 100, $P_e = 0.02$, $P_m = 0.6$, the number of exchange arcs = 3, the search range = 300, and the roulette wheel value $K = 5$. We also tested another scenario with a search range = 100, $P_p = 0.05$, and the other parameters being the same as the above. Tests are performed on three large-scale networks (100, 150, and 500), each with D values of 0.2, 0.5, and 0.8. Consequently, a total of 9 networks and 36 problem instances are tested.

The results show $P_p = 0.01$ yields the best solutions (i.e. one immigrant is allowed to enter the population in every generation), with an average error of 3.30%. $P_p = 0$ yields the worst solutions (i.e. no immigrant enters the population), with an average error of 4.87%. The $P_p = 0.1$ solutions are in between, with an average error of 4.53%. The solutions obtained for a search range = 100 and $P_p = 0.05$, with an average error of 3.38%, are slightly worse that for a search range = 300 and $P_p = 0.01$.

6.2.6. *Comparison of LCISA and CCISA*

We first compare the objective values solved by LCISA and CCISA and their influences on the GA's performance. We set P_A to be the proportion of initial solutions in the initial population generated by CCISA. Similarly, P_B is set to be the proportion of immigrant solutions in the population of the second and later generations generated by CCISA. For example, when $P_A = 0.6$, it means that 60% of the initial solutions are generated using CCISA and 40% using LCISA. When $P_B = 0.6$, it means that 60% of the solutions are generated using CCISA and 40% using LCISA, for immigration into the second and later generations. Note that the number of solutions that immigrate into a offspring generation using CCISA is equal to $N_p{*}P_p{*}P_B$, where N_p is the population size.

If we let N_p be 100 and use network 150_05 and 9 combinations of (P_A, P_B), such as (0.6, 1), (0.6, 0.6), (0.6, 0), (1, 1), (1, 0.6), (1, 0), (0, 1), (0, 0.6), and (0, 0) for tests, the results will show that CCISA generates

better solutions than LCISA, although the former is more time-consuming than the latter. As the LCISA solutions in the population increase, the average solution quality decreases. Twenty-seven problem instances, including nine combinations of (P_A, P_B) and three parameter scenarios (J, O, and W), are tested further. The results show that a mixture of CCISA and LCISA solutions in the initial population will yield, on average, better solutions, probably due to the consideration of both solution quality and diversity. In particular, the combination of (P_A, P_B) = (0.6, 1.0) results in the best solution quality, on average. Moreover, pure CCISA immigrants (i.e. $P_B = 1$) yields better solutions than mixed CCISA and LCISA immigrants. In conclusion, the best solution quality is yielded when the initial population is composed of 60% CCISA solutions and 40% LCISA solutions, with 100% CCISA solutions as immigrants in each child generation.

6.2.7. *Evolution of Generations and Convergence*

The number of evolutional generations directly affects the solution quality of GA. Two indices are used to evaluate the GA's performance. One is the best objective value of the population for a generation, f_g. The other is the average objective value of the population for a generation, f. In each generation we use f_g and f, which are solved using GA with the parameters from Scenario A, and the best solution F_g obtained from all the parameter scenarios, to discuss the GA's convergence under the three scale networks, 500_08, 105_05 and 50_02. Note that similar results are obtained for the other networks so they are not detailed here. As shown in Figures 2, 3 and 4, GA converges at 1000 generations for the large network (500_08), 500 generations for the medium network (150_05) and 100 generations for the small network (50_02). We also find that f declines slowly in the large network and rapidly in the small and middle networks, as shown in Figures 2,

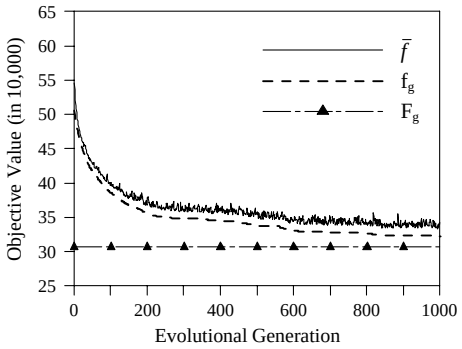

Figure 2. Convergence conditions for network (500_08).

