Computers & Industrial Engineering 70 (2014) 124–133

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem ☆

Yong Wang *

School of Renewable Energy, North China Electric Power University, Beijing 102206, China

ARTICLE INFO

Article history:
Received 11 April 2013
Received in revised form 23 January 2014
Accepted 27 January 2014
Available online 1 February 2014

Keywords:
Traveling salesman problem
Genetic algorithm
Local optimization strategies

ABSTRACT

Traveling salesman problem (TSP) is proven to be NP-complete in most cases. The genetic algorithm (GA) is improved with two local optimization strategies for it. The first local optimization strategy is the four vertices and three lines inequality, which is applied to the local Hamiltonian paths to generate the shorter Hamiltonian circuits (HC). After the HCs are adjusted with the inequality, the second local optimization strategy is executed to reverse the local Hamiltonian paths with more than 2 vertices, which also generates the shorter HCs. It is necessary that the two optimization strategies coordinate with each other in the optimization process. The two optimization strategies are operated in two structural programs. The time complexity of the first and second local optimization strategies are $O(n)$ and $O(n^3)$, respectively. The two optimization strategies are merged into the traditional GA. The computation results show that the hybrid genetic algorithm (HGA) can find the better approximate solutions than the GA does within an acceptable computation time.

1. Introduction

The objective of traveling salesman problem (TSP) is to find the shortest tour visiting each city once and exactly once in a tourist map. The cities are connected with routes in the map. The tour including each city once is named as Hamiltonian circuit (HC) and the shortest Hamiltonian circuit is the optimal Hamiltonian circuit (OHC) for Euclidean TSP. As we know, TSP has been proven to be NP-complete (Zhang & Korf, 1996). TSP has been widely studied in the fields of combinatorial mathematics, graph theory and computer science due to its theoretical and practical values (Rodríguez & Ruiz, 2012; Wang & Liu, 2010). However, there is no polynomial algorithms for the NP-complete problems unless $NP = P$ (Berman & Karpinski, 2006). The research on the efficient algorithms for TSP is still one frontier subject.

Because of its hardness, TSP becomes one of the best platforms to test the performance of all kinds of algorithms. Exact algorithms, approximate algorithms and intelligent algorithms are extensively designed for TSP. Many literatures illustrated that the exact algorithms, such as the depth-first search graph algorithm (Douglas, 2006), the integer programming methods (Climer & Zhang, 2006) and dynamic programming methods, are feasible to tackle the TSP with less than 1000 cities (Johnson & McGeoch, 2004). They can find the OHC within an acceptable computation time using powerful Turing machines. When the scale of TSP becomes larger, the approximate algorithms demonstrate their good performance. The local search rules used by the approximate algorithms are efficient to allow them find the OHC or near OHC in a polynomial computation time. The experiments show that the k-opt algorithms (Verhoeven, Aarts, & Swinkels, 1995), the LKH algorithm has dealt with the large scale of TSP with thousands of cities, even more than 3,000,000 cities (Helsgaun, xxxx). On the other hand, the local search rules, such as the neighborhood information (Liu, 2008), will make the algorithms trap into the local minima. Therefore, the quality of the solutions cannot be evaluated due to the lack of the OHCs.

The intelligent algorithm is another resolution for TSP. They find the best or approximate solutions based on the evolutionary rules which are different from the local search rules. Almost all of the intelligent algorithms, including the anterior artificial neural network (Ghaziri & Osman, 2003) and the recent particle swarm optimization (Chen et al., 2010), have been applied to TSP. The genetic algorithm (GA) is one of the competitive intelligent algorithms for discrete optimization problems. It evolves to the optimal solution with the crossover operation and mutation operation (Schmitt & Amini, 1998). The crossover operation guarantees the better genes evolve to the offspring to obtain better solutions. The mutation operation maintains the diversity of the offspring to reproduce

☆ This manuscript was processed by Area Editor Mitsuo Gen.
* Tel.: +86 010 6177 2051.
E-mail address: wangyyong100@163.com

http://dx.doi.org/10.1016/j.cie.2014.01.015
0360-8352/© 2014 Elsevier Ltd. All rights reserved.

