Generalized chromosome genetic algorithm for generalized traveling salesman problems and its applications for machining

Chunguo Wu,[1,2] Yanchun Liang,[1,2,*] Heow Pueh Lee,[2,3] and Chun Lu[2]
[1]*College of Computer Science, Jilin University, Changchun 130012, China*
[2]*Institute of High Performance Computing, Singapore 117528, Singapore*
[3]*Department of Mechanical Engineering, National University of Singapore, 9 Engineering Drive 1, Singapore 119260, Singapore*
(Received 28 November 2003; published 1 July 2004)

Traveling salesman problems (TSP) and generalized traveling salesman problems (GTSP) are two kinds of well known and challenging combinatorial optimization problems with much diversified application fields. Between the two application problems the GTSP is more complex than TSP. Many researchers have studied TSP extensively, but relatively fewer studies pay attention to GTSP, and also its solution using genetic algorithm (GA). In this paper, the structure of conventional chromosome is generalized to be a chromosome termed as a generalized chromosome (GC). A genetic scheme named as generalized-chromosome-based genetic algorithm (GCGA) is also presented. The proposed GCGA enables GTSP and TSP to be solved under a uniform algorithm mode. Forty one benchmark test problems have been solved with the known optimal solutions using the proposed algorithm to verify its validity. The test results show that GCGA can directly solve GTSP without the need of intermediate transformation to TSP.

DOI: 10.1103/PhysRevE.70.016701 PACS number(s): 02.70.Rr, 02.70.Hm, 05.10.−a, 05.65.+b

I. INTRODUCTION

The generalized traveling salesman problem (GTSP) represents a kind of combinatorial optimization problem, which has been introduced by Henry-Labordere [1], Saksena [2], and Srivastava [3] in the context of computer record balancing and of visit sequencing through welfare agencies since 1960s. The GTSP can be described as the problem of seeking a special Hamiltonian cycle with lowest cost in a complete weighted graph. Let $\mathbf{G}=(\mathbf{V},\mathbf{E},\mathbf{W})$ be a complete weighted graph where $\mathbf{V}=\{v_1,v_2,\ldots,v_n\}$ $(n\geqslant 3)$, $\mathbf{E}=\{e_{ij}|v_i,v_j\in\mathbf{V}\}$ and $\mathbf{W}=\{w_{ij}|w_{ij}\geqslant 0$ *and* $w_{ii}=0,\forall i,j\in\mathbf{N}(n)\}$ are vertex set, edge set, and cost set, respectively. And v_i, e_{ij}, w_{ij} are the ith vertex, the edge connecting vertices v_i and v_j, and the cost/weight corresponding to edge e_{ij}, respectively. $\mathbf{N}(n)$ is the subset $\{1,2,\ldots,n\}$ of the natural number set. The vertex set $\mathbf{V}$ is partitioned into m possibly intersecting groups $\mathbf{V}_1,\mathbf{V}_2,\ldots,\mathbf{V}_m$ with $|\mathbf{V}_j|\geqslant 1$ and $\mathbf{V}=\cup_{j=1}^{m}\mathbf{V}_j$, where $|\ldots|$ is the element number of a limited set. The special Hamiltonian cycle is required to pass through all of the groups, but not all of the vertices differing from that of TSP. For convenience, we also call $\mathbf{W}$ as the cost matrix and take it as $\mathbf{W}=(w_{ij})_{n\times n}$. There are two different kinds of GTSP under the abovementioned framework of the special Hamiltonian cycle [4,5]: (1) the cycle passes exactly one vertex in each group (refer to Fig. 1) and (2) the cycle passes at least one vertex in each group (refer to Fig. 2). The first kind of GTSP is also known as E-GTSP, where E stands for equality [5]. In this paper we only discuss the GTSP for the first case and will still call it as GTSP for convenience.

GTSP has extensive application fields. Laport *et al.* [4], Lien *et al.* [6], and Castelino *et al.* [7] reported the applications of GTSP. Just as mentioned in Ref. [6], "for many real-world problems that are inherently hierarchical, the GTSP offers a more accurate model than the TSP." Generally, GTSP provides a more ideal modeling tool for many real problems. Furthermore, GTSP can include the grouped and isolated vertices at the same time according to our present extension. Therefore, GTSP includes TSP theoretically (see Fig. 3) and application fields of GTSP are wider than those of TSP.

Although since late 1960s GTSP has been proposed [1–3], the related reported works are very limited compared with those on TSP [8–11] and the existing algorithms for GTSP are mainly based on dynamic programming techniques [1–3,5,12,13]. However, because of its NP-hard quality, only a few solutions of modest-size problems are supported by the current hardware technology and most of them fail to obtain the results due to the huge memory required in dynamic programming algorithms and the problem of lengthy computational time. The main methodology of the dynamic programming algorithms is to transform the GTSP into TSP and then to solve the TSP using existing algorithms [5,13–15].

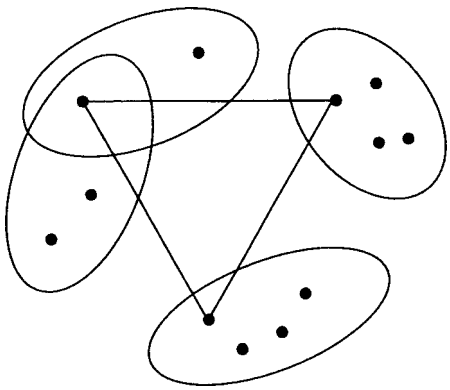

FIG. 1. Exactly one vertex is visited in a GTSP cycle.

*Corresponding author. Electronic address: liangyc@ihpc.a-star.edu.sg

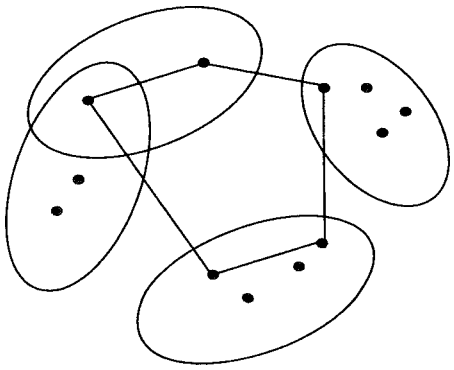

FIG. 2. At least one vertex is visited in a GTSP cycle.

The shortcomings of these methods are that the transformation increases the problem dimension dramatically and in some cases the dimension would expand up to more than three times of the original [6,16–18]. Therefore, although theoretically the GTSP could be solved using the corresponding transformed TSP, the technological limitation ruins its practical feasibility. Some studies have been performed to discuss and solve the problem [19–21].

Genetic algorithm (GA) is one of the powerful tools to deal with NP-hard combinatorial optimization problems and has been widely applied for finding the solution of TSP due to its high efficiency and strong searching ability. However, theoretical and application studies related to using GA methods to solve GTSP are very few. The authors of Ref. [19] proposed a hybrid GTSP solving algorithm based on random-key GA [22] and local search method [23] (HRKGA). However, we have noted that the intensive local search schemes have resulted in lengthy computation, which may hamper the method to be used to handle large scale problems. To explore the application of GA on GTSP solution, especially, to enable GA-based methods to deal with some problems arising from industrial applications, this paper extends the structure of the conventional chromosome used in GA and proposes a GA based on the extended chromosome. We name the chromosome as a generalized chromosome (GC), and the proposed GA as generalized-chromosome-based GA (GCGA). The advantages of the GCGA are that it does not require the transformation from GTSP to TSP and remove the limitation of triangle inequality of the cost matrix, which enables the GCGA to be able to run with high efficiency. Furthermore, the GCGA provides a consistent mode to solve GTSP, TSP, and even GTSP-TSP hybrid problems within GA framework.

II. METHODOLOGY

The commonly used decimal integral encoding chromosome in the GA for solving TSP has quite a few advantages such as intuition, easy understanding, convenient operating, and high efficiency, which promote the wide applications of GA to the TSP. However, it looses its predomination when it is used for the GTSP. Because the vertices need to be grouped and usually a group contains more than one vertex, the chromosome must contain information to decide which vertex is included in the current tour in each group besides the visiting sequence. The proposed GCGA with generalized chromosome can be used to deal with this request easily.

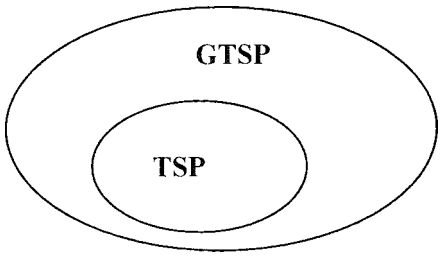

FIG. 3. Relationship between GTSP and TSP.

We present the designed GC in Sec. II A, and then propose the GCGA in Sec. II B.