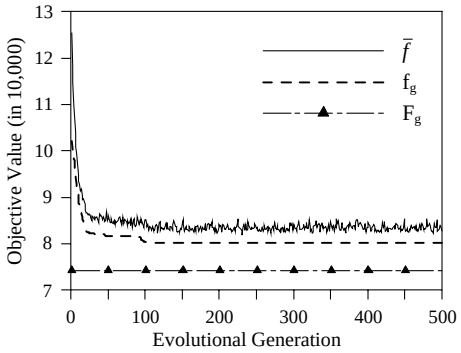

Figure 3. Convergence conditions for network (150_05).

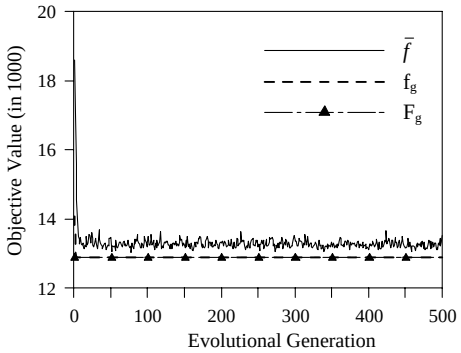

Figure 4. Convergence conditions for network (50_02).

3 and 4. Note that in Figure 4 f_g and F_g are the same when the evolutional generation is greater than 2.

6.2.8. *Summary*

1. A mixture of CCISA and LCISA solutions in the initial population will yield, on average, better solutions than either pure CCISA or LCISA solutions.
2. A population size of 100 is better than 150 or others, in terms of solution quality and computation time.
3. The number of exchange arcs only slightly affects the computation time. The use of three exchange arcs and an arc exchange rate of 0.9 yield similar computation times, and are better than the others.
4. A suitable consideration of different objective value probabilities during the selection step is better than random selection, without consideration of the objective value. In particular, a division of 5 groups or 30 groups in the roulette wheel method yields better solutions than others.

5. Crossover and mutation coordination implies the combination of a global search with a local search and usually yields good solutions. In addition, in mutation searching a larger area in one move usually yields better solutions than searching smaller areas in more moves.
6. Of the 26 tested parameter scenarios, J, O and W are superior to the others. The seven parameters used in these three scenarios are shown in Table 2.

6.3. ADVANCED LOCAL SEARCH ALGORITHMS

In this section, we perform tests to find suitable parameters for advanced local search algorithms. To prevent the results from being significantly affected by the application of random probability, while still considering computation times, we perform three tests for each parameter scenario. Note that the parameters obtained are only suitable for problems tested with the proposed algorithms. More tests must be performed and verified when applied to other problems in the future.

6.3.1. *Initial Solution*

We generated 1000 solutions using CCISA, and selected the best solution among them as the initial solution. To save time, the tests were performed on two medium networks (100 and 150), each with D values of 0.2, 0.5, and 0.8. Consequently, a total of six networks and six problem instances were tested. The results showed that there was still significant difference between each initial solution and the best solution, with the error gap between 14.06 and 36.8%.

6.3.2. *System parameters for TA and GDA*

We design nine scenarios (L1–L9) to test for such TA and GDA parameters as the initial threshold value (T_0), the initial deluge level (G_0), the

Table 2. Parameter combinations in Scenarios J, O and W

Parameter	Scenario		
	J	O	W
Population size	100	100	100
Elite ratio, P_e	0.02	0.02	0.02
Immigrant ratio, P_p	0.01	0.01	0.01
Mutation probability, P_m	0.6	0.9	0.6
Arc exchange (number) probability	(3)	90%	90%
Search range	300	300	300
Number of roulette wheel groups, K	5	5	30

number of different thresholds or deluge levels (num), EPOCH, iter, and no_update. For each scenario, three tests are performed and the best solution selected for comparison. The results show the TA error gaps to be between 2.32 and 4.88%, and the GDA error gaps to be between 2.70 and 6.40%. Of all the TA scenarios, Scenario L2 ($T_0 = 0.5$, num = 60, EPOCH = $2n$, no_update = 30) yields better TA solutions than the others, with an average error of 2.32%. Scenario L5 ($G_0 = 0.39$, num = 100, EPOCH = n, no_update = 50) yields better GDA solutions than the others, with an average error of 2.70%.

6.3.3. *System parameters for TTA and TGDA*

In addition to the parameters used in TA and GDA, TTA and TGDA also include the tabu list length (tabu_num). Ten scenarios (M1–M10) are designed to test for the TTA and TGDA parameters. Each scenario is tested three times and the best solution was selected for comparison. The results show the TTA error gaps to be between 2.05 and 4.27%, and the TGDA error gaps to be between 2.31 and 4.84%.