better solutions. Although it has many merits, the genetic algorithm usually detects an approximate OHC for TSP. It is always improved to acquire the better performance. The good initial solutions facilitate the genetic algorithm to evolve to the best solution. Liu (Liu, 2010) designed three kinds of initial solution generators under the GA framework for TSP. It is found that the various computation times are consumed and different results are obtained with different initial solutions. To enhance the performance of traditional GA, the local search rules are widely taken into consideration. The 1-shift and 2-opt local search rules are merged into the GA by Liu (2010). Also, the feasible search space is computed by Choi, Kim, & Kim (2003) through determining the infeasible solutions with sub-tours. The memetic algorithm is introduced by Bontoux, Artigues, & Feillet (2010). The cities are partitioned into a set of clusters first, and the minimum tour visiting each cluster is found with the crossover procedure based on a dynamic programming algorithm. The cluster method is also used by Ding, Cheng, & He (2007) in their two-level genetic algorithm, in which the first level is to generate the shortest sub-tours and the second level is to generate the shortest tour with the sub-tours. The immigration, local and global optimization strategies are defined and inserted into the GA (Xing et al., 2008). In addition, the other intelligent algorithms are combined with GA to reinforce its performance. The genetic algorithm is improved by Liu & Zeng (2009) with the reinforcement mutation which relies on the reinforcement learning. The genetic algorithm, simulated annealing, ant colony optimization and particle swarm optimization are integrated together to utilize their total advantages (Chen & Chien, 2011).

Different from the above researches, the genetic algorithm with two local optimization strategies is designed for TSP. The first local optimization strategy is the four vertices and three lines inequality. The four point conditions for symmetrical TSP has been summarized by Deineko, Klinz, & Woeginger (2006) under their assumptions. The approximate algorithms with the four point conditions are seldom regarded. The four vertices and three lines inequality is the extension of one of the fundamental four point conditions. It is the constraint of the local optimal Hamiltonian path connected by four vertices in the OHC or an approximate OHC. When the local Hamiltonian paths in the HCs are transformed into the local optimal Hamiltonian paths with the four vertices and three lines inequality, the HCs will become shorter. The computation complexity of the algorithm based on the local optimization strategy is $O(n)$. After the HCs are adjusted with the four vertices and three lines inequality, the adjacent vertices in the HCs cannot be exchanged to produce the shorter HCs. On the other hand, the nonadjacent vertices in the HCs are allowed to exchange to generate the next shorter HCs. The second optimization strategy is the reversion of a selected local Hamiltonian paths including over 2 vertices. The algorithm is designed according to the optimization strategy and the maximum computation complexity is $O(n^3)$.

The two strategies are merged into the genetic algorithm to construct a hybrid genetic algorithm (HGA). The first local optimization strategy is executed after the HCs are generated to produce the shorter HCs. The second optimization strategy is merged into the mutation operation to generate the shorter HCs. The HGA are designed and tested with the TSP instances downloaded from the TSPLIB. The results show that the OHCs of most of the small scale of TSP instances are found within an acceptable computation time. For the large scale of TSP, the errors of the detected approximate OHCs to the given OHCs are very small.

2. Basic knowledge on symmetrical TSP

In graph theory, it is to find the OHC in a weighted graph (WG) and only the simple graph G is considered for TSP. The cities and routes are mapped into the vertices and edges in the WG. For a graph G including n vertices, it is generally represented as $G = \langle V,E\rangle$, where $V = \langle v_1, v_2, \ldots, v_n\rangle$ are the vertices sets and $E = [e_{ij}]_{n\times n}$ are the edges sets. v_i $(1 \leqslant i \leqslant n)$ is the vertex and e_{ij} $(1 \leqslant i,j \leqslant n)$ is the edge connecting the two vertices v_i and v_j. The relationships of the vertices in graph G are represented as an adjacent matrix $A(G) = [a_{ij}]_{n\times n}$ $(1 \leqslant i,j \leqslant n)$, where $a_{ij} = 1$ if $(v_i, v_j) \in E(G)$ and v_i and v_j are adjacent in the graph G. Otherwise, $a_{ij} = 0$. If the edges are assigned with weights $W = [w_{ij}]_{n\times n}$, the graph G becomes one WG. The weight w_{ij} is often taken as distance, cost, etc. for various kinds of TSP. If $w_{ij} = w_{ji}$, the WG is symmetrical. Otherwise, it is asymmetrical. It is considered that the symmetrical TSP is more difficult than the asymmetrical TSP (Helsgaun, xxxx). The report said the asymmetrical TSP with 500,000 cities had been resolved whereas the symmetrical TSP with only 7397 cities was resolved at that time.