A. Generalized chromosome

Some definitions used in this paper are presented as follows.

Definition 1. A vertex in graph $\mathbf{G}=(\mathbf{V},\mathbf{E},\mathbf{W})$ is said to be an original vertex if it is denoted in the form of v_i $[i \in \mathbf{N}(n)]$.

Definition 2. A group $\mathbf{V}_j$ $[j \in \mathbf{N}(m)]$ is said to be a super vertex if $|\mathbf{V}_j|>1$.

Definition 3. An original vertex v_i $[i \in \mathbf{N}(n)]$ is said to be a scattering vertex if $v_i \in \mathbf{V}_j$ and $|\mathbf{V}_j|=1$ $[j \in \mathbf{N}(m)]$.

Definition 4. An original vertex v_i $[i \in \mathbf{N}(n)]$ is said to be an element of certain super vertex if it belongs to the super vertex.

Definition 5. Either a super vertex or scattering vertex is said to be a generalized vertex if dealing with it in a uniform way is needed.

Definition 6. A cycle is said to be a GTSP cycle if it contains all of the generalized vertices.

Definition 7. A GTSP cycle is said to be a valid GTSP cycle if it satisfies passing each original vertex at most one time.

Definition 8. A GTSP cycle is said to be an invalid GTSP cycle if it is not a valid one.

To facilitate the discussion, the groups are further distinguished by their vertex numbers according to definitions 2 and 3 in this paper. Then the group number m can be decomposed into two parts, i.e.,

$$m = \hat{m} + \tilde{m}, \tag{1}$$

where $\hat{m}$ is the number of the super vertices, $\tilde{m}$ the number of the scattering vertices. We refer to a GTSP with $\hat{m} \neq 0$ and $\tilde{m} \neq 0$ as a GTSP-TSP hybrid problem. Denote all of the super vertices as

$$u_1, u_2, \ldots, u_{\hat{m}}, \tag{1a}$$

where

$$u_i = \{u_{i,1}, u_{i,2}, \ldots, u_{i,k_i}\}, \tag{1b}$$

k_i $[i \in \mathbf{N}(\hat{m})]$ represents the number of elements contained in the ith super vertex, $u_{i,l}$ $[l \in \mathbf{N}(k_i)]$ the lth element belonging

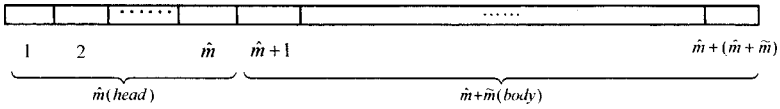

FIG. 4. Mode of the generalized chromosome.

to the super vertex u_i, l the index of $u_{i,l}$ in the super vertex u_i. Denote all of the scattering vertices as

$$\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_{\tilde{m}}. \tag{1c}$$

The super vertices and scattering vertices are sequenced as follows:

$$u_1, u_2, \ldots, u_{\hat{m}}, \tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_{\tilde{m}}. \tag{1d}$$

The corresponding generalized vertices are denoted as

$$w_1, w_2, \ldots, w_{\hat{m}+\tilde{m}}. \tag{1e}$$

The relationship among the generalized vertices, super vertices and scattering vertices are as follows

$$w_k = \begin{cases} u_k & 1 \leqslant k \leqslant \hat{m} \\ \tilde{u}_{k-\hat{m}} & \hat{m}+1 \leqslant k \leqslant \hat{m}+\tilde{m}, \quad k \in \mathbf{N}(\hat{m}+\tilde{m}). \end{cases} \tag{2}$$

Therefore, we also refer to the super vertex u_i $(i=1,2,\ldots,\hat{m})$ as the ith generalized vertex, and scattering vertex $\tilde{u}_i$ $(i=1,2,\ldots,\tilde{m})$ as the $(i+\hat{m})$th generalized vertex. Obviously, the number of generalized vertices is $\hat{m}+\tilde{m}$, i.e., the original number of the groups. Notice that an original vertex may belong to more than one super vertex at the same time, hence, the following inequality holds

$$n \leqslant \tilde{m} + \sum_{i=1}^{\hat{m}} k_i. \tag{3}$$

The mode of the designed GC is shown in Fig. 4. The GC consists of two parts. The left part contains $\hat{m}$ genes and is named as head; the right part contains $\hat{m}+\tilde{m}$ genes and is named as body. When $0<i\leqslant\hat{m}$, the ith gene lies in the head part, which stores the index l $[l \in \mathbf{N}(k_i)]$ of the element visited by the current tour in the super vertex u_i. When $\hat{m}<i \leqslant \hat{m}+(\hat{m}+\tilde{m})$, the ith gene lies in the body part, which stores the index k $[k \in \mathbf{N}(\hat{m}+\tilde{m})]$ of the generalized vertex. From Eq. (2), it follows that if i satisfies

$$i - \hat{m} \leqslant \hat{m},$$

then the gene represents the super vertex $u_{i-\hat{m}}$ and if i satisfies

$$i - \hat{m} > \hat{m},$$

then the gene represents the scattering vertex $\tilde{u}_{(i-\tilde{m})-\tilde{m}}$.

In the decoding process, the body part defines a GTSP cycle and the head part is used to determine the visited vertex in each super vertex. If we denote

$$\mathbf{H} = \{h | h=[h(1), h(2), \ldots, h(\hat{m})],\ h(i) \in \mathbf{N}(k_i), \quad i \in \mathbf{N}(\hat{m})\}, \tag{4}$$

where $[\ldots]$ is a limited sequence, $h(i)$ represents an element index belonging to the super vertex u_i, then $\mathbf{H}$ is the set consisting of all of the feasible gene segments lying in the head part. And if we denote

$$\mathbf{B} = \{b | b = \Pi(m)\}, \tag{5}$$

where $b=\Pi(m)=[b_1, b_2, \ldots, b_m]$ is a permutation of $1,2,\ldots,m$, then b represents a GTSP cycle. Therefore $\mathbf{B}$ is the set consisting of all feasible gene segments lying in the body part. Then we can denote the set $\mathbf{D}$ consisting of all of the GCs corresponding to feasible GTSP cycles as

$$\mathbf{D} = \{x | x = h \oplus b,\ h \in \mathbf{H},\ b \in \mathbf{B}\}, \tag{6}$$

where $x=h\oplus b$ represents a GC compounded of two gene segments belonging to $\mathbf{H}$ and $\mathbf{B}$, respectively.

As an illustration, let us consider a vertex set $\mathbf{V}=\{1,2,\ldots,20\}$. Assume that parts of its vertices are partitioned into three groups as follows:

$$\mathbf{V}_1 = \{1,2,3,4,12,13,15,17\}, \tag{7}$$

$$\mathbf{V}_2 = \{3,4,6,7,16,19\}, \tag{8}$$

$$\mathbf{V}_3 = \{8,9,10,11\}, \tag{9}$$

and the scattering vertices are 5, 14, 18, 20 (Fig. 5). Then we have

$$\begin{gathered} u_1 = \{u_{1,1}, u_{1,2}, u_{13}, u_{1,4}, u_{1,5}, u_{1,6}, u_{1,7}, u_{1,8}\} \\ u_2 = \{u_{2,1}, u_{2,2}, u_{2,3}, u_{2,4}, u_{2,5}, u_{2,6}\} \\ u_3 = \{u_{3,1}, u_{3,2}, u_{3,3}, u_{3,4}\} \\ \tilde{u}_1 = 5, \tilde{u}_2 = 14, \tilde{u}_3 = 18, \tilde{u}_4 = 20, \end{gathered} \tag{10}$$

where

$$u_{1,1} = 1, u_{1,2} = 2, u_{13} = 3, u_{1,4} = 4, u_{1,5} = 12, u_{1,6} = 13, u_{1,7} = 15, u_{1,8} = 17$$

$$u_{2,1} = 3, u_{2,2} = 4, u_{2,3} = 5, u_{2,4} = 6, u_{2,5} = 16, u_{2,6} = 19$$

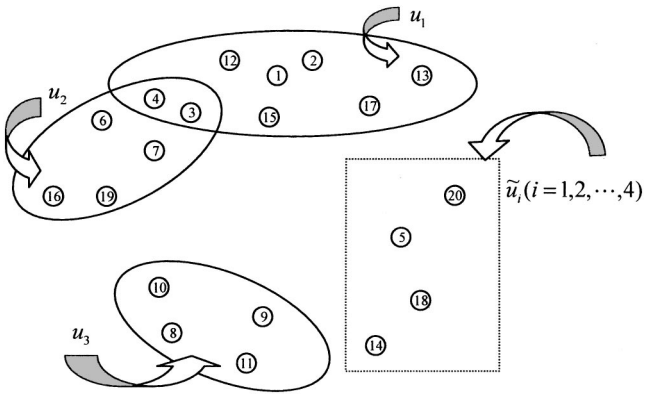

FIG. 5. Super and scattering vertices.

