

Chapter 2

Basic Network Models

Network design is one of the most important and most frequently encountered classes of optimization problems [1]. It is a combinatorial field in graph theory and combinatorial optimization. A lot of optimization problems in network design arose directly from everyday practice in engineering and management: determining shortest or most reliable paths in traffic or communication networks, maximal or compatible flows, or shortest tours; planning connections in traffic networks; coordinating projects; and solving supply and demand problems.

Furthermore, network design is also important for complexity theory, an area in the common intersection of mathematics and theoretical computer science which deals with the analysis of algorithms. However, there is a large class of network optimization problems for which no reasonable fast algorithms have been developed. And many of these network optimization problems arise frequently in applications. Given such a hard network optimization problem, it is often possible to find an efficient algorithm whose solution is approximately optimal. Among such techniques, the genetic algorithm (GA) is one of the most powerful and broadly applicable stochastic search and optimization techniques based on principles from evolution theory.

2.1 Introduction

Network design is used extensively in practice in an ever expanding spectrum of applications. Network optimization models such as shortest path, assignment, max-flow, transportation, transshipment, spanning tree, matching, traveling salesman, generalized assignment, vehicle routing, and multi-commodity flow constitute the most common class of practical network optimization problems.

In this chapter, we introduce some of the core models of network design (Fig. 2.1), such as shortest path models (*i.e.*, node selection and sequencing), spanning tree models (*i.e.*, arc selection) and maximum flow models (*i.e.*, arc selection and flow assignment) *etc.* These network models are used most extensively in applica-

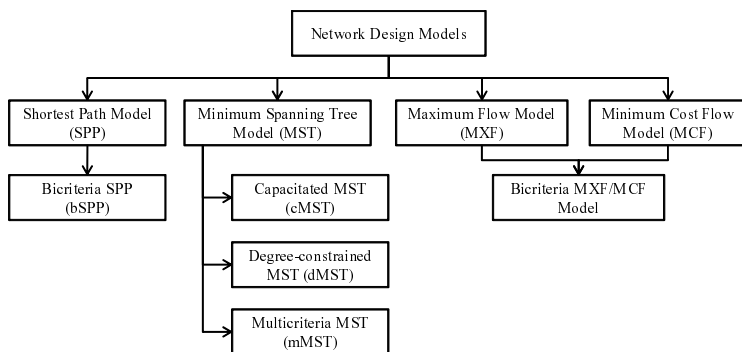


Fig. 2.1 The core models of network design introduced in this chapter

tions and differentiated by their structural characteristics. We assume that a connected graph $G = (N, A)$ is given, and it is a structure consisting of a finite set N of elements called nodes and a set A of unordered pairs of nodes called arcs. The descriptions of the models are as follows.

2.1.1 Shortest Path Model: Node Selection and Sequencing

It is desired to find a set of nodes. What is the shortest path between two specified nodes of G ?

The *shortest path model* is the heart of network design optimization. They are alluring to both researchers and practitioners for several reasons: (1) they arise frequently in practice since in a wide variety of application settings we wish to send some material (*e.g.*, a computer data packet, a telephone call, a vehicle) between two specified points in a network as quickly, as cheaply, or as reliably as possible; (2) as the simplest network models, they capture many of the most salient core ingredients of network design problems and so they provide both a benchmark and a point of departure for studying more complex network models; and (3) they arise frequently as subproblems when solving many combinatorial and network optimization problems [1]. Even though shortest path problems are relatively easy to solve, the design and analysis of most efficient algorithms for solving them requires considerable ingenuity. Consequently, the study of shortest path problems is a natural starting point for introducing many key ideas from network design problems, including the use of clever data structures and ideas such as data scaling to improve the worst case algorithmic performance. Therefore, in this chapter, we begin our discussion of network design optimization by studying shortest path models.

Applications

Shortest path problems arise in a wide variety of practical problem settings, both as stand-alone models and as subproblems in more complex problem settings.

1. *Transportation planning*: How to determine the route road that have prohibitive weight restriction so that the driver can reach the destination within the shortest possible time.
2. *Salesperson routing*: Suppose that a sales person want to go to Los Angeles from Boston and stop over in several city to get some commission. How can she determine the route?
3. *Investment planning*: How to determine the invest strategy to get an optimal investment plan.
4. *Message routing in communication systems*: The routing algorithm computes the shortest (least cost) path between the router and all the networks of the internet-work. It is one of the most important issues that has a significant impact on the network's performance.

2.1.2 Spanning Tree Model: Arc Selection

It is desired to find a subset of arcs such that when these arcs are removed from G , the graph remains connected. For what subset of arcs is the sum of the arc lengths minimized?

Spanning tree models play a central role within the field of network design. It generally arises in one of two ways, directly or indirectly. In some direct applications, we wish to connect a set of points using the least cost or least length collection of arcs. Frequently, the points represent physical entities such as components of a computer chip, or users of a system who need to be connected to each other or to a central service such as a central processor in a computer system [1]. In indirect applications, we either (1) wish to connect some set of points using a measure of performance that on the surface bears little resemblance to the minimum spanning tree objective (sum of arc costs), or (2) the problem itself bears little resemblance to an “optimal tree” problem – in these instances, we often need to be creative in modeling the problem so that it becomes a minimum spanning tree problem.

Applications

1. *Designing physical systems*: Connect terminals in cabling the panels of electrical equipment. How should we wire the terminals to use the least possible length of wire? Constructing a pipeline network to connect a number of towns, how should we using the smallest possible total length of pipeline.

2. *Reducing data storage*: We wish to store data specified in the form of a two-dimensional array more efficiently than storing all the elements of the array (to save memory space). The total storage requirement for a particular storage scheme will be the length of the reference row (which we can take as the row with the least amount of data) plus the sum of the differences between the rows. Therefore, a minimum spanning tree would provide the least cost storage scheme.
3. *Cluster analysis*: The essential issue in cluster analysis is to partition a set of data into “natural groups”; the data points within a particular group of data, or a cluster, should be more “closely related” to each other than the data points not in the cluster. We can obtain n partitions by starting with a minimum spanning tree and deleting tree arcs one by one in nonincreasing order of their lengths. Cluster analysis is important in a variety of disciplines that rely on empirical investigations.

2.1.3 Maximum Flow Model: Arc Selection and Flow Assignment

It is desired to send as much flow as possible between two special nodes of G , without exceeding the capacity of any arc.

The *maximum flow model* and the shortest path model are complementary. They are similar because they are both pervasive in practice and because they both arise as subproblems in algorithms for the minimum cost flow problem. The two problems differ because they capture different aspects. Shortest path problems model arc costs but not arc capacities; maximum flow problems model capacities but not costs. Taken together, the shortest path problem and the maximum flow problem combine all the basic ingredients of network design optimization. As such, they have become the cores of network optimization [1].

Applications

The maximum flow problems arise in a wide variety of situations and in several forms. For example, sometimes the maximum flow problem occurs as a subproblem in the solution of more difficult network problems, such as the minimum cost flow problems or the generalized flow problem. The problem also arises directly in problems as far reaching as machine scheduling, the assignment of computer modules to computer processors, the rounding of census data to retain the confidentiality of individual households, and tanker scheduling.

1. *Scheduling on uniform parallel machines*: The feasible scheduling problem, described in the preceding paragraph, is a fundamental problem in this situation and can be used as a subroutine for more general scheduling problems, such as the maximum lateness problem, the (weighted) minimum completion time problem, and the (weighted) maximum utilization problem.

2. *Distributed computing on a two-processor computer*: Distributed computing on a two-processor computer concerns assigning different modules (subroutines) of a program to two processors in a way that minimizes the collective costs of interprocessor communication and computation.
3. *Tanker scheduling problem*: A steamship company has contracted to deliver perishable goods between several different origin-destination pairs. Since the cargo is perishable, the customers have specified precise dates (*i.e.*, delivery dates) when the shipments must reach their destinations.

2.1.4 Representing Networks

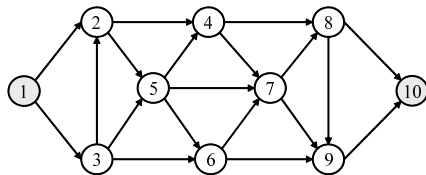


Fig. 2.2 A digraph G

Definition 2.1. Edge Lists

A *directed graph* (or *digraph*) G on the nodes set $N = \{1, 2, \dots, n\}$ is specified by:

1. Its number of nodes n ;
2. The list of its arcs, given as a sequence of ordered m pairs $A = \{(i, j), (k, l), \dots, (s, t)\}$.

The digraph G of Fig. 2.2 may then be given as follows:

1. Its number of nodes $n = 10$;
2. The list of its arcs $A = \{(1, 2), (1, 3), (2, 4), (2, 5), (3, 2), (3, 5), (3, 6), (4, 7), (4, 8), (5, 4), (5, 6), (5, 7), (6, 7), (6, 9), (7, 8), (7, 9), (8, 9), (8, 10), (9, 10)\}$.

Definition 2.2. Adjacency Lists

A digraph with node set $N = \{1, 2, \dots, n\}$ is specified by:

1. The number of nodes n ;
2. n lists $S_1, \dots, S_i, \dots, S_n$, where S_i contains all nodes j for which G contains an arc (i, j) .

The digraph G of Fig. 2.2 may be represented by *adjacency lists* as follows:

1. Its number of nodes $n = 10$;
2. The adjacency lists: $S_1 = \{2, 3\}$, $S_2 = \{4, 5\}$, $S_3 = \{2, 5, 6\}$, $S_4 = \{7, 8\}$, $S_5 = \{4, 6, 7\}$, $S_6 = \{7, 9\}$, $S_7 = \{8, 9\}$, $S_8 = \{9, 10\}$, $S_9 = \{10\}$ and $S_{10} = \phi$.

Definition 2.3. Adjacency Matrices

A digraph with node set $N = \{1, 2, \dots, n\}$ is specified by an $(n \times n)$ -matrix $A = (a_{ij})$, where $a_{ij} = 1$ if and only if (i, j) is an arc of G , and $a_{ij} = 0$ otherwise. A is called the *adjacency matrix* of G . The digraph G of Fig. 2.2 may then be given as follows:

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{matrix} & \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Adjacency matrices can be implemented simply as an array $[1, \dots, n, 1, \dots, n, \dots]$. As they need a lot of space in memory (n^2 places), they should only be used (if at all) to represent digraphs having many arcs. Though adjacency matrices are of little practical interest, they are an important theoretical tool for studying digraphs.

2.1.5 Algorithms and Complexity

2.1.5.1 Algorithms

Algorithms are techniques for solving problems.

Here the term problem is used in a very general sense: a problem class comprises infinitely many instances having a common structure. In general, an algorithm is a technique which can be used to solve each instance of a given problem class. An algorithm should have the following properties [3]:

1. *Finiteness of description*: The technique can be described by a finite text.
2. *Effectiveness*: Each step of the technique has to be feasible (mechanically) in practice.
3. *Termination*: The technique has to stop for each instance after a finite number of steps.
4. *Determinism*: The sequence of steps has to be uniquely determined for each instance.

Of course, an algorithm should also be correct, that is, it should indeed solve the problem correctly for each instance. Moreover, *an algorithm should be efficient, which means it should work as fast and economically as possible* [3].

2.1.5.2 Complexity

Computational complexity theory is the study of the complexity of problems – that is, the difficulty of solving them. Problems can be classified by *complexity class* according to the time it takes for an algorithm to solve them as function of the *problem size*. For example, the travelling salesman problem can be solved in time $O(n^2 2^n)$ (where n is the size of the network to visit).

Even though a problem may be solvable computationally in principle, in actual practice it may not be that simple. These problems might require large amounts of time or an inordinate amount of *space*. Computational complexity may be approached from many different aspects. Computational complexity can be investigated on the basis of *time*, memory or other resources used to solve the problem. Time and space are two of the most important and popular considerations when problems of complexity are analyzed. The *time complexity* of a problem is the number of steps that it takes to solve an instance of the problem as a function of the size of the input (usually measured in bits), using the most efficient algorithm. To understand this intuitively, consider the example of an instance that is n bits long that can be solved in n^2 steps. In this example we say the problem has a time complexity of n^2 . Of course, the exact number of steps will depend on exactly what machine or language is being used. To avoid that problem, the Big O notation is generally used. If a problem has time complexity $O(n^2)$ on one typical computer, then it will also have complexity $O(n^2)$ on most other computers, so this notation allows us to generalize away from the details of a particular computer.

2.1.6 NP-Complete

In complexity theory, the *NP-complete* problems are the most difficult problems in NP (“non-deterministic polynomial time”) in the sense that they are the smallest subclass of NP that could conceivably remain outside of P, the class of deterministic polynomial-time problems. The reason is that a deterministic, polynomial-time solution to any NP-complete problem would also be a solution to every other problem in NP. The complexity class consisting of all NP-complete problems is sometimes referred to as NP-C. A more formal definition is given below.

A decision problem C is NP-complete if it is complete for NP, meaning that:

1. It is in NP and
2. It is NP-hard, *i.e.* every other problem in NP is reducible to it.

“Reducible” here means that for every problem L , there is a polynomial-time many-one reduction, a deterministic algorithm which transforms instances $l \in L$ into instances $c \in C$, such that the answer to c is Yes if and only if the answer to l is Yes. To prove that an NP problem A is in fact an NP-complete problem it is sufficient to show that an already known NP-complete problem reduces to A . A consequence of this definition is that if we had a polynomial time algorithm for C , we could solve all problems in NP in polynomial time.

This definition was given by Cook [5], though the term “NP-complete” did not appear anywhere in his paper. At that computer science conference, there was a fierce debate among the computer scientists about whether NP-complete problems could be solved in polynomial time on a deterministic Turing machine. John Hopcroft brought everyone at the conference to a consensus that the question of whether NP-complete problems are solvable in polynomial time should be put off to be solved at some later date, since nobody had any formal proofs for their claims one way or the other. This is known as the question of whether $P=NP$.

2.1.7 List of NP-complete Problems in Network Design

Routing Problems

Bottleneck traveling salesman, Chinese postman for mixed graphs, Euclidean traveling salesman, K most vital arcs, K -th shortest path, Metric traveling salesman, Longest circuit, Longest path, Prize Collecting Traveling Salesman, Rural Postman, Shortest path in general networks, Shortest weight-constrained path, Stacker-crane, Time constrained traveling salesman feasibility, Traveling salesman problem, Vehicle Routing

Spanning Trees

Degree constrained spanning tree, Maximum leaf spanning tree, Shortest total path length spanning tree, Bounded diameter spanning tree, Capacitated spanning tree, Geometric capacitated spanning tree, Optimum communication spanning tree, Isomorphic spanning tree, K -th best spanning tree, Bounded component spanning forest, Multiple choice branching, Steiner tree, Geometric Steiner tree, Cable Trench Problem

Flow Problems

Minimum edge-cost flow, Integral flow with multipliers, Path constrained network flow, Integral flow with homologous arcs, Integral flow with bundles, Undirected flow with lower bounds, Directed two-commodity integral flow, Undirected two-

commodity integral flow, Disjoint connecting paths, Maximum length-bounded disjoint paths, Maximum fixed-length disjoint paths, Unsplittable multicommodity flow

Cuts and Connectivity

Graph partitioning, Acyclic partition, Max weight cut, Minimum cut into bounded sets, Biconnectivity augmentation, Strong connectivity augmentation, Network reliability, Network survivability, Multiway Cut, Minimum k -cut

2.2 Shortest Path Model

Shortest path problem (SPP) is at the heart of network optimization problems. Being the simplest network model, shortest path can capture most significant ingredients of network optimization problem. Even though it is relatively easy to solve a shortest path problem, the analysis and design of efficient algorithms requires considerable ingenuity. Consequently, the study of models for shortest path is a natural beginning of network models for introducing innovative ideas, including the use of appropriate data structures and data scaling to improve algorithmic performance in the worst cases.

Furthermore, the traditional routing problem has been a single objective problem having as a goal the minimization of either the total distance or travel time. However, in many applications dealing with the design and efficient use of networks, the complexity of the social and economic environment requires the explicit consideration of objective functions other than cost or travel time. In other words, it is necessary to take into account that many real world problems are multi-criteria in nature. The objective functions related to cost, time, accessibility, environmental impact, reliability and risk are appropriated for selecting the most satisfactory (*i.e.*, “textitbest compromise”) route in many network optimization problems [6].

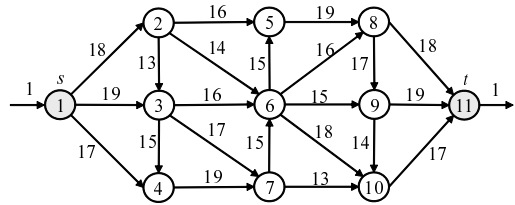
Example 2.1: A Simple Example of SPP

A simple example of shortest path problem is shown in Fig. 2.3, and Table 2.1 gives the data set of the example.

Table 2.1 The data set of illustrative shortest path problem

| | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| i | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | 4 | 5 | 6 | 6 | 6 | 6 | 7 | 7 | 8 | 8 | 9 | 9 | 10 |
| j | 2 | 3 | 4 | 3 | 5 | 6 | 4 | 6 | 7 | 7 | 8 | 5 | 8 | 9 | 10 | 6 | 10 | 9 | 11 | 10 | 11 | 11 |
| c_{ij} | 18 | 19 | 17 | 13 | 16 | 14 | 15 | 16 | 17 | 19 | 19 | 15 | 16 | 15 | 18 | 15 | 13 | 17 | 18 | 14 | 19 | 17 |

Fig. 2.3 A simple example of shortest path problem



Traditional Methods

A method to solve SPP is sometimes called a *routing algorithm*. The most important algorithms for solving this problem are:

Dijkstra's algorithm: Solves single source problem if all edge weights are greater than or equal to zero. Without worsening the run time, this algorithm can in fact compute the shortest paths from a given start point s to all other nodes.

Bellman-Ford algorithm: Computes single-source shortest paths in a weighted digraph (where some of the edge weights may be negative). Dijkstra's algorithm accomplishes the same problem with a lower running time, but requires edge weights to be nonnegative. Thus, Bellman-Ford is usually used only when there are negative edge weights.

Floyd-Warshall algorithm: An algorithm to solve the all pairs shortest path problem in a weighted, directed graph by multiplying an adjacency-matrix representation of the graph multiple times.

Recently, to address SPP, neural networks (NNs) and GAs (and other evolutionary algorithms) received a great deal of attention regarding their potential as optimization techniques for network optimization problems [7]–[9]. They are often used to solve optimization problems in communication network optimization problems, including the effective approaches on the *shortest path routing* (SPR) problem [10]–[14], *multicasting routing* problem [11], *ATM bandwidth allocation* problem [15], *capacity and flow assignment* (CFA) problem [16], and the *dynamic routing* problem [17]. It is noted that all these problems can be formulated as some sort of a combinatorial optimization problem.

2.2.1 Mathematical Formulation of the SPP Models

Let $G = (N, A)$ be a *directed network*, which consists of a finite set of nodes $N = \{1, 2, \dots, n\}$ and a set of directed arcs $A = \{(i, j), (k, l), \dots, (s, t)\}$ connecting m pairs of nodes in N . Arc (i, j) is said to be incident with nodes i and j , and is directed from node i to node j . Suppose that each arc (i, j) has been assigned to a nonnegative value c_{ij} , the cost of (i, j) . The SPP can be defined by the following assumptions:

- A1. The network is directed. We can fulfil this assumption by transforming any undirected network into a directed one.
- A2. All transmission delay and all arc costs are nonnegative.
- A3. The network does not contain parallel arcs (*i.e.*, two or more arcs with the same tail and head nodes). This assumption is essentially for notational convenience.

Indices

i, j, k : index of node $(1, 2, \dots, n)$

Parameters

n : number of nodes
 c_{ij} : transmission cost of arc (i, j)
 d_{ij} : transmission delay of arc (i, j)

Decision variables

x_{ij} : the link on an arc $(i, j) \in A$.

2.2.1.1 Shortest Path Model

The SPP is to find the minimum cost z from a specified *source node* 1 to another specified *sink node* n , which can be formulated as follows in the form of integer programming:

$$\min z = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \quad (2.1)$$

$$\text{s. t. } \sum_{j=1}^n x_{ij} - \sum_{k=1}^n x_{ki} = \begin{cases} 1 & (i=1) \\ 0 & (i=2, 3, \dots, n-1) \\ -1 & (i=n) \end{cases} \quad (2.2)$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i, j \quad (2.3)$$

2.2.1.2 Bicriteria Shortest Path Model

The *bicriteria shortest path problem* (bSPP) is formulated as follows, in which the objectives are minimizing cost function z_1 and minimizing delay function z_2 from source node 1 to sink node n :

$$\min z_1 = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \quad (2.4)$$

$$\min z_2 = \sum_{i=1}^n \sum_{j=1}^n d_{ij} x_{ij} \quad (2.5)$$

$$\text{s. t. } \sum_{j=1}^n x_{ij} - \sum_{k=1}^n x_{ki} = \begin{cases} 1 & (i = 1) \\ 0 & (i = 2, 3, \dots, n-1) \\ -1 & (i = n) \end{cases} \quad (2.6)$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i, j \quad (2.7)$$

with constraints at Eqs. 2.2 and 2.6, a flow conservation law is observed at each of the nodes other than s or t . That is, what goes out of node i , $\sum_{j=1}^n x_{ij}$ must be equal to what comes in, $\sum_{k=1}^n x_{ki}$.

2.2.2 Priority-based GA for SPP Models

Let $P(t)$ and $C(t)$ be parents and offspring in current generation t , respectively. The overall procedure of *priority-based GA* (priGA) for solving SPP models is outlined as follows:

procedure: priGA for SPP models

input: network data (N, A, c, d) ,

GA parameters ($popSize, maxGen, p_M, p_C, p_I$)

output: Pareto optimal solutions E

begin

$t \leftarrow 0$;

initialize $P(t)$ by priority-based encoding routine;

calculate objectives $z_i(P), i = 1, \dots, q$ by priority-based decoding routine;

create Pareto $E(P)$;

evaluate $eval(P)$ by *interactive adaptive-weight fitness assignment routine*;

while (not terminating condition) **do**

create $C(t)$ from $P(t)$ by weight mapping crossover routine;

create $C(t)$ from $P(t)$ by insertion mutation routine;

create $C(t)$ from $P(t)$ by immigration routine;

calculate objectives $z_i(C), i = 1, \dots, q$ by priority-based decoding routine;

update Pareto $E(P, C)$;

evaluate $eval(P, C)$ by *interactive adaptive-weight fitness assignment routine*;

select $P(t+1)$ from $P(t)$ and $C(t)$ by roulette wheel selection routine;

$t \leftarrow t + 1$;

end

output Pareto optimal solutions $E(P, C)$

end

2.2.2.1 Genetic Representation

How to encode a solution of the network design problem into a chromosome is a key issue for GAs. In Holland's work, encoding is carried out using binary strings. For many GA applications, especially for the problems from network design problems, the simple approach of GA was difficult to apply directly. During the 10 years, various nonstring encoding techniques have been created for network routing problems [8].

We need to consider these critical issues carefully when designing a new non-binary application string coding so as to build an effective GA chromosome. How to encode a path in a network is also critical for developing a GA application to network design problems, it is not easy to find out a nature representation. Special difficulty arises from (1) a path contains variable number of nodes and the maximal number is $n-1$ for an n node network, and (2) a random sequence of edges usually does not correspond to a path.

Variable-length Encoding

Recently, to encode a diameter-constrained path into a chromosome, various common encoding techniques have been created. Munemoto *et. al.* proposed a *variable-length encoding* method for network routing problems in a wired or wireless environment [10]. Ahn and Ramakrishna developed this variable-length representation and proposed a new crossover operator for solving the shortest path routing (SPR) problem [12]. The variable-length encoding method consists of sequences of positive integers that represent the IDs of nodes through which a path passes. Each locus of the chromosome represents an order of a node (indicated by the gene of the locus) in a path. The length of the chromosome is variable, but it should not exceed the maximum length n , where n is the total number of nodes in the network. The advantage of variable-length encoding is the mapping from any chromosome to solution (decoding) belongs to *1-to-1 mapping* (uniqueness). The disadvantages are: (1) in general, the genetic operators may generate infeasible chromosomes (illegality) that violate the constraints, generating loops in the paths; (2) repairing techniques are usually adopted to convert an illegal chromosome to a legal one.

An example of the variable-length chromosome and its decoded path are shown in Fig. 2.4, in terms of the directed network in Fig. 2.3.

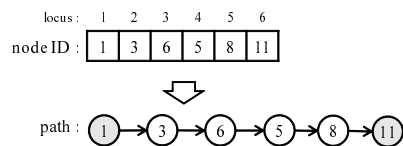


Fig. 2.4 An example of variable-length chromosome and its decoded path

Fixed-length Encoding

Inagaki *et al.* proposed a *fixed-length encoding* method for multiple routing problems [11]. The proposed method are sequences of integers and each gene represents the node ID through which it passes. To encode an arc from node i to node j , put j in the i -th locus of the chromosome. This process is reiterated from the source node 1 and terminating at the sink node n . If the route does not pass through a node x , select one node randomly from the set of nodes which are connected with node x , and put it in the x -th locus. The advantages of fixed-length encoding are: (1) any path has a corresponding encoding (completeness); (2) any point in solution space is accessible for genetic search; (3) any permutation of the encoding corresponds to a path (legality) using the special genetic operators. The disadvantages are: (1) in some cases, n -to-1 mapping may occur for the encoding; (2) in general, the genetic operators may generate infeasible chromosomes (illegality), and special genetic operator phase is required. Therefore we lose feasibility and heritability. An example of fixed-length chromosome and its decoded path is shown in Fig. 2.5, for the undirected network shown in Fig. 2.3.

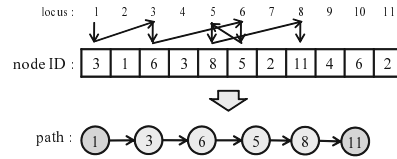


Fig. 2.5 An example of fixed-length chromosome and its decoded path

2.2.2.2 Priority-based Encoding and Decoding

Cheng and Gen proposed a priority-based encoding method firstly for solving Resource-constrained Project Scheduling Problem (rcPSP) [22]. Gen *et al.* also adopted priority-based encoding for the solving bSPP problem [23]. Recently, Lin and Gen proposed a *priority-based encoding* method [24]. As is known, a gene in a chromosome is characterized by two factors: locus, *i.e.*, the position of the gene located within the structure of chromosome, and allele, *i.e.*, the value the gene takes. In this encoding method, the position of a gene is used to represent node ID and its value is used to represent the priority of the node for constructing a path among candidates. A path can be uniquely determined from this encoding.