Given a WG including n vertices, there are total $(n-1)!/2$ HCs for the symmetrical TSP. The OHC is taken as the HC whose length is the shortest among all of the HCs for the Euclidean TSP. Given the HC including n vertices, it is represented as $\mathrm{HC}^{n+1} = (v_1, v_2, v_3, \ldots, v_n, v_1)$. The end vertex v_1 emerges repeatedly once to form the HC. Given $l_{i\times j}$ is the distance between the two adjacent vertices v_i and v_j in the HC, the computation model of the symmetrical TSP is given as formula (1).

$$\left.\begin{array}{l} L_{\min} = \min(L(\mathrm{HC})) = \min \sum_{i,j=1}^{n} l_{i\times j} \\ \text{Subject to} \quad v_i \neq v_j \quad \text{and} \quad e_{i\times j} \in E(\mathrm{HC}) \end{array}\right\} \tag{1}$$

where $L(\mathrm{HC})$ is the length of the HC, $e_{i\times j}(1 \leqslant i,j \leqslant n)$ is the edge linking the two adjacent vertices v_i and v_j in the HC. For the simple WG, it is equal to the standard form of integer programming for TSP (Climer & Zhang, 2006). With the model, all the HCs will be traversed and evaluated to find the OHC.

The HC consists of the local Hamiltonian paths (LHP) and the OHC must be composed of the local optimal Hamiltonian paths (LOHP). The LHP or LOHP including i $(1 \leqslant i \leqslant n)$ vertices is represented as LHP^i or $\mathrm{LOHP}^i = (v_1, v_2, v_3, \ldots, v_i)$. The vertices v_1 and v_i are the end vertices, and the other vertices between them are the middle vertices. There are no identical vertices in the LOHPs or LHPs.

For general WG, the orders of the vertices are determined in the OHC. For an arbitrary LOHP in the OHC, the orders of these vertices are also concluded as well as its two end vertices. Its length is the minimum among those of the LHPs including the same vertices in condition that their two end vertices are identical. If i is big, it is also NP-complete to detect a LOHP^i. If i is small, it is relatively convenient to compute a LOHP^i. But not all of the LOHPs belong to the OHC.

3. The genetic algorithm with the two local optimization strategies

3.1. The traditional genetic algorithm

Genetic algorithm originates from the evolutionary rules of the nature population and it is one of the unconstrained optimization methods (Holland, 1975). The solution of optimization problem is encoded as the chromosome and the concrete parameters of solution are taken as the genes of the chromosome. For most of the optimization problems, the solutions are often encoded as the binary strings. However, it is believed that the binary string is not suitable to represent an HC (Michalewicz, 1994). The binary code of the cities will make the GA become complex for TSP. The literatures (Gen & Cheng, 1997; Michalewicz, 1994) summarized the current representations of the HCs, such as the adjacency representation, ordinal representation, path representation and random-ken representation. The path representation is the direct and meaningful representation of HCs. In addition, there are several typical crossover and mutation operators for path representation model. With the path representation model, the vertices (cities) are coded as numbers and HC is the sequence of the vertex numbers. The vertex numbers are mapped into the genes and the HC is considered as the chromosome. In the optimization process, the HCs are changed through crossover and mutation operations whereas the vertex numbers are kept unchangeable.

Before the optimization process is executed, the maximum number of computational iterations $N_{\max}$ and population size S_P must be set. The optimal or approximate solution is searched using the permutation of the population within the computational iterations. In general, the more approximate optimal solution will be found with the bigger size of population and more computational iterations and the more computation time will be demanded. The permutation of chromosome is based on four operations, such as the evaluation, the selection, crossover and mutation. Each chromosome has a fitness value computed with its fitness function. The chromosomes are evaluated based on their fitness values and the better chromosomes are selected for crossover and mutation to generate their next offspring.

3.2. The framework of the hybrid genetic algorithm

The framework of the hybrid genetic algorithm (HGA) for TSP is given in Table 1. The initializations are accomplished in the first three steps. They include the generation of initial HCs and assignment of the HGA parameters. The initial HC contains n cities which are represented as numbers and produced at random. The parameter including the maximum computational iteration $N_{\max}$, population size S_P, the probabilities of the selection P_s, crossover P_c and mutation P_m are assigned by designers. It notes that the values of these parameters play an important role to affect the convergence speed of the algorithm and solutions' quality. The computational iteration $N_{\max}$ can be tested with the TSP instances. It is not set too big, or the computation time will be lengthened.