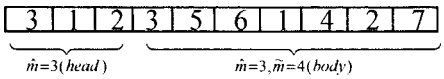

FIG. 6. Generalized chromosome of the first example.

$$u_{3,1} = 8, u_{3,2} = 9, u_{3,3} = 10, u_{3,4} = 11$$

$$\tilde{u}_1 = 5, \tilde{u}_2 = 14, \tilde{u}_3 = 18, \tilde{u}_4 = 20.$$

The original vertices can be sorted randomly in each super vertex. An increasing sequence is recommended here. Similarly, the scattering vertices are sorted in the same way. In the example, $\hat{m}$ and $\tilde{m}$ are taken as 3 and 4, respectively, which means that the head part has three ($\hat{m}=3$) genes and the body part has seven ($\hat{m}+\tilde{m}=7$) genes in a GC. To illustrate the GC more clearly, we design two GC instances. Let

$$h_1 = h_2 = [3,1,2], \tag{11}$$

$$b_1 = [3,5,6,1,4,2,7], \tag{12}$$

$$b_2 = [3,5,6,4,1,2,7], \tag{13}$$

then we can obtain two GCs as follows:

$$x_1 = h_1 \oplus b_1, \tag{14}$$

$$x_2 = h_2 \oplus b_2. \tag{15}$$

Their structures are shown in Figs. 6 and 7, and the corresponding GTSP cycles are shown in Figs. 8 and 9. We take x_1 as an example to interpret the decoding process of the GC. There are three genes 3, 1, and 2 lying in the head of the GC, which represent that the vertex $u_{1,3}$ in super vertex u_1, $u_{2,1}$ in super vertex u_2, and $u_{3,2}$ in super vertex u_3 are visited in the GCGA cycle. There are seven genes 3, 5, 6, 1, 4, 2, and 7 lying in the body of the GC, which determine the visited sequence of the seven generalized vertices. The tour corresponding to x_1 is

$$\text{a1:}w_3 \to w_5 \to w_6 \to w_1 \to w_4 \to w_2 \to w_7 \to w_3,$$

that is,

$$\text{a2:}u_3 \to \tilde{u}_2 \to \tilde{u}_3 \to u_1 \to \tilde{u}_1 \to u_2 \to \tilde{u}_4 \to u_3.$$

If the index of a generalized vertex is not larger than $\hat{m}$, it is a super vertex; if its index is larger than $\hat{m}$, it is a scattering vertex. Hence, according to the above decoding rules, the GC x_1 stands for the tour

$$\text{a3:}u_{3,2} \to \tilde{u}_2 \to \tilde{u}_3 \to u_{1,3} \to \tilde{u}_1 \to u_{2,1} \to \tilde{u}_4 \to u_{3,2},$$

whose original-vertex form is

$$\text{a4:}v_9 \to v_{14} \to v_{18} \to v_3 \to v_5 \to v_3 \to v_{20} \to v_9,$$

i.e.,

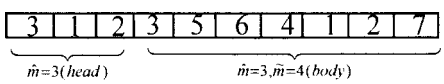

FIG. 7. Generalized chromosome of the second example.

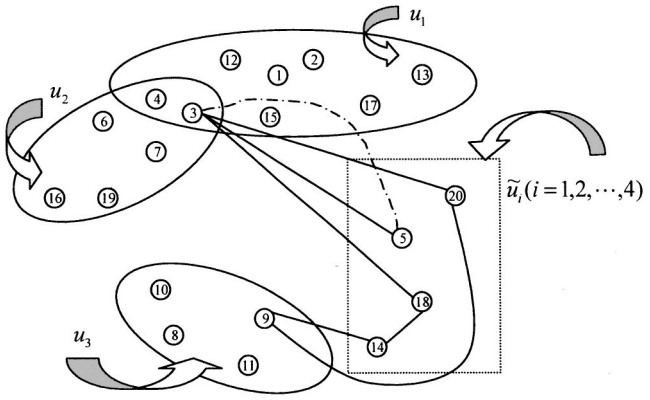

FIG. 8. GTSP cycle corresponding to the first example.

$$\text{a5:}9 \to 14 \to 18 \to 3 \to 5 \to 3 \to 20 \to 9.$$

Similarly, according to the above decoding rules, five different forms of the GTSP cycles corresponding to GC x_2 are, respectively,

$$\text{b1:}w_3 \to w_5 \to w_6 \to w_4 \to w_1 \to w_2 \to w_7 \to w_3,$$

$$\text{b2:}u_3 \to \tilde{u}_2 \to \tilde{u}_3 \to \tilde{u}_1 \to u_1 \to u_2 \to \tilde{u}_4 \to u_3,$$

$$\text{b3:}u_{3,2} \to \tilde{u}_2 \to \tilde{u}_3 \to \tilde{u}_1 \to u_{1,3} \to u_{2,1} \to \tilde{u}_4 \to u_{3,2},$$

$$\text{b4:}v_9 \to v_{14} \to v_{18} \to v_5 \to v_3 \to v_3 \to v_{20} \to v_9,$$

$$\text{b5:}9 \to 14 \to 18 \to 5 \to 3 \to 3 \to 20 \to 9.$$

It can be seen that there is a subtour containing v_3 and v_5 in both tours a4 and a5, which has been marked in dotted line in Fig. 8, and that there is an edge connecting itself on vertex v_3 in both b4 and b5. The vertex v_3 is visited twice of the two cases mentioned above. The edge with the same starting and ending vertices is referred to as a ring in the graph theory. The dashed line in Fig. 9 shows a ring for vertex v_3. These cases both result from the overlap between the super vertices u_1 and u_2. From the definition of the cost matrix it can be seen that the ring corresponds to a zero cost, consequently, the cost of x_1 is higher than that of x_2. If all of the rings in a GC are removed in the decoding process, a valid GTSP cycle

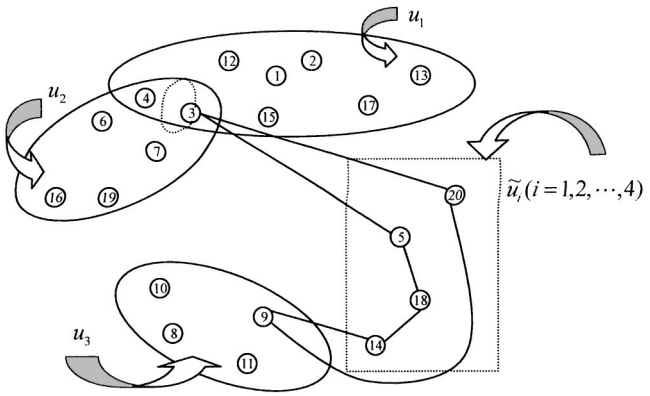

FIG. 9. GTSP cycle corresponding to the second example.

will be obtained. Although these cases are invalid in TSP, they are valid in GTSP, such as a1, a2, a3 and b1, b2, b3. Hence, we permit their existence in the population, but not rectify them into absolutely legal ones in each generation. In fact, because the cost of x_1 is larger than that of x_2, the fitness value of x_1 must be lower than that of x_2. The generation renewal will remove this kind of GCs such as x_1 with relatively lower fitness. Moreover, those GCs such as x_2 are valid GTSP cycles as long as all of their rings are removed at the last generation. Hence, the proposed GCGA omits the rectification during the intermediate generations, which reduces the work on algorithm coding, increases the computing efficiency, keeps the diversity of gene segments and promotes the generation of excellent chromosomes.

B. Generalized-chromosome-based genetic algorithm

The special form of the designed chromosome makes the standard genetic operations not available, therefore, a set of genetic operations is proposed in this paper. The proposed algorithm is named as generalized-chromosome-based genetic algorithm (GCGA). To make the GCGA to be understood easily by researchers acquainted with standard GA, the work flow of the conventional algorithm is adopted [24], which can be described as follows.

(a) Initialize population: Utilize the initializing operator on the head set and the body set, and denote the size of the population as S_p.

(b) Select reproductive population: Select the individual repeatedly from the old population according to "Survival of the fittest" and put the selected individual into a temporary population, named reproductive population, till its size equals to S_p.

(c) Crossover process: Select two individuals randomly in the reproductive population and generate a random number in $[0,1]$, denote it as ρ. If ρ is smaller than or equal to the crossover probability, utilize the crossover operator on the two selected individuals to get two offspring individuals, if ρ is larger than the crossover probability, take the two unchanged selected individuals as offspring individuals.