Illustration of priority-based chromosome and its decoded path are shown in Fig. 2.6. At the beginning, we try to find a node for the position next to source node 1. Nodes 2, 3 and 4 are eligible for the position, which can be easily fixed according to adjacent relation among nodes. The priorities of them are 1, 10 and 3, respectively. Node 3 has the highest priority and is put into the path. The possible nodes next to

node 3 are nodes 4, 6 and 7. Because node 6 has the largest priority value, it is put into the path. Then we form the set of nodes available for next position and select the one with the highest priority among them. Repeat these steps until we obtain a complete path, (1-3-6-5-8-11).

The advantages of the priority-based encoding method are: (1) any permutation of the encoding corresponds to a path (feasibility); (2) most existing genetic operators can be easily applied to the encoding; (3) any path has a corresponding encoding (legality); (4) any point in solution space is accessible for genetic search. However, there is a disadvantage as that n -to-1 mapping (uniqueness) may occur for the encoding at some case. For example, we can obtain the same path, (1-3-6-5-8-11) by different chromosomes, ($v_1 = [11, \underline{1}, 10, 3, 8, 9, \underline{5}, 7, 4, 2, 6]$ and $v_2 = [11, \underline{5}, 10, 3, 8, 9, \underline{1}, 7, 4, 2, 6]$).

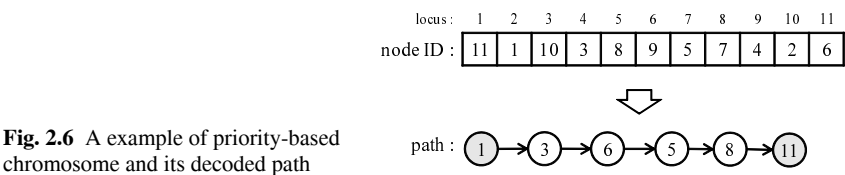


Fig. 2.6 A example of priority-based chromosome and its decoded path

In general, there are two ways to generate the initial population, *heuristic initialization* and *random initialization*. However, the mean fitness of the heuristic initialization is already high so it may help the GAs to find solutions faster. Unfortunately, in most large scale problems, it may just explore a small part of the solution space and it is difficult to find global optimal solutions because of the lack of diversity in the population [12]. Therefore, random initialization is effected in this research so that the initial population is generated with the priority-based encoding method. The encoding process and decoding process of the priority-based GA (priGA) are shown in Figures 2.7 and 2.8, respectively.

Depending on the properties of encodings (introduced on page 9), we summarize the performance of the priority-based encoding method and other introduced encoding methods in Table 2.2.

Table 2.2 Summarizes the performance of encoding methods

| Chromosome Design | | Space | Time | Feasibility | Uniqueness | Locality | Heritability |
|--------------------------|----------------------------|-------|---------------|-------------|-------------------|----------|--------------|
| Variable length-based GA | Ahn and Ramakrishna [13] | m | $O(m \log m)$ | Poor | 1-to-1 mapping | Worst | Worst |
| Fixed length-based GA | Inagaki <i>et al.</i> [12] | n | $O(n \log n)$ | Worst | n -to-1 mapping | Worst | Worst |
| Priority-based GA | Gen <i>et al.</i> [14] | n | $O(n \log n)$ | Good | n -to-1 mapping | Good | Good |

Fig. 2.7 Pseudocode of priority-based encoding method

```

procedure: priority-based encoding
input: number of nodes  $n$ 
output:  $k$ th initial chromosome  $v_k$ 
begin
  for  $i = 1$  to  $n$ 
     $v_k[i] \leftarrow i$ ;
  for  $i = 1$  to  $\lceil n/2 \rceil$ 
    repeat
       $j \leftarrow \text{random}[1, n]$ ;
       $l \leftarrow \text{random}[1, n]$ ;
    until ( $j \neq l$ );
    swap( $v_k[j], v_k[l]$ );
  output  $v_k$ ;
end

```

```

procedure: priority-based decoding
input: number of nodes  $n$ , chromosome  $v_k$ ,
        the set of nodes  $S_i$  with all nodes adjacent to node  $i$ 
output: path  $P_k$ 
begin
  initialize  $i \leftarrow 1, l \leftarrow 1, P_k[l] \leftarrow i$ ; //  $i$ : source node,  $l$ : length of path  $P_k$ .
  while  $S_i \neq \emptyset$  do
     $j' \leftarrow \text{argmax} \{v_k[j], j \in S_i\}$ ;
    //  $j'$ : the node with highest priority among  $S_i$ .
    if  $v_k[j'] \neq 0$  then
       $P_k[l] \leftarrow j'$ ; // chosen node  $j'$  to construct path  $P_k$ .
       $l \leftarrow l + 1$ ;
       $v_k[j'] \leftarrow 0$ ;
       $i \leftarrow j'$ ;
    else
       $S_i \leftarrow S_i \setminus j'$ ; // delete the node/ adjacent to node  $i$ .
       $v_k[i] \leftarrow 0$ ;
       $l \leftarrow l - 1$ ;
      if  $l \leq 1$  then  $l \leftarrow 1$ , break;
       $i \leftarrow P_k[l]$ ;
  output path  $P_k$ 
end

```

Fig. 2.8 Pseudocode of priority-based decoding method

2.2.2.3 Genetic Operators

Genetic operators mimic the process of heredity of genes to create new offspring at each generation. Using the different genetic operators has very large influence on GA performance [17]. Therefore it is important to examined different genetic operators.

For priority-based representation as a permutation representation , several crossover operators have been proposed, such as *partial-mapped crossover* (PMX), *order crossover* (OX), *cycle crossover* (CX), *position-based crossover* (PX), *heuristic crossover*, etc. Roughly, these operators can be classified into two classes:

- Canonical approach
- Heuristic approach

The canonical approach can be viewed as an extension of two-point or multi-point crossover of binary strings to permutation representation. Generally, permutation representation will yield illegal offspring by two-point or multi-point crossover in the sense of that some priority values may be missed while some priority values may be duplicated in the offspring. *Repairing procedure* is embedded in this approach to resolve the illegitimacy of offspring.

The essence of the canonical approach is the blind random mechanism. There is no guarantee that an offspring produced by this kind of crossover is better than their parents. The application of heuristics in crossover intends to generate an improved offspring.

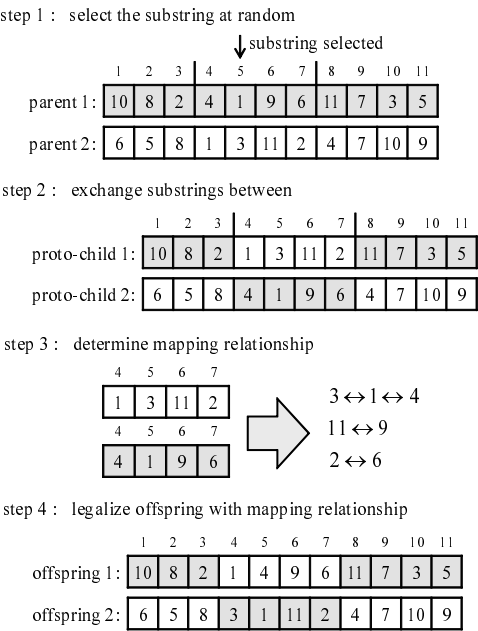
Partial-mapped Crossover (PMX)

PMX was proposed by Goldberg and Lingle [19]. PMX can be viewed as an extension of two-point crossover for binary string to permutation representation. It uses a special repairing procedure to resolve the illegitimacy caused by the simple two-point crossover. So the essentials of PMX are a simple two-point crossover plus a repairing procedure. The procedure is illustrated in Fig. 2.9. PMX works as follows:

| | |
|----------------------|--|
| Procedure PMX | |
| Input: | two parents |
| Output: | offspring |
| Step 1: | Select two positions along the string uniformly at random. The substrings defined by the two positions are called the <i>mapping sections</i> . |
| Step 2: | Exchange two substrings between parents to produce proto-children. |
| Step 3: | Determine the <i>mapping relationship</i> between two mapping sections. |
| Step 4: | Legalize offspring with the <i>mapping relationship</i> . |

Priority values 3, 11 and 2 are duplicated in *proto-child* 1 while values 4, 9 and 6 are missing from it. According to the mapping relationship determined in step 3,

Fig. 2.9 Illustration of the PMX operator



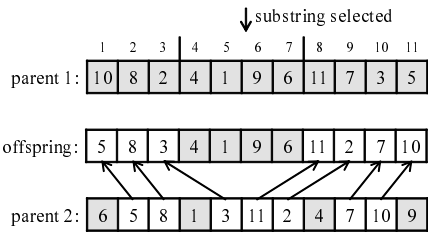
the excessive values 3, 11 and 2 should be replaced by the missing values 4, 9 and 6, respectively, while keeping the swapped substring unchanged.

Order Crossover (OX)

OX was proposed by Davis [20]. It can be viewed as a kind of variation of PMX with a different repairing procedure. The procedure is illustrated in Fig. 2.10. OX works as follows:

| | |
|---------------------|--|
| Procedure OX | |
| Input: | two parents |
| Output: | offspring |
| Step 1: | Select a substring from one parent at random. |
| Step 2: | Produce a proto-child by copying the substring into the corresponding positions of it. |
| Step 3: | Delete the nodes which are already in the substring from the second parent. The resulted sequence of nodes contains the nodes that the proto-child needs. |
| Step 4: | Place the nodes into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce an offspring. |

Fig. 2.10 Illustration of the OX operator



2.2.2.4 Position-based Crossover (PX)

Position-based crossover was proposed by Syswerda [21]. It is essentially a kind of uniform crossover for permutation representation together with a repairing procedure. It can also be viewed as a kind of variation of OX where the nodes are selected inconsecutively. The procedure is illustrated in Fig. 2.11. PX works as follows:

Procedure PX

- Input:** two parents
- Output:** offspring
- Step 1:** Select a set of positions from one parent at random.
- Step 2:** Produce a proto-child by copying the nodes on these positions into the corresponding positions of it.
- Step 3:** Delete the nodes which are already selected from the second parent. The resulted sequence of nodes contains the nodes the proto-child needs
- Step 4:** Place the nodes into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce one offspring.

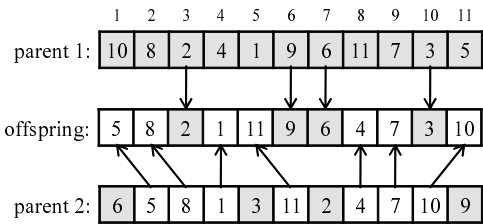


Fig. 2.11 Illustration of the PX operator

Weight Mapping Crossover (WMX):

In all the above crossover operators, the mechanism of the crossover is not the same as that of the conventional one-cut point crossover. Some offspring may be generated that did not succeed on the character of the parents, thereby retarding the process of evolution.

Lin and Gen proposed a *weight mapping crossover* (WMX) [24]; it can be viewed as an extension of one-cut point crossover for permutation representation. At one-cut point crossover, two chromosomes (parents) would choose a random-cut point and generate the offspring by using a segment of its own parent to the left of the cut point, then remap the right segment based on the weight of other parent of right segment. Fig. 2.12 shows the crossover process of WMX, and an example of the WMX is given in Fig. 2.13.

```

procedure : weight mapping crossover (WMX)
input : two parents  $v_1[\cdot], v_2[\cdot]$ , the length of chromosome  $n$ 
output : offspring  $v_1'[\cdot], v_2'[\cdot]$ 
begin
   $p \leftarrow \text{random}[1, n];$       //  $p$ : a random cut-point
   $l \leftarrow n - p;$            //  $l$ : the length of right segments of chromosomes
   $v_1' \leftarrow v_1[1 : p] // v_2[p+1 : n];$ 
   $v_2' \leftarrow v_2[1 : p] // v_1[p+1 : n];$  // exchange substrings between parents.
   $s_1[\cdot] \leftarrow \text{sorting}(v_1[p+1 : n]);$ 
   $s_2[\cdot] \leftarrow \text{sorting}(v_2[p+1 : n]);$  // sorting the weight of the right segments.
  for  $i = 1$  to  $l$ 
    for  $j = 1$  to  $l$ 
      if  $v_1'[p+i] = s_2[j]$  then  $v_1'[p+i] \leftarrow s_1[j];$ 
    for  $j = 1$  to  $l$ 
      if  $v_2'[p+i] = s_1[j]$  then  $v_2'[p+i] \leftarrow s_2[j];$ 
  output offspring  $v_1'[\cdot], v_2'[\cdot];$ 
end

```

Fig. 2.12 Pseudocode of weight mapping crossover

As showed in Fig. 2.12, first we choose a random cut point p ; and calculate l that is the length of right segments of chromosomes, where n is the number of nodes in the network. Then we get a mapping relationship by sorting the weight of the right segments $s_1[\cdot]$ and $s_2[\cdot]$. As a one-cut point crossover, it generates the offspring $v_1'[\cdot], v_2'[\cdot]$ by exchange substrings between parents $v_1[\cdot], v_2[\cdot]$; legalized offspring with mapping relationship and then two new chromosomes are produced eventually. In

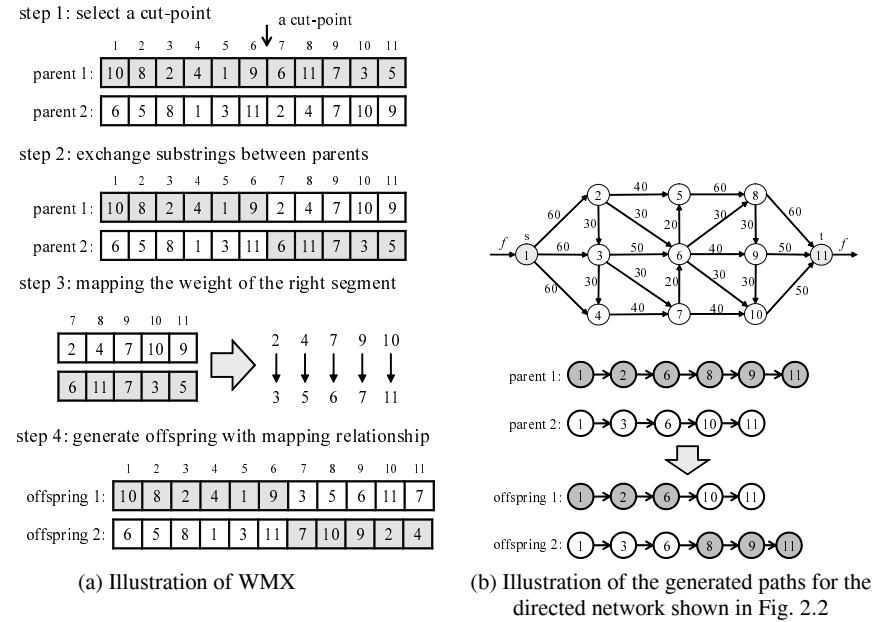


Fig. 2.13 Example of WMX procedure

some cases, WMX is the same as that of the conventional one-cut point crossover and can generate two new paths that exchanged sub-route from two parents.

For permutation representation, it is relatively easy to produce some mutation operators. Several mutation operators have been proposed for permutation representation, such as swap mutation, inversion mutation, insertion mutation, *etc.*

Inversion Mutation

Inversion mutation selects two positions within a chromosome at random and then inverts the substring between these two positions. It is illustrated in Fig. 2.14.

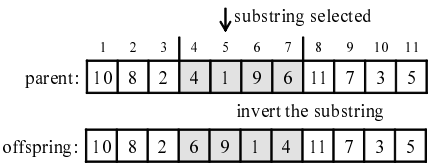


Fig. 2.14 Illustration of the inversion mutation

Insertion Mutation

Insertion mutation selects an element at random and inserts it in a random position as illustrated in Fig. 2.15.

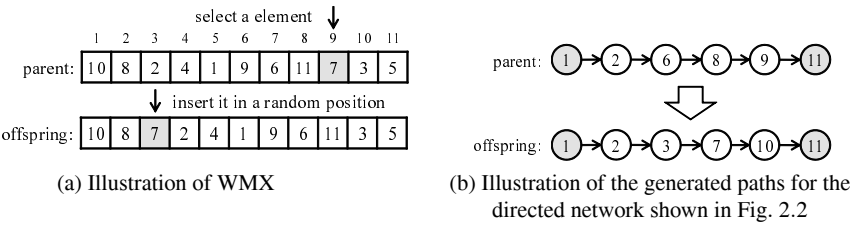
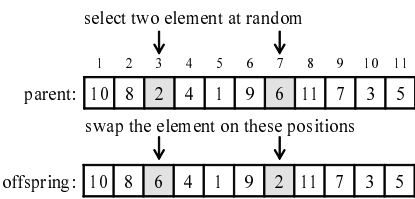


Fig. 2.15 Example of the insertion mutation

Swap Mutation

Swap mutation selects two elements at random and then swaps the elements on these positions as illustrated in Fig. 2.16.



2.2.2.5 Heuristic Mutation

Heuristic mutation was proposed by Gen and Cheng [7]. It is designed with a neighborhood technique in order to produce an improved offspring. A set of chromosomes transformable from a given chromosome by exchanging no more than λ elements are regarded as the neighborhood. The best one among the neighborhood is used as offspring produced by the mutation. The procedure is illustrated in Fig. 2.17. The mutation operator works as follows:

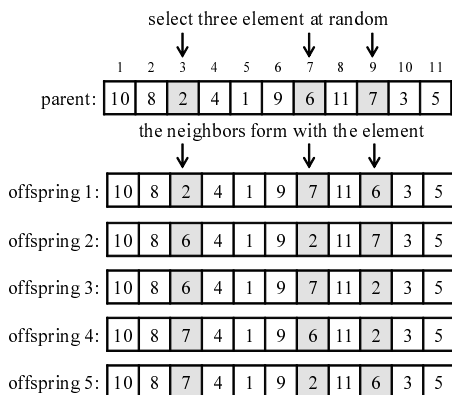
Procedure Heuristic Mutation**Input:** parent**Output:** offspring**Step 1:** Pick up λ genes at random.**Step 2:** Generate neighbors according to all possible permutation of the selected genes.**Step 3:** Evaluate all neighbors and select the best one as offspring.**Step 4:** Fulfill the child with the remaining nodes.

Fig. 2.17 Illustration of the heuristic mutation

Immigration Operator

The *immigration operator* is modified to (1) include immigration routine, in each generation, (2) generate and (3) evaluate $popSize \cdot p_I$ random chromosomes, and (4) replace the $popSize \cdot p_I$ worst chromosomes of the current population (p_I , called the immigration probability). The detailed introduction is showed on page 23.

Interactive Adaptive-weight Fitness Assignment

The *interactive adaptive-weight fitness assignment mechanism* assigns weights to each objective and combines the weighted objectives into a single objective function. It is an improved adaptive-weight GA [8] with the consideration of the disadvantages of weighted-sum approach and Pareto ranking-based approach. The detailed introduction is showed on page 39.

Selection Operator

We adopt the roulette wheel selection (RWS). It is to determine selection probability or survival probability for each chromosome proportional to the fitness value. A

model of the roulette wheel can be made displaying these probabilities. The detailed introduction is showed on page 13.

2.2.3 Computational Experiments and Discussions

Usually during the GA design phase, we are only concerned with the design of genetic representations, neglecting the design of more effective genetic operators with depend on the characteristics of the genetic representations. In the first experiment, the effectiveness of different genetic operators will be demonstrated. Then to validate the effectiveness of different genetic representations, priority-based GA with Ahn and Ramakrishna's algorithm [12] are compared. Finally, to validate the multiobjective GA approaches for solving bSPR problems, i-awGA with three different fitness assignment approaches (spEA, nsGA II and rwGA) are compared. For each algorithm, 50 runs with Java are performed on a Pentium 4 processor (3.40-GHz clock), 3.00GA RAM.

2.2.3.1 Performance Comparisons with Different Genetic Operators

In this experiment, the different genetic operators for priority-based genetic representation are combined; there are partial-mapped crossover (PMX), order crossover (OX), position-based crossover (PX), swap mutation, insertion mutation and immigration operator. We used six shortest path problems [12, 61], and Dijkstra's algorithm used to calculate the optimal solution to each problem. GA parameter settings were taken as follows: population size, $popSize = 10$; crossover probability, $p_C = 0.30$; mutation probability, $p_M = 0.30$; immigration rate, $p_I = 0.30$. In addition, there are two different terminating criteria employed. One of them is the number of maximum generations, $maxGen = 1000$. Another stopping criteria is the terminating condition $T = 200$. That is, if the best solution is not improved until successive 200 generations, the algorithm will be stopped. In order to evaluate the results of each test, the following four performance measures are adopted: average of the best solutions (ABS), standard deviation (SD), percent deviation from optimal solution (PD) and a statistical analysis by analysis of variance (ANOVA) [25].

There are seven kinds of combinations of genetic operators: PMX + Swap (Alg. 1), PMX + Insertion (Alg. 2), OX + Swap (Alg. 3), OX + Insertion (Alg. 4), PX + Swap (Alg. 5), PX + Insertion (Alg. 6) and WMX + Insertion + Immigration (Alg. 7). Figure 2.18 showed the ABS on the six test problems. The values of PD, SD and ANOVA analysis with 50 runs of 7 algorithms on 6 test problems are showed in Table 2.3.

As depicted in Fig. 2.18, all PD with Alg. 7 are better than each of the other combinations of genetic operators. Also all of the SD of 50 runs with Alg. 7 are smaller than each of the other combinations of genetic operators. In addition, the SD of 50 runs with Alg. 7 were 0 in test problem #1 – #3 and #5; that means we can obtain the

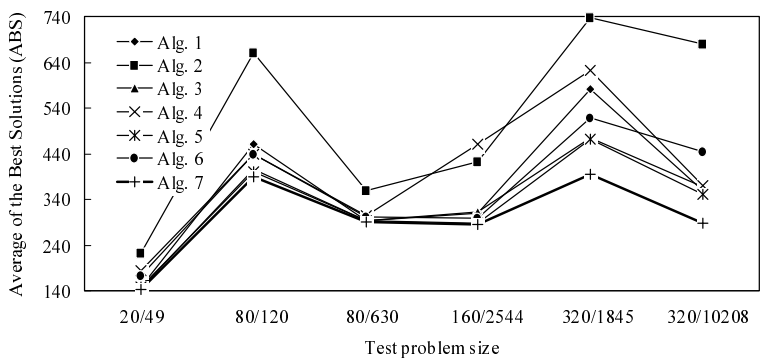


Fig. 2.18 Effectiveness comparing of ABS with seven kinds of combinations of genetic operators

Table 2.3 The Average of PD, SD for combinations of genetic operators and ANOVA analysis

| Test problems | Alg. 1 | Alg. 2 | Alg. 3 | Alg. 4 | Alg. 5 | Alg. 6 | Alg. 7 |
|-----------------------------|----------------|----------------|-------------|------------|--------------|--------------|--------|
| 20 / 49 | 4.10 | 55.87 | 7.27 | 28.69 | 8.92 | 21.63 | 0.00 |
| 80 / 120 | 18.12 | 69.98 | 8.96 | 12.91 | 8.50 | 12.72 | 0.00 |
| 80 / 632 | 1.07 | 23.32 | 0.76 | 4.53 | 2.53 | 3.79 | 0.00 |
| 160 / 2544 | 8.80 | 49.08 | 19.28 | 62.58 | 12.49 | 5.58 | 0.07 |
| 320 / 1845 | 47.20 | 86.82 | 24.95 | 57.70 | 24.52 | 31.24 | 0.00 |
| 320 /10208 | 24.26 | 136.34 | 43.37 | 28.12 | 37.82 | 53.99 | 0.04 |
| Mean | 17.26 | 70.24 | 17.43 | 32.42 | 15.80 | 21.49 | 0.02 |
| SD | 34.59 | 147.45 | 60.38 | 111.68 | 54.17 | 124.50 | 0.12 |
| Variance | 1196.14 | 21740.92 | 3645.65 | 12471.66 | 2934.87 | 15500.29 | 0.02 |
| Sum of squares | 358841.24 | 6522274.63 | 1093695.70 | 3741497.46 | 880462.44 | 4650087.65 | 4.55 |
| Factors | Sum of squares | Freedom degree | Mean square | F | F (α = 0.05) | t (α = 0.05) | |
| Between groups | 881925.72 | 6 | 146987.62 | 17.84 | 2.10 | 1.96 | |
| Within-groups | 17246863.66 | 2093 | 8240.26 | | | | |
| Total | 18128789.38 | 2099 | | | | | |
| LSD | 14.53 | | | | | | |
| Mean Difference with Alg. 7 | 17.24 | 70.22 | 17.41 | 32.40 | 15.78 | 21.47 | |

global optimum for each run. In the other two test problems, #4 and #6, the SD were close to 0. In other words, Alg. 7 has good stability. In Table 2.3, ANOVA analysis depended on the average of 50 PDs with 6 test problems and are shown to analyze the differences between the quality of solutions obtained by various combinations of genetic operators. At the significant level of $\alpha = 0.05$, $F=17.84$ is greater than the reference value of ($F=2.10$). The difference between Alg. 7 and each of the other combinations is greater than $LSD=14.53$ (Least Significant Difference). The Alg. 7 (WMX + Insertion + Immigration) is indeed statistically better than the others.

2.2.3.2 Comparisons with Different Encoding Methods

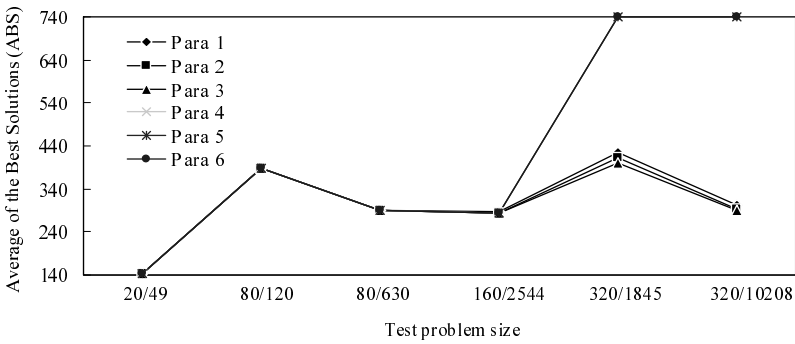
How to encode a solution of the problem into a chromosome is a key issue in GAs. The different chromosome representations will have a very big impact on GA designs. In the second experiment, the performance comparisons between priority-based GA approach (priGA) and ahnGA are showed. In priGA, WMX + Insertion + Immigration are used as genetic operators. The six shortest path problems, as with the first experiment, are combined. In order to evaluate the results of each test, four performance measures are adopted: average of the best solutions (ABS), standard deviation (SD), percent deviation from optimal solution (PD) and a statistical analysis by analysis of variance (ANOVA) [25].