A moderate population size S_P will guarantee the enough diversity of the later generations in the optimization process and a satisfactory convergence rate. Oliveto & Witt (2012) resolve the OneMax problem with the general GA and found that the GA will have exponential runtime once the population size is above $n^{1/8} - \varepsilon$, where n is the scale of problem and ε is a small number. Some experiments show that the fitness has small improvement after the population size is above 30 (Reeves, 1994). Hence, we will assign a value near to 30 as the population size S_P. The other three parameters P_s, P_c and P_m will be set in Sections 3.3, 3.4 and 3.5, respectively.

Table 1
The procedure of the hybrid genetic algorithm.

Step	The pseudo codes of the hybrid genetic algorithm
1	Input: the iterations $N_{\max}$ and population size S_P
2	Input: the probabilities of the selection P_s, crossover P_c and mutation P_m
3	Generate the initial population of HCs at random and compute their length L
4	While (computational iterations less than $N_{\max}$)
5	Evaluate the HCs with their probabilities of selection
6	Select the HCs according to P_s until the HCs pool is full
7	Execute the crossover operation to the HCs
8	Execute the mutation operation with the second local optimization algorithm to the offspring of HCs
9	Apply the four vertices and three lines inequality to the LHP^{4}s in the HCs to make them become shorter
10	End and output the results

The following five steps from step 5 to step 9 are used to generate the OHC or an approximate OHC, which includes the evaluation, selection, the crossover, the mutation containing the second local optimization algorithm and the first local optimization algorithm. The first local optimization algorithm is designed based on the four vertices and three lines inequality, and the second local optimization algorithm is used to reverse the LHPs with more than 2 vertices in the HCs to generate the shorter HCs. The hybrid algorithm is executed until the terminal conditions are met.

3.3. The evaluation and selection operations

In the optimization process, the chromosomes will be evaluated according to the defined fitness function and the good chromosomes will be selected into the next generation of for crossover and mutation. For the minimum TSP, the fitness function of HC is defined as formula (2). It is the reciprocal of the length of the HC, where $L(\mathrm{HC})$ represents the length of the HC. The bigger the fitness value is, the shorter the HC is.

$$f = 1/L(\mathrm{HC}) \tag{2}$$

Given S_P HCs, their fitness values will be computed first. Then these HCs are ordered according to their fitness values. The HCs with big fitness values are taken as good HCs. There are many methods to select the good HCs, such as the roulette-wheel selection, tournament selection, steady state selection and rank selection (Srinivas & Patnaik, 1994). Each of them has their own advantages and disadvantages. For example, the roulette-wheel selection is not suitable to the case that the fitness value of the best HC is 85% of the roulette wheel. In this case, the other HCs have little chance to be selected into the next generation. We assume that the HCs in the same generation do not have much distinction in view of the fitness values, especially in the later evolutionary process. Many HCs will have big fitness values and they should be selected into the next generation. The steady state selection (Sharma & Mehta, 2013) is this kind of method. It is used to select the HCs with big fitness values. We will give a big value to the probability of selection P_s to select more number of HCs into the next generation. The HC with small fitness will be discarded. The HCs are selected according to their probabilities computed with the defined fitness function. The HCs in the last generation are evaluated with their probabilities of selection computed as formula (3).

$$P_i = f_i \Big/ \sum_{j=1}^{S_P} f_j \tag{3}$$

where P_i is the probability of selection of HC i, and f_i is its fitness value and S_P is the size of the population. After the probability of selection P_s is given, the $P_s \times S_P$ HCs will be selected into the next generation. The bigger the P_i is, the more possible the HC i will be selected.

3.4. The crossover operation

The crossover operation is the first primary evolutionary rule to generate the novel offspring. In Section 7.3.4.2 of literature (Yu & Gen, 2010), they introduced eight kinds of crossover operators which are used to break a permutation. Some of them are widely used in various GA algorithms, such as partially mapped crossover (PMX), order crossover (OX), position-based crossover (PBX) and cycle crossover (CX). The PMX is proposed by Goldberg & Lingle (1985) and it is convenient to change the old HCs into the new HCs. Here the simple PMX called two point crossover is used as the crossover operator to generate the next offspring. To compare the performance of the HGA and the traditional GA, we do not improve the PMX for HCs.