Of the scenarios, Scenario M7 ($T_0 = 0.39$, num = 100, EPOCH = n, no_update = 50, tabu_num = 3) yields better TTA solutions than the others, with an average error of 2.05%. Scenario M3 ($G_0 = 0.5$, num = 100, EPOCH = n, no_update = 50, tabu_num = 7) yields better TGDA solutions than the others, with an average error of 2.31%. The results also show that TTA/TGDA with the incorporation of the TS technique is improved over TA/GDA. Note that a tabu list length of 3, on average, leads to better solutions than that of 7 or other alternatives.

6.3.4. *Summary*

The test results are summarized as follows:

1. It is time-consuming to detect cycles and augment flows in neighborhood searches. Randomly choosing a number of candidate arcs for neighborhood searches seems to be better, in terms of solution quality and computation time, than enumerating all arcs in neighborhood.
2. The TS method is useful for resolving degeneracy problems in network flows. In particular, TTA and TGDA are an improvement over TA and GDA. The tabu list length needs not necessarily to be long, but is dependent on the problem. A short tabu list may allow a searching process with fewer constraints. The test results show that a tabu list length of 3 is better than others.
3. The best parameter scenario, for each algorithm, is shown in Table 3. The error gaps for the initial solutions, and solutions that use the algorithms with their best parameters on different networks, are shown in

Table 3. The best TA, GDA, TTA and TGDA parameters

Parameter	Algorithm (scenario)			
	TA (L2)	TGDA (M3)	GDA (L5)	TTA (M7)
Initial threshold value, T_0	0.5	0.5	0.39	0.39
No. of thresholds, num	60	100	100	100
EPOCH	$2n$	N	n	n
no_update	30	50	50	50
tabu_num	N/A	7	N/A	3

Note: N/A means not applicable.

Table 4. Test results for the four advanced local search algorithms

Network problem	Initial solution (CCISA)	Algorithm (scenario)			
		TA(L2) (%)	TTA(M7) (%)	GDA(L5) (%)	TGDA(M3) (%)
100_02	14.60	4.20	4.20	4.20	4.20
100_05	27.10	2.50	3.80	4.00	0.00
100_08	30.80	1.30	0.00	1.90	3.10
150_02	30.90	0.90	0.90	2.00	0.90
150_05	32.20	2.90	0.80	0.80	4.00
150_08	36.80	2.10	2.60	3.20	1.60
Average	28.74	2.32	2.05	2.70	2.31

Table 4. We see from Table 4 that TTA (with Scenario M7) performs the best, with an average error of 2.05%, while GDA (with Scenario L5) performs the worst, with an average error of 2.70%.

6.4. COMPARISON OF GLOBAL AND LOCAL SEARCH ALGORITHMS

To compare the performance of the global search algorithm with that of the advanced local search algorithms, we perform an additional test. The best individual in the initial population of the global search algorithm is selected as the initial solution for the local search algorithms. Every test problem is then solved using the global search algorithm GA (with three scenarios J, O, W) and the advanced local search algorithms TA (with scenario L2), TTA (with scenario M7), GDA (with scenario L5), and TGDA (with scenario M3).

Besides the 12 medium and large networks, the test networks also include small size networks, such as networks 10_05 and 20_05. Note that the optimal solutions of the small networks 10_05 and 20_05 can be found by manual examination. As a result, the testing includes five algorithms, seven parameter scenarios, 14 networks and 98 problem instances. The results, in terms of the objective value error (compared with the best solutions founded previously), are shown in Table 5.