In addition, as we all know, in general the result of GA decrease with increasing the problem size means that we must increase the GA parameter settings. Therefore if the parameter setting of a GA approach does not increase the problem size, then we can say this GA approach has a very good search capability of obtaining optimal results. The effectiveness comparing with six kinds of different GA parameter settings are shown as follows:

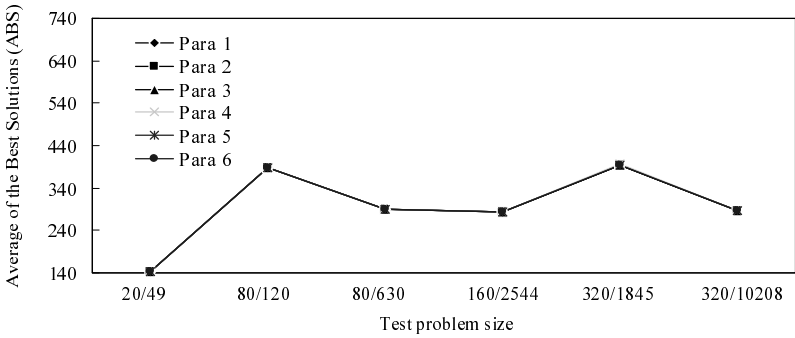
- Para 1: $maxGen=1000, popSize=10, p_C=0.3, p_M=0.3, p_I = 0.3, T=200$.
- Para 2: $maxGen=1000, popSize=10, p_C=0.5, p_M=0.5, p_I = 0.3, T=200$.
- Para 3: $maxGen=1000, popSize=10, p_C=0.7, p_M=0.7, p_I = 0.3, T=200$.
- Para 4: $maxGen=1000, popSize=20, p_C=0.3, p_M=0.3, p_I = 0.3, T=200$.
- Para 5: $maxGen=1000, popSize=20, p_C=0.5, p_M=0.5, p_I = 0.3, T=200$.
- Para 6: $maxGen=1000, popSize=20, p_C=0.7, p_M=0.7, p_I = 0.3, T=200$.

To see clearly the difference between priGA and ahnGA with the different GA parameter settings, Fig. 2.19 shows the ABS of six test problems. The values of PD, SD and ANOVA analysis are showed in Table 2.4. Because of an memory error, the last two large-scale test problems with large GA parameter setting were not solved by ahnGA. As depicted in Fig. 2.19a and Table 2.4, ahnGA can solve the first four test problems successfully, but for solving last two test problems, GA parameter setting affected the efficiency of ahnGA. In addition, the memory requirements are too high and even impossible to use with the higher GA parameter settings. Figure 2.19b and Table 2.4 show the efficiency of priGA with all the test problems successfully solved without any effect grow GA parameter settings by priGA.

In Table 2.4, ANOVA analysis depends on the average of 50 PD with 6 test problems to analyze the differences between the quality of solutions obtained by two kinds of chromosome representations with six kinds of different GA parameter settings. At the significant level of $\alpha = 0.05$, $F=16.27$ is greater than the reference value of ($F=1.79$). The difference between the lower GA parameter setting (Para 1, Para 2 or Para 3) and higher GA parameter setting (Para 4, Para 5 or Para 6) is greater than the $LSD=0.48$ by ahnGA. The GA parameter settings indeed statistically affect the efficiency of ahnGA. However, all of the difference between the GA parameter settings is smaller than $LSD=0.48$ by priGA, meaning that the GA parameter settings did not affect the efficiency of priGA. Futhermore, the difference between ahnGA and priGA is greater than $LSD=0.48$, combining the lower GA parameter setting



(a) Combine with ahnGA



(b) Combine with priGA

Fig. 2.19 Effectiveness comparing of ABS with six kinds of GA parameter settings

Table 2.4 The average of PD, SD for different chromosome representations and ANOVA analysis

| Test problems | ahnGA | | | | | | priGA | | | | | |
|----------------|----------------|----------------|-------------|--------|-------------------------------------|--------|--------|--------|--------|--------|--------|--------|
| | Para 1 | Para 2 | Para 3 | Para 4 | Para 5 | Para 6 | Para 1 | Para 2 | Para 3 | Para 4 | Para 5 | Para 6 |
| 20/ 49 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 80/ 120 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 80/ 632 | 0.08 | 0.01 | 0.01 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 160/ 2544 | 1.04 | 0.14 | 0.24 | 0.39 | 0.04 | 0.00 | 0.07 | 0.00 | 0.00 | 0.03 | 0.00 | 0.00 |
| 320/ 1845 | 7.44 | 4.87 | 3.13 | - | - | - | 0.00 | 0.00 | 0.00 | 0.54 | 0.00 | 0.00 |
| 320/10208 | 5.33 | 1.27 | 0.83 | - | - | - | 0.04 | 0.00 | 0.01 | 0.01 | 0.01 | 0.01 |
| Mean | 2.31 | 1.05 | 0.70 | 0.10 | 0.01 | 0.00 | 0.02 | 0.00 | 0.00 | 0.10 | 0.00 | 0.00 |
| SD | 7.76 | 4.77 | 3.47 | 0.53 | 0.09 | 0.00 | 0.12 | 0.00 | 0.03 | 1.57 | 0.03 | 0.02 |
| Variance | 60.25 | 22.72 | 12.01 | 0.28 | 0.01 | 0.00 | 0.02 | 0.00 | 0.00 | 2.45 | 0.00 | 0.00 |
| Sum of squares | 18075.87 | 6815.52 | 3603.02 | 55.78 | 1.47 | 0.00 | 4.55 | 0.00 | 0.24 | 735.90 | 0.24 | 0.12 |
| Factors | Sum of squares | Freedom degree | Mean square | F | F (α = 0.05) t (α = 0.05) LSD | | | | | | | |
| Between groups | 1594.87 | 11 | 144.99 | 16.27 | | | | | | | | |
| Within-groups | 29292.69 | 3288 | 8.91 | | | | | | | | | |
| Total | 30887.56 | 3299 | | | | | | | | | | |

“-” means out of memory error

(Para 1 Para 2 or Para 3). priGA was indeed statistically better than ahnGA with lower GA parameter settings.

2.2.3.3 Performance Validations of Multiobjective GAs

In the third experimental study, the performance comparisons of different multi-objective GA (moGA) approaches are shown for solving bSPR problems by different fitness assignment approaches; there are strength Pareto evolutionary algorithm (spEA), non-dominated sorting genetic algorithm II (nsGA II), random-weight genetic algorithm (rwGA) and interactive adaptive-weight genetic algorithm (i-awGA). In each GA approach, priority-based encoding was used, WMX, insertion, immigration and auto-tuning operators were used as genetic operators. All test network topologies were constructed by Beasley and Christofides [61, 62]. The two objectives, costs and delay, are represented as random variables depending on the distribution functions. The network characteristics for test networks are shown in Table 2.5.

Table 2.5 Network characteristics # of nodes n , # of arcs m , cost c and delay d for the test networks

| ID | n | m | Cost c | Delay d |
|----|-----|------|----------------------|----------------------|
| 1 | 100 | 955 | runif(m , 10, 50) | runif(m , 5, 100) |
| 2 | | 959 | rnorm(m , 50, 10) | rnorm(m , 50, 5) |
| 3 | | 990 | rlnorm(m , 1, 1) | rlnorm(m , 2, 1) |
| 4 | | 999 | 10*rexp(m , 0.5) | 10*rexp(m , 2) |
| 5 | 200 | 2040 | runif(m , 10, 50) | runif(m , 5, 100) |
| 6 | | 1971 | rnorm(m , 50, 10) | rnorm(m , 50, 5) |
| 7 | | 2080 | rlnorm(m , 1, 1) | rlnorm(m , 2, 1) |
| 8 | | 1960 | 10*rexp(m , 0.5) | 10*rexp(m , 2) |
| 9 | 500 | 4858 | runif(m , 10, 50) | runif(m , 5, 100) |
| 10 | | 4978 | rnorm(m , 50, 10) | rnorm(m , 50, 5) |
| 11 | | 4847 | rlnorm(m , 1, 1) | rlnorm(m , 2, 1) |
| 12 | | 4868 | 10*rexp(m , 0.5) | 10*rexp(m , 2) |

Uniform Distribution: runif(m , min, max)

Normal Distribution: rnorm(m , mean, sd)

Lognormal Distribution: rlnorm(m , meanlog, sdlog)

Exponential Distribution: rexp(m , rate)

Performance Measures

In these experiments, the following performance measures are considered to evaluate the efficiency of the different fitness assignment approaches. The detailed explanations have been introduced on page 41.

Number of obtained solutions $|S_j|$: Evaluate each solution set depend on the number of obtained solutions.

Ratio of nondominated solutions $R_{NDS}(S_j)$: This measure simply counts the number of solutions which are members of the Pareto-optimal set S^* .

Average distance $D1_R(S_j)$: Instead of finding whether a solution of S_j belongs to the set S^* or not, this measure finds an average distance of the solutions of S_j from S^* .

For finding the reference solution set S^* , we used the following GA parameter settings in all of four fitness assignment approaches (spEA, nsGA II, rwGA and our i-awGA) : population size, $popSize=50$; crossover probability, $p_C=0.90$; mutation probability, $p_M=0.90$; immigration rate, $p_I=0.60$; stopping criteria: evaluation of 100,000 generations.

Table 2.6 The ABS of 50 runs by different fitness assignment approaches

| ID | $ S_j $ | | | | $R_{NDS}(S_j)$ | | | | $D1_R(S_j)$ | | | |
|----|---------|------|------|--------|----------------|------|------|--------|-------------|------|------|--------|
| | spEA | nsGA | rwGA | i-awGA | spEA | nsGA | rwGA | i-awGA | spEA | nsGA | rwGA | i-awGA |
| 1 | 1.64 | 1.70 | 1.64 | 1.84 | 1.00 | 1.00 | 1.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 5.00 | 5.08 | 4.98 | 5.64 | 0.18 | 0.16 | 0.22 | 0.38 | 0.18 | 0.23 | 0.17 | 0.10 |
| 3 | 3.30 | 3.04 | 3.22 | 3.48 | 0.91 | 0.93 | 0.92 | 0.91 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 7.36 | 7.40 | 7.12 | 7.46 | 0.04 | 0.02 | 0.04 | 0.04 | 0.06 | 0.06 | 0.05 | 0.05 |
| 5 | 3.26 | 3.22 | 3.12 | 3.46 | 1.00 | 1.00 | 1.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 1.74 | 2.40 | 2.20 | 1.54 | 0.28 | 0.14 | 0.18 | 0.30 | 0.17 | 0.24 | 0.22 | 0.15 |
| 7 | 4.16 | 3.96 | 3.66 | 3.70 | 0.52 | 0.59 | 0.66 | 0.68 | 0.40 | 0.42 | 0.40 | 0.05 |
| 8 | 5.90 | 4.80 | 5.30 | 5.16 | 0.05 | 0.13 | 0.07 | 0.10 | 1.10 | 0.89 | 0.96 | 0.86 |
| 9 | 1.16 | 1.24 | 1.28 | 1.36 | 0.99 | 0.96 | 0.91 | 0.99 | 0.00 | 0.01 | 0.01 | 0.00 |
| 10 | 2.60 | 2.42 | 2.62 | 2.30 | 0.11 | 0.18 | 0.16 | 0.33 | 1.17 | 0.76 | 0.99 | 0.59 |
| 11 | 2.86 | 2.90 | 2.70 | 3.22 | 0.31 | 0.30 | 0.30 | 0.43 | 0.01 | 0.01 | 0.01 | 0.00 |
| 12 | 5.82 | 6.02 | 6.14 | 6.20 | 0.03 | 0.03 | 0.04 | 0.05 | 0.19 | 0.19 | 0.20 | 0.19 |

As depicted in Table 2.6, most results of ABS of 50 runs by i-awGA are better than each of the other fitness assignment approach. In addition, it is difficult to say the efficiency of the approach only depends on the performance measure $|S_j|$ or $R_{NDS}(S_j)$. It is necessary to demonstrate the efficiency both of the performance measure $|S_j|$ and $R_{NDS}(S_j)$.

In Tables 2.7 and 2.8, ANOVA analysis depended on the $|S_j|$ and $R_{NDS}(S_j)$ in 50 with test problem 11 runs to analyze the difference and was shown the quality of solutions obtained by four kinds of different fitness assignment approaches. At the significance level of $\alpha = 0.05$, the $F=3.56$ and 3.12 is greater than the reference value of $F=2.84$, respectively. The difference between i-awGA and each of

Table 2.7 ANOVA analysis with $|S_j|$ in test problem 11

| | spEA | nsGA II | rwGA | i-awGA |
|-----------------------------|----------------|----------------|-------------|--------|
| # of data | 50 | 50 | 50 | 50 |
| Mean | 2.86 | 2.90 | 2.70 | 3.22 |
| SD | 0.92 | 0.83 | 0.78 | 0.64 |
| Variance | 0.84 | 0.69 | 0.61 | 0.41 |
| Sum of squares | 42.02 | 34.50 | 30.50 | 20.58 |
| Factors | Sum of squares | Freedom degree | Mean square | F |
| Between groups | 7.12 | 3 | 2.37 | 3.65 |
| Within-groups | 127.60 | 196 | 0.65 | |
| Total | 134.72 | 199 | | |
| F ($\alpha = 0.05$) | 2.68 | | | |
| t ($\alpha = 0.05$) | 1.98 | | | |
| LSD | 0.31 | | | |
| Mean Difference with i-awGA | 0.36 | 0.32 | 0.52 | |

Table 2.8 ANOVA analysis with $R_{NDS}(S_j)$ in test problem 11.

| | spEA | nsGA II | rwGA | i-awGA |
|-----------------------------|----------------|----------------|-------------|--------|
| # of data | 50 | 50 | 50 | 50 |
| Mean | 0.31 | 0.30 | 0.30 | 0.43 |
| SD | 0.27 | 0.22 | 0.26 | 0.23 |
| Variance | 0.07 | 0.05 | 0.07 | 0.05 |
| Sum of squares | 3.62 | 2.43 | 3.33 | 2.62 |
| Factors | Sum of squares | Freedom degree | Mean square | F |
| Between groups | 0.57 | 3 | 0.19 | 3.12 |
| Within-groups | 12.01 | 196 | 0.06 | |
| Total | 12.58 | 199 | | |
| F ($\alpha = 0.05$) | 2.68 | | | |
| t ($\alpha = 0.05$) | 1.98 | | | |
| LSD | 0.10 | | | |
| Mean Difference with i-awGA | 0.11 | 0.13 | 0.13 | |

the other approaches (spEA, nsGA II or rwGA) is greater than the LSD=0.31 and 0.10, respectively. That means i-awGA is indeed statistically better than the other approaches.

2.3 Minimum Spanning Tree Models

Given a connected, undirected graph, a *spanning tree* of that graph is a subgraph which is a tree and connects all the nodes together. A single graph can have many different spanning trees. We can also assign a weight to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A *minimum spanning tree* (MST) is a spanning tree with weight less than or equal to the weight of every other spanning tree [4].

Example 2.2: A Simple Example of MST

A simple example of minimum spanning tree problem is shown in Fig. 2.20, and Table 2.9 gives the data set of the example.

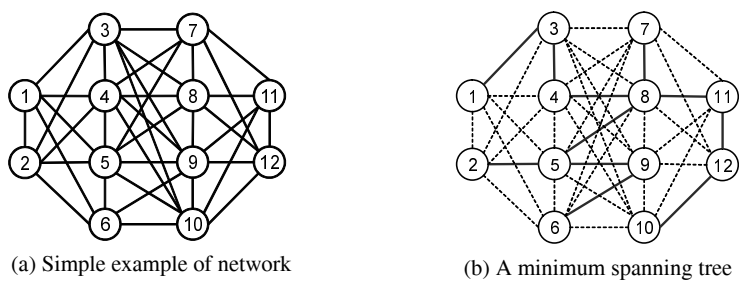


Fig. 2.20 A simple example of minimum spanning tree problem

Table 2.9 The data set of illustrative minimu spanning tree problem

| k | edge (i, j) | weight w_{ij} | k | edge (i, j) | weight w_{ij} | k | edge (i, j) | weight w_{ij} | k | edge (i, j) | weight w_{ij} |
|-----|---------------|-----------------|-----|---------------|-----------------|-----|---------------|-----------------|-----|---------------|-----------------|
| 1 | (1, 2) | 35 | 11 | (3, 7) | 51 | 21 | (5, 7) | 26 | 31 | (7, 12) | 41 |
| 2 | (1, 3) | 23 | 12 | (3, 8) | 23 | 22 | (5, 8) | 35 | 32 | (8, 9) | 62 |
| 3 | (1, 4) | 26 | 13 | (3, 9) | 64 | 23 | (5, 9) | 63 | 33 | (8, 11) | 26 |
| 4 | (1, 5) | 29 | 14 | (3, 10) | 28 | 24 | (5, 10) | 23 | 34 | (8, 12) | 30 |
| 5 | (1, 6) | 52 | 15 | (4, 5) | 54 | 25 | (6, 7) | 27 | 35 | (9, 10) | 47 |
| 6 | (2, 3) | 34 | 16 | (4, 7) | 24 | 26 | (6, 8) | 29 | 36 | (9, 11) | 68 |
| 7 | (2, 4) | 23 | 17 | (4, 8) | 47 | 27 | (6, 9) | 65 | 37 | (9, 12) | 33 |
| 8 | (2, 5) | 68 | 18 | (4, 9) | 53 | 28 | (6, 10) | 24 | 38 | (10, 11) | 42 |
| 9 | (2, 6) | 42 | 19 | (4, 10) | 24 | 29 | (7, 8) | 38 | 39 | (10, 12) | 26 |
| 10 | (3, 4) | 23 | 20 | (5, 6) | 56 | 30 | (7, 11) | 52 | 40 | (11, 12) | 51 |

Traditional Methods

Kruskal's Algorithm: Examines edges in nondecreasing order of their lengths and include them in MST if the added edge does not form a cycle with the edges already chosen. The proof of the algorithm uses the path optimality conditions. Attractive algorithm if the edges are already sorted in increasing order of their lengths [26]. Table 2.10 presents the trace table for the generating process of MST for solving a simple network example in Fig. 2.21. The procedure of Kruskal's algorithm is shown in Fig. 2.22.

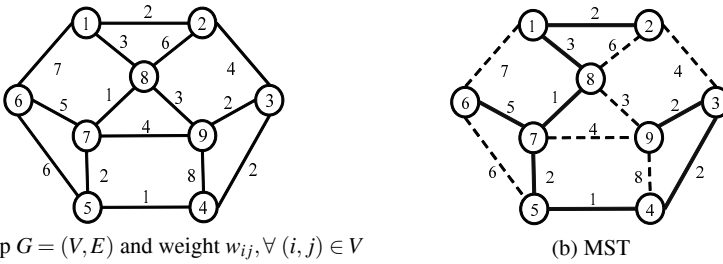


Fig. 2.21 A small-scale example of MST

```

procedure :Kruskal's Algorithm
input: graph  $G = (V, E)$ , weight  $w_{ij}, \forall (i, j) \in V$ 
output: spanning tree  $T$ 
begin
   $T \leftarrow \phi$ ;
   $A \leftarrow E$ ;    //  $A$ : eligible edges
  while  $|T| < |V| - 1$  do
    choose an edge  $(u, v) \leftarrow \text{argmin} \{w_{ij} | (i, j) \in A\}$ ;
     $A \leftarrow A \setminus \{(u, v)\}$ ;
    if  $u$  and  $v$  are not yet connected in  $T$  then
       $T \leftarrow T \cup \{(u, v)\}$ ;
  output spanning tree  $T$ ;
end

```

Fig. 2.22 Pseudocode of Kruskal's algorithm

Prim's Algorithm: Maintains a tree spanning a subset S of node and adds minimum cost edges in the cut $[S, S]$. The proof of the algorithm uses the cut optimality condi-

Table 2.10 Trace table for Kruskal's algorithm

| Step | Eligible edges \mathcal{A} | Weight $\{w_e\}$ | $ Z $ | Edge (u, v) | Spanning tree T | Fitness z |
|------|--|---|-------|---------------|--|-------------|
| 1 | $\{(1, 2), (1, 6), (1, 8), (2, 3), (2, 8), (3, 4), (3, 9), (4, 5), (4, 9), (5, 6), (5, 7), (6, 7), (7, 8), (7, 9), (8, 9)\}$ | $\{2, 7, 3, 4, 6, 2, 2, 1, 8, 6, 2, 5, 1, 4, 3\}$ | 0 | (4, 5) | $\{(4, 5)\}$ | 1 |
| 2 | $\{(1, 2), (1, 6), (1, 8), (2, 3), (2, 8), (3, 4), (3, 9), (4, 9), (5, 6), (5, 7), (6, 7), (7, 8), (7, 9), (8, 9)\}$ | $\{2, 7, 3, 4, 6, 2, 2, 8, 6, 2, 5, 1, 4, 3\}$ | 1 | (7, 8) | $\{(4, 5), (7, 8)\}$ | 2 |
| 3 | $\{(1, 2), (1, 6), (1, 8), (2, 3), (2, 8), (3, 4), (3, 9), (4, 9), (5, 6), (5, 7), (6, 7), (7, 9), (8, 9)\}$ | $\{2, 7, 3, 4, 6, 2, 2, 8, 6, 2, 5, 4, 3\}$ | 2 | (1, 2) | $\{(4, 5), (7, 8), (1, 2)\}$ | 4 |
| 4 | $\{(1, 6), (1, 8), (2, 3), (2, 8), (3, 4), (3, 9), (4, 9), (5, 6), (5, 7), (6, 7), (7, 9), (8, 9)\}$ | $\{7, 3, 4, 6, 2, 2, 8, 6, 2, 5, 4, 3\}$ | 3 | (3, 4) | $\{(4, 5), (7, 8), (1, 2), (3, 4)\}$ | 6 |
| 5 | $\{(1, 6), (1, 8), (2, 3), (2, 8), (3, 9), (4, 9), (5, 6), (5, 7), (6, 7), (7, 9), (8, 9)\}$ | $\{7, 3, 4, 6, 2, 8, 6, 2, 5, 4, 3\}$ | 4 | (3, 9) | $\{(4, 5), (7, 8), (1, 2), (3, 4), (3, 9)\}$ | 8 |
| 6 | $\{(1, 6), (1, 8), (2, 3), (2, 8), (4, 9), (5, 6), (5, 7), (6, 7), (7, 9), (8, 9)\}$ | $\{7, 3, 4, 6, 8, 6, 2, 5, 4, 3\}$ | 5 | (5, 7) | $\{(4, 5), (7, 8), (1, 2), (3, 4), (3, 9), (5, 7)\}$ | 10 |
| 7 | $\{(1, 6), (1, 8), (2, 3), (2, 8), (4, 9), (5, 6), (6, 7), (7, 9), (8, 9)\}$ | $\{7, 3, 4, 6, 8, 6, 5, 4, 3\}$ | 6 | (1, 8) | $\{(4, 5), (7, 8), (1, 2), (3, 4), (3, 9), (5, 7), (1, 8)\}$ | 13 |
| 8 | $\{(1, 6), (2, 3), (2, 8), (4, 9), (5, 6), (6, 7), (7, 9), (8, 9)\}$ | $\{7, 4, 6, 8, 6, 5, 4, 3\}$ | 7 | (6, 7) | $\{(4, 5), (7, 8), (1, 2), (3, 4), (3, 9), (5, 7), (1, 8), (6, 7)\}$ | 18 |

tions. Can be implemented using a variety of heaps structures; the stated time bound is for the Fibonacci heap data structure [26]. The procedure of Kruskal's algorithm is shown in Fig. 2.23. Table 2.11 presents the trace table for the generating process of MST for solving a simple network example in Fig. 2.21.

Table 2.11 Trace table for Prim's algorithm

| Step | Set of connected nodes C | Eligible edges \mathcal{A} | Weight $\{w_e\}$ | Node u | Node v | Spanning tree T | Fitness z |
|------|------------------------------|--|---------------------------------|----------|----------|--|-------------|
| 1 | $\{5\}$ | $\{(5, 4), (5, 6), (5, 7)\}$ | $\{1, 6, 2\}$ | 5 | 6 | $\{(5, 4)\}$ | 1 |
| 2 | $\{5, 4\}$ | $\{(5, 6), (5, 7), (4, 3), (4, 9)\}$ | $\{6, 2, 2, 8\}$ | 4 | 3 | $\{(5, 4), (4, 3)\}$ | 3 |
| 3 | $\{5, 4, 3\}$ | $\{(5, 6), (5, 7), (4, 9), (3, 9), (2, 3)\}$ | $\{6, 2, 8, 2, 4\}$ | 3 | 9 | $\{(5, 4), (4, 3), (3, 9)\}$ | 5 |
| 4 | $\{5, 4, 3, 9\}$ | $\{(5, 6), (5, 7), (4, 9), (2, 3), (7, 9), (8, 9)\}$ | $\{6, 2, 8, 4, 4, 3\}$ | 5 | 7 | $\{(5, 4), (4, 3), (3, 9), (5, 7)\}$ | 7 |
| 5 | $\{5, 4, 3, 9, 7\}$ | $\{(5, 6), (4, 9), (2, 3), (7, 9), (8, 9), (6, 7), (7, 8)\}$ | $\{6, 8, 4, 4, 3, 5, 1\}$ | 7 | 8 | $\{(5, 4), (4, 3), (3, 9), (5, 7), (7, 8)\}$ | 8 |
| 6 | $\{5, 4, 3, 9, 7, 8\}$ | $\{(5, 6), (4, 9), (2, 3), (7, 9), (8, 9), (6, 7), (1, 8), (2, 8)\}$ | $\{6, 8, 4, 4, 3, 5, 3, 6\}$ | 8 | 1 | $\{(5, 4), (4, 3), (3, 9), (5, 7), (7, 8), (1, 8)\}$ | 11 |
| 7 | $\{5, 4, 3, 9, 7, 8, 1\}$ | $\{(5, 6), (4, 9), (2, 3), (7, 9), (8, 9), (6, 7), (2, 8), (1, 2), (1, 6)\}$ | $\{6, 8, 4, 4, 3, 5, 6, 2, 7\}$ | 1 | 2 | $\{(5, 4), (4, 3), (3, 9), (5, 7), (7, 8), (1, 8), (1, 2)\}$ | 13 |
| 8 | $\{5, 4, 3, 9, 7, 8, 1, 2\}$ | $\{(5, 6), (4, 9), (2, 3), (7, 9), (8, 9), (6, 7), (2, 8), (1, 6)\}$ | $\{6, 8, 4, 4, 3, 5, 6, 7\}$ | 7 | 6 | $\{(5, 4), (4, 3), (3, 9), (5, 7), (7, 8), (1, 8), (1, 2), (6, 7)\}$ | 18 |

Sollin's Algorithm: Maintains a collection of node-disjoint trees: in each iteration, adds the minimum cost edge emanating from each such tree. The proof of the algorithm uses the cut optimality conditions [1].