(d) Mutation process: Generate a random number in $[0,1]$ and denote it as η. If η is smaller than or equal to the mutation probability, utilize the mutation operator on the individuals going through the crossover process. If η is larger than the mutation probability, keep the individuals unchanged.

(e) Inversion process: Utilize the inversion operator on the individuals going through the mutation process.

(f) Denote the current size of the new population as S_c. If S_c is smaller than S_p, return to step c; else copy the new population into the old one.

(g) Denote the current generation number as k and the maximal generation number as K. If k is smaller than K, return to step B; else output the results including the individuals with respect to the maximal, average and minimal fitness, and then stop.

The flow chart of the proposed GCGA algorithm is shown in Fig. 10 and the work behaviors of its key genetic operators are explained as follows.

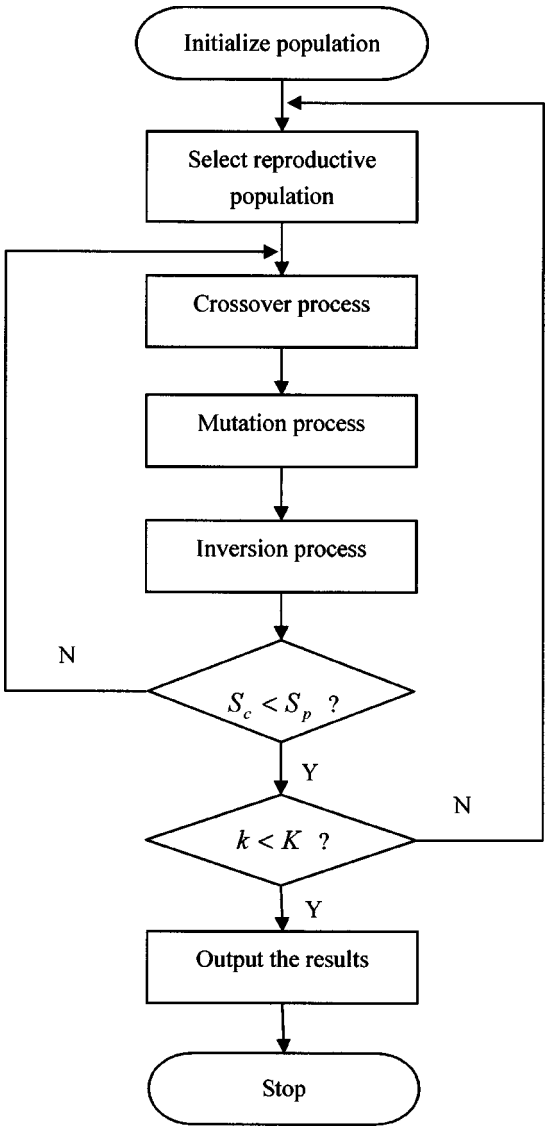

FIG. 10. Flow chart of the GCGA algorithm.

1. Initializing operator **P**

Initializing operator **P** is used to generate an initial population. It is a two-element random operator. Its two variables are **H** and **B**, and its result is a subset of **D**. Denoting P as a population, then the initialization of P can be represented as

$$P = \mathbf{P}_N(\mathbf{H},\mathbf{B}), \tag{16}$$

where $\mathbf{P}_N$ is an operator to generate an initial population with size N. Its flow chart is shown in Fig. 11.

2. Generalized crossover operator **C**

To implement the crossover operation and generate new chromosomes, a generalized crossover operator is defined as

$$\mathbf{C}:\mathbf{D}\times\mathbf{D}\rightarrow\mathbf{D}\times\mathbf{D}. \tag{17}$$

It is a two-element random operator. Its variables are the elements of **D**. If $x,y\in\mathbf{D}$, then **C** generates a pair of new GCs as

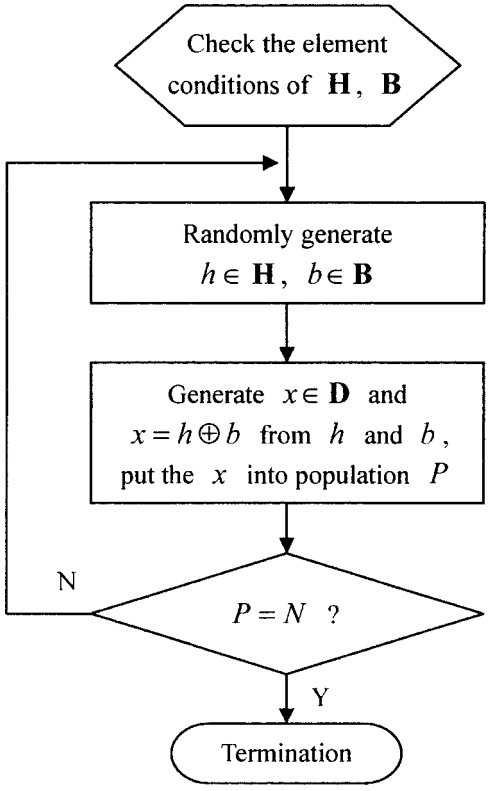

FIG. 11. Flow chart of the initializing operator.

$$(x',y') = \mathbf{C}(x,y), \tag{18}$$

where x',y' are the two offspring chromosomes generated by operator **C** at the same time based on the parental chromosomes x and y. Generally, if

$$(x_1',y_1') = \mathbf{C}(x,y), \tag{19}$$

$$(x_2',y_2') = \mathbf{C}(x,y) \tag{20}$$

are generated at different time, the following two equations

$$x_1' = x_2', \tag{21}$$

$$y_1' = y_2', \tag{22}$$

do not necessarily hold, because the generalized crossover operator **C** is a random operator. Equations (21) and (22) hold if and only if the two parental chromosomes and the crossover segment positions are exactly the same.

Let $x = h_x \oplus b_x$ and $y = h_y \oplus b_y$, then the flow chart of the generalized crossover operator **C** is shown in Fig. 12. The behavior of the operator is somewhat similar to the two-point crossover in the standard GA. Let the two crossover points selected randomly be i_1 and i_2 (assume $i_1 < i_2$), where $i_1 = \text{random}(\hat{m} + (\hat{m} + \tilde{m}))$, $i_2 = \text{random}(\hat{m} + (\hat{m} + \tilde{m}))$, and random$(n)$ is a generator of random numbers with uniform distribution within $\mathbf{N}(n)$.

If $i_1 > \hat{m}$ then the crossover takes place in the body parts. In this case, the effect of operator **C** is equal to the conventional crossover in some extent, because the body parts of GC are equivalent to two normal chromosomes b_x and b_y with length of $\hat{m} + \tilde{m}$. But because the head parts are omitted, the crossover points should be translated as

$$i_1' \Leftarrow i_1 - \hat{m}, \tag{22a}$$

$$i_2' \Leftarrow i_2 - \hat{m}. \tag{22b}$$

Denote the crossover on two normal chromosomes as

$$b_x \otimes b_y \rightarrow (b_x', b_y'), \tag{22c}$$

where $b_x', b_y' \in \mathbf{B}$. Then combining b_x' and b_y' with the original head parts of x and y, respectively, offspring GCs corresponding to the GTSP cycles can be obtained

$$x' = h_x \oplus b_x', \tag{23}$$

$$y' = h_y \oplus b_y'. \tag{24}$$

If $i_2 \leqslant \hat{m}$, then the crossover takes place in the head parts. In this case, it is only needed to exchange the genes within the crossover segments. The process can be denoted as

$$h_x \otimes h_y \rightarrow (h_x', h_y'), \tag{24a}$$

where $h_x', h_y' \in \mathbf{H}$. Then combining h_x' and h_y' with the original body parts of x and y, offspring GCs corresponding to the GTSP cycles can be generated as

$$x' = h_x' \oplus b_x, \tag{25}$$

$$y' = h_y' \oplus b_y. \tag{26}$$

If $i_1 \leqslant \hat{m}$ and $i_2 > \hat{m}$, then the generalized crossover can be treated as the combination of the above cases. One crossover operation takes place on the gene segment from $(\hat{m}+1)$ to i_2, another from i_1 to $\hat{m}$. The two crossovers can be implemented asynchronously in two steps. Combining the new head and body parts obtained in the two steps, we can have the offspring GCs corresponding to the GTSP cycles

$$x' = h_x' \oplus b_x', \tag{27}$$

$$y' = h_y' \oplus b_y'. \tag{28}$$

3. Generalized mutation operator **M**

To increase the diversity of the gene segments, the generalized mutation operator **M** is designed based on the insertion mutation used in standard GA

$$\mathbf{M}:\mathbf{D} \rightarrow \mathbf{D}. \tag{29}$$

It is a one-element random operator, and its variable is an element belonging to GC set **D**. If $x \in \mathbf{D}$, then under the effect of generalized mutation operator **M** it will result in a new GC