Table 5. The test results for GA, TA, TTA, GDA and TGDA with the appropriate parameters

Network problem	Initial solution (CCISA) (%)	Algorithm (scenario)						
		GA(J) (%)	GA(O) (%)	GA(W) (%)	TA(L2) (%)	TTA(M7) (%)	GDA(L5) (%)	TGDA(M3) (%)
10_05	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20_05	1.1	0.0	0.0	0.0	0.2	0.2	0.2	0.2
50_02	7.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50_05	14.7	0.0	2.3	0.0	3.5	0.0	4.1	3.5
50_08	24.5	0.0	0.0	0.0	1.8	3.0	0.0	1.8
100_02	22.5	5.6	3.7	5.2	12.6	9.9	5.3	12.6
100_05	33.0	7.7	4.4	3.7	5.9	8.1	4.3	5.7
100_08	32.8	3.4	3.6	1.0	10.0	9.5	9.8	7.8
150_02	38.5	5.2	5.1	0.3	10.2	8.7	8.6	8.7
150_05	41.8	2.5	1.6	5.4	12.6	9.8	7.5	9.7
150_08	46.0	0.0	3.1	0.5	3.2	6.2	5.2	3.5
500_02	69.3	0.0	2.8	6.1	2.7	4.8	4.7	7.0
500_05	69.4	3.5	1.3	3.0	8.9	4.9	4.2	5.9
500_08	65.9	2.9	1.5	6.2	5.0	3.8	1.6	6.2
Average	33.39	2.20	2.10	2.24	5.47	4.92	3.96	5.19

Note: The shadowed values indicate the relative minimum errors for the associated networks solved using the associated algorithms.

As shown in Table 5, the global search algorithm can find optimal solutions for the two small-scale networks 10_05 and 20_05. The four advanced local search algorithms can find optimal solutions for the small network 10_05, while near-optimal solutions, with a relative error of less than 0.2% are found for network 20_05, which implies that the advanced local search algorithms cannot efficiently "jump" out of local optima.

The global search algorithm results for scenarios J, O and W, are similar to the previously tested results. Scenario O yields the best solution quality, with an average error of 2.10%, scenario J is second, with an average error of 2.20%, and scenario W is third, with an average error of 2.24%. The average computation times for scenarios J, O and W are 1011.2, 1178.4 and 465.2 s, respectively.

Of the four advanced local search algorithms, GDA (scenario L5) yields the best solution quality, with an average error of 3.96%, TTA (scenario M7) is second, with an average error of 4.92%, TGDA (scenario M3) is third, with an average error of 5.19%, while TA (scenario L2) is the worst, with an average error of 5.47%. The average computation times for TA, TTA, GDA and TGDA are 173.0, 258.7, 247.5 and 248.5 s, respectively.

To further compare the performance of the global search algorithm with that of the advanced local search algorithms, we perform another test similar to the previous one. We first generate 14 test networks similar to the previous ones. The network types are shown in Table 6. For simplicity,

Table 6. Additional tested networks

No. of nodes	No. of arcs	Network density (%)
10	22	22.0
20	44	11.0
50	274	11.0
100	816	8.2
150	1328	5.9
500	5544	2.2

every test problem is then solved using the global search algorithm GA (with three scenarios J, O, W) and the advanced local search algorithms TA (with scenario L2), TTA (with scenario M7), GDA (with scenario L5), and TGDA (with scenario M3). The testing thus includes five algorithms, seven parameter scenarios, 14 networks and 98 problem instances. The results, in terms of the objective value error (compared with the best solutions found for the seven parameter scenarios), are shown in Table 7.

Similar to the above tests, the global search algorithm performs, on average, better than the local search algorithms. In particular, for the global search algorithm, scenario O yields the best solution quality, with an average error of 0.66%, scenario J is second, with an average error of 1.30%, and scenario W is third, with an average error of 1.97%. The average computation times for scenarios J, O and W are 1122.5, 1317.6 and 548.2 s, respectively.

Table 7. Additional test results for GA, TA, TTA, GDA and TGDA, with the appropriate parameters

Network problem	Initial solution (CCISA) (%)	Algorithm (scenario)						
		GA(J) (%)	GA(O) (%)	GA(W) (%)	TA(L2) (%)	TTA(M7) (%)	GDA(L5) (%)	TGDA(M3) (%)
10_05	10.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20_05	7.2	1.4	0.0	3.0	3.0	3.0	3.0	3.0
50_02	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50_05	19.3	0.0	0.0	0.0	6.6	6.7	1.4	7.2
50_08	16.1	0.3	0.0	0.0	0.0	0.0	0.0	0.0
100_02	19.3	0.0	0.0	0.0	3.0	0.2	0.4	3.0
100_05	27.6	3.3	0.0	4.4	11.1	2.9	4.9	3.2
100_08	47.3	1.5	0.0	1.8	4.3	4.8	2.5	6.6
150_02	29.2	0.0	5.1	2.9	3.5	1.7	2.7	4.8
150_05	32.8	2.9	2.0	0.0	6.3	3.9	3.4	1.0
150_08	30.3	2.6	0.0	3.6	1.8	0.8	0.0	3.4
500_02	77.8	4.4	0.0	3.4	9.3	1.2	2.3	5.8
500_05	68.9	0.0	2.2	4.0	4.7	1.1	5.3	4.5
500_08	46.8	1.8	0.0	4.5	6.2	3.9	2.8	3.2
Average	31.26	1.30	0.66	1.97	4.27	2.16	2.05	3.26

Note: The shadowed values indicate the relative minimum errors for the associated networks solved using the associated algorithms.