Recently, the genetic algorithm (GA) and other evolutionary algorithms (EAs) have been successfully applied to solve constrained spanning tree problems of real-life instances and have also been used extensively in a wide variety of communica-

```

procedure : Prim's Algorithm
input: graph  $G = (V, E)$ , weight  $w_g, \forall (i, j) \in V$ 
output: spanning tree  $T$ 
begin
   $T \leftarrow \emptyset$ ;
  choose a random starting node  $s \in V$ ;
   $C \leftarrow C \cup \{s\}$ ; //  $C$ : set of connected nodes
   $A \leftarrow A \cup \{(s, v), \forall v \in V\}$ ; //  $A$ : eligible edges
  while  $C \neq V$  do
    choose an edge  $(u, v) \leftarrow \text{argmin} \{w_g \mid (i, j) \in A\}$ ;
     $A \leftarrow A \setminus \{(u, v)\}$ ;
    if  $v \notin C$  then
       $T \leftarrow T \cup \{(u, v)\}$ ;
       $C \leftarrow C \cup \{v\}$ ;
       $A \leftarrow A \cup \{(v, w) \mid (v, w) \wedge w \notin C\}$ ;
    output spanning tree  $T$ ;
  end

```

Fig. 2.23 Pseudocode of Prim's algorithm

tion network design problems [27, 28, 29]. For example, Abuali *et al.* [30] developed a new encoding scheme for *probabilistic MST* problem. Zhou and Gen [31] gave an effective GA approach to the *quadratic MST* problem. Zhou and Gen [32] and Fernandes and Gouveia [33] investigated the *leaf-constrained MST* problem with GA approaches. Raidl and Drexe [34] and Ahuja and Orlin [35] gave the EAs for the *capacitated MST* problem occurring in telecommunication applications. Zhou and Gen [36, 37], Raidl [38, 39], Knowles and Corne [40], Chou *et al.* [41], and Raidl and Julstrom [42] investigated the different encoding methods and gave the performance analyzes for the *degree-constraint MST* problem, respectively. Raidl and Julstrom [43] and Julstrom [44] developed EAs for the bounded-diameter MST problem. Zhou and Gen [45], Knowles and Corne [40], Lo and Chang [46], and Neumann [47] investigated the *multicriteria MST* problem with GA approaches and other EAs. EAs were also applied to solve other communication network design problem such as *two graph problem*, *one-max tree problem* and communication spanning tree problem [48]–[51].

2.3.1 Mathematical Formulation of the MST Models

The MST model attempts to find a minimum cost tree that connects all the nodes of the network. The links or edges have associated costs that could be based on their distance, capacity, and quality of line. Let $G = (V, E, W)$ be a connected weighted undirected graph with $n = |V|$ nodes and $m = |E|$ edges, and $w_{ij} \in W$ represent the weight or cost of each edge $(i, j) \in E$ where the weight is restricted to be a nonnegative real number.

Indices

$i, j = 1, 2, \dots, n$, index of node

Parameters

$n = |V|$: number of nodes
 $m = |E|$: number of edges
 $w_{ij} \in W$: weight of each edge $(i, j) \in E$
 $c_{ij} \in C$: cost of each edge $(i, j) \in E$
 u_i : weight capacity of each node i
 $d_{ij} \in D$: delay of each edge $(i, j) \in E$
 d_i : degree constraint on node i

Decision Variables

x_{ij} : the 0,1 decision variable; 1, if edge $(i, j) \in E$ is selected, and 0, otherwise
 y_{ij} : degree value on edge (i, j)

2.3.1.1 Minimum Spanning Tree Model

Let A_S denote the set of edges contained in the subgraph of G induced by the node set S (i.e., A_S is the set of edges of E with both endpoints in S). The *integer programming* formulation of the MST problem can be formulated as follows:

$$\min z = \sum_{(i,j) \in E} w_{ij} x_{ij} \quad (2.8)$$

$$\text{s. t. } \sum_{(i,j) \in E} x_{ij} = n - 1 \quad (2.9)$$

$$\sum_{(i,j) \in A_S} x_{ij} \leq |S| - 1 \quad \text{for any set } S \text{ of nodes} \quad (2.10)$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E \quad (2.11)$$

In this formulation, the 0-1 variable x_{ij} indicates whether we select edge (i, j) as part of the chosen spanning tree (note that the second set of constraints with $|S| = 2$ implies that each $x_{ij} \leq 1$). The constraints at Eq. 2.9 is a cardinality constraint implying that we choose exactly $n - 1$ edges, and the packing constraint at Eq. 2.10 implies that the set of chosen edges contain no cycles (if the chosen solution contained a cycle, and S were the set of nodes on a chosen cycle, the solution would violate this constraint).

2.3.1.2 Capacitated Minimum Spanning Tree Model

Capacitated minimum spanning tree (cMST) problem is an extended case of MST. Given a finite connected network, the problem is to find a MST, where the capacity of nodes are satisfied. Mathematically, the problem is reformulated as follows:

$$\min z = \sum_{(i,j) \in E} c_{ij}x_{ij} \quad (2.12)$$

$$\text{s. t. } \sum_{(i,j) \in E} x_{ij} = n - 1 \quad (2.13)$$

$$\sum_{(i,j) \in A_S} x_{ij} \leq |S| - 1 \quad \text{for any set } S \text{ of nodes} \quad (2.14)$$

$$\sum_{j=1}^n w_{ij}x_{ij} \leq u_i, \quad \forall i \quad (2.15)$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E \quad (2.16)$$

In this formulation, the constraint at Eq. 2.15 guarantees that the total link weight of each node i does not exceed the upper limit u_i .

2.3.1.3 Degree-constrained Minimum Spanning Tree Model

Degree-constrained minimum spanning tree (dMST) is a special case of MST. Given a finite connected network, the problem is to find an MST where the upper bounds of the number of edges to a node is satisfied. The dMST problem is NP-hard and traditional heuristics have had only limited success in solving small to midsize problems. Mathematically, the dMST problem is reformulated as follows:

$$\min z = \sum_{(i,j) \in E} w_{ij}x_{ij} \quad (2.17)$$

$$\text{s. t. } \sum_{(i,j) \in E} x_{ij} = n - 1 \quad (2.18)$$

$$\sum_{(i,j) \in A_S} x_{ij} \leq |S| - 1 \quad \text{for any set } S \text{ of nodes} \quad (2.19)$$

$$\sum_{j=1}^n y_{ij} \leq d_i, \quad \forall i \quad (2.20)$$

$$x_{ij}, y_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E \quad (2.21)$$

In this formulation, the constraint at Eq. 2.20 guarantees that the total edges to a node i does not exceed the degree d_i .

2.3.1.4 Multicriteria Minimum Spanning Tree Model

In the real world, there are usually such cases that one has to consider simultaneously multicriteria in determining an MST because there are multiple attributes defined on each edge, and this has become subject to considerable attention. Each edge has q associated positive real numbers, representing q attributes defined on it and denoted by w_{ij}^k . In practice w_{ij}^k may represent the weight, distance, cost and so on. The *multicriteria minimum spanning tree* (mMST) model can be formulated as follows:

$$\min z_k = \sum_{(i,j) \in E} w_{ij}^k x_{ij} \quad k = 1, 2, \dots, q \quad (2.22)$$

$$\text{s. t. } \sum_{(i,j) \in E} x_{ij} = n - 1 \quad (2.23)$$

$$\sum_{(i,j) \in A_S} x_{ij} \leq |S| - 1 \quad \text{for any set } S \text{ of nodes} \quad (2.24)$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in E \quad (2.25)$$

2.3.2 PrimPred-based GA for MST Models

Let $P(t)$ and $C(t)$ be parents and offspring in current generation t , respectively. The overall procedure of PrimPred-based GA for solving minimum spanning tree models is outlined as follows:

2.3.2.1 Genetic Representation

In GAs literature, whereas several kinds of encoding methods were used to obtain MSTs, most of them cannot effectually encode or decode between chromosomes and legality spanning trees. Special difficulty arises from (1) a cardinality constraint implying that we choose exactly $n - 1$ edges, and (2) implying any set of chosen

```

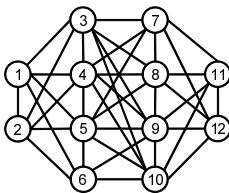
procedure: PrimPred-based GA for MST models
input: network data ( $V, E, W$ ),
        GA parameters ( $popSize, maxGen, p_M, p_C$ )
output: a minimum spanning tree
begin
     $t \leftarrow 0$ ;
    initialize  $P(t)$  by PrimPred-based encoding routine;
    evaluate  $P(t)$  by PrimPred-based decoding routine;
    while (not terminating condition) do
        create  $C(t)$  from  $P(t)$  by Prim-based crossover routine;
        create  $C(t)$  from  $P(t)$  by LowestCost mutation routine;
        evaluate  $C(t)$  by PrimPred-based decoding routine;
        select  $P(t+1)$  from  $P(t)$  and  $C(t)$  by roulette wheel selection routine;
         $t \leftarrow t+1$ ;
    end
    output a minimum spanning tree
end

```

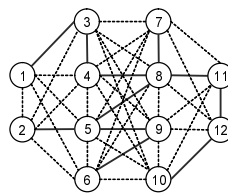
edges containing no cycles. We need to consider these critical issues carefully when designing an appropriate encoding method so as to build an effective GA. How to encode a spanning tree T in a graph G is critical for developing a GA to network design problem, it is not easy to find out a nature representation. We summarized the several kinds of classification of encoding methods as follows:

1. Characteristic vectors-based encoding
2. Edge-based encoding
3. Node-based encoding

Considering this classification, the following subsections review several recent encoding methods concerning how to represent a spanning tree. Figure 2.24 presents a simple network with 12 nodes and 40 arcs.



(a) Example with 12 nodes and 40 edges [42]



(b) A generated example of spanning tree

Fig. 2.24 A simple undirected graph

Characteristic Vectors-based Encoding

Davis *et al.* [53] and Piggott and Suraweera [54] have used a *binary-based encoding method* to represent spanning trees in GAs. A binary-based encoding requires space proportional to m and the time complexities of binary-based encoding is $O(m)$. The mapping from chromosomes to solutions (decoding) may be *1-to-1 mapping*. In a complete graph, $m = n(n-1)/2$ and the size of the search space is $2^{n(n-1)/2}$. However, only a tiny fraction of these chromosomes represent feasible solutions, since a complete graph G has only n^{n-2} distinct spanning trees. Repairing strategies have been more successful, but they require additional computation and weaken the encoding heritability. An example of the binary-based encoding is shown in Fig. 2.25

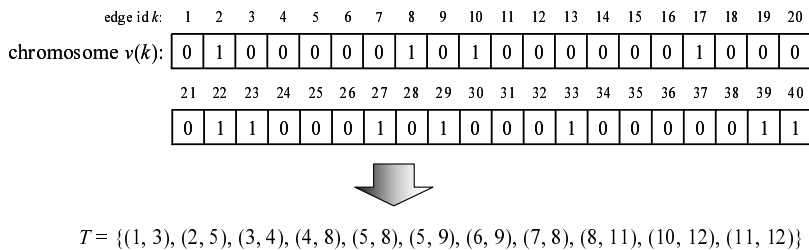


Fig. 2.25 An example of the binary-based encoding

Bean [55] described a *random keys-based encoding method* for encoding ordering and scheduling problems. Schindler *et al.* [49] and Rothlauf *et al.* [51] further investigated network random keys in an evolution strategy framework. In this encoding, a chromosome is a string of real-valued weights, one for each edge. To decode a spanning tree, the edges are sorted by their weights, and Kruskal's algorithm considers the edges are sorted order. As for binary-based encoding, random keys-based encoding requires space proportional to m and the time complexities is $O(m)$. Whereas all chromosomes represent feasible solutions, the uniqueness of the mapping from chromosomes to solutions may be n -to-1 mapping. In a complete graph with only n^{n-2} distinct spanning trees, consider random keys-based encoding, the size of the search space is $(n(n-1)/2)!$ For this reason, most different chromosomes represent the same spanning tree (*i.e.*, n is a very large number with n -to-1 mapping from chromosomes to solutions) and it weaken the encoding heritability. An example of the random keys-based encoding is shown in Fig. 2.26

Edge-based Encoding

Edge-based encoding is an intuitive representation of a tree. A general edge-based encoding requires space proportional to $n-1$ and the time complexities is $O(m)$. The

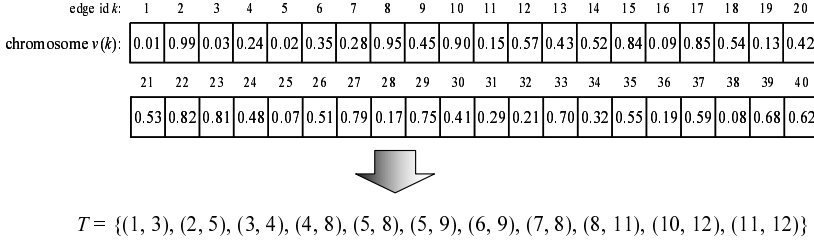


Fig. 2.26 An example of the random keys-based encoding

mapping from chromosomes to solutions (decoding) may be *1-to-1 mapping*. In a complete graph, $m = n(n-1)/2$ and the size of the search space is $2^{n(n-1)/2}$. Edge-based encoding and binary-based encoding have very similar performance in theory. However, there are $2^{n(n-1)/2}$ possible values for a tree, and only a tiny fraction of these chromosomes represent feasible solutions, and weaken the encoding heritability. Recently, researchers investigated and developed edge-based representations in GAs for designing several MST-related problems. Considering the disadvantage of edge-based encoding, the heuristic method is adopted in the chromosome generating procedure. These methods can ensure feasibility; all chromosomes can be represented by feasible solutions.

Knowles and Corne [40] proposed a method which improves edge-based encoding. The basis of this encoding is a spanning-tree construction algorithm which is *randomized primal method* (RPM), based on the Prim's algorithm. Raidl and Julstrom [43] gave the method depending on an underlying random spanning-tree algorithm. Three choices for this algorithm, based on Prim's algorithm, Kruskal's algorithm and on random walks, respectively, are examined analytically and empirically. Raidl [39], Chou *et al.* [41], Raidl and Julstrom [43] and Julstrom [44] described the other similar heuristic methods which ensure feasibility of the chromosomes. All of this heuristic method based encoding requires space proportional to $n-1$ and the time complexities is $O(n)$. The mapping from chromosomes to solutions (decoding) may be 1-to-1 mapping. In a complete graph, $m = n(n-1)/2$ and the size of the search space is n^{n-1} . These encoding methods offer efficiency of time complexity, feasibility and uniqueness. However, offspring of simple crossover and mutation should represent infeasible solutions. Several special genetic operator and repair strategies have been successful, but their limitations weaken the encoding heritability. An example of the edge-based encoding is shown in Fig. 2.27

Node-based Encoding

Prüfer number-based encoding: Cayley [56] proved the following formula: the number of spanning trees in a complete graph of n nodes is equal to n^{n-2} . Prüfer [57] presented the simplest proof of Cayley's formula by establishing a *1-to-1 correspon-*

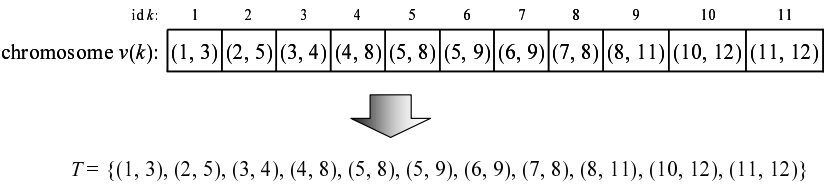


Fig. 2.27 An example of the edge-based encoding

dence between the set of spanning trees and a set of sequences of $n - 2$ integers, with each integer between 1 and n inclusive. The sequence of n integers for encoding a tree is known as the Prüfer number. An example of the Prüfer number-based encoding is shown in Fig. 2.28. The trace table of Prüfer number encoding process and decoding process are shown in Tables 2.12 and 2.13, respectively.

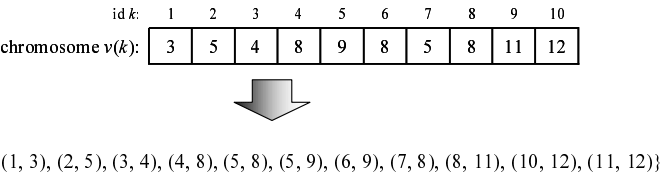


Fig. 2.28 An example of the Prüfer number-based encoding

Table 2.12 The trace table of Prüfer encoding process considering the example graph in Fig. 2.24

| A tree T | i | j | Prüfer number P |
|---|-----|-----|----------------------------------|
| $\{(1, 3), (3, 4), (4, 8), (7, 8), (8, 11), (11, 12), (12, 10), (8, 5), (5, 2), (5, 9), (9, 6)\}$ | 1 | 3 | [3] |
| $\{(3, 4), (4, 8), (7, 8), (8, 11), (11, 12), (12, 10), (8, 5), (5, 2), (5, 9), (9, 6)\}$ | 2 | 5 | [3, 5] |
| $\{(3, 4), (4, 8), (7, 8), (8, 11), (11, 12), (12, 10), (8, 5), (5, 9), (9, 6)\}$ | 3 | 4 | [3, 5, 4] |
| $\{(4, 8), (7, 8), (8, 11), (11, 12), (12, 10), (8, 5), (5, 9), (9, 6)\}$ | 4 | 8 | [3, 5, 4, 8] |
| $\{(7, 8), (8, 11), (11, 12), (12, 10), (8, 5), (5, 9), (9, 6)\}$ | 6 | 9 | [3, 5, 4, 8, 9] |
| $\{(7, 8), (8, 11), (11, 12), (12, 10), (8, 5), (5, 9)\}$ | 7 | 8 | [3, 5, 4, 8, 9, 8] |
| $\{(8, 11), (11, 12), (12, 10), (8, 5), (5, 9)\}$ | 9 | 5 | [3, 5, 4, 8, 9, 8, 5] |
| $\{(8, 11), (11, 12), (12, 10), (8, 5)\}$ | 5 | 8 | [3, 5, 4, 8, 9, 8, 5, 8] |
| $\{(8, 11), (11, 12), (12, 10)\}$ | 8 | 11 | [3, 5, 4, 8, 9, 8, 5, 8, 11] |
| $\{(11, 12), (12, 10)\}$ | 10 | 12 | [3, 5, 4, 8, 9, 8, 5, 8, 11, 12] |
| $\{(11, 12)\}$ | | | |

i : lowest labeled leaf node; j : predecessor node of leaf node i

Table 2.13 Trace table of Prüfer decoding process considering the example graph in Fig. 2.24

| Prüfer number P | Set of residual nodes P' | Spanning tree T |
|----------------------------------|----------------------------|--|
| [3, 5, 4, 8, 9, 8, 5, 8, 11, 12] | [1, 2, 6, 7, 10] | {(1,3)} |
| [5, 4, 8, 9, 8, 5, 8, 11, 12] | [2, 3, 6, 7, 10] | {(1,3), (2, 5)} |
| [4, 8, 9, 8, 5, 8, 11, 12] | [3, 6, 7, 10] | {(1,3), (2, 5), (3, 4)} |
| [8, 9, 8, 5, 8, 11, 12] | [4, 6, 7, 10] | {(1,3), (2, 5), (3, 4), (4, 8)} |
| [9, 8, 5, 8, 11, 12] | [6, 7, 10] | {(1,3), (2, 5), (3, 4), (4, 8), (6, 9)} |
| [8, 5, 8, 11, 12] | [7, 9, 10] | {(1,3), (2, 5), (3, 4), (4, 8), (6, 9), (7, 8)} |
| [5, 8, 11, 12] | [9, 10] | {(1,3), (2, 5), (3, 4), (4, 8), (6, 9), (7, 8), (5, 9)} |
| [8, 11, 12] | [5, 10] | {(1,3), (2, 5), (3, 4), (4, 8), (6, 9), (7, 8), (5, 9), (5, 8)} |
| [11, 12] | [8, 10] | {(1,3), (2, 5), (3, 4), (4, 8), (6, 9), (7, 8), (5, 9), (5, 8), (8, 11)} |
| [12] | [10, 11] | {(1,3), (2, 5), (3, 4), (4, 8), (6, 9), (7, 8), (5, 9), (5, 8), (8, 11), (10, 12)} |
| ϕ | [11, 12] | {(1,3), (2, 5), (3, 4), (4, 8), (6, 9), (7, 8), (5, 9), (5, 8), (8, 11), (10, 12), (11, 12)} |

Many researchers have encoded spanning trees as Prüfer numbers in GAs for a variety of problems. These include the degree-constrained MST problems, multicriteria MST problems, and leaf-constrained MST problems, *etc.* However, researchers have pointed out that Prüfer number is a poor representation of spanning trees for evolutionary search, and their major disadvantages as follows:

1. It needs a complex encoding process and decoding process between chromosomes and solutions (computational cost);
2. It is difficult to consist mostly of substructures of their parents' phenotypes (poor heritability);
3. It contains no useful information such as degree, connection, *etc.*, about a tree;
4. It also represents infeasible solutions if the graph G is incomplete graph.

Predecessor-based Encoding: A more compact representation of spanning trees is the predecessor or determinant encoding, in which an arbitrary node in G is designated the root, and a chromosome lists each other node's predecessor in the path from the node to the root in the represented spanning tree: if $\text{pred}(i)$ is j , then node j is adjacent to node i and nearer the root. Thus, a chromosome is string of length $n - 1$ over $1, 2, \dots, n$, and when such a chromosome decodes a spanning tree, its edges can be made explicit in time that is $O(n \log n)$.

Applied to such chromosomes, positional crossover and mutation operators will generate infeasible solutions, requiring again penalization or repairing. As shown in [42], Abuali *et al.* described a repairing mechanism that applies to spanning trees on sparse, as well as complete graphs. However, these strategies require additional computation and weaken the encoding heritability, and these repairing process have to spend much computational cost.

PrimPred-based Encoding

Take a predecessor-based encoding string $v[]$, with length of $n - 1$, where n represents number of nodes. Assume $v(i)$ represents the allele of the i -th fixed position in chromosome $v[]$, where i starts from 2 to n . As discussed in Chou *et al.*'s research paper [41], predecessor-based encoding generates some chromosomes that are illegal (*i.e.*, not a spanning tree). Combining the simple random initialization, most of the chromosomes will be illegal for three reasons: missing node i , self-loop, or cycles. They gave the repairing function for illegal chromosomes. However, the complex repairing process will be used at each generation (computational cost), and after repairing, the offspring of the crossover and mutation are difficult to represent solutions that combine substructures of their parental solutions (worst heritability and locality).

```

procedure : PrimPred-based encoding
input: number of nodes  $n$ ,
        node set  $S_i$  with all nodes adjacent to node  $i$ 
output: chromosome  $v$ 
begin
    node  $i \leftarrow 0$ ;
    assigned node set  $C \leftarrow \{i\}$ ;
     $t \leftarrow 1$ ;
    while ( $t \leq n$ )
        eligible edge set  $E \leftarrow \{(i, j) \mid j \in S_i\}$ 
        choose edge  $(i, j) \in E$  at random;
         $v(j) \leftarrow i$ ;
         $C \leftarrow C \cup \{j\}$ ;
         $i \leftarrow j$ ;
         $E \leftarrow E \cup \{(i, j) \mid j \in S_i\}$ ;
         $E \leftarrow E \setminus \{(i, j) \mid i \in C \ \& \ j \in C\}$ ;
         $t \leftarrow t + 1$ ;
    end
    output chromosome  $v$ ;
end

```

Fig. 2.29 Pseudocode of PrimPred-based encoding

```

procedure : PrimPred-based decoding
input: chromosome  $v$ , length of chromosome  $l$ 
output: spanning tree  $T$ 
begin
    spanning tree  $T \leftarrow \emptyset$ ;
    for  $i = 1$  to  $l$ 
         $v(j) \leftarrow i$ ;
         $T \leftarrow T \cup \{i, v(i)\}$ ;
    end
    output spanning tree  $T$ ;
end

```

Fig. 2.30 Pseudocode of predecessor-based decoding

When a GA searches a space of spanning trees, its initial population consists of chromosomes that represent random trees. It is not as simple as it might seem to choose spanning trees of a graph so that all are equal by likely. Prim's algorithm greedily builds a minimum spanning tree from a start node by repeatedly appending the lowest cost edge that joins a new node to the growing tree. Applying Prim's algorithm to the GAs chooses each new edge at random rather than according to its cost [42]. The encoding procedure and decoding procedure are shown in Figures

2.29 and 2.30, respectively. An example of generated chromosome and its decoded spanning tree is shown in Fig. 2.31 for the undirected network shown in Fig. 2.24.

In general, there are two ways to generate the initial population, heuristic initialization and random initialization. However, the mean fitness of the heuristic initialization is already high so that it may help the GAs to find solutions faster. Unfortunately, in most large scale problems, for example network communication designing, it may just explore a small part of the solution space and it is difficult to find global optimal solutions because of the lack of diversity in the population. Therefore, random initialization is effected in this research so that the initial population is generated with the PrimPred-based encoding method.

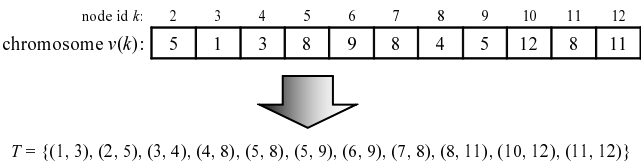


Fig. 2.31 An example of the PrimPred-based encoding

Depending on the properties of encodings (introduced on page 9), we summarize the performance of proposed PrimPred-based encoding method and other introduced encoding methods in Table 2.14.