$$x' = \mathbf{M}(x). \tag{30}$$

Similarly to the generalized crossover operator **C**, if

$$x_1' = \mathbf{M}(x) \quad \text{and} \tag{31}$$

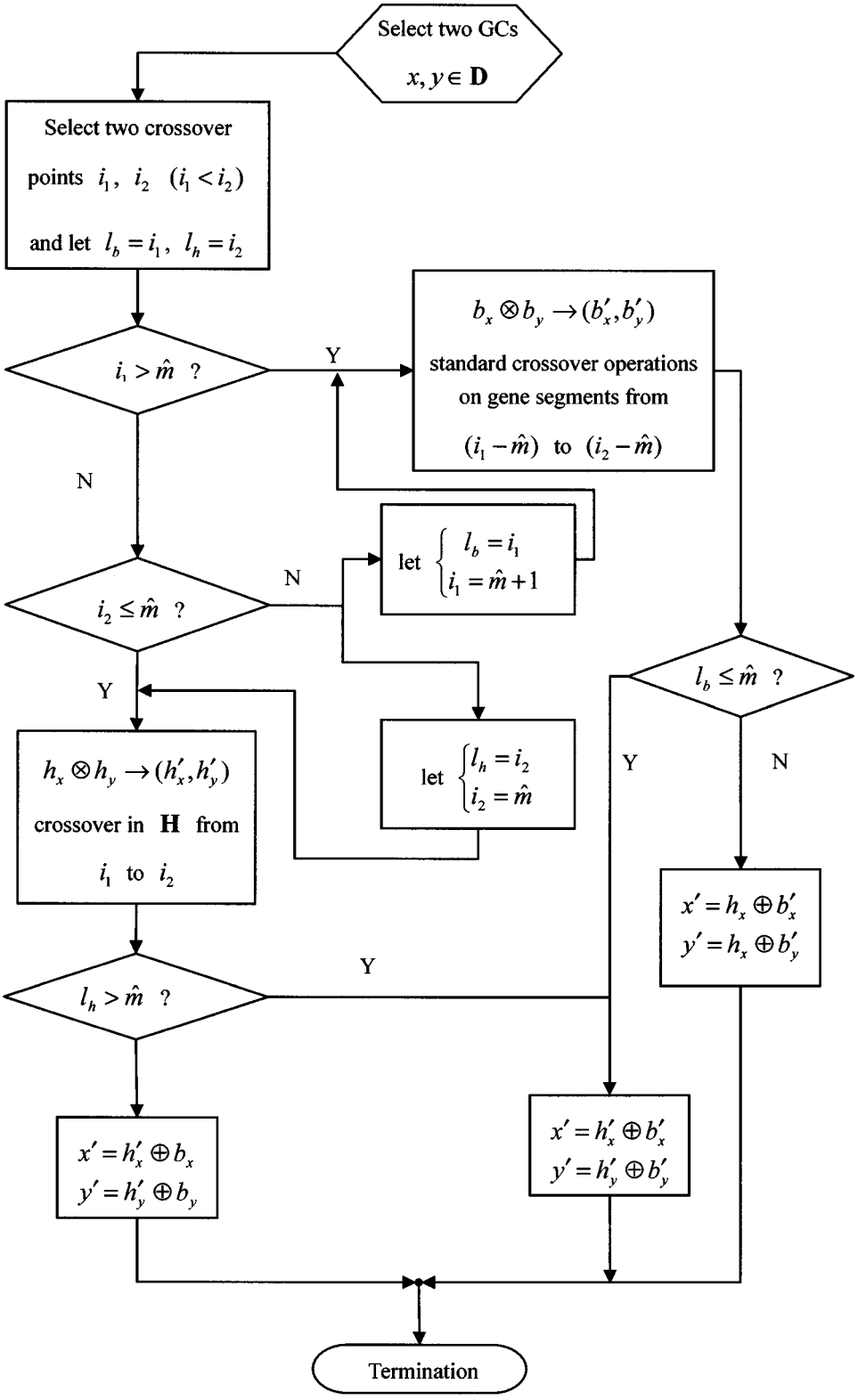

FIG. 12. Flow chart of the generalized crossover operator.

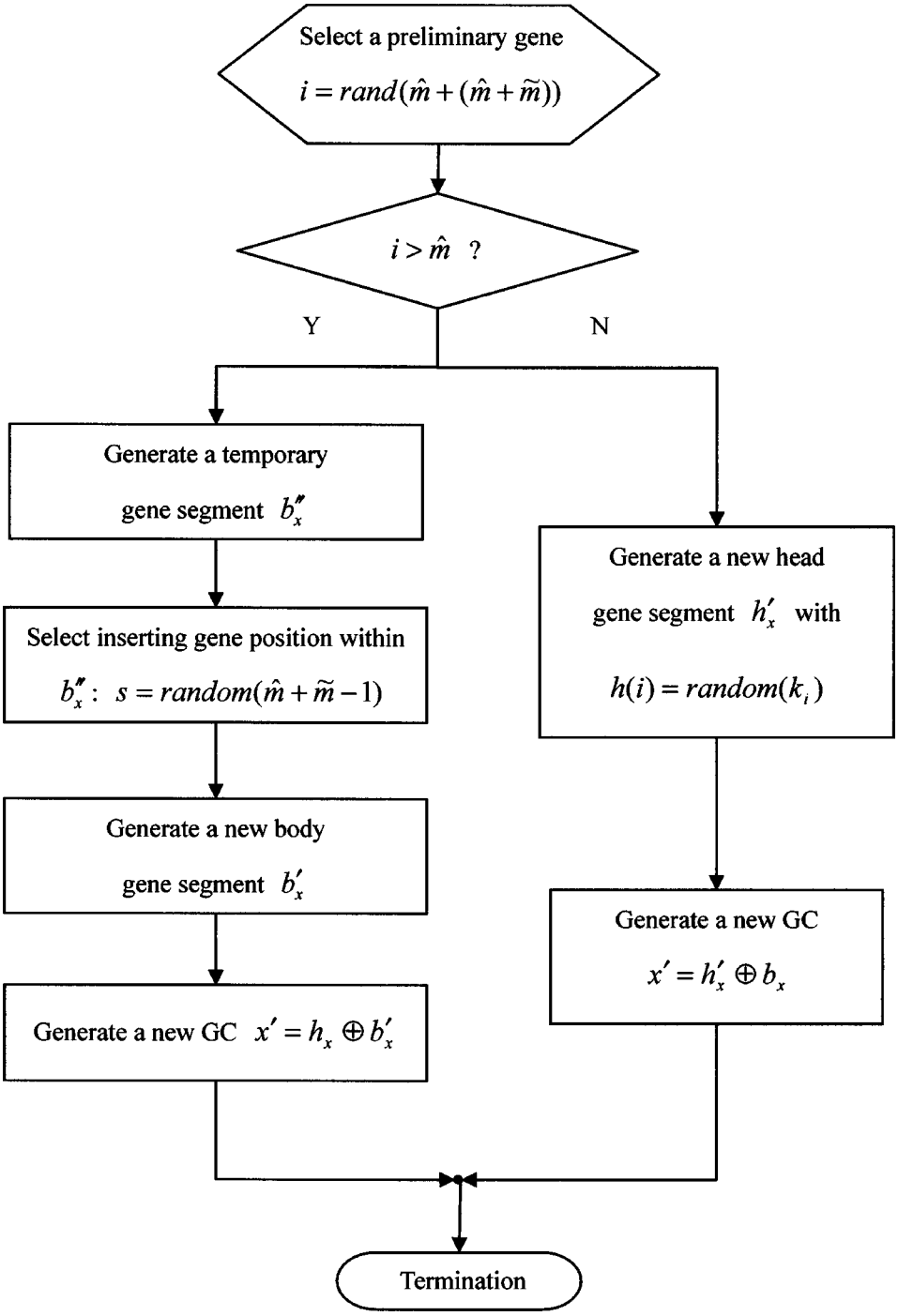

FIG. 13. Flow chart of the generalized mutation operator.

$$x_2' = \mathbf{M}(x) \tag{32}$$

are two new GCs generated by **M** based on x, generally,

$$x_1' = x_2', \tag{33}$$

does not necessarily hold. Equation (33) holds if and only if the preliminary and insertion genes are exactly identical, respectively.

Let $x = h_x \oplus b_x$ and

$$h_x = [h(1), h(2), \ldots, h(\hat{m})], \tag{34}$$

$$b_x = [b(1), b(2), \ldots, b(\hat{m} + \tilde{m})]. \tag{35}$$

The flow chart of operator **M** is shown in Fig. 13. A preliminary gene i is randomly selected, $i = \text{random}(\hat{m} + (\hat{m} + \tilde{m}))$, which is taken as the gene to be inserted.