Of the four advanced local search algorithms, GDA (scenario L5) yields the best solution quality, with an average error of 2.05%, TTA (scenario M7) is second, with an average error of 2.16%, TGDA (scenario M3) is third, with an average error of 3.26%, while TA (scenario L2) is the worst, with an average error of 4.27%. The average computation times for TA, TTA, GDA and TGDA are 196.0, 292.3, 276.1 and 277.2 s, respectively.

To summarize the test results, for all the networks, GA performs better than the four advanced local search algorithms in terms of solution quality, although the former needs more computation time than the latter. However, for planning issues, the solution quality is far more important than the computation time in practice. In particular, the average objective value errors yielded by the three GA scenarios, in the first test, are between 2.10 and 2.24%, which is smaller than the 3.96–5.47% range yielded by the four advanced local search algorithms. In the second test, the average objective value errors yielded by the three GA scenarios in the first test are between 0.66% and 1.97%, which is smaller than the 2.05% to 4.27% range yielded by the four advanced local search algorithms. Note that, in order to save time, each network problem is solved only once in this additional test, which might lead to the results being affected by random disturbances. However, for the seven parameter scenarios, 28 tested networks and 196 tested problem instances, the average errors for the global search algorithm are all superior to the four advanced local search algorithms. This indicates that, on average, the global search algorithm is better than the four advanced local search algorithms.

7. Conclusions and Suggestions

Based on the problem's characteristics, we have developed a global search algorithm to solve the minimum cost network flow problems with concave arc costs. To evaluate the effectiveness of the proposed global search algorithm, we also developed four advanced local search algorithms, based on the threshold accepting algorithm, the GDA, and the tabu search algorithm, for comparison purposes. A randomized network generator is designed to produce test problems. All the computer programs are written in C++ language and tested on a personal computer. The results indicate that the proposed global search algorithm is more effective than the four advanced local search algorithms for solving the concave cost transshipment problems. Some of the most notable conclusions are as follows:

1. The minimum cost network flow problem with concave arc costs is a non-linear program (NLP) and is characterized as NP-hard. It does not belong to the traditional combinatorial optimization problems often formulated as integer programs. However, due to the problem's

properties, its optimal solutions are located at extreme points that are integers given all integer parameters. Hence, similar to traditional integer programming problems, the candidates for optimal solutions are spread throughout the linear solution set. As a result, traditional combinatorial optimization techniques, such as global and local search algorithms, which have rarely been applied to the problem, can be employed to solve it.

2. The proposed encoding and crossover methods discussed in this paper are different from those in the literature. In addition to the breadth of the crossover search, the proposed global search algorithm enhances the depth search by incorporating a mutation method that employs a local search technique, which improves the solutions generated by the crossover. With the population composition strategies, as well as the initial solutions being tailored to the problem characteristics, the proposed global search algorithm is able to solve for good solutions to minimum cost network flow problem with concave arc costs, as the test results show.
3. Four advanced local search algorithms, TA, TTA, GDA, and TGDA, are tested and yield good solutions. Nevertheless, the proposed global search algorithm performs still better than the four advanced local search algorithms in terms of solution quality.

Several directions of future research are suggested as follows:

1. The assumptions made in this research, such as the concave arc cost function of $c_{ij}/\sqrt{x_{ij}}$, and the arc flow without an upper bound can be relaxed, and the algorithms suitably modified in future. Moreover, similar global search algorithms may be developed for other types of concave cost objective functions.
2. Many problem instances of different sizes are generated using a random network generator; the small-scale networks could be optimally solved by manual examination. As for medium- or large-scale networks, since their optimal solutions are unknown, we only use the solutions for each algorithm to define the relative errors for comparison purposes. Other methods, such as the branch-and-bound method, could be developed in the future, to obtain good upper and lower bounds, which are useful for evaluating the accuracy of a solution.