Table 2.14 Summary of the performance of encoding methods

| Representation | | Space | Time | Feasibility | Uniqueness | Locality | Heritability |
|------------------------------|-------------------------------|-------|---------------|-------------|-------------------|----------|--------------|
| Characteristic Vectors-based | Binary-based encoding | m | $O(m)$ | Worst | 1-to-1 mapping | Worst | Worst |
| | Random keys-based encoding | m | $O(m)$ | Good | n -to-1 mapping | Worst | Worst |
| Edge-based | General edge-based encoding | n | $O(m)$ | Worst | 1-to-1 mapping | Worst | Worst |
| | Heuristic edge-based encoding | n | $O(n)$ | Good | 1-to-1 mapping | Poor | Poor |
| Node-based | Prüfer number-based encoding | n | $O(n \log n)$ | Good | 1-to-1 mapping | Worst | Worst |
| | Predecessor-based Encoding | n | $O(n \log n)$ | Poor | 1-to-1 mapping | Worst | Worst |
| | PrimPred-based Encoding | n | $O(n \log n)$ | Good | 1-to-1 mapping | Poor | Poor |

2.3.2.2 Genetic Operators

Genetic operators mimic the process of heredity of genes to create new offspring at each generation. Using the different genetic operators has very large influence on GA performance [8]. Therefore it is important to examine different genetic operators.

Prim-based Crossover

Crossover is the main genetic operator. It operates on two parents (chromosomes) at a time and generates offspring by combining both chromosomes' features. In network design problems, crossover plays the role of exchanging each partial route of two chosen parents in such a manner that the offspring produced by the crossover still represents a legal network.

To provide good heritability, a crossover operator must build an offspring spanning tree that consists mostly or entirely of edges found in the parents. In this subsection, we also apply Prim's algorithm described in the previous subsection to the graph $G' = (V, T_1 \cup T_2)$, where T_1 and T_2 are the edge sets of the parental trees, respectively. Figure 2.32 gave the pseudocode of crossover process. An example of Prim-based crossover is shown in Fig. 2.34

```

procedure : Prim-based crossover
input: parent  $v_1[]$ ,  $v_2[]$ 
output: offspring  $v'[]$ 
begin
    spanningtree  $T_1 \leftarrow \text{decode}(v_1)$ ;
    // decode the chromosomes to spanning tree
    spanningtree  $T_2 \leftarrow \text{decode}(v_2)$ ;
    edge set  $E \leftarrow T_1 \cup T_2$ ;
    subgraph  $G \leftarrow (V, E)$ ;
    offspring  $v' \leftarrow \text{PrimPredEncode}(G)$ ;
    // routine 1: PrimPred-based encoding
    output offspring  $v'[]$ ;
end

```

Fig. 2.32 Pseudocode of Prim-based crossover

```

procedure : LowestCost mutation
input: parent  $v[]$ , nodeset  $V = \{1, 2, \dots, n\}$ , cost  $w_{ij}$  of each edge  $(i, j) \in E$ 
output: offspring  $v'[]$ 
begin
    spanning tree  $T \leftarrow \text{decode}(v)$ ;
     $T \leftarrow T \setminus \{(u, v) \mid (u, v) \in T\}$  at random;
    allocate node  $A(k) \leftarrow 0, \forall k \in V$ ;
    while ( $T \neq \emptyset$ )
        choose edge  $(i, j) \in T$ ;
         $T \leftarrow T \setminus \{(i, j)\}$ ;
        if  $A(i) = \emptyset \ \& \ A(j) = \emptyset$  then
             $l \leftarrow \min\{i, j\}$ ;
             $A(i) \leftarrow l$ ;
             $A(j) \leftarrow l$ ;
        else if  $A(i) = \emptyset \ \& \ A(j) \neq \emptyset$  then
             $A(i) \leftarrow A(j)$ ;
        else if  $A(i) \neq \emptyset \ \& \ A(j) = \emptyset$  then
             $A(j) \leftarrow A(i)$ ;
        else if  $A(i) \neq \emptyset \ \& \ A(j) \neq \emptyset$  then
            if  $A(i) < A(j)$  then
                 $A(k) \leftarrow A(i), \text{ for all } k \text{ with } A(k) = A(j)$ ;
            if  $A(i) > A(j)$  then
                 $A(k) \leftarrow A(j), \text{ for all } k \text{ with } A(k) = A(i)$ ;
        end
    end
    nodeset  $C_1 \leftarrow \{k \mid A(k) = A(u)\}$ ;
    nodeset  $C_2 \leftarrow \{k \mid A(k) = A(v)\}$ ;
    edge  $(x, y) \leftarrow \arg \min\{w_{ij} \mid i \in C_1, j \in C_2\}$ ;
     $T \leftarrow T \cup \{(x, y)\}$ ;
    offspring  $v' \leftarrow \text{encode}(T)$ ;
    output offspring  $v'[]$ ;
end

```

Fig. 2.33 Pseudocode of LowestCost mutation

LowestCost Mutation

Mutation is a background operator which produces spontaneous random changes in various chromosomes. A simple way to achieve mutation would be to alter one or more genes. In GAs, mutation serves the crucial role of either replacing the genes lost from the population during the selection process so that they can be tried in a new context or providing the genes that were not present in the initial population. In this subsection, it is relatively easy to produce some mutation operators for permutation representation. Several mutation operators have been proposed for permutation representation, such as *swap mutation*, *inversion mutation*, and *insertion*

mutation, and so on [7]. Swap mutation selects two positions at random. For example, to represent a feasible solutions after mutation, and to improve the heritability of offspring, a new mutation operator is proposed , first removing a randomly chosen edge, determining the separated components, then reconnecting them by the lowest cost edge. An example of LowestCost mutation is shown in Fig. 2.35. Figure 2.33 is the pseudocode of mutation process.

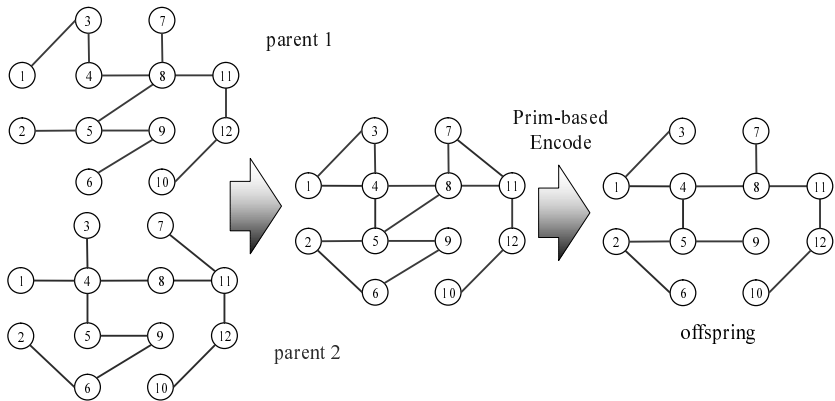


Fig. 2.34 An example of Prim-based crossover

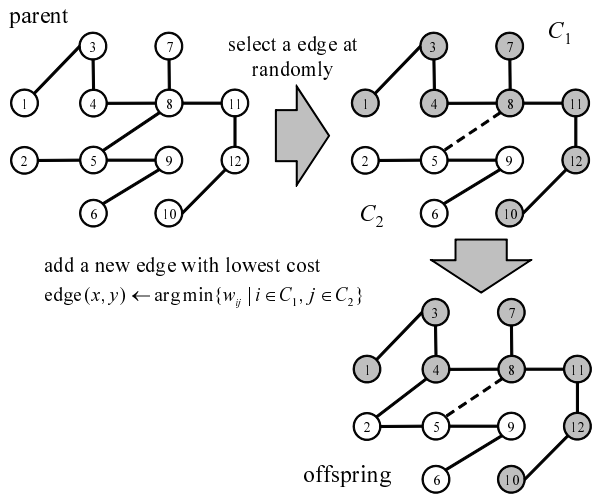


Fig. 2.35 An example of lowestcost mutation

2.3.3 Computational Experiments and Discussions

In this section, PrimPred-based GA is compared with Zhou and Gen [36] and Raidl and Julstrom [42] for solving several *large scale minimum spanning tree* (MST) problems. All the simulations were performed with Java on a Pentium 4 processor (2.6-GHz clock), 3.00GA RAM.

For examining the effectiveness of different encoding methods, PrimPred-based GA, Zhou and Gen's Prüfer number-based encoding method and Raidl and Julstrom's edge-based encoding method are applied to six test problems [61]. Prüfer number-based encoding with one-cut point crossover and swap mutation is combined, and edge-based encoding using two kinds of mutation operators is combined which is included in [42], and for initializing the chromosomes based on the edge set, Raidl and Julstrom's PrimRST (Prim random spanning tree) is combined. Each algorithm was run 20 times using different initial seeds for each test problems. And Prim's algorithm has been used to obtain optimum solutions for the problems. The GA parameter is setting as follows:

Population size: $popSize = 10$;
 Crossover probability: $p_C = 0.30, 0.50$ or 0.70 ;
 Mutation probability: $p_M = 0.30, 0.50$ or 0.70 ;
 Maximum generation: $maxGen = 1000$;

The experimental study was realized to investigate the effectiveness of the different encoding method; the interaction of the encoding with the crossover operators and mutation operators, and the parameter settings affect its performance. Table 2.15 gives computational results for four different encoding methods on six test problems by three kinds of parameter settings. In the columns of the best cost of four encoding methods, it is possible to see that whereas the Prüfer number-based approach is faster than the others, it is difficult to build from the substructures of their parents' phenotypes (poor heritability), and the result is very far from the best one. Two kinds of mutation are used in edge-based encoding, the second one (depends on the cost) giving better performance than the first. For considering the computational cost (CPU time), because of the LowestCost mutation in the proposed approach, spending a greater CPU time to find the edge with the lowest cost they always longer than other algorithms. However, PrimPred-based GA developed in this study gives a better cost than other algorithms.

2.4 Maximum Flow Model

The maximum flow problem (MXF) is to find a feasible flow through a single-source, single-sink flow network that is maximum. The MXF model and the shortest path model are complementary. The two problems are different because they capture different aspects: in the shortest path problem model all arcs are costs but not arc capacities; in the maximum flow problem model all arcs are capacities but not costs. Taken together, the shortest path problem and the maximum flow problem combine

Table 2.15 Performance comparisons with different GA approaches: Prüfer number-based encoding (Zhou and Gen [36]), edge-based 1 and edge-based 2 with different mutation operators (Raidl and Julstrom [42]) and PrimPred-based GA

| Test Problem <i>n</i> / <i>m</i> | Optimal Solutions | <i>p_C</i> | <i>p_M</i> | Prüfer Num-based | | Edge-based 1 | | Edge-based 2 | | PrimPred-based | |
|-------------------------------------|-------------------|----------------------|----------------------|------------------|--------|--------------|-----------|--------------|-----------|----------------|-----------|
| | | | | avg. | time | avg. | time | avg. | time | avg. | time |
| 40/780 | 470 | 0.30 | 0.30 | 1622.20 | 72.20 | 1491.80 | 1075.20 | 495.60 | 1081.40 | 470.00 | 1100.20 |
| | | 0.50 | 0.50 | 1624.40 | 87.60 | 1355.80 | 2184.40 | 505.80 | 2175.00 | 470.00 | 2256.40 |
| | | 0.70 | 0.70 | 1652.60 | 134.80 | 1255.20 | 3287.40 | 497.60 | 3281.40 | 470.00 | 3316.00 |
| 40/780 | 450 | 0.30 | 0.30 | 1536.60 | 74.80 | 1458.20 | 1118.60 | 471.60 | 1093.80 | 450.00 | 1106.20 |
| | | 0.50 | 0.50 | 1549.20 | 78.20 | 1311.40 | 2190.80 | 480.20 | 2175.00 | 450.00 | 2200.20 |
| | | 0.70 | 0.70 | 1564.40 | 122.00 | 1184.40 | 3287.60 | 466.40 | 3262.40 | 450.00 | 3275.00 |
| 80/3160 | 820 | 0.30 | 0.30 | 3880.40 | 150.00 | 3760.20 | 5037.80 | 923.20 | 5059.60 | 820.00 | 5072.00 |
| | | 0.50 | 0.50 | 3830.00 | 184.40 | 3692.00 | 10381.20 | 871.00 | 10494.20 | 820.00 | 10440.60 |
| | | 0.70 | 0.70 | 3858.20 | 231.20 | 3483.80 | 16034.80 | 899.20 | 15871.80 | 820.00 | 15984.60 |
| 80/3160 | 802 | 0.30 | 0.30 | 3900.60 | 131.40 | 3853.00 | 5125.00 | 894.60 | 4934.20 | 802.00 | 5071.80 |
| | | 0.50 | 0.50 | 3849.60 | 206.20 | 3515.20 | 10325.20 | 863.00 | 10268.80 | 802.00 | 10365.60 |
| | | 0.70 | 0.70 | 3818.40 | 222.00 | 3287.20 | 16003.00 | 868.00 | 15965.40 | 802.00 | 15947.20 |
| 120/7140 | 712 | 0.30 | 0.30 | 5819.40 | 187.40 | 5536.60 | 15372.00 | 871.80 | 15306.40 | 712.00 | 15790.40 |
| | | 0.50 | 0.50 | 5717.20 | 293.80 | 5141.00 | 31324.80 | 805.40 | 30781.40 | 712.00 | 31503.20 |
| | | 0.70 | 0.70 | 5801.40 | 316.00 | 5035.20 | 47519.00 | 804.20 | 47047.20 | 712.00 | 47865.80 |
| 160/12720 | 793 | 0.30 | 0.30 | 7434.80 | 284.40 | 7050.40 | 41993.60 | 1353.60 | 42418.60 | 809.60 | 42628.20 |
| | | 0.50 | 0.50 | 7361.00 | 421.80 | 7111.60 | 87118.80 | 1061.60 | 86987.40 | 793.00 | 86828.40 |
| | | 0.70 | 0.70 | 7517.00 | 403.20 | 6735.00 | 163025.00 | 955.40 | 161862.40 | 793.00 | 154731.20 |

avg.: average solution of 20 runs; time: average computation time in millisecond (ms)

all the basic ingredients of network models [1]. As such, they have become the nuclei of network optimization models.

Example 2.3: A Simple Example of MXF

A simple example of maximum flow model is shown in Fig. 2.36, and Table 2.16 gives the data set of the example.

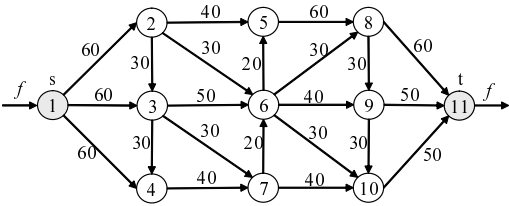


Fig. 2.36 A simple example of maximum flow model

Table 2.16 The data set of illustrative maximum flow model

| i | j | u_{ij} | i | j | u_{ij} |
|-----|-----|----------|-----|-----|----------|
| 1 | 2 | 60 | 6 | 5 | 20 |
| 1 | 3 | 60 | 6 | 8 | 30 |
| 1 | 4 | 60 | 6 | 9 | 40 |
| 2 | 3 | 30 | 6 | 10 | 30 |
| 2 | 5 | 40 | 7 | 6 | 20 |
| 2 | 6 | 30 | 7 | 10 | 40 |
| 3 | 4 | 30 | 8 | 9 | 30 |
| 3 | 6 | 50 | 8 | 11 | 60 |
| 3 | 7 | 30 | 9 | 10 | 30 |
| 4 | 7 | 40 | 9 | 11 | 50 |
| 5 | 8 | 60 | 10 | 11 | 50 |

Traditional Methods

If we interpret u_{ij} as the maximum flow rate of arc (i, j) , MXF identifies the maximum steady-state flow that the network can send from node s to node t per unit time. There are several ways of solving this problem:

Ford-Fulkerson algorithm: The idea behind the algorithm is very simple. As long as there is a path from the source (start node s) to the sink (end node t), with available capacity on all edges in the path, we send flow along one of these paths. Then we find another path, and so on. A path with available capacity is called an *augmenting path*.

Edmonds-Karp algorithm: The algorithm is identical to the Ford-Fulkerson algorithm, except that the search order when finding the augmenting path is defined. The path found must be the shortest path which has available capacity. This can be found by a *breadth-first search*, as we let edges have unit length. The running time is $O(nm^2)$.

Relabel-to-front algorithm: It is in the class of *push-relabel algorithms* for maximum flow which run in $O(n^2m)$. For dense graphs it is more efficient than the Edmonds-Karp algorithm.

The MXF appears to be more challenging in applying GAs and other evolutionary algorithms than many other common network optimization problems because of its several unique characteristics. For example, a flow at each edge can be anywhere between zero and its flow capacity. Munakata and Hashier [58] give a GA approach to applied the MXF problems. In this approach, each solution is represented by a flow matrix, and the fitness function is defined to reflect two characteristics—balancing vertices and the saturation rate of the flow. Akiyama *et al.* [59] gave a mixed approach based on neural network (NN) and GA for solving a *transport safety planning problem* that is an extended version of the MXF model. Ericsson *et al.* [60] presented a GA to solve the *open shortest path first* (OSPF) weight set-

ting problem that is also an extended version of MXF model which seeks a set of weights.

2.4.1 Mathematical Formulation

Let $G = (N, A)$ be a directed network, consisting of a finite set of nodes $N = 1, 2, \dots, n$ and a set of directed arcs $A = \{(i, j), (k, l), \dots, (s, t)\}$ joining m pairs of nodes in N . Arc (i, j) is said to be incident with nodes i and j , and is directed from node i to node j . Suppose that each arc (i, j) has assigned to it nonnegative numbers u_{ij} , the capacity of (i, j) . This capacity can be thought of as representing the maximum amount of some commodity that can “flow” through the arc per unit time in a steady-state situation. Such a flow is permitted only in the indicated direction of the arc, *i.e.*, from i to j .

Consider the problem of finding maximal flow from a source node s (node 1) to a sink node t (node n), which can be formulated as follows: Let x_{ij} = the amount of flow through arc (i, j) . The assumptions are given as follows:

- A1. The network is directed. We can fulfil this assumption by transforming any undirected network into a directed network.
- A2. All capacities are nonnegative.
- A3. The network does not contain a directed path from node s to node t composed only of infinite capacity arcs. Whenever every arc on a directed path P from node s to node t has infinite capacity, we can send an infinite amount of flow along this path, and therefore the maximum flow value is unbounded.
- A4. The network does not contain parallel arcs (*i.e.*, two or more arcs with the same tail and head nodes).

This assumption is essentially a notational convenience.

Indices

$i, j, k = 1, 2, \dots, n$ index of node

Parameters

n number of nodes
 u_{ij} capacity of arc (i, j)

Decision Variables

x_{ij} the amount of flow through arc $(i, j) \in A$.

Maximum Flow Model

The mathematical model of the MXF problem is formulated as a *linear programming*, in which the objective is to maximize total flow z from source node 1 to sink node n as follows:

$$\max z = v \quad (2.26)$$

$$\begin{aligned} \text{s. t. } \sum_{j=1}^n x_{ij} - \sum_{k=1}^n x_{ki} &= \begin{cases} v & (i = 1) \\ 0 & (i = 2, 3, \dots, n-1) \\ -v & (i = n) \end{cases} \quad (2.27) \\ 0 \leq x_{ij} \leq u_{ij} & \quad (i, j = 1, 2, \dots, n) \quad (2.28) \end{aligned}$$

where the constraint at Eq. 2.27, a conservation law, is observed at each of the nodes other than s or t . That is, what goes out of node i , $\sum_{j=1}^n x_{ij}$ must be equal to what comes in, $\sum_{k=1}^n x_{ki}$. The constraint at Eq. 2.28 is flow capacity. We call any set of numbers $\mathbf{x}=(x_{ij})$ which satisfy Eqs. 2.27 and 2.28 a feasible flow, and v is its value.

2.4.2 Priority-based GA for MXF Model

Let $P(t)$ and $C(t)$ be parents and offspring in current generation t , respectively. The overall procedure of priority-based GA (priGA) for solving MXF model is outlined as follows:

2.4.2.1 Genetic Representation

We have summarized the performance of the priority-based encoding method and other encoding methods introduced for solving the shortest path model in a previous section (page 62).

In this section, the special difficulty of the MXF is that a solution of the MXF is presented by various numbers of paths. Table 2.17 shows two examples of the solutions of the MXF problem with various paths, for solving a simple network with 11 nodes and 22 arcs as shown in Fig. 2.36.

Until now, for presenting a solution of MXF with various paths, the general idea of chromosome design is adding several shortest path based-encoding to one chromosome. The length of these representations is variable depending on various paths, and most offspring are infeasible after crossover and mutation operations. Therefore

```

procedure: priGA for MXF model
input: network data ( $N, A, U$ ),
        GA parameters ( $popSize, maxGen, p_M, p_C, p_I$ )
output: the best solution
begin
     $t \leftarrow 0$ ;
    initialize  $P(t)$  by priority-based encoding routine;
    evaluate  $P(t)$  by overall-path growth decoding routine;
    while (not terminating condition) do
        create  $C(t)$  from  $P(t)$  by WMX routine;
        create  $C(t)$  from  $P(t)$  by insertion mutation routine;
        evaluate  $C(t)$  by overall-path growth decoding routine;
        select  $P(t+1)$  from  $P(t)$  and  $C(t)$  by roulette wheel selection routine;
         $t \leftarrow t+1$ ;
    end
    output the best solution
end

```

Table 2.17 Examples of the solutions with various paths

| Solution | # of paths K | Path P | Flow f | Total flow z |
|----------|----------------|---------------|----------|----------------|
| 1 | 3 | 1-3-6-5-8-11 | 20 | 60 |
| | | 1-3-6-8-11 | 30 | |
| | | 1-3-7-6-9-11 | 10 | |
| 2 | 8 | 1-3-6-5-8-11 | 20 | 160 |
| | | 1-3-6-8-11 | 30 | |
| | | 1-3-7-6-9-11 | 10 | |
| | | 1-4-7-6-9-11 | 10 | |
| | | 1-4-7-10-11 | 30 | |
| | | 1-2-5-8-11 | 10 | |
| | | 1-2-5-8-9-11 | 30 | |
| | | 1-2-6-9-10-11 | 20 | |

these kinds of representations are lost regarding the feasibility and legality of chromosomes.

In this section, the priority-based chromosome is adopted; it is an effective representation to present a solution with various paths. For the decoding process, first, a *one-path growth procedure* is introduced that obtains a path based on the chromosome generated with given network in Fig. 2.37.

Then an *overall-path growth procedure* is presented in Fig. 2.38 that removes the flow used from each arc and deletes the arcs where its capacity is 0. Based on

updated network, we obtain a new path by one-path growth procedure, and repeat these steps until we obtain the overall possible path.

```

procedure: one-path growth
input: chromosome  $v$ , number of nodes  $n$ , the set of nodes  $S_i$  with all nodes adjacent to node  $i$ 
output: a path  $P_k$ 
begin
  initialize  $i \leftarrow 1, l_k \leftarrow 1, P_k[l] \leftarrow i$ ; //  $i$ : source node,  $l_k$ : length of path  $P_k$ .
  while  $S_i \neq \emptyset$  do
     $j' \leftarrow \text{argmax}\{v[j], j \in S_i\}$ ; //  $j'$ : the node with highest priority among  $S_i$ .
    if  $v[j'] \neq 0$  then
       $P_k[l_k] \leftarrow j'$ ; // chosen node  $j'$  to construct path  $P_k$ .
       $l_k \leftarrow l_k + 1$ ;
       $v[j'] \leftarrow 0$ ;
       $i \leftarrow j'$ ;
    else
       $S_i \leftarrow S_i \setminus j'$ ; // delete the node  $j'$  adjacent to node  $i$ .
       $v[i] \leftarrow 0$ ;
       $l_k \leftarrow l_k - 1$ ;
      if  $l_k \leq 1$  then  $l_k \leftarrow 1$ , break;
       $i \leftarrow P_k[l_k]$ ;
  output a path  $P_k[]$ 
end

```

Fig. 2.37 Pseudocode of one-path growth

By using the priority-based chromosome which is shown in Fig. 2.39, a trace table of decoding process is shown in Table 2.18 for a simple network (shown in Fig. 2.36). The final result is shown in Fig. 2.40. The objective function value is 160

2.4.2.2 Genetic Operations

Crossover Operator

In this section, we adopt the *weight mapping crossover* (WMX) proposed on page 68. WMX can be viewed as an extension of one-cut point crossover for permutation representation. As a one-cut point crossover, two chromosomes (parents) would choose a random-cut point and generate the offspring by using segment of own par-

procedure: overall-path growth for MXF

input: network data (N, A, U) , chromosome $v_k[\cdot]$, the set of nodes S_i with all nodes adjacent to node i

output: no. of paths L_k and the flow f_i^k of each path, $i \in L_k$

begin

$l \leftarrow 0$; // l : number of paths.

while $S_i \neq \emptyset$ **do**

$l \leftarrow l + 1$;

generate a path $P_l^k[\cdot]$ by **procedure** (one - path growth);

select the sink node a of path P_l^k ;

if $a = n$ **then**

$f_i^k \leftarrow f_{i-1}^k + \min\{u_{ij} \mid (i, j) \in P_l^k\}$; // calculate the flow f_i^k of the path P_l^k .

$\tilde{u}_{ij} \leftarrow u_{ij} - (f_i^k - f_{i-1}^k)$; // make a new flow capacity \tilde{u}_{ij} .