As a special case of the PMX, the two point crossover technique (Majumdar & Bhunia, 2011; wiazda, 2006) is apt to generate the next generations. For farther and mother HCs, two random numbers are generated to position the two LHPs in the same sites for crossover. Then the two LHPs in the same positions of the father and mother HCs are exchanged and the two offspring HCs are obtained. It notes that the illegal offspring HCs with two identical vertices may be produced with the two-point crossover operation. At this time, the illegal offspring HCs should be repaired and reassembled to form the legal HCs. The two point crossover is shown with one simple example with 9 cities (vertices).

One father HC_f^{10} and mother HC_m^{10} are selected at step one. Two integer numbers are generated to position the two LHPs to be exchanged at step two. The two LHPs for exchange are illustrated in two vertical lines. At step 3, the two LHPs in father and mother HCs are exchanged and two offspring HCs are generated. If the sub-tours are generated in the offspring HCs, the partial map vertices, i.e., the allele pair at same locus of the sub-tours, are used to replace the repetitive vertices. For example, the vertex 2 in $HC_{\text{off}_1}^{10}$ is replaced by the vertex 4 in $HC_{\text{off}_2}^{10}$ at step 4. The vertex 4 in $HC_{\text{off}_2}^{10}$ is also replaced by the vertex 2 in the $HC_{\text{off}_1}^{10}$. After the first and end vertices are updated, the two legal offspring HCs are obtained.

$$\text{Step 1}\quad HC_{\text{f}}^{10} = (3,2,6,4,9,1,5,7,8,3)^{10}$$

$$HC_{\text{f}}^{10} = (4,3,9,2,6,5,7,8,1,4)^{10}$$

$$\text{Step 2}\quad HC_{f}^{10} = (3,2,6,|4,9,1|,5,7,8,3)^{10}$$

$$HC_{m}^{10} = (4,3,9,|2,6,5|,7,8,1,4)^{10}$$

$$\text{Step 3}\quad HC_{\text{off}_1}^{10} = (3,2,6,|2,6,5|,5,7,8,3)^{10}$$

$$HC_{\text{off}_2}^{10} = (4,3,9,|4,9,1|,7,8,1,4)^{10}$$

$$\text{Step 4}\quad HC_{\text{off}_1}^{10} = (3,4,9,2,6,5,1,7,8,3)^{10}$$

$$HC_{\text{off}_2}^{10} = (2,3,6,4,9,1,7,8,5,2)^{10}$$

In this paper, one father and mother HC are selected orderly for crossover once to reproduce two offspring HCs. The probability P_c of crossover operation is set to restrict the number of the chromosomes, i.e., HCs for crossover. It is the ratio of the number of the HCs for crossover to the total number of the HCs. To preserve the good local LHPs in the parent HCs, we assign the P_c a big value above 0.5. The total $P_c \times S_P$ HCs will be selected for crossover operation and the good LHPs of parent HCs will be saved to the offspring HCs as many as possible.

3.5. The mutation operation

Mutation operation is the second primary evolutionary rule to reproduce the novel offspring. The function of mutation is to add the diversity of the population and prevent the chromosomes falling into the local minima. Different from crossover operation, mutation operation is usually used to change an HC into another HC. In the previous GAs, the exchange mutation (EM), insertion mutation (ISM), displacement mutation (DM) and inversion mutation (IM) Michalewicz, 1994; Yu & Gen, 2010 operators are widely used. The EM is used to exchange two vertices in the original HC and an offspring HC is obtained. For example, the initial HC^{10} is $(2, 3, 6, 9, 4, 1, 5, 8, 7, 2)^{10}$. Two random numbers are generated as 2 and 5, which means that the 2nd vertex and the 5th vertex will be exchanged. After the 2nd vertex and the 5th vertex are exchanged, the offspring HC^{10} is $(2, 4, 6, 9, 3, 1, 5, 8, 7, 2)^{10}$. In our traditional GA, the EM operation is used. After the two optimization strategies are incorporated into the GA, the HGA is designed.

Unlike the selection and crossover, the mutation operation is performed with a small probability (Črepinšek, Liu, & Mernik, 2013). If the mutation probability is too big, the GA is seldom converged. If it is too small, the GA will easily trap into the near optimum solutions (Ise website, 2013). The probability P_m of mutation controls the number of the HCs for mutation. It is the ratio of the number of the HCs for mutation to the total number of the HCs. In this paper, we assign P_m a small value which is less than 0.2.