Acknowledgements

This research was supported by a grant (NSC-91-2211-E-008 -042) from the National Science Council of Taiwan. We would like to thank the two anonymous referees for their helpful comments and suggestions on the presentation of the paper.

References

1. Abuali, F.N., Wainwright, R.L. and Schoenefeld D.A. (1995), Determinant Factorization: a new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem, *Proceedings of the Sixth International Conference on Genetic Algorithms* 470–477.
2. Ahuja, R.K., Maganti, T.L. and Orlin, J.B. (1993), *Network Flows, Theory, Algorithms, and Applications,* Prentice Hall, Englewood Cliffs.
3. Amiri, A. and Pirkul, H. (1997), New formulation and relaxation to solve a concave cost network flow problem, *Journal of the Operational Research Society* 48, 278–287.
4. Balakrishnan, A. and Graves S.C. (1989), A composite algorithm for a concave-cost network flow problem, *Networks* 19, 175–202.
5. Blumenfeld, D.E., Burns, L.D., Diltz, J.D. and Daganzo, C.F. (1985), Analyzing trade-offs between transportation, inventory, and production costs on freight network, *Transportation Research* 19B, 361–380.
6. Booker, L.B. (1987), Improving search in genetic algorithms, In: Davis, L. (ed.), *Genetic Algorithms and Simulated Annealing*, Pitman, London, pp. 61–73.
7. Charon, I. and Hurdy, O. (1993), The noising method: a new method for combinatorial optimization, *Operations Research Letters* 14, 133–137.
8. Cheng, C.P., Liu, C.W. and Liu, C.C. (2000), Unit commitment by Lagrangian relaxation and genetic algorithms, *IEEE Transactions on Power Systems* 15, (2).
9. Davis, L. (1987), *Genetic Algorithm and Simulated Annealing*, Morgan Kaufman Publishers, Los Altos, CA.
10. Davis, L. (1989), Adapting operator probabilities in genetic algorithms, *Proceedings of the Third International Conference on Genetic Algorithms*, 61–69.
11. Davis, L. (1991), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.
12. DeJong, K.A. (1975), Analysis of the behavior of a class of genetic adaptive systems, Ph.D. dissertation, University of Michigan.
13. Dueck, G. (1993), New optimization heuristics: the great deluge algorithm and the record-to-record travel, *Journal of Computational Physics* 104, 86–92.
14. Dueck, G. and Scheuer, T. (1990), Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, *Journal of Computational Physics* 90, 161–175.
15. Kim, D. and Pardalos, P.M. (2000), Dynamic slope scaling and trust interval techniques for solving concave piecewise linear network flow problems, *Networks* 35, 216–222.
16. Gallo, G., Sandi C. and Sodini, C. (1980), An algorithm for the min concave cost flow problem, *European Journal of Operation Research* 4, 248–255.
17. Gallo, G. and Sandi, C. (1979), Adjacent extreme flows and application to min concave cost flow problems, *Networks* 9, 95–121.
18. Gen, M. and Cheng, R. (1997), *Genetic Algorithms and Engineering Design*, Wiley Interscience Publication, MA.
19. Glover, F. and Laguna, M. (1997), *Tabu Search*, Kluwer Academic Publishers, Massachusetts.
20. Glover, F. (1989), Tabu Search, Part I, *ORSA Journal on Computing* 1(3), 190–206.
21. Glover, F. (1990), Tabu Search- Part II, *ORSA Journal on Computing* 2(1), 4–32.
22. Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading MA.
23. Golden, B.L., and Skiscim, C.C. (1986), Using stimulated annealing to solve routing and location problems, *Naval Research Logistic Quarterly* 33, 261–279.
24. Gu, J. and Huang, X. (1994), Efficient local search with search space smoothing: a case study of the traveling salesman problem (TSP), *IEEE Transaction on Systems, Man and Cybernetics* 24, 728–739.