$S_i \leftarrow S_i \setminus j$, $(i, j) \in P_l^k$ & $\tilde{u}_{ij} = 0$; // delete the arcs which its capacity is 0.

else

$S_i \leftarrow S_i \setminus a$, $\forall i$; // update the set of nodes S_i .

output no. of paths $L_k \leftarrow l - 1$ and the flow f_i^k , $i \in L_k$.

end

Fig. 2.38 Pseudocode of overall-path growth for FMX

| | | | | | | | | | | | |
|------------|---|---|---|---|----|---|---|----|---|----|----|
| node ID : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| priority : | 2 | 1 | 6 | 4 | 11 | 9 | 8 | 10 | 5 | 3 | 7 |

Fig. 2.39 The best chromosome for MXF example

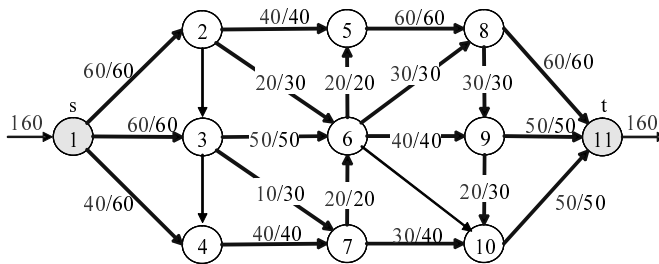


Fig. 2.40 Final result of the MXF example

Table 2.18 A trace table of decoding process for the MXF problem

| k | i | S_i | j' | P_k | S_1 | f_i^k |
|-----|-----|---------------|------|---------------|---------------|---------|
| 1 | 0 | | | 1 | | |
| | 1 | {2, 3, 4} | 3 | 1-3 | | |
| | 2 | {4, 6, 7} | 6 | 1-3-6 | | |
| | 3 | {5, 8, 9, 10} | 5 | 1-3-6-5 | | |
| | 4 | {8} | 8 | 1-3-6-5-8 | | |
| | 5 | {9, 11} | 11 | 1-3-6-5-8-11 | {2, 3, 4} | 20 |
| 2 | 0 | | | 1 | | |
| | 1 | {2, 3, 4} | 3 | 1-3 | | |
| | 2 | {4, 6, 7} | 6 | 1-3-6 | | |
| | 3 | {8, 9, 10} | 8 | 1-3-6-8 | | |
| | 4 | {9, 11} | 11 | 1-3-6-8-11 | {2, 3, 4} | 50 |
| 3 | ... | ... | ... | 1-3-7-6-9-11 | {2, 4} | 60 |
| 4 | ... | ... | ... | 1-4-7-6-9-11 | {2, 4} | 70 |
| 5 | ... | ... | ... | 1-4-7-10-11 | {2} | 100 |
| 6 | ... | ... | ... | 1-2-5-8-11 | {2} | 110 |
| 7 | ... | ... | ... | 1-2-5-8-9-11 | {2} | 140 |
| 8 | ... | ... | ... | 1-2-6-9-10-11 | \varnothing | 160 |

ent to the left of the cut point, then remap the right segment based on the weight of other parent of right segment.

Mutation Operator:

As introduced on page 70, insertion mutation selects a gene at random and inserts it in another random position. We adopt this insertion mutation to generate offspring in this section.

The immigration operator is modified to (1) include immigration routine, in each generation, (2) generate and (3) evaluate $popSize \cdot p_I$ random chromosomes, and (4) replace the $popSize \cdot p_I$ worst chromosomes of the current population (p_I , called the immigration probability). The detailed introduction is shown on page 23.

Selection Operator:

We adopt the roulette wheel selection (RWS). It is to determine selection probability or survival probability for each chromosome proportional to the fitness value. A model of the roulette wheel can be made displaying these probabilities. The detailed introduction is shown on page 13.

2.4.3 Experiments

The numerical examples, presented by Munakata and Hashier, were adopted [58]. All the simulations were performed with Java on a Pentium 4 processor (1.5-GHz clock). The parameter specifications used are as follows:

- Population size: $popSize = 10$
- Crossover probability: $p_C = 0.50$
- Mutation probability: $p_M = 0.50$
- Maximum generation: $maxGen = 1000$
- Terminating condition: 100 generations with same fitness.

2.4.3.1 Test Problem 1

The first test problem is given in Fig. 2.41. This network contains 25 nodes and 49 arcs with no bottlenecks or loops. The maximum flow is 90.

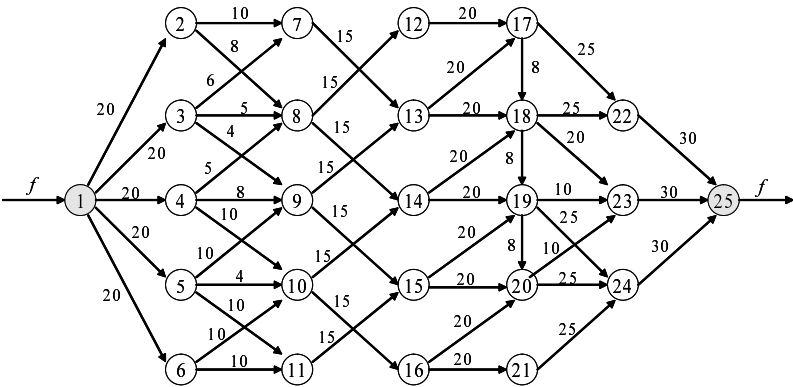


Fig. 2.41 First example of maximum flow model with 25 nodes and 49 arcs

The best chromosome using priority-based encoding is shown in Fig. 2.42 and the final result by overall-path growth decoding is shown in Fig. 2.43.

| | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| node ID : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| priority : | 8 | 25 | 2 | 12 | 15 | 20 | 1 | 16 | 21 | 14 | 7 | 6 | 18 |
| | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | |
| | 23 | 3 | 13 | 4 | 17 | 5 | 11 | 9 | 24 | 19 | 10 | 22 | |

Fig. 2.42 The best priority-based chromosome

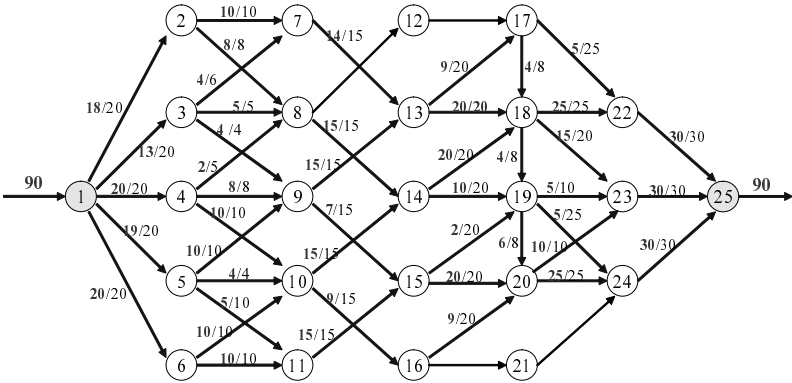


Fig. 2.43 Final result of maximum flow model with 25 nodes and 49 arcs

2.4.3.2 Test Problem 2

The second test problem is given in Fig. 2.44. This network contains 25 nodes and 56 arcs with loops but no bottlenecks. The maximum flow is 91.

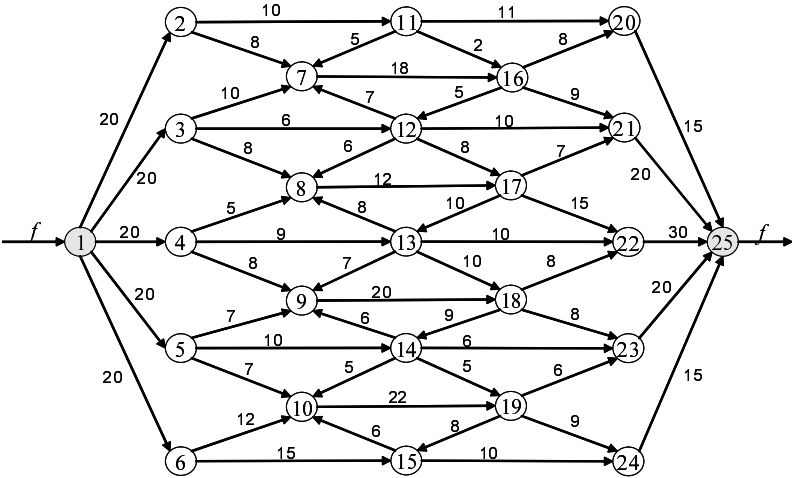


Fig. 2.44 Second example of maximum flow model with 25 nodes and 56 arcs

The best chromosome using priority-based encoding is shown in Fig. 2.45 and the final result by overall-path growth decoding is shown in Fig. 2.46.

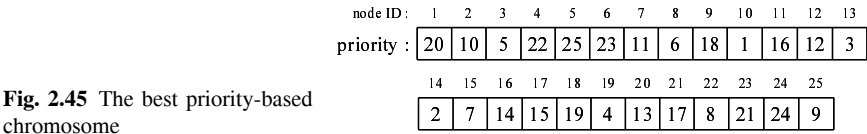


Fig. 2.45 The best priority-based chromosome

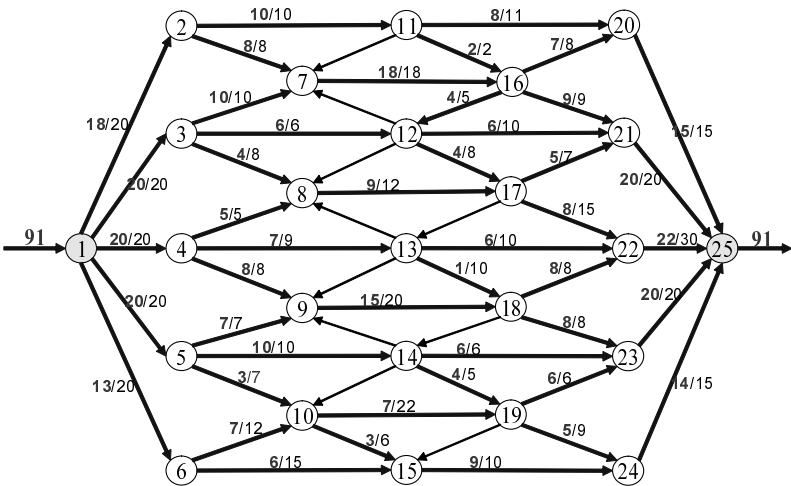


Fig. 2.46 Final result of maximum flow model with 25 nodes and 56 arcs

2.5 Minimum Cost Flow Model

The *minimum cost flow* (MCF) model is the most fundamental one of all network optimization problems. The *shortest path problem* (SPP) and *maximum flow problem* (MXF) can be formulated as two special cases of MCF: SPP considers arc flow costs but not flow capacities; MXF considers capacities but only the simplest cost structure. The MCF is finding the cheapest possible way of sending a certain amount of flow through a network.

Example 2.4: A Simple Example of MCF

A simple example of minimum cost flow problem is shown in Fig. 2.47 and Table 2.19 gives the data set of the example.

Traditional Methods

Successive shortest path algorithm: The successive shortest path algorithm maintains optimality of the solution at every step and strives to attain feasibility.

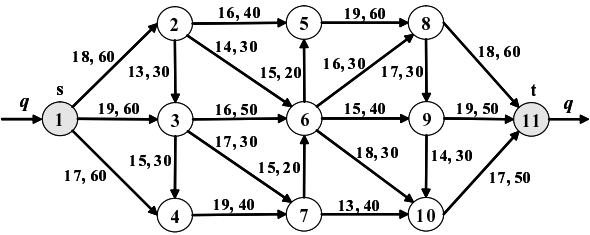


Fig. 2.47 A simple example of minimum cost flow model

Table 2.19 The data set of illustrative minimum cost network model

| <i>i</i> | <i>j</i> | <i>c_{ij}</i> | <i>u_{ij}</i> | <i>i</i> | <i>j</i> | <i>c_{ij}</i> | <i>u_{ij}</i> |
|----------|----------|-----------------------|-----------------------|----------|----------|-----------------------|-----------------------|
| 1 | 2 | 18 | 60 | 6 | 5 | 15 | 20 |
| 1 | 3 | 19 | 60 | 6 | 8 | 16 | 30 |
| 1 | 4 | 17 | 60 | 6 | 9 | 15 | 40 |
| 2 | 3 | 13 | 30 | 6 | 10 | 18 | 30 |
| 2 | 5 | 16 | 40 | 7 | 6 | 15 | 20 |
| 2 | 6 | 14 | 30 | 7 | 10 | 13 | 40 |
| 3 | 4 | 15 | 30 | 8 | 9 | 17 | 30 |
| 3 | 6 | 16 | 50 | 8 | 11 | 18 | 60 |
| 3 | 7 | 17 | 30 | 9 | 10 | 14 | 30 |
| 4 | 7 | 19 | 40 | 9 | 11 | 19 | 50 |
| 5 | 8 | 19 | 60 | 10 | 11 | 17 | 50 |

Primal-dual algorithm: The primal-dual algorithm for the minimum cost flow problem is similar to the successive shortest path algorithm in the sense that it also maintains a pseudo flow that satisfies the reduced cost optimality conditions and gradually converts it into a flow by augmenting flows along shortest paths.

Out-of-kilter algorithm: The out-of-kilter algorithm satisfies only the mass balance constraints, so intermediate solutions might violate both the *optimality conditions* and the *flow bound restrictions*.

2.5.1 Mathematical Formulation

Let $G=(N,A)$ be a directed network, consisting of a finite set of nodes $N = \{1,2,\cdots,n\}$ and a set of directed arcs $A = \{(i,j),(k,l),\cdots,(s,t)\}$ joining m pairs of nodes in N . Arc (i,j) is said to be incident with nodes i and j , and is directed from node i to node j . Suppose that each arc (i,j) has assigned to it nonnegative numbers c_{ij} , the cost of (i,j) and u_{ij} , the capacity of (i,j) . Associate this with an integer number q representing the supply/demand. The assumptions are given as follows:

- A1. The network is directed. We can fulfil this assumption by transforming any undirected network into a directed network.

- A2. All capacities, all arc costs and supply/demand q are nonnegative.
- A3. The network does not contain a directed path from node s to node t composed only of infinite capacity arcs. Whenever every arc on a directed path P from node s to node t has infinite capacity, we can send an infinite amount of flow along this path, and therefore the maximum flow value is unbounded.
- A4. The network does not contain parallel arcs (*i.e.*, two or more arcs with the same tail and head nodes).

Indices

$i, j, k = 1, 2, \dots, n$ index of node

Parameters

n : number of nodes
 q : supply/demand requirement
 c_{ij} : unit cost of arc (i, j)
 u_{ij} : capacity of arc (i, j)

Decision Variables

x_{ij} : the amount of flow through arc (i, j)

Minimum Cost Flow Model:

The minimum cost flow model is an optimization model formulated as follows:

$$\min z = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \quad (2.29)$$

$$\begin{aligned} \text{s. t. } & \sum_{j=1}^n x_{ij} - \sum_{k=1}^n x_{ki} \\ & = \begin{cases} q & (i = 1) \\ 0 & (i = 2, 3, \dots, n-1) \\ -q & (i = n) \end{cases} \end{aligned} \quad (2.30)$$

$$0 \leq x_{ij} \leq u_{ij} \quad (i, j = 1, 2, \dots, n) \quad (2.31)$$

where the constraint at Eq. 2.30, a conservation law is observed at each of the nodes other than s or t . That is, what goes out of node i , $\sum_{j=1}^n x_{ij}$ must be equal to what

comes in, $\sum_{k=1}^n x_{ki}$. Constraint at Eq. 2.31 is flow capacity. We call any set of numbers $\mathbf{x}=(x_{ij})$ which satisfy Eqs. 2.30 and 2.31 a feasible flow, and q is its value.

2.5.2 Priority-based GA for MCF Model

2.5.2.1 Genetic Representation

We have summarized the performance of the priority-based encoding method and other encoding methods for solving shortest path model in a previous section (page 62). In this section, MCF is the same special difficult with MXF that a solution of the MCF is presented by various numbers of paths. We adopt the priority-based chromosome; it is an effective representation for presenting a solution with various paths. The initialization process is same pseudocode of Fig. 2.7.

```

procedure 6 : overall - path growth
input : network data  $(V, A, C, U, q)$ , chromosome  $v_k$ , the set of nodes  $S_i$  with all nodes adjacent to node  $i$ 
output : number of paths  $L_k$  and the cost  $c_i^k$  of each path,  $i \in L_k$ 
begin
  initialize  $l \leftarrow 0$ ; //  $l$  : number of paths.
  while  $f_i^k \geq q$  do
     $l \leftarrow l + 1$ ;
    generate a path  $P_l^k$  by procedure 4 (one - path growth);
    select the sink node  $a$  of path  $P_l^k$ ;
    if  $a = n$  then
       $f_i^k \leftarrow f_{i-1}^k + \min\{u_{ij} \mid (i, j) \in P_l^k\}$ ; // calculate the flow  $f_i^k$  of the path  $P_l^k$ .
       $c_i^k \leftarrow c_{i-1}^k + \sum_{j=1}^n \sum_{i=1}^n c_{ij}(f_i^k - f_{i-1}^k)$ ,  $\forall (i, j) \in P_l^k$ ; // calculate the cost  $c_i^k$  of the path  $P_l^k$ .
       $\tilde{u}_{ij} \leftarrow u_{ij} - (f_i^k - f_{i-1}^k)$ ; // make a new flow capacity  $\tilde{u}_{ij}$ .
       $S_i \leftarrow S_i \setminus j$ ,  $(i, j) \in P_l^k$  &  $\tilde{u}_{ij} = 0$ ; // delete the arcs which its capacity is 0.
    else
       $S_i \leftarrow S_i \setminus a$ ,  $\forall i$ ; // update the set of nodes  $S_i$ .
    output number of paths  $L_k \leftarrow l - 1$  and the cost  $c_i^k$ ,  $i \in L_k$ .
end

```

Fig. 2.48 Pseudocode of overall-path growth for MCF

For the decoding process, firstly, we adopt a *one-path growth procedure* that presents same the priority-based decoding as MXF. It is given in a previous sec-

tion (on page 103). Then we present an extended version of *overall-path growth procedure* that removes the used flow from each arc and deletes the arcs when its capacity is 0. Based on an updated network, we obtain a new path by one-path growth procedure, and repeat these steps until obtaining the feasible flow equal to q . The pseudocode of overall-path growth is shown in Fig. 2.48. By using the best chromosome for the MCF example which is shown in Fig. 2.49, we show a trace table of decoding process which is shown in Table 2.20 for a simple network (shown in Fig. 2.47). Where the supply/demand requirement is 60. The final result is shown in Fig. 2.50. The objective function value is

$$z = 20 \times 87 + 30 \times 69 + 10 \times 85 = 4660$$

Fig. 2.49 The best chromosome for MCF example

| | | | | | | | | | | | |
|------------|---|---|---|---|----|---|---|----|---|----|----|
| node ID : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| priority : | 2 | 1 | 6 | 4 | 11 | 9 | 8 | 10 | 5 | 3 | 7 |

Table 2.20 A trace table of decoding process for the MCF example

| k | i | S_i | l | P_k | S_l | f_l^k | c_l^k |
|-----|-----|-------------|-----|-------------------|---------|---------|---------|
| 1 | 0 | | | 1 | | | |
| | 1 | 2, 3, 4 | 3 | 1, 3 | | | |
| | 2 | 4, 6, 7 | 6 | 1, 3, 6 | | | |
| | 3 | 5, 8, 9, 10 | 5 | 1, 3, 6, 5 | | | |
| | 4 | 8 | 8 | 1, 3, 6, 5, 8 | | | |
| | 5 | 9, 11 | 11 | 1, 3, 6, 5, 8, 11 | 2, 3, 4 | 20 | 87 |
| 2 | 0 | | | 1 | | | |
| | 1 | 2, 3, 4 | 3 | 1, 3 | | | |
| | 2 | 4, 6, 7 | 6 | 1, 3, 6 | | | |
| | 3 | 8, 9, 10 | 8 | 1, 3, 6, 8 | | | |
| | 4 | 9, 11 | 11 | 1, 3, 6, 8, 11 | 2, 3, 4 | 30 | 69 |
| 3 | 0 | | | 1 | | | |
| | 1 | 2, 3, 4 | 3 | 1, 3 | | | |
| | 2 | 4, 7 | 7 | 1, 3, 7 | | | |
| | 3 | 6, 10 | 6 | 1, 3, 7, 6 | | | |
| | 4 | 9, 10 | 9 | 1, 3, 7, 6, 9 | | | |
| | 5 | 10, 11 | 11 | 1, 3, 7, 6, 9, 11 | 2, 4 | 10 | 85 |

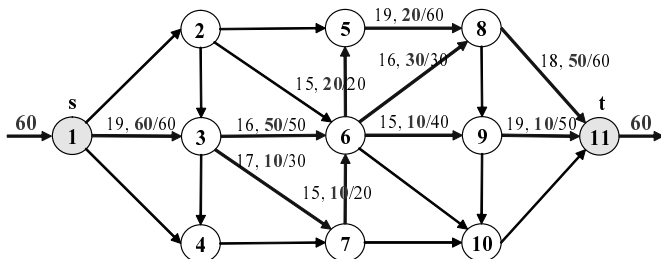


Fig. 2.50 Final result of the simple MCF example

2.5.2.2 Genetic Operators

Crossover Operator

In this section, we adopt the *weight mapping crossover* (WMX) proposed on page 68. WMX can be viewed as an extension of one-cut point crossover for permutation representation. As one-cut point crossover, two chromosomes (parents) would choose a random-cut point and generate the offspring by using segment of own parent to the left of the cut point, then remapping the right segment based on the weight of other parent of right segment.

Mutation Operator

As introduced on page 70, insertion mutation selects a gene at random and inserts it in another random position. We adopt this insertion mutation to generate offspring in this section.

The immigration operator is modified to (1) include the immigration routine, in each generation, (2) generate and (3) evaluate $popSize \cdot p_I$ random chromosomes, and (4) replace the $popSize \cdot p_I$ worst chromosomes of the current population (p_I , called the immigration probability). The detailed introduction is showed on page 23.

Selection Operator

We adopt the roulette wheel selection (RWS). It is to determine selection probability or survival probability for each chromosome proportional to the fitness value. A model of the roulette wheel can be made displaying these probabilities. The detailed introduction is showed on page 13.

2.5.3 Experiments

Two MCF test problems are generated based on the MXF network structure presented by [19], and randomly assigned unit shipping costs on each arc. All the simulations were performed with Java on a Pentium 4 processor (1.5-GHz clock). The parameter specifications are used as follows:

- Population size: $popSize = 10$
- Crossover probability: $p_C = 0.50$
- Mutation probability: $p_M = 0.50$
- Maximum generation: $maxGen = 1000$
- Terminating condition: 100 generations with same fitness.

2.5.3.1 Test Problem 1

The first test problem is given in Fig. 2.51. This network contains 25 nodes and 49 arcs with no bottlenecks or loops. The supply/demand requirement is 70.

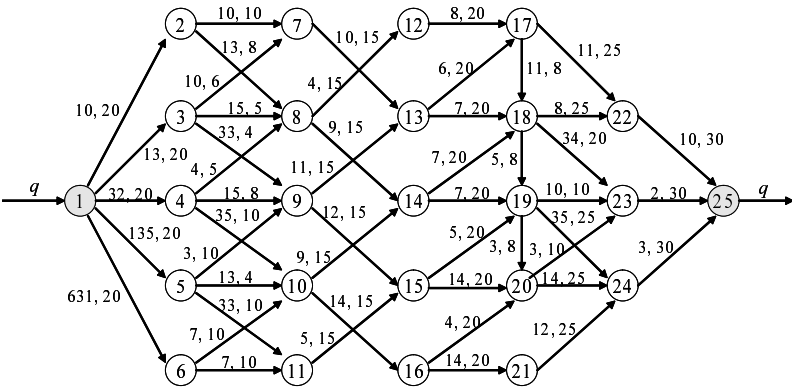


Fig. 2.51 First example of MCF model with 25 nodes and 49 arcs

The best chromosome by the priority-based encoding is shown in Fig. 2.52, the best result is 6969 and the final result by using overall-path growth decoding is shown in Fig. 2.53.

| | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| node ID : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| priority : | 1 | 16 | 11 | 9 | 6 | 5 | 7 | 8 | 15 | 10 | 3 | 12 | 13 |
| | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | |
| | 21 | 4 | 22 | 14 | 18 | 20 | 24 | 17 | 25 | 23 | 2 | 19 | |

Fig. 2.52 The best priority-based chromosome

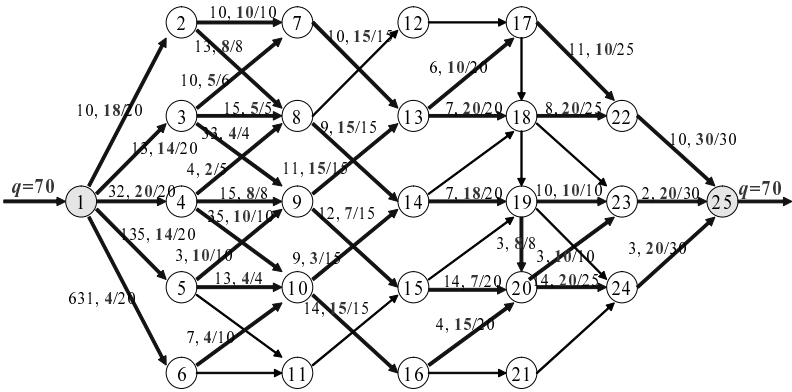


Fig. 2.53 Final result of first MCF example with 25 nodes and 49 arcs

2.5.3.2 Test Problem 2

The second test problem is given in Fig. 2.54. This network contained 25 nodes and 56 arcs with loops but no bottlenecks. The supply/demand requirement is 72.

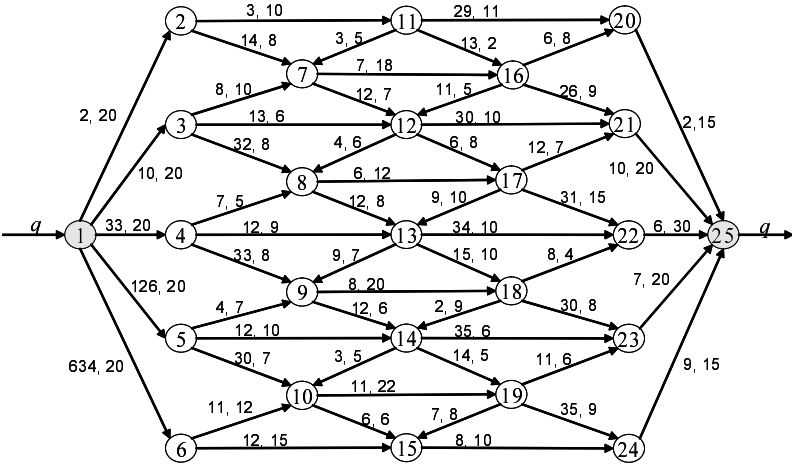


Fig. 2.54 Second example of minimum cost flow model with 25 nodes and 56 arcs

The best chromosome by the priority-based encoding is shown in Fig. 2.55, the best result is 5986 and the final result by using overall-path growth decoding is shown in Fig. 2.56.