The difference between GCGA and standard GA is that if $i > \hat{m}$ then the preliminary gene lies in the body part and it can be determined that the preliminary gene is the $(i - \hat{m})$th component $b(i - \hat{m})$ of b_x. Removing the preliminary gene from b_x and forwardly moving genes behind $b(i - \hat{m})$ one bit sequentially, a temporary gene segment can be generated as follows

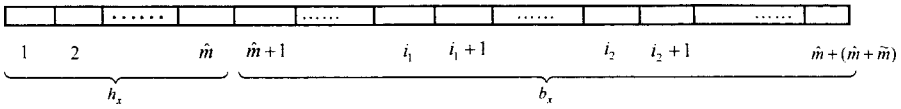

FIG. 14. Gene selection of the generalized reversion operator.

$$b_x'' = [b(1), b(2), \ldots, b(i-\hat{m}-1), b(i-\hat{m}+1), \ldots, b(\hat{m}+\tilde{m})]. \tag{36}$$

Then an inserted gene position s on b_x'' with length of $\hat{m}+\tilde{m}-1$ is randomly selected as

$$s = \text{random}(\hat{m}+\tilde{m}-1). \tag{37}$$

After getting the inserting gene position, we use the generalized mutation operator to move the genes behind $b(s)$ backwardly and then copy the preliminary gene into the component $b(s+1)$. The mutation process can be denoted as

$$x' = \mathbf{M}(x) = h_x \oplus b_x', \tag{38}$$

where

$$b_x' = [b(1), b(2), \ldots, b(i-\hat{m}-1), b(i-\hat{m}+1) \ldots, b(s), b(i-\hat{m}), b(s+1), \ldots, b(\hat{m}+\tilde{m})]. \tag{39}$$

If $i \leqslant \hat{m}$, then the preliminary gene lies in the head part, and it can be determined that the gene is the ith component $h(i)$ of h_x. In this case, no inserting gene is required. The generalized mutation operator can be used to select an integer in $\mathbf{N}(k_i)$ randomly and $h(i)$ can be replaced by the random selected integer. The mutation process can be denoted as

$$x' = h_x' \oplus b_x, \tag{40}$$

where

$$h_x = [h(1), h(2), \ldots, h(i-1), \text{random}(k_i), h(i+1), \ldots, h(\hat{m})]. \tag{41}$$

4. Generalized reversion operator **R**

To enhance the convergent speed of the GCGA, the generalized reversion operator is designed

$$\mathbf{R}: \mathbf{D} \rightarrow \mathbf{D}. \tag{42}$$

It is a one-element random operator, and its variable is also an element belonging to the set **D**. If $x \in \mathbf{D}$, then after implementing the generalized reversion operator **R**, we can obtain a new GC

$$x' = \mathbf{R}(x). \tag{43}$$

Similarly to generalized mutation operator **M**, if

$$x_1' = \mathbf{R}(x), \tag{44}$$

$$x_2' = \mathbf{R}(x), \tag{45}$$

are two new GCs generated by **R** based on x,

$$x_1' = x_2' \tag{46}$$

dose not necessarily hold generally. Equation (46) holds if and only if the two reversion points are exactly identical. It is different from the generalized mutation operator **M** that **R** affects only the body part of the GC. Let the two reversion points be i_1 and i_2 $(i_1 < i_2)$ as shown in Fig. 14. Then the flow chart of the generalized reversion operator **R** can be shown in Fig. 15.

Similarly to the conventional reversion operation, operator **R** can be used to select two reversion points i_1 and i_2 randomly

$$i_1 = \text{random}(\hat{m}+\tilde{m}), \tag{47}$$

$$i_2 = \text{random}(\hat{m}+\tilde{m}). \tag{48}$$

If any of $b(i_1)$ and $b(i_2)$ is super vertex, then the following cost computation is related to the original vertex visited in its super vertex where $b(i_1)$ or $b(i_2)$ appears. Denote

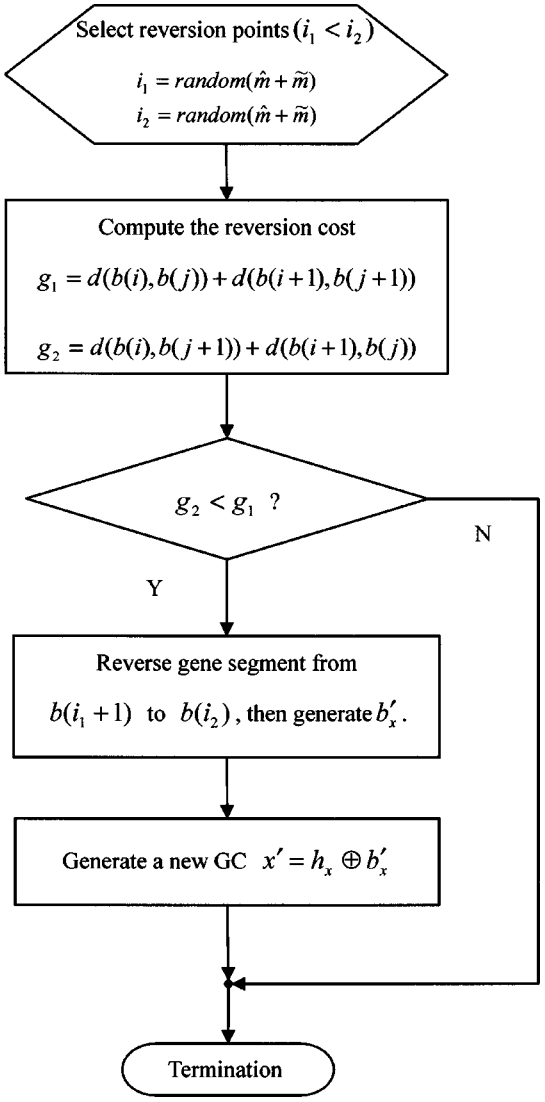

FIG. 15. Flow chart of the generalized reversion operator.

TABLE I. Results and comparisons for benchmark test problems.

		Five runs					Statistical information				Time(s)	
Problem	Opt	1	2	3	4	5	Max	Min	Ave	Err(%)	GCGA	HRKGA
10ATT48	5394	5394	5394	5394	5394	5394	5394	5394	5394	0.00	0.78	2.08
10GR48	1834	1860	1834	1834	1834	1834	1860	1834	1839	0.27	0.23	1.81
10HK48	6386	6592	6573	6386	6386	6386	6592	6386	6464	1.22	0.19	2.91
11EIL51	174	176	176	177	176	176	177	176	176	1.15	0.87	2.04
12BRAZIL58	15332	15332	15332	15332	15332	15332	15332	15332	15332	0.00	0.26	2.03
14ST70	316	316	316	316	316	316	316	316	316	0.00	1.23	2.03
16EIL76	209	214	222	214	214	214	222	214	215	2.87	1.45	1.71
16PR76	64925	64925	64925	64925	64925	64925	64925	64925	64925	0.00	1.29	2.36
20RAT99	497	497	497	497	500	497	500	497	498	0.10	2.02	3.40
20KROA100	9711	9758	9758	9758	10019	9822	10019	9758	9823	1.15	1.93	2.80
20KROB100	10328	10492	10335	10465	10328	10328	10492	10328	10389	0.59	1.79	3.35
20KROC100	9554	9554	9554	9554	9570	9554	9570	9554	9557	0.03	1.92	4.12
20KROD100	9450	9450	9450	9450	9450	9451	9451	9450	9450	0.00	1.96	5.11
20KROE100	9523	9523	9802	9637	9523	9541	9802	9523	9605	0.86	2.00	4.07
20RD100	3650	3653	3653	3653	3810	3653	3810	3653	3684	0.93	1.66	2.03
21EIL101	249	253	251	251	251	251	253	251	251	0.80	1.79	3.96
21LIN105	8213	8213	8213	8213	8213	8213	8213	8213	8213	0.00	2.10	3.68
22PR107	27898	27901	27901	27898	27898	27950	27950	27898	27909	0.04	2.25	4.07
24GR120	2769	2769	2797	2805	2779	2837	2792	2837	2779	1.01	1.19	6.04
25PR124	36605	40337	36605	36605	36762	36605	40337	36605	37382	2.12	2.51	5.11
26BIER127	72418	72418	72418	91819	72498	73252	91819	72418	76481	5.61	2.49	2.85
28PR136	42570	42570	42570	42570	42570	42570	42570	42570	42570	0.00	2.72	5.71
29PR144	45886	45890	46657	45890	45891	45891	46657	45890	46043	0.34	2.67	9.83
30KROA150	11018	11132	11602	11117	11737	11451	11737	11117	11407	3.53	3.26	4.72
30KROB150	12196	12453	12211	12428	12376	13728	13728	12211	12639	3.63	3.29	12.36
31PR152	51576	51576	51820	51628	51610	51576	51820	51576	51642	0.13	2.09	14.11
32U159	22664	22664	22981	22667	22664	22664	22981	22664	22728	0.28	2.90	9.72
39RAT195	854	870	866	874	871	870	874	866	870	1.87	4.89	17.09
40D198	10557	10626	10620	10574	10631	10557	10631	10557	10601	0.42	3.50	11.32
40KROA200	13406	13763	14848	13514	13734	13735	14848	13514	13918	3.82	4.73	16.42
40KROB200	13111	13265	13222	13164	13122	13117	13265	13117	13178	0.51	4.93	13.78
45TS225	68340	68643	68756	70886	70097	68756	70886	68643	69427	1.59	1.46	16.31
46PR226	64007	64007	65423	64007	64007	66074	66074	64007	64703	1.09	4.99	14.67
53GIL262	1013	1051	1096	1051	1040	1135	1135	1040	1074	6.02	4.76	22.03
53PR264	29549	29894	29894	30478	29549	29725	30478	29549	29908	1.21	4.26	22.79
60PR299	22615	23508	23086	23446	23089	22762	23508	22762	23178	2.49	2.96	24.6
64LIN318	20765	20977	21165	25330	21870	21237	25330	20977	22115	6.50	2.04	43.61
80RD400	6361	6465	6599	6795	6630	6535	6795	6465	6604	3.82	5.30	87.11
84FL417	9651	9763	9757	9730	9670	9706	9763	9670	9725	0.77	8.21	177.3
88PR439	60099	61395	61030	70416	60816	60529	70416	60529	62837	4.56	3.51	234.54
89PCB442	21657	22564	23062	24041	22976	23836	24041	22564	23296	7.57	5.06	88.71