25. Guisewite, G.M. and Pardalos, P.M. (1993), A polynomial time solvable concave network flow problems, *Networks* 23, 143–147.
26. Hall, R.W. (1983), Direct versus terminal freight routing on network with concave costs, GMR-4517, Transportation Research Dept., GM Research Laboratories.
27. Jeffrey, A.J. and Christopher, R.H. (1994), On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's, Department of Industrial Engineering North Carolina State University, NC 27695–7906.
28. Jordan, W.C. (1986), Scale economies on multi-commodity networks, GMR-5579, Operating Systems Research Dept., GM Research Laboratories.
29. Kershenbaum, A. (1997), When genetic algorithms work best, *INFORMS Journal of Computing* 9(3), 253–254.
30. Kirkpatrick, S., Gelatt, C.D. and Vecchi, M.P. (1983), Optimization by simulated annealing, *Science* 220, 671–680.
31. Kuhn, H.W. and Baumol, W.J. (1962), An approximate algorithm for the fixed-charge transportation problem, *Naval Res. Logistics Quarterly* 9, 1–16.
32. Larsson, T., Migdalas, A. and Ronnqvist, M. (1994), A Lagrangian heuristic for the capacitated concave minimum cost network flow problem, *European Journal of Operational Research* 78, 116–129.
33. Mathias, K.E. and Whitley, L.D. (1994), Initial performance comparisons for the delta coding algorithm, *The First IEEE Conference on Evolutionary Computation*, Orlando, Florida.
34. Nourie, F.J. and Guder, F. (1994), A restricted-entry method for a transportation problem with piecewise-linear concave cost, *Computer & Operations Research* 21, 723–733.
35. Osman, I.H. and Kelly, J.P. (1996), *Meta-Heuristics: An Overview*, *Meta-Heuristics: Theory & Applications*, Kluwer Academic Publishers, Boston, London, Dordrecht, 1–21.
36. Palmer, C.C. and Kershenbaum, A. (1995), An approach to a problem in network design using genetic algorithms, *Networks* 26, 151–163.
37. Rech, P. and Barton, L.G. (1970), A Non-convex transportation algorithm, In: Beale, E.M. (ed.), *Applications of Mathematical Programming Techniques*.
38. Reeves, C. (1997), Genetic algorithms for the operations researcher, *INFORMS Journal on Computing* 9(3), 231–250.
39. Reeves, C.R. (1993), *Modern Heuristic Techniques for Combinatorial Problems*, John Wiley and Sons, Inc.
40. Rudolph, G. (1994), Convergence properties of canonical genetic algorithms, *IEEE Transactions On Neural Networks* 5, 96–101.
41. Seiichi, K., Maggie, K. and Wayne, W.D. (1995), *Genetic Simulated Annealing and Application to Non-slicing Floor plan Design*, Baskin Center for Computer Engineering & Information Sciences University of California, Santa Cruz, CA 95064.
42. Srinivas, M. and Patnaik, L.M. (1994), Adaptive probabilities of crossover and mutation in genetic algorithms, *IEEE Transaction On System* Man, and Cybern 24, 656–667.
43. Suwan, R. and Sawased, T. (1999), Link capacity assignment in packet-switched networks: the case of piecewise linear concave cost function, *IEICE Transaction Communications* E82-B(10).
44. Taguhi, T., Ida. K. Gen, M. (1998), A genetic algorithm for optimal flow assignment in computer network, *Computers and Industrial Engineering* 35(3–4), 535–538.
45. Thach, P.T. (1992), A decomposition method using a pricing mechanism for min concave cost flow problems with a hierarchical structure, *Mathematical Programming* 53, 339–359.
46. Thierens, D. and Goldberg D. (1994), Elitist recombination: an integrated selection recombination GA, *The First IEEE Conference on Evolutionary Computation*, Orlando, Florida.

47. Yaged, B. (1971), Minimum cost routing for static network models, *Networks* 1, 139–172.
48. Yan, S. (1996), Approximating reduced costs under degeneracy in a network flow problem with side constraints, Networks 27, 267–278.
49. Yan, S., and Luo, S.C. (1998), A tabu search-based algorithm for concave cost transportation network problems, *Journal of the Chinese Institute of Engineers* 21, 327–335.
50. Yan, S. and Luo, S.C. (1999), Probabilistic local search algorithms for concave cost transportation network problems, *European Journal of Operational Research* 117, 511–521.
51. Yan, S. and Yang, D.H. (1996), A decision support framework for handling schedule perturbation, *Transportation Research* 30B, 405–419.
52. Zangwill, W.I. (1968), Minimum concave cost flows in certain networks, *Management Science* 14, 429–450.