Fig. 2.55 The best priority-based chromosome

| | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| node ID : | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| priority : | 10 | 25 | 15 | 24 | 11 | 4 | 7 | 8 | 12 | 6 | 5 | 9 | 13 |
| | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | |
| | 14 | 3 | 16 | 17 | 18 | 1 | 20 | 22 | 19 | 23 | 21 | 2 | |

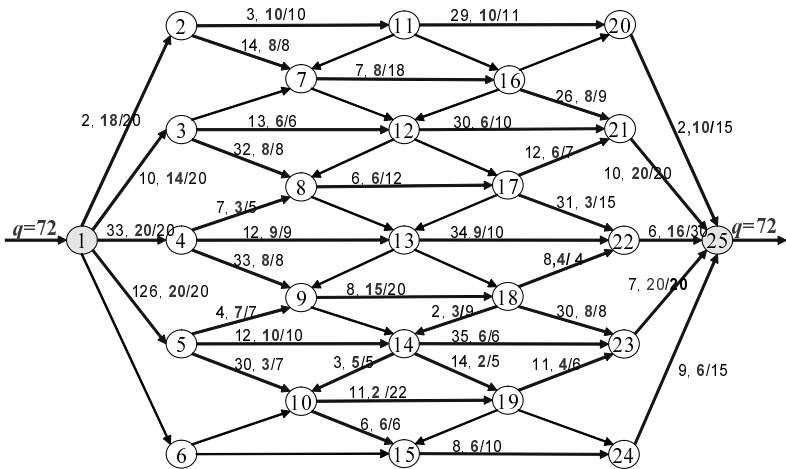


Fig. 2.56 Final result of second MCF example with 25 nodes and 56 arcs

2.6 Bicriteria MXF/MCF Model

Network design problems where even one cost measure must be minimized are often NP-hard [63]. However, in practical applications it is often the case that the network to be built is required to multiobjective. In the following, we introduce three core bicriteria network design models. (1) *Bicriteria shortest path* (bSP) model is one of the basic multi-criteria network design problems. It is desired to find a diameter-constrained path between two specified nodes with minimizing two cost functions. Hansen presented the first bSP model [64]. Recently, Skriver and Andersen examined the correlative algorithms for the bSP problems [65]; Azaron presented a new methodology to find the bicriteria shortest path under the steady-state condition [66]. (2) *Bicriteria spanning tree* (bST) model play a central role within the field of multi-criteria network modes. It is desired to find a subset of arcs which is a tree and connects all the nodes together with minimizing two cost functions. Marathe et al. presented a general class of bST model [67], and Balint proposed a non-approximation algorithm to minimize the diameter of a spanning sub-graph subject to the constraint that the total cost of the arcs does not exceed a given budget [68]. (3) *Bicriteria network flow* (bNF) model and bSP model are mutual complementary topics. It is desired to send as much flow as possible between two special

nodes without exceeding the capacity of any arc. Lee and Pulat presented algorithm to solve a bNF problem with continuous variables [69].

As we know, the shortest path problem (SPP) considers arc flow costs but not flow capacities; the maximum flow (MXF) problem considers capacities but only the simplest cost structure. SPP and MXF combine all the basic ingredients of network design problems. Bicriteria MXF/MCF model is an *integrated bicriteria network design* (bND) model integrating these nuclear ingredients of SPP and MXF. This bND model considers the flow costs, flow capacities and multiobjective optimization. This bND model provide a useful way to model real world problems, which are extensively used in many different types of complex systems such as communication networks, manufacturing systems and logistics systems. For example, in a communication network, we want to find a set of links which consider the connecting cost (or delay) and the high throughput (or reliability) for increasing the network performance [70, 71]; as an example in the manufacturing application described in [72], the two criteria under consideration are cost, that we wish to minimize, and manufacturing yield, that we wish to maximize; in a logistics system, the main drive to improve logistics productivity is the enhancement of customer services and asset utilization through a significant reduction in order cycle time (lead time) and logistics costs [82].

Review of Related Work

This bND model is the most complex case of the multi-criteria network design models. The traditional bSP model and bST model only consider the node selection for a diameter-constrained path or spanning tree, and the traditional bNF model only considers the flow assignment for maximizing the total flows. But the bND model considers not only the node selection but also flow assignment for two conflicting objectives, maximizing the total flow and minimizing the total cost simultaneously. In network design models, even one measure of flow maximization is often NP-hard problem [58]. For example, a flow at each edge can be anywhere between zero and its flow capacity, *i.e.*, it has more “freedom” to choose. In many other problems, selecting an edge may mean simply to add fixed distance. It has been well studied using a variety of methods by parallel algorithm with a worst case time of $O(n^2 \log n)$ (Shiloach and Vishkin), distributed algorithms with a worst case time of $O(n^2 \log n)$ to $O(n^3)$ (Yeh and Munakata), and sequential algorithms *etc.*, with n nodes [58]. In addition, as one of multiobjective optimization problems, this bND problem is not simply an extension from single objective to two objectives. In general, we cannot get the optimal solution of the problem because these objectives conflict with each other in practice. The real solutions to the problem are a set of Pareto optimal solutions (Chankong and Haimes [52]). To solve this bND problem, the set of efficient solutions may be very large and possibly exponential in size. Thus the computational effort required to solve it can increase exponentially with the problem size.

Recently, genetic algorithm (GA) and other evolutionary algorithms (EAs) have been successfully applied in a wide variety of network design problems [73]. For

example, Ahn and Ramakrishna developed a variable-length chromosomes and a new crossover operator for shortest path routing problem [12], Wu and Ruan proposed a gene-constrained GA for solving shortest path problem [74], Li *et al.* proposed a specific GA for *optimum path planning in intelligent transportation system* (ITS) [75], Kim *et al.* proposed a new path selection scheme which uses GA along with the modified roulette wheel selection method for *MultiProtocol label switching* (MPLS) network [76], Hasan *et al.* proposed a *novel heuristic GA* to solve the *single source shortest path* (ssSP) problem [77], Ji *et al.* developed a *simulation-based GA* to find multi-objective paths with minimizing both expected travel time and travel time variability in ITS [78], Chakraborty *et al.* developed *multiobjective genetic algorithm* (moGA) to find out simultaneously several alternate routes depending on distance, contains minimum number of turns, path passing through mountains [79], Garrozi and Araujo presented a moGA to solve the *multicast routing problem* with maximizing the common links in source-destination routes and minimizing the route sizes [80], and Kleeman *et al.* proposed a *modified nondominated sorting genetic algorithm II* (nsGA II) for the *multicommodity capacitated network design problem* (mcNDP), the multiple objectives including costs, delays, robustness, vulnerability, and reliability [81]. Network design problems where even one cost measure must be minimized are often NP-hard [63]. However, in practical applications, it is often the case that the network to be built is required to multiobjective. In the following, we introduce three core bicriteria network design models. (1) *Bicriteria shortest path* (bSP) model is one of the basic multi-criteria network design problems. It is desired to find a diameter-constrained path between two specified nodes with minimizing two cost functions. Hansen presented the first bSP model [64]. Recently, Skriver and Andersen examined the correlative algorithms for the bSP problems [65]; Azaron presented a new methodology to find the bicriteria shortest path under the steady-state condition [66]. (2) *Bicriteria spanning tree* (bST) model play a central role within the field of multi-criteria network modes. It is desired to find a subset of arcs which is a tree and connects all the nodes together with minimizing two cost functions. Marathe *et al.* presented a general class of bST model [67], and Balint proposed a non-approximation algorithm to minimize the diameter of a spanning sub-graph subject to the constraint that the total cost of the arcs does not exceed a given budget [68]. (3) *Bicriteria network flow* (bNF) model and bSP model are mutual complementary topics. It is desired to send as much flow as possible between two special nodes without exceeding the capacity of any arc. Lee and Pulat presented algorithm to solve a bNF problem with continuous variables [69].

As we know, shortest path problem (SPP) considers arc flow costs but not flow capacities; maximum flow (MXF) problem considers capacities but only the simplest cost structure. SPP and MXF combine all the basic ingredients of network design problems. In this chapter, we formulate an *integrated bicriteria network design* (bND) model with integrating these nuclear ingredients of SPP and MXF. This bND model considers the flow costs, flow capacities and multiobjective optimization. This bND model provide a useful way to modeling real world problems, which are extensively used in many different types of complex systems such as communication networks, manufacturing systems and logistics systems. For example, in a

communication network, we want to find a set of links which consider the connecting cost (or delay) and the high throughput (or reliability) for increasing the network performance [70, 71]; as an example in the manufacturing application described in [72], the two criteria under consideration are cost, that we wish to minimize, and manufacturing yield, that we wish to maximize; in a logistics system, the main drive to improve logistics productivity is the enhancement of customer services and asset utilization through a significant reduction in order cycle time (lead time) and logistics costs [82].

2.6.1 Mathematical Formulations

Let $G=(N,A)$ be a directed network, consisting of a finite set of nodes $N = 1, 2, \dots, n$ and a set of directed arcs $A = (i,j), (k,l), \dots, (s,t)$ joining m pairs of nodes in N . Arc (i,j) is said to be incident with nodes i and j , and is directed from node i to node j . Suppose that each arc (i,j) has assigned to it nonnegative numbers c_{ij} , the cost of (i,j) and u_{ij} , the capacity of (i,j) . This capacity can be thought of as representing the maximum amount of some commodity that can “flow” through the arc per unit time in a steady-state situation. Such a flow is permitted only in the indicated direction of the arc, *i.e.*, from i to j .

Consider the problem of finding maximal flow and minimal cost from a source node s (node 1) to a sink node t (node n), which can be formulated as follows. Let x_{ij} = the amount of flow through arc (i,j) . The assumptions are given as following:

- A1. The network is directed. We can fulfil this assumption by transforming any undirected network into a directed network.
- A2. All capacities and all arc costs are nonnegative.
- A3. The network does not contain a directed path from node s to node t composed only of infinite capacity arcs. Whenever every arc on a directed path P from node s to node t has infinite capacity, we can send an infinite amount of flow along this path, and therefore the maximum flow value is unbounded.
- A4. The network does not contain parallel arcs (*i.e.*, two or more arcs with the same tail and head nodes). This assumption is essentially a notational convenience.

Indices

$i, j, k = 1, 2, \dots, n$ index of node

Parameters

c_{ij} : unit cost of arc (i,j)

u_{ij} : capacity of arc (i,j)

Decision Variables

x_{ij} : the amount of flow through arc (i, j)
 f : the total flow

The bND problem is formulated as a mixed integer programming model, in which the objectives are maximizing total flow z_1 and minimizing total cost z_2 from source node 1 to sink node n as follows:

$$\max z_1 = f \quad (2.32)$$

$$\min z_2 = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \quad (2.33)$$

$$\text{s. t. } \sum_{j=1}^n x_{ij} - \sum_{k=1}^n x_{ki} = \begin{cases} f & (i = 1) \\ 0 & (i = 2, 3, \dots, n-1) \\ -f & (i = n) \end{cases} \quad (2.34)$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \quad (2.35)$$

$$x_{ij} = 0 \text{ or } 1 \quad \forall i, j \quad (2.36)$$

where the constraint at Eq. 2.34, a conservation law is observed at each of the nodes other than s or t . That is, what goes out of node i , $\sum_{j=1}^n x_{ij}$ must be equal to what comes in, $\sum_{k=1}^n x_{ki}$. Constraint at Eq. 2.35 is flow capacity. We call any set of numbers $\mathbf{x}=(x_{ij})$ which satisfy Eqs. 2.34 and 2.35 a feasible flow, or simply a flow, and f is its value.

2.6.2 Priority-based GA for bMXF/MCF Model

The overall procedure of priority-based GA for solving bicriteria MXF/MCF model is outlined as follows.

2.6.2.1 Genetic Representation

To solve this bND problem, the special difficulties of the bND problem are (1) a solution is presented by various numbers of paths, (2) a path contains various numbers of nodes and the maximal number is $n-1$ for an n node network, and (3) a random sequence of edges usually does not correspond to a path. Thus a nature genetic representation is very difficult to use to represent a solution of bND problem.

We have summarized the performance of the priority-based encoding method and also introduced encoding methods for solving shortest path routing problems on page 62.

In this section, we consider the special difficulty of the bND problem in that a solution of the bND problem is presented by various numbers of paths. Figure

procedure: priGA for bMXF/MCF model

input: network data (N, A, C, U) , GA parameters $(popSize, maxGen, p_M, p_C, p_I)$

output: Pareto optimal solutions E

begin

$t \leftarrow 0$;

initialize $P(t)$ by priority-based encoding routine;

calculate objectives $z_i(P), i = 1, \dots, q$ by priority-based decoding routine;

create Pareto $E(P)$;

evaluate $eval(P)$ by *interactive adaptive-weight fitness assignment routine*;

while (not terminating condition) **do**

create $C(t)$ from $P(t)$ by weight mapping crossover routine;

create $C(t)$ from $P(t)$ by insertion mutation routine;

create $C(t)$ from $P(t)$ by immigration routine;

calculate objectives $z_i(C), i = 1, \dots, q$ by priority-based decoding routine;

update Pareto $E(P, C)$;

evaluate $eval(P, C)$ by *interactive adaptive-weight fitness assignment routine*;

select $P(t+1)$ from $P(t)$ and $C(t)$ by roulette wheel selection routine;

$t \leftarrow t + 1$;

end

output Pareto optimal solutions $E(P, C)$

end

2.57 presents a simple network with 11 nodes and 22 arcs and Table 2.21 shows 2 examples of the solutions of bND with various paths.

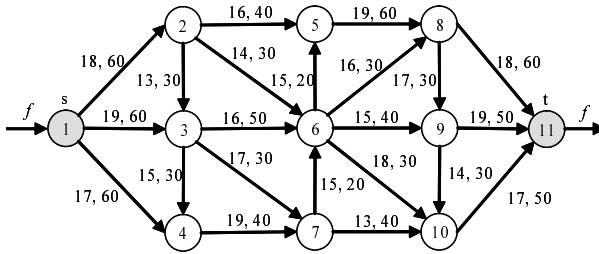


Fig. 2.57 A simple network with 11 nodes and 22 edges

Until now, to present a solution of bND problem with various paths, the general idea of chromosome design is adding several shortest path based-encodings to one chromosome. The length of these representations is variable depending on various paths, and most offspring are infeasible after crossover and mutation. Therefore these kinds of representations lose the feasibility and legality of chromosomes.

In this research, the priority-based chromosome is adopted; it is an effective representation for present a solution with various paths. For the decoding process, first,

Table 2.21 Examples of the solutions with various paths

| Solution | # of paths K | Path $P(\cdot)$ | Total flow z_1 | Total cost z_2 |
|----------|----------------|--|------------------|------------------|
| 1 | 3 | (1-3-6-5-8-11), (1-3-6-8-11), (1-3-7-6-9-11) | 60 | 4660 |
| 2 | 8 | (1-3-6-5-8-11), (1-3-6-8-11), (1-3-7-6-9-11), (1-4-7-6-9-11), (1-4-7-10-11), (1-2-5-8-11), (1-2-5-8-9-11), (1-2-6-9-10-11) | 160 | 12450 |

a *one-path growth procedure* is combined that obtains a path base on the generated chromosome with given network in Fig. 2.58.

Then present an *overall-path growth procedure* in Fig. 2.59 that removes the used flow from each arc and deletes the arcs where its capacity is 0. Based on updated network, a new path can be obtained by the one-path growth procedure, and repeat these steps until we obtain overall possible path. A trace table of the decoding process is shown in Table 2.22, for a simple network (shown in Fig. 2.57).

procedure: one-path growth

input: chromosome v , number of nodes n , the set of nodes S_i with all nodes adjacent to node i

output: path P_k , length of path P_k l_k

begin

initialize $i \leftarrow 1, l_k \leftarrow 1, P_k[l] \leftarrow i$; // i : source node, l_k : length of path P_k .

while $S_i \neq \emptyset$ **do**

$j' \leftarrow \arg \max \{v[j], j \in S_i\}$; // j' : the node with highest priority among S_i .

if $v[j'] \neq 0$ **then**

$P_k[l_k] \leftarrow j'$; // chosen node j' to construct path P_k .

$l_k \leftarrow l_k + 1$;

$v[j'] \leftarrow 0$;

$i \leftarrow j'$;

else

$S_i \leftarrow S_i \setminus j'$; // delete the node j' adjacent to node i .

$v[i] \leftarrow 0$;

$l_k \leftarrow l_k - 1$;

if $l_k \leq 1$ **then** $l_k \leftarrow 1$, **break**;

$i \leftarrow P_k[l_k]$;

output path P_k and path length l_k

end

Fig. 2.58 Pseudocode of one-path growth

procedure: overall-path growth for bicriteria MXF/MCF
input: network data (V, A, C, U) , chromosome $v_k[\cdot]$,
 set of nodes S_i with all nodes adjacent to node i
output: no. of paths L_k , flow f_i^k and cost c_i^k of each path, $i \in L_k$
begin
 $l \leftarrow 0$; // l : number of paths.
while $S_i \neq \emptyset$ **do**
 $l \leftarrow l + 1$;
 generate a path P_l^k by **procedure** (one-path growth);
 select the sink node α of path P_l^k ;
if $\alpha = n$ **then**
 $f_l^k \leftarrow f_{l-1}^k + \min\{u_{ij} \mid (i, j) \in P_l^k\}$; // calculate the flow f_l^k of the path P_l^k .
 $c_l^k \leftarrow c_{l-1}^k + \sum_{i=1}^n \sum_{j=1}^n c_{ij} (f_i^k - f_{i-1}^k)$, $\forall (i, j) \in P_l^k$; // calculate the cost c_l^k of the path P_l^k .
 $\tilde{u}_{ij} \leftarrow u_{ij} - (f_i^k - f_{i-1}^k)$; // make a new flow capacity \tilde{u}_{ij} .
 $S_i \leftarrow S_i \setminus j$, $(i, j) \in P_l^k$ & $\tilde{u}_{ij} = 0$; // delete the arcs which its capacity is 0.
else
 $S_i \leftarrow S_i \setminus \alpha$, $\forall i$; // update the set of nodes S_i .
output no. of paths $L_k \leftarrow l - 1$, the flow f_i^k and the cost c_i^k , $i \in L_k$.
end

Fig. 2.59 Pseudocode of overall-path growth

2.6.2.2 Genetic Operators

Crossover Operator

In this section, we adopt the weight mapping crossover (WMX) proposed on page 68. WMX can be viewed as an extension of one-cut point crossover for permutation representation. With one-cut point crossover, two chromosomes (parents) would be chose a random-cut point and generate the offspring by using segment of one parent to the left of the cut point, then remapping the right segment based on the weight of the other parent of right segment.

Mutation Operator

As introduced on page 70, insertion mutation selects a gene at random and inserts it in another random position. We adopt this insertion mutation to generate offspring in this section.

The immigration operator is modified to (1) include immigration routine, in each generation, (2) generate and (3) evaluate $popSize \cdot p_I$ random chromosomes, and (4)

Table 2.22 Examples of the solutions with various paths

| k | i | S_i | j' | P_k | S_1 | z_1^k | z_2^k |
|-----|-----|---------------|------|---------------|---------------|---------|---------|
| 1 | 0 | | | 1 | | | |
| | 1 | {2, 3, 4} | 3 | 1-3 | | | |
| | 2 | {4, 6, 7} | 6 | 1-3-6 | | | |
| | 3 | {5, 8, 9, 10} | 5 | 1-3-6-5 | | | |
| | 4 | {8} | 8 | 1-3-6-5-8 | | | |
| | 5 | {9, 11} | 11 | 1-3-6-5-8-11 | {2, 3, 4} | 20 | 1740 |
| 2 | 0 | | | 1 | | | |
| | 1 | {2, 3, 4} | 3 | 1-3 | | | |
| | 2 | {4, 6, 7} | 6 | 1-3-6 | | | |
| | 3 | {8, 9, 10} | 8 | 1-3-6-8 | | | |
| | 4 | {9, 11} | 11 | 1-3-6-8-11 | {2, 3, 4} | 50 | 3810 |
| 3 | ... | ... | ... | 1-3-7-6-9-11 | {2, 4} | 60 | 4660 |
| 4 | ... | ... | ... | 1-4-7-6-9-11 | {2, 4} | 70 | 5510 |
| 5 | ... | ... | ... | 1-4-7-10-11 | {2} | 100 | 7490 |
| 6 | ... | ... | ... | 1-2-5-8-11 | {2} | 110 | 8200 |
| 7 | ... | ... | ... | 1-2-5-8-9-11 | {2} | 140 | 10870 |
| 8 | ... | ... | ... | 1-2-6-9-10-11 | \varnothing | 160 | 12430 |

replace the $popSize \cdot p_I$ worst chromosomes of the current population (p_I , called the immigration probability). The detailed introduction is shown on page 23.

Selection Operator

We adopt the roulette wheel selection (RWS). It is to determine selection probability or survival probability for each chromosome proportional to the fitness value. A model of the roulette wheel can be made displaying these probabilities. The detailed introduction is shown in page 13.

2.6.3 *i-awGA for bMXF/MCF Model*

As described above, a characteristic of the priority-based decoding method is its ability to generate various paths by one chromosome. The several non-dominated solutions can be combined by these paths. *i.e.*, one chromosome mapping various fitnesses. However, most of the fitness assignment methods can only be applied the case of one chromosome mapping one fitness assignment as the fitness assignment methods above (rwGA, awGA, spEA, nsGAII, *etc.*) cannot be adopted successfully to solve bND problems by using the priority-based encoding method.

To develop Pareto ranking-based fitness assignment for bND problems, two special difficulties are encountered. (1) For creating Pareto rank of each solution set,

Pareto ranking-based fitness assignment needs the process which sorts popSize (number of chromosomes) solution sets. It is a disadvantage for considering computation time. In this research, for solving bND problems, we need a more complex Pareto rank sorting process. Because of there are $\sum_{k=1}^{popSize} L_k$ solution sets in each generation, where L_k is number of solution set in k -th chromosome. The computational effort required to solve bND problem can increase with the problem size. (2) After created Pareto rank of each solution set, there are L_k Pareto ranks in k -th chromosome, and how to assign the fitness of each chromosome is another difficulty.

Different from Pareto ranking-based fitness assignment, weighted-sum based fitness assignment is to assign weights to each objective function and combine the weighted objectives into a single objective function. It is easier to calculate the weight-sum based fitness and the sorting process becomes unnecessary; thus it is effective for considering the computation time to solve the problems. In addition, another characteristic of weighted-sum approach is its use to adjust genetic search toward the Pareto frontier.

In this research, we propose an interactive adaptive-weight fitness assignment approach, which is an improved adaptive-weight fitness assignment approach with the consideration of the disadvantages of weighted-sum approach and Pareto ranking-based approach. We combine a penalty term to the fitness value for all dominated solutions. To solve bND problems, we first define two extreme points: the maximum extreme point $z^+ \leftarrow \{z_1^{\max}, z_2^{\max}\}$ and the minimum extreme point $z^- \leftarrow \{z_1^{\min}, z_2^{\min}\}$ in criteria space, where $z_1^{\max}, z_2^{\max}, z_1^{\min}$ and z_2^{\min} are the maximum value and the minimum value for objective 1 (maximum total flow) and the reciprocal of objective 2 (minimum total cost) in the current population. Calculate the adaptive weight $w_1 = 1/(z_1^{\max} - z_1^{\min})$ for objective 1 and the adaptive weight $w_2 = 1/(z_2^{\max} - z_2^{\min})$ for objective 2. Afterwards, calculate the penalty term $p(v_k)=0$, if v_k is a nondominated solution in the nondominated set P . Otherwise $p(v_k)=1$ for dominated solution v_k . Last, calculate the fitness value of each chromosome by combining the method as follows adopt roulette wheel selection as supplementary to the interactive adaptive-weight genetic algorithm (i-awGA):

$$eval(v_k) = 1 / \left(\sum_{i=1}^{L_k} 1 / \left(w_1 (f_i^k - z_1^{\min}) + w_2 (1/c_i^k - z_2^{\min}) \right) + p(x_i^k) \right) / L_k$$

$\forall k \in popSize$

where x_i^k is the i -th solution set in chromosome v_k , f_i^k is the flow of x_i^k and c_i^k is the cost of x_i^k . We adopted the roulette wheel selection as the supplementary operator to this interactive adaptive-weight assignment approach.

2.6.4 Experiments and Discussion

In the experimental study, the performance comparisons of multiobjective GAs is demonstrated for solving bND problems by different fitness assignment approaches; there are Strength Pareto Evolutionary Algorithm (spEA), non-dominated sorting Genetic Algorithm II (nsGA II), random-weight Genetic Algorithm (rwGA) and interactive adaptive-weight Genetic Algorithm (i-awGA) proposed. Two maximum flow test problems are presented by [58], and randomly assigned unit shipping costs along each arc. In each GA approach, priority-based encoding was used, and WMX + Insertion + Immigration were used as genetic operators.

2.6.4.1 Performance Measures

In these experiments, the following performance measures are considered to evaluate the efficiency of the different fitness assignment approaches. The detailed explanations were introduced on page 41.

Number of obtained solutions $|S_j|$: Evaluate each solution set depending on the number of obtained solutions.

Ratio of nondominated solutions $R_{NDS}(S_j)$: This measure simply counts the number of solutions which are members of the Pareto-optimal set S^* .

Average distance $D1_R(S_j)$: Instead of finding whether a solution of S_j belongs to the set S^* or not, this measure finds an average distance of the solutions of S_j from S^* .

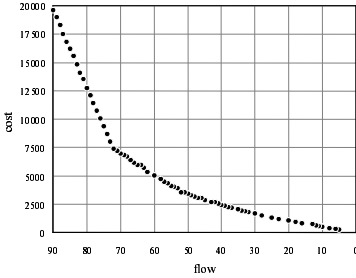
In order to make a large number of solutions in the reference set S^* , first we calculate the solution sets with special GA parameter settings and very long computation times for each approach used in comparison experiments; then we combine these solution sets to calculate the reference set S^* . Furthermore, we will combine small but reasonable GA parameter settings for comparison experiments. This ensures the effectiveness of the reference set S^* .

In this research, the following GA parameter settings are used in all of 4 fitness assignment approaches (spEA, nsGA II, rwGA and our i-awGA) for finding the reference solution set S^* : population size, $popSize = 50$; crossover probability, $p_C = 0.90$; mutation probability, $p_M = 0.90$; immigration rate, $p_I = 0.60$; stopping criteria: evaluation of 100000 generations.

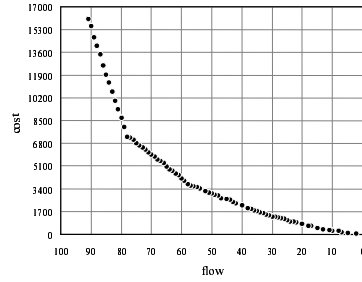
The number of the obtained reference solutions for two test problems is summarized in Table 2.23. We chose nondominated solutions as reference solutions from four solution sets of the four algorithms for each test problem. We show the obtained reference solution sets for the 25-node/49-arc test problem in Fig. 2.60a, 25-node/56-arc test problem in Fig. 2.60b, respectively. We can observe the existence of a clear tradeoff between the two objectives in each Figure. We can also see that the obtained reference solution set for each test problem has a good distribution on the tradeoff front in the objective space.