$$g_1 = d[b(i), b(j)] + d[b(i+1), b(j+1)], \tag{49}$$

$$g_2 = d[b(i), b(j+1)] + d[b(i+1), b(j)], \tag{50}$$

where $d(\cdot,\cdot)$ is the cost/weight between two corresponding vertices. If $g_2 < g_1$, then reverse the gene segment from $b(i_1+1)$ to $b(i_2)$ and a new body part can be obtained

$$b'_x = [b(1), \ldots, b(b_1)b(i_2), b(i_2-1), \ldots, b(i_1+2), b(i_1+1), b(i_2+1), \ldots, b(\hat{m}+\tilde{m})]. \tag{51}$$

Combining the new body part and the original head part, we can obtain a new GC

$$x' = h_x \oplus b'_x; \tag{52}$$

if else, the generalized reversion operator does nothing.

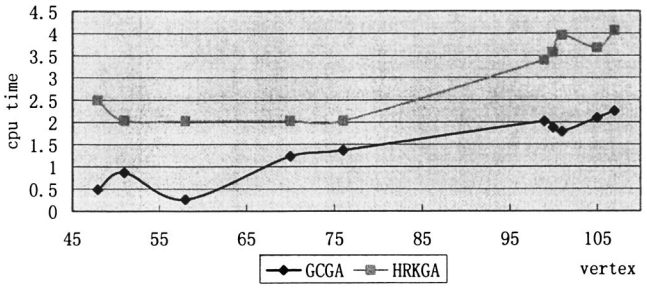

FIG. 16. Forward segments of time consumption trends of GCGA and HRKGA.

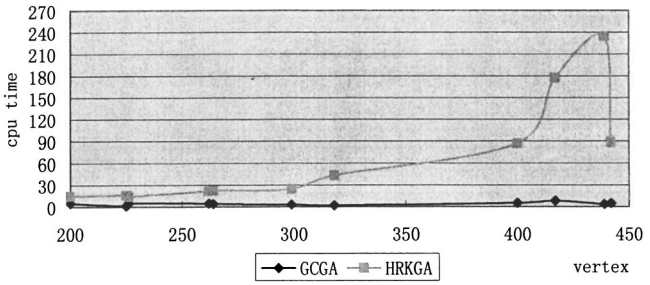

FIG. 18. Backward segments of time consumption trends of GCGA and HRKGA.

5. Generalized fitness function

In this paper, we still take the fitness function similar to that used in conventional GA, but with a little alteration

$$f(x) = \alpha\sqrt{\hat{m} + \tilde{m}}A/T(x), \tag{53}$$

where $x \in \mathbf{D}$, α is a preset constant, $\hat{m}$ and $\tilde{m}$ are the numbers of super vertex and scattering vertex, respectively, A is the edge length of the minimal regular polytope containing the vertex set $\mathbf{V}$. If $\mathbf{V}$ is a point set in a two-dimension Euclidian space, then A is the edge length of minimal square containing the point set $\mathbf{V}$, $T(x)$ is the objective function, i.e., the real cost of a GTSP cycle, which has the form

$$T(x) = T(h_x \oplus b_x) = d[b(1), b(\hat{m} + \tilde{m})] + \sum_{i=2}^{\hat{m}+\tilde{m}} d[b(i), b(i-1)]. \tag{54}$$

Notice that the difference between GTSP and conventional GA fitness functions is that the GTSP fitness function may include only part of the vertices within $\mathbf{V}$, whereas the conventional GA fitness exactly includes all of the vertices within $\mathbf{V}$. Moreover, in the fitness function of GTSP if some super vertices are included, their corresponding cost is computed with the original vertices that the current GTSP cycle visits.

III. NUMERICAL EXPERIMENTS

To verify the validity of the proposed GCGA, we first calculate all the instances used in Ref. [19] on a PC with 1.4 GHz processor and 256 M memory. These instances can be obtained from TSPLIB library [25] and were originally generated for testing standard TSP algorithms. To test GTSP algorithms, Fischetti *et al.* [5] provided a partition algorithm to convert the instances used in TSP to those which could be used in GTSP. Because the partition algorithm can generate the same results at different running provided that the data order are the same, the partition algorithm can be used to generate test data for different algorithms. In the following experiments, we take the population size as 100, maximal generation as 200, crossover probability as 0.89, and mutation probability as 0.003.

Table I presents the results and some comparisons between the proposed GCGA and HRKGA, in which the first column stands for the names of the test instances, the second for exact optimal tour length for each problem given in Ref. [5], the third to the seventh for the result in each run time, the eighth to the eleventh for some statistical information including the maximum (Max), minimum (Min), average (Ave) length for each five run and the relative error (Err), respectively, where the relative error is calculated as

$$\mathrm{Err} = \frac{\mathrm{Ave} - \mathrm{Opt}}{\mathrm{Opt}} \times 100\,\%,$$

and the last two columns for the average run time used by the GCGA and HRKGA in five runs, respectively. Table I shows that the GCGA can be used to solve GTSP effectively and efficiently. From Table I it can be seen that among 41 test problems there are 7 instances (10ATT48, 12BRAZIL58, 14ST70, 16PR76, 20KROD100, 21LIN105, and 28PR136)

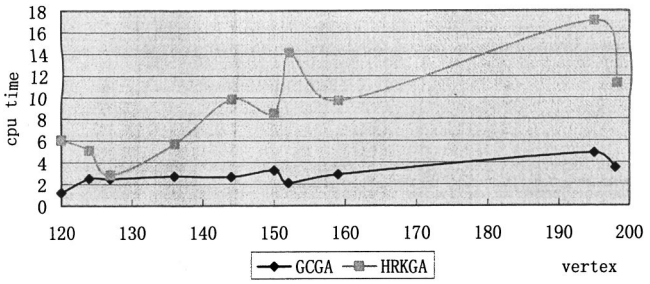

FIG. 17. Middle segments of time consumption trends of GCGA and HRKGA.

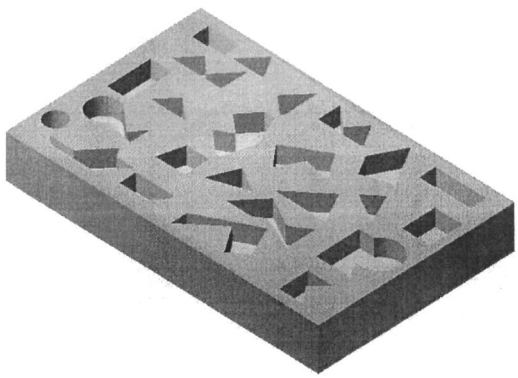

FIG. 19. Real picture of the block with some machined simple shapes.

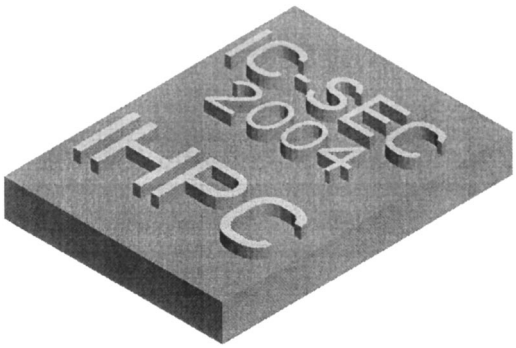

FIG. 20. Suggested machining tour of the block machining obtained by GCGA.

with zero relative errors for five runs. It shows that if the GCGA is integrated with some proper local improvement methods, good results could be obtained using GCGA. In all the simulations the maximum relative error is 7.57% and the average relative error of all the 41 test problems is 1.68%. To examine the time consumption trends clearly, Figs. 16–18 show the time consumption trends when using GCGA and HRKGA, respectively. When vertex numbers are small, the time consumption trends of the GCGA and HRKGA have almost the same increasing ratio, but the time consumption of GCGA corresponding to any given vertex number is smaller than that of HRKGA as shown in Fig. 16. With the increasing of the vertex numbers, the time consumption increasing ratio of the HRKGA has exceeded that of the GCGA as shown in Fig. 17. In the last segment as shown in Fig. 18, the time consumption increasing ratio of the HRKGA becomes nearly quadratic except for the last instance with 442 cities, however, that of the GCGA is still as linear.

Two real application examples are also implemented using the GCGA. One is to machine some simple geometrical shapes in a rectangular plate as shown in Fig. 19. Another application example is to machine an international conference logo as shown in Fig. 20. To obey the technical restrictions and obtain high efficiency and smooth contours, the cutter should go down into a shape and come back to the starting point after going continuously along the designed pattern in slots (Fig. 19) or protruding block patterns (in Fig. 20). A geometrical shape or a letter can be regarded as a super vertex, and then the machine cutter should try to minimize the jump or the time spent from one shape (letter) to another shape (letter) as the time spent is deemed to be unproductive. The GTSP is used to model both of the two examples. The proposed GCGA algorithm is employed to obtain the minimal jumping length. The results are shown in Figs. 21 and 22.

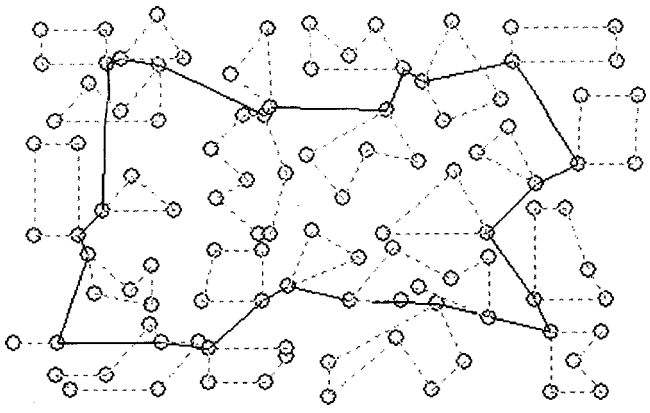

FIG. 21. Real picture of the internal conference logo.

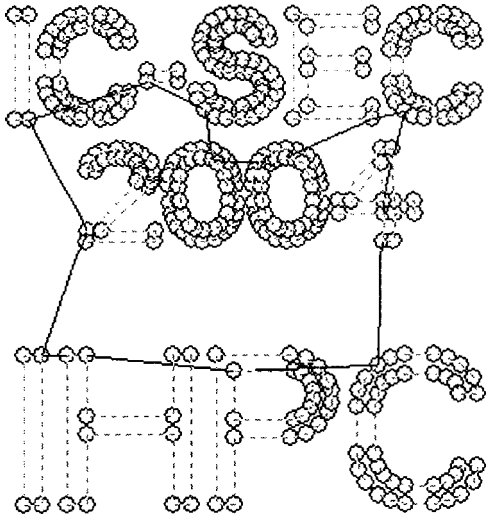

FIG. 22. Suggested machining tour of the conference obtained by GCGA.

IV. CONCLUSIONS AND DISCUSSIONS

In order to deal with the GTSP instances using GA efficiently, this paper designs a generalized chromosome at first, and then proposes the generalized-chromosome-based genetic algorithm. Simulations for both benchmark test problems and real application instances show that the proposed algorithm can be used to solve the GTSP effectively and efficiently. Furthermore, the proposed GCGA could solve TSP and GTSP instances in a unison mode. When $\hat{m}=0$, i.e., the number of super vertices is zero, the algorithm runs on an entire TSP mode. And when $\tilde{m}=0$, i.e., the number of the scattering vertices is zero, the algorithm runs on an entire GTSP mode. When both $\hat{m}\neq 0$ and $\tilde{m}\neq 0$ hold, the algorithm deals with the GTSP-TSP hybrid cases.

Compared with Ref. [19], it can be seen that the proposed GCGA is superior to the existing HRKGA in running time. The time consumption trend of the GCGA is approximately linear, whereas that of the HRKGA is nearly quadratic when the vertices exceed some number. Numerical simulations show that the local search could further improve the solution quality, however, the time consumption problem is serious. The proposed algorithm could be an ideal choice to find a quasioptimal route with high efficiency for machining applications.

ACKNOWLEDGMENTS

The authors would like to thank Lim Tiong Hwa and Yap Poh Heng for providing the pictures of machining. The first two authors were grateful to the support of the science-technology development project of Jilin Province of China under Grant No. 20030520, the doctoral funds of the Na tional Education Ministry of China under Grant No. 20030183060, and the key science-technology project of the National Education Ministry of China under Grant No. 02090.

[1] A. L. Henry-Labordere, RIRO B 2, 43 (1969).
[2] J. P. Saskena, CORS J. 8, 185 (1970).
[3] S. S. Srivastava, S. Kumar, R. C. Garg, and P. Sen, CORS J. 7, 97 (1969).
[4] G. Laporte, A. Asef-vaziri, and C. Sriskandarajah, J. Oper. Res. Soc. **47**, 1461 (1996).
[5] M. Fischetti, J. J. Salazar, and P. Toth, Oper. Res. **45**, 378 (1997).
[6] Y. N. Lien, E. Ma, and B. W.-S. Wah, J. Chem. Inf. Comput. Sci. **74**, 177 (1993).
[7] K. Castelino, R. D'Souza, and P. K. Wright, http://kingkong.me.berkeley.edu/~kenneth/
[8] N. E. Bowler, T. M. A. Fink, and R. C. Ball, Phys. Rev. E **68**, 036703 (2003).
[9] M. Andrecut and M. K. Ali, Phys. Rev. E **63**, 047103 (2001).
[10] T. Munakata and Y. Nakamura, Phys. Rev. E **64**, 046127 (2001).
[11] J. Bentner, G. Bauer, G. M. Obermair, I. Morgenstern, and J. Schneider, Phys. Rev. E **64**, 036701 (2001).
[12] G. Laporte and Y. Nobert, INFOR 21, 61 (1983).
[13] C. E. Noon and J. C. Bean, Oper. Res. **39**, 623 (1991).
[14] D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo, and A. Zverovitch, Int. J. Prod. Res. **41**, 2581 (2003).
[15] D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo, and A. Zverovitch, Oper. Res. Lett. **31**, 357 (2003).
[16] V. Dimitrijevic and Z. Saric, J. Chem. Inf. Comput. Sci. **102**, 105 (1997).
[17] G. Laporte and F. Semet, INFOR 37, 114 (1999).
[18] C. E. Noon and J. C. Bean, INFOR 31, 39 (1993).
[19] L. V. Snyder and M. S. Daskin, A *Random-key genetic algorithm for the generalized traveling salesman problem* (Northwestern University, see, l-snyder3@northweatern.edu, m-daskin@northwestern.edu).
[20] O. Jellouli, in IEEE International Conference on Systems, Man, and Cybernetics, 2001 (IEEE, Piscataway, NJ, 2001), Vol. 4, pp. 2765–2768.
[21] Y. Matsuyama, Trans. Inst. Electron., Inf. Commun. Eng. D-II **J74D-II**, 416 (1991).
[22] J. C. Bean, ORSA J. Comput. **6**, 154 (1994).
[23] D. Levine, Comput. Oper. Res. 23, 547 (1996).
[24] Y. C. Liang, H. W. Ge, C. G. Zhou, H. P. Lee, W. Z. Lin, S. P. Lim, and K. H. Lee, Prog. Nat. Sci. **13**, 1 (2003).
[25] G. Reinelt, ORSA J. Comput. **3**, 376 (1991).