Table 2.23 Number of obtained reference solutions and the their range width for each objective

| Test Problems (# of nodes / # of arcs) | # of obtained solutions $ S_j $ | range width $W_{f_i}(S^*)$ | |
|--|------------------------------------|----------------------------|----------|
| | | $f_1(r)$ | $f_2(r)$ |
| 25 / 49 | 69 | 85 | 19337 |
| 25 / 56 | 77 | 89 | 16048 |



(a) 25-node/49-arc test problem



(b) 25-node/56 arc test problem

Fig. 2.60 Reference solutions obtained from the four GA approaches

The range width of the i -th objective over the reference solution set S^* is defined as

$$W_{f_i}(S^*) = \max \{f_i(r) | r \in S^*\} - \min \{f_i(r) | r \in S^*\}$$

2.6.4.2 Discussion of the Results

The i-awGA with spEA, nsGA II and rwGA are combined through computational experiments on the 25-node/49-arc and 25-node/56-arc test problems under the same GA parameter settings: population size, $popSize = 20$; crossover probability, $p_C = 0.70$; mutation probability, $p_M = 0.70$; immigration rate, $p_I = 0.30$; stopping condition, evaluation of 5000 solutions. Each simulation was run 50 times. In order to evaluate the results of each test, we consider the average results and standard deviation (SD) for three performance measures ($|S_j|$, $R_{NDS}(S_j)$ and $D1_R(S_j)$), and average of computation times (ACT) in millisecond (ms). Finally we give statistical analysis with the results of $R_{NDS}(S_j)$ and $D1_R(S_j)$ by ANOVA.

As depicted in Figures 2.61 and 2.62, the best results of $|S_j|$, $R_{NDS}(S_j)$ and $D1_R(S_j)$ are obtained by our i-awGA. As described above, the difference is in their fitness calculation mechanisms and the generation update mechanisms. While the spEA and nsGA II used fitness calculation mechanisms based on the Pareto-dominance relation and the concept of crowding, we need a more complex Pareto rank sorting process for solving bND problem; rwGA and i-awGA used a simple

scalar fitness function. Figures 2.61 and 2.62 show the results of comparing computation time; i-awGA and rwGA are always faster than spEA and nsGA II. The selection process of rwGA is simplest; therefore the computation time is also fastest. However for combining weighted-sum based fitness assignment, assigning the effective weight for each objective function is difficult. As a result of computational experiments, the rwGA cannot obtain a better result than the three performance measures.

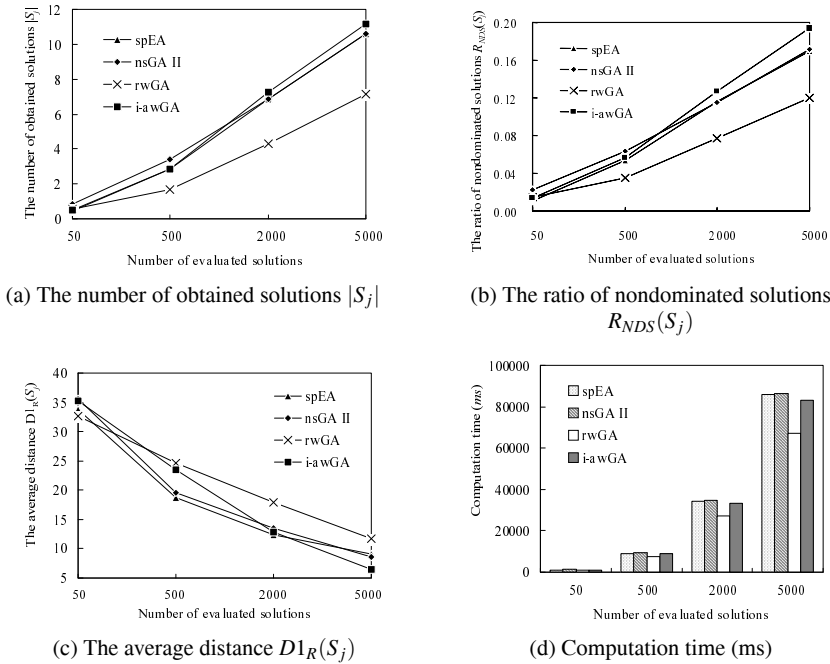


Fig. 2.61 Performance evaluation of fitness assignment approaches using the performance measures for the 25-node/49-arc test problem

In Tables 2.24 and 2.25, ANOVA analysis is used depending on the average of $R_{NDS}(S_j)$ for 50 times with 2 test problems to analyze the difference among the quality of solutions obtained by various 4 kinds of different fitness assignment approaches. At the significant level of $\alpha=0.05$, the $F=21.99$ and 16.59 is greater than the reference value ($F=2.68$), respectively. The difference between our i-awGA and each of the other approaches (spEA, nsGA II or rwGA) is greater than the $LSD=0.019$ and 0.02 , respectively. Similarly with ANOVA analysis depending on the average of $R_{NDS}(S_j)$, as depicted in Tables 2.26 and 2.27, we use ANOVA analysis depended on the average of $D1_R(S_j)$ for 50 times with 2 test problems to analyze the difference among the quality of solutions obtained by four kinds of different fitness assignment approaches. At the significant level of $\alpha=0.05$, the $F=11.22$ and

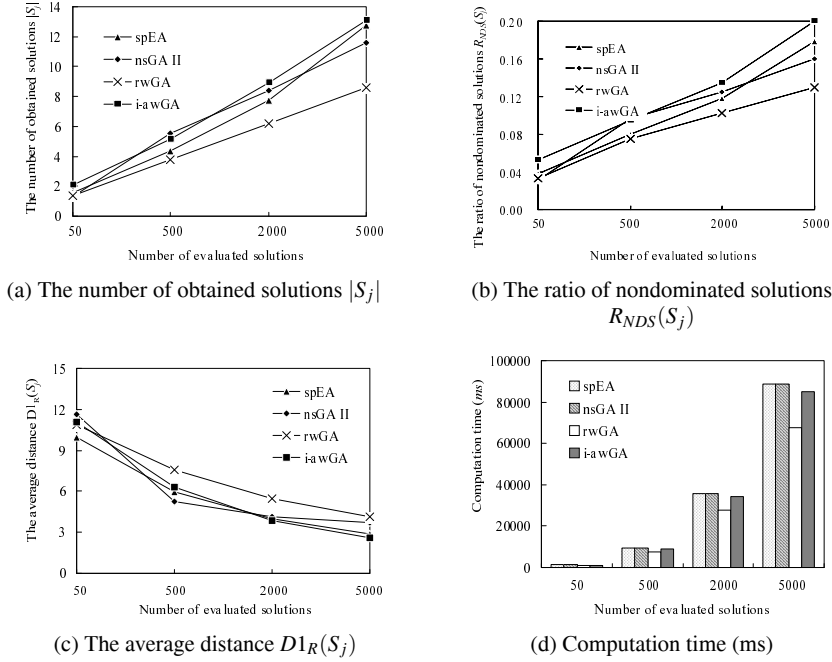


Fig. 2.62 Performance evaluation of fitness assignment approaches using the performance measures for the 25-node/56-arc test problem

8.38 is greater than the reference value of ($F=2.68$), respectively. The difference between i-a wGA and each of the other approaches (spEA, nsGA II or rwGA) is greater than the $LSD=1.18$ and 0.70 , respectively. It is shown that i-a wGA is indeed statistically better than the other approaches.

2.7 Summary

In this chapter, we introduced some of the core models of network design, such as shortest path models (*i.e.*, node selection and sequencing), spanning tree models (*i.e.*, arc selection) and maximum flow models (*i.e.*, arc selection and flow assignment) etc. These network models are used most extensively in applications and differentiated between with their structural characteristics.

To solve the different type models of network design, we introduced the performance analysis of different encoding methods in GAs. Furthermore we gave a priority-based encoding method with special decoding method for different models, and a predecessor-based encoding depended on an underlying random spanning-tree algorithm for different types of minimum spanning tree models. The effectiveness

Table 2.24 ANOVA analysis with the average of $R_{NDS}(S_j)$ in 25-node/49-arc test problem

| | spEA | nsGA II | rwGA | i-awGA |
|-----------------------------|----------------|----------------|-------------|--------|
| # of data | 50 | 50 | 50 | 50 |
| Mean | 0.17 | 0.17 | 0.12 | 0.19 |
| SD | 0.05 | 0.05 | 0.04 | 0.05 |
| Variance | 0.00 | 0.00 | 0.00 | 0.00 |
| Sum of squares | 0.10 | 0.13 | 0.08 | 0.13 |
| Factors | Sum of squares | Freedom degree | Mean square | F |
| Between groups | 0.15 | 3 | 0.05 | 21.99 |
| Within-groups | 0.43 | 196 | 0.00 | |
| Total | 0.58 | 199 | | |
| F ($\alpha = 0.05$) | 2.68 | | | |
| t ($\alpha = 0.05$) | 1.98 | | | |
| LSD | 0.019 | | | |
| Mean Difference with i-awGA | 0.02 | 0.02 | 0.07 | |

Table 2.25 ANOVA analysis with the average of $R_{NDS}(S_j)$ in 25-node/56-arc test problem

| | spEA | nsGA II | rwGA | i-awGA |
|-----------------------------|----------------|----------------|-------------|--------|
| # of data | 50 | 50 | 50 | 50 |
| Mean | 0.18 | 0.16 | 0.13 | 0.20 |
| SD | 0.05 | 0.05 | 0.04 | 0.06 |
| Variance | 0.00 | 0.00 | 0.00 | 0.00 |
| Sum of squares | 0.13 | 0.14 | 0.09 | 0.16 |
| Factors | Sum of squares | Freedom degree | Mean square | F |
| Between groups | 0.13 | 3 | 0.04 | 16.59 |
| Within-groups | 0.52 | 196 | 0.00 | |
| Total | 0.66 | 199 | | |
| F ($\alpha = 0.05$) | 2.68 | | | |
| t ($\alpha = 0.05$) | 1.98 | | | |
| LSD | 0.02 | | | |
| Mean Difference with i-awGA | 0.02 | 0.04 | 0.07 | |

and efficiency of GA approaches were investigated with various scales of network problems by comparing with recent related researches.

Table 2.26 ANOVA analysis with the average of $D1_R(S_j)$ in 25-node/49-arc test problem

| | spEA | nsGA II | rwGA | i-awGA |
|-----------------------------|----------------|----------------|-------------|--------|
| # of data | 50 | 50 | 50 | 50 |
| Mean | 9.03 | 8.59 | 11.67 | 6.48 |
| SD | 5.17 | 4.04 | 4.94 | 3.43 |
| Variance | 26.70 | 16.33 | 24.37 | 11.79 |
| Sum of squares | 1335.01 | 816.55 | 1218.41 | 589.42 |
| Factors | Sum of squares | Freedom degree | Mean square | F |
| Between groups | 679.97 | 3 | 226.66 | 11.22 |
| Within-groups | 3959.39 | 196 | 20.20 | |
| Total | 4639.36 | 199 | | |
| F ($\alpha = 0.05$) | 2.68 | | | |
| t ($\alpha = 0.05$) | 1.98 | | | |
| LSD | 1.78 | | | |
| Mean Difference with i-awGA | 2.55 | 2.10 | 5.18 | |

Table 2.27 ANOVA analysis with the average of $D1_R(S_j)$ in 25-node/56-arc test problem

| | spEA | nsGA II | rwGA | i-awGA |
|-----------------------------|----------------|----------------|-------------|--------|
| # of data | 50 | 50 | 50 | 50 |
| Mean | 2.84 | 3.73 | 4.16 | 2.63 |
| SD | 1.78 | 1.98 | 1.90 | 1.28 |
| Variance | 3.17 | 3.91 | 3.60 | 1.64 |
| Sum of squares | 158.42 | 195.28 | 179.78 | 82.25 |
| Factors | Sum of squares | Freedom degree | Mean square | F |
| Between groups | 79.02 | 3 | 26.34 | 8.38 |
| Within-groups | 615.73 | 196 | 3.14 | |
| Total | 694.75 | 199 | | |
| F ($\alpha = 0.05$) | 2.68 | | | |
| t ($\alpha = 0.05$) | 1.98 | | | |
| LSD | 0.70 | | | |
| Mean Difference with i-awGA | 0.22 | 1.11 | 1.53 | |

References

1. Ahuj, R. K., Magnanti, T. L. & Orlin, J. B. (1993). *Network Flows*, New Jersey: Prentice Hall.
2. Bertsekas, D. P. (1998). *Network Optimization: Continuous and Discrete Models*, Massachusetts: Athena Scientific.
3. Bauer, F.L. & Wossner, H. (1982). *Algorithmic Language and Program Development*, Berlin: Springer.
4. Wikipedia [Online] http://en.wikipedia.org/wiki/Main_Page

5. Cook, S. (1971). The complexity of theorem-proving procedures, *Proceeding of the 3rd Annual ACM Symposium on Theory of Computing*, 151–158.
6. Climaco, J. C. N., Jose M. F. Craveirinha. & Pascoal, M. B. (2003). A bicriterion approach for routing problems in multimedia networks, *Networks*, 41(4), 206–220.
7. Gen, M. & Cheng, R. (1997). *Genetic Algorithms and Engineering Design*. New York, John Wiley & Sons.
8. Gen, M. & Cheng, R. (2000). *Genetic Algorithms and Engineering Optimization*. New York, John Wiley & Sons.
9. Gen, M. & Cheng, R. (2003). Evolutionary network design: hybrid genetic algorithms approach. *International Journal of Computational Intelligence and Applications*, 3(4), 357–380.
10. Munemoto, M., Takai, Y. & Sate, Y. (1997). An adaptive network routing algorithm employing path genetic operators, *Proceeding of the 7th International Conference on Genetic Algorithms*, 643–649.
11. Inagaki, J., Haseyama, M. & Kitajim, H. (1999). A genetic algorithm for determining multiple routes and its applications. *Proceeding of IEEE International Symposium. Circuits and Systems*, 137–140.
12. Ahn, C. W. & Ramakrishna, R. (2002). A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE Transacion on Evolutionary Computation.*, 6(6), 566–579.
13. Gen, M., Cheng, R. & Wand, D. (1997). Genetic algorithms for solving shortest path problems. *Proceeding of IEEE International Conference on Evolutionary Computer*, 401–406.
14. Gen, M. & Lin, L. (2005). Priority-based genetic algorithm for shortest path routing problem in OSPF. *Proceeding of Genetic & Evolutionary Computation Conference*.
15. Bazaraa, M., Jarvis, J. & Sherali, H. (1990). *Linear Programming and Network Flows*, 2nd ed., New York, John Wiley & Sons.
16. Mostafa, M. E. & Eid, S. M. A. (2000). A genetic algorithm for joint optimization of capacity and flow assignment in Packet Switched Networks. *Proceeding of 17th National Radio Science Conference*, C5–1–C5–6.
17. Shimamoto, N., Hiramatsu, A. & Yamasaki, K. (1993). A Dynamic Routing Control Based On a Genetic Algorithm. *Proceeding of IEEE International Conference Neural Networks*, 1123–1128.
18. Michael, C. M., Stewart, C. V. & Kelly, R. B. (1991). Reducing the Search Time of A Steady State Genetic Algorithm Using the Immigration Operator. *Proceeding IEEE International Conference Tools for AI*, 500–501.
19. Goldberg, D. & Lingle, R., Alleles. (1985). logic and the traveling salesman problem. *Proceeding of the 1st Internatinal Conference on GA*, 154–159.
20. Davis, L. (1995). Applying adaptive algorithms to domains. *Proceeding of the International Joint Conference on Artificial Intelligence*, 1, 162–164.
21. Syswerda, G. (1990). Schedule Optimization Using Genetic Algorithms, *Handbook of Genetic Algorithms*, Van Nostran Reinhold, New York.
22. Cheng, R. & Gen, M. (1994). Evolution program for resource constrained project scheduling problem, *Proceedings of IEEE International Conference of Evolutionary Computation*, 736–741.
23. Gen, M., Cheng, R. & Wang, D. (1997). Genetic algorithms for solving shortest path problems, *Proceedings of IEEE International Conference of Evolutionary Computation*, pp. 401–406.
24. Lin, L. & Gen, M. (2007). Bicriteria network design problem using interactive adaptive-weight GA and priority-based encoding method. *IEEE Transactions on Evolutionary Computation in Reviewing*.
25. Lindman, H. R. (1974). *Analysis of variance in complex experimental designs*. San Francisco: W. H. Freeman & Co..
26. Thomas, H., Charles, E. & Rivest, L. (1996) *Introduction to Algorithms*, MIT Press.
27. Gen, M., Cheng, R. & Oren, S. S. (1991). Network design techniques using adapted genetic algorithms. *Advances in Engineering Software*, 32(9), 731–744.

28. Gen, M., Kumar, A. & Kim, R. (2005). Recent network design techniques using evolutionary algorithms. *International Journal Production Economics*, 98(2), 251–261.
29. Gen, M. (2006). *Study on Evolutionary Network Design by Multiobjective Hybrid Genetic Algorithm*. Ph.D dissertation, Kyoto University, Japan.
30. Abuali, F., Wainwright, R. & Schoenefeld, D. (1995). Determinant factorization: a new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. *Proceeding of 6th International Conference Genetic Algorithms*, 470–475.
31. Zhou, G. & Gen, M. (1998). An effective genetic algorithm approach to the quadratic minimum spanning tree problem. *Computers and Operations Research*, 25(3), 229–247.
32. Zhou, G. & Gen, M. (1998). Leaf-constrained spanning tree problem with genetic algorithms approach. *Beijing Mathematics*, 7(2), 50–66.
33. Fernandes, L. M. & Gouveia, L. (1998). Minimal spanning trees with a constraint on the number of leaves. *European Journal of Operational Research*, 104, 250–261.
34. Raidl, G. & Drexe, C. (2000). A predecessor coding in an evolutionary algorithm for the capacitated minimum spanning tree problem. *Proceeding of Genetic and Evolutionary Computation conference*, 309–316.
35. Ahuja, R. & Orlin, J. (2001). Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91, 71–97.
36. Zhou, G. & Gen, M. (1997). Approach to degree-constrained minimum spanning tree problem using genetic algorithm. *Engineering Design and Automation*, 3(2), 157–165.
37. Zhou, G. & Gen, M. (1997). A note on genetic algorithm approach to the degree-constrained spanning tree problems. *Networks*, 30, 105–109.
38. Raidl, G. (2000). A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. *Proceedings Symposium on Applied*, 1, 440–445.
39. Raidl, G. (2000). An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem. *Proceedings Congress on Evolutionary Computation*, 1, 104–111.
40. Knowles, J. & Corne, D. (2000). A new evolutionary approach to the degree-constrained minimum spanning tree problem. *IEEE Transaction Evolutionary Computation*, 4(2), 125–134.
41. Chou, H., Premkumar, G. & Chu, C. (2001). Genetic algorithms for communications network design - an empirical study of the factors that influence performance. *IEEE Transaction Evolutionary Computation*, 5(3), 236–249.
42. Raidl, G. R. & Julstrom, B. A. (2003). Edge Sets: An Effective Evolutionary Coding of Spanning Trees. *IEEE Transaction on Evolutionary Computation*, 7(3), 225–239.
43. Raidl, G. R. & Julstrom, B. (2003). Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. *Proceeding SAC*, 747–752.
44. Julstrom, B. (2003). A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. *Proceeding of Genetic and Evolutionary Computation Conference*, 2–7.
45. Zhou, G. & Gen, M. (1999). Genetic algorithm approach on multi-criteria minimum spanning tree problem. *European Journal of Operational Research*, 114(1), 141–152.
46. Lo, C. & Chang, W. (2000). A multiobjective hybrid genetic algorithm for the capacitated multipoint network design problem. *IEEE Transaction on Systems, Man and Cybernetic*, 30(3).
47. Neumann, F. (2004). Expected runtimes of a simple evolutionary algorithm for the multi-objective minimum spanning tree problem. *Proceeding 8th International Conference of Parallel Problem Solving from Nature*, 80–89.
48. Julstrom, B. & Raidl, G. (2001). Weight-biased edge-crossover in evolutionary algorithms for two graph problems. *Proceeding the 16th ACM Symposium on Applied Computing*, 321–326.
49. Schindler, B., Rothlauf, F. & Pesch, H. (2002). Evolution strategies, network random keys, and the one-max tree problem. *Proceeding Application of Evolutionary Computing on EvoWorkshops*, 143–152.
50. Rothlauf, F., Goldberg, D. & Heinz, A. (2002). Network random keys a tree network representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation*, 10(1), 75–97.

51. Rothlauf, F., Gerstaecker, J. & Heinzl, A. (2003). On the optimal communication spanning tree problem. *IlligAL Technical Report*, University of Illinois.
52. Chankong, V. & Haimes, Y. Y. (1983). *Multiobjective Decision Making Theory and Methodology*. North-Holland, New York.
53. Davis, L., Orvosh, D., Cox, A. & Y. Qiu. (1993). A genetic algorithm for survivable network design. *Proceeding 5th International Conference Genetic Algorithms*, 408–415.
54. Piggott, P. I. & Suraweera, F. (1995). Encoding graphs for genetic algorithms: an investigation using the minimum spanning tree problem. *Progress in Evolutionary Computation.*, X. Yao, Ed. Springer, New York, 956, LNAI, 305–314.
55. Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA J. Computing*, 6(2), 154–160.
56. Cayley, A. (1889). A theorem on tree. *Quarterly Journal of Mathematics & Physical sciences*, 23, 376–378.
57. Prüfer, H. (1918). Neuer beweis eines Satzes über Permutationen. *Archives of Mathematical Physica*, 27, 742–744.
58. Munakata, T. & Hashier, D. J. (1993). A genetic algorithm applied to the maximum flow problem *Proceeding 5th International Conference Genetic Algorithms*, 488–493.
59. Akiyama, T., Kotani, Y. & Suzuki, T. (2000). The optimal transport safety planning with accident estimation process. *Proceeding of the 2nd international Conference on Traffic and Transportation Studies*, 99–106.
60. Ericsson, M., Resende, M.G.C. & Pardalos, P.M. (2001). A genetic algorithm for the weight setting problem in OSPF routing. *Journal of Combinatorial Optimization*, 6:299–333.
61. OR-Library. [Online]. Available: <http://people.brunel.ac.uk/mastjbjb/jeb/info.html>
62. Beasley, J. E., & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, 19, 379–394.
63. Garey, M. R. & Johnson, D. S. *Computers and Intractability: a guide to the theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.
64. Hansen, P. (1979). Bicriterion path problems. *Proceeding 3rd Conference Multiple Criteria Decision Making Theory and Application*, 109–127.
65. Skriver, A. J. V. & Andersen, K. A. (2000). A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research*, 27(6) 507–524.
66. Azaron, A. (2006). Bicriteria shortest path in networks of queues. *Applied Mathematics & Comput*, 182(1) 434–442.
67. Marathe, M. V., Ravi, R., Sundaram R., Ravi S. S., Rosenkrantz, D. J. & Hunt, H. B. (1998). Bicriteria network design problems. *Journal of Algorithms*, 28(1) 142–171.
68. Balint, V. (2003). The non-approximability of bicriteria network design problems. *Journal of Discrete Algorithms*, 1(3,4) 339–355.
69. Lee, H. & Pulat, P. S. (1991). Bicriteria network flow problems: continuous case. *European Journal of Operational Research*, 51(1) 119–126.
70. Yuan, D. (2003). A bicriteria optimization approach for robust OSPF routing. *Proceeding IEEE IP Operations & Management*, 91–98.
71. Yang, H., Maier, M., Reisslein, M. & Carlyle, W. M. (2003). A genetic algorithm-based methodology for optimizing multiservice convergence in a metro WDM network. *J. Light-wave Technol.*, 21(5) 1114–1133.
72. Raghavan, S., Ball, M. O. & Trichur, V. (2002). Bicriteria product design optimization: an efficient solution procedure using AND/OR trees. *Naval Research Logistics*, 49, 574–599.
73. Gen, M., Cheng, R. & Oren, S.S. (2001). Network Design Techniques using Adapted Genetic Algorithms. *Advances in Engineering Software*, 32(9), 731–744.
74. Wu, W. & Ruan, Q. (2004). A gene-constrained genetic algorithm for solving shortest path problem. *Proceeding 7th International Conference Signal Processing*, 3, 2510–2513.
75. Li, Q., Liu, G., Zhang, W., Zhao, C., Yin Y. & Wang, Z. (2006). A specific genetic algorithm for optimum path planning in intelligent transportation system. *Proceeding 6th International Conference ITS Telecom*, 140–143.

76. Kim, S. W., Youn, H. Y., Choi, S. J. & Sung, N. B. (2007). GAPS: The genetic algorithm-based path selection scheme for MPLS network. *Proceeding of IEEE International Conference on Information Reuse & Integration*, 570–575.
77. Hasan, B. S., Khamees, M. A. & Mahmoud, A. S. H. (2007). A heuristic genetic algorithm for the single source shortest path problem. *Proceeding IEEE/ACS International Conference on Computer Systems & Applications*, 187–194. d
78. Ji, Z., Chen, A. & Subprasom, K. (2004). Finding multi-objective paths in stochastic networks: a simulation-based genetic algorithm approach. *Proceedings of IEEE Congress on Evolutionary Computation*, 1, 174–180.
79. Chakraborty, B., Maeda, T. & Chakraborty, G. (2005). Multiobjective route selection for car navigation system using genetic algorithm. *Proceeding of IEEE Systems, Man & Cybernetics Society*, 190–195.
80. Garrozi, C. & Araujo, A. F. R. (2006). Multiobjective genetic algorithm for multicast routing. *Proceeding IEEE Congress on Evolutionary Computation*, 2513–2520.
81. Kleeman, M. P., Lamont, G. B., Hopkinson, K. M. & Graham, S. R. (2007). Solving multicommodity capacitated network design problems using a multiobjective evolutionary algorithm. *Proceeding IEEE Computational Intelligence in Security & Defense Applications*, 33–41.
82. Zhou, G., Min, H. & Gen, M. (2003). A genetic algorithm approach to the bi-criteria allocation of customers to warehouses. *International Journal of Production Economics*, 86, 35–45.



<http://www.springer.com/978-1-84800-180-0>

Network Models and Optimization
Multiobjective Genetic Algorithm Approach
Gen, M.; Cheng, R.; Lin, L.
2008, XIV, 692 p., Hardcover
ISBN: 978-1-84800-180-0