To reduce the computation time, $100 \times P_m$ HCs are selected randomly for mutation in each computation cycle. To generate the OHC or approximate OHC, the second local optimization strategy is applied to the HCs and integrated into the mutation process besides the EM operation. The second local optimization algorithm is given in Table 2.

After a HC is adjusted with the four vertices and three lines inequality, the LHP^4s in the HC become the $LOHP^4$s. The adjacent vertices in the adjusted HC are not allowed to exchange. Otherwise, the LHP^4s will be generated and the approximation may become longer. In Table 2, an integer variable d is set to represent the number of vertices between two appointed vertices v_i and v_j of an LHP. The value of the integer variable d is assigned above 2 to assure that the LHP includes more than 4 vertices. In the 5th step, the LHP except the two end vertices v_i and v_j is reversed and an offspring HC is generated. For example, an $LHP^5 = (2, 3, 6, 9, 4)$ is selected from the $HC^{10} = (2, 3, 6, 9, 4, 1, 5, 8, 7, 2)^{10}$. After the inner path (3, 6, 9) is reversed, the offspring is $HC^{10} = (2, 9, 6, 3, 4, 1, 5, 8, 7, 2)^{10}$. The initial HC and its offspring HC will be compared in the 6th step and the shorter HC will be maintained. The four vertices and three lines inequality ensure the LHP^4s in the HCs to change into the $LOHP^4$s. An HC will become shorter if it includes the LHP^4s. The second optimization strategy is used to reverse the inner path of the LHPs with more than four vertices and maintain the $LOHP^4$s as many as possible in an HC. Though we do not know which $LOHP^4$ belongs to the OHC, we can conclude that the OHC is composed of some $LOHP^4$s but none of the other LHP^4s.

The maximum computation complexity of the second local optimization strategy is $O(n^3)$. To reduce the computation time, the second local optimization strategy will be not called in each computation cycle for large scale of TSP. After the computational iterations $N_{\max}$ is appointed, the second local optimization strategy can be called $N_{\max}/m$ times and m is an integer variable assigned by designers. For large scale of TSP, the smaller the m is, the better the

Table 2
The second local optimization strategy applied to the HCs.

Step	The pseudo codes of the second local optimization strategy
1	Choose one HC and set an integer variable $d \in [2, n]$
2	For($d = 2$; $d < n$; d++)
3	For($i = 0$; $i < n - d$; i++)
4	For($j = i + 2$; $j \leqslant i + d$; j++)
5	Reverse the LHP between vertex v_i and v_j and an offspring HC is generated
6	The offspring HC and the original HC are evaluated and maintain the shorter HC
7	End
8	End
9	End and output the results

result will be obtained and the longer the computation time will be spent.

3.6. The first local optimization strategy

The first local optimization strategy is placed at step 9 after the mutation operation in Table 1. It is derived from the fact that the OHC is composed of the LOHPs as well as n LOHP^{4}s. The first local optimization strategy is given in Table 3. There are n LHP^{4}s (or LOHP^{4}s) in an HC and they will be changed into the n LOHP^{4}s sequentially.

For the symmetrical TSP, it is easy to derive the number of the LOHPs with i vertices as formula (4). Where $i!$ is the factorial numbers of the integer number i, C_n^i is the number of the combinations in the case that i vertices are selected from the n vertices. The denominator $(i-2)!$ denotes the number of the LHPs composed of the i vertices excluding its two end vertices. It implies that a path is composed of at least two vertices and a HC consists of at least three vertices. The formula illustrates that the number of LOHP^{i}s changes as the binomial coefficient multiplied by a factor. It increases at first, and then decreases. It is smaller than the total number of LHP^{i}s $n!/(n-i)!$ in the complete graph with n vertices. When i is equal to n, the LOHP^{i}s become the Hamiltonian paths LOHP^{n}s. Total $n\times(n-1)/2$ LOHP^{n}s will be computed based on formula (4) and the $n\times(n-1)/2$ HC^{n+1}s will be obtained after the two end vertices of LOHP^{n}s are connected. There are n OHC among the $n\times(n-1)/2$ HC^{n+1}s. Evaluate the length of these HC^{n+1}s and the OHC will be found.

$$N_{\text{LOHP}^i}=\frac{i!C_n^i}{2\times(i-2)!} \tag{4}$$

In view of formula (4), the number of LOHP^{i}s is still large, and it is also NP-complete to search the LOHP^{i}s when i is near to $n/2$ (or $(n+1)/2$). When i is small, the number of LOHP^{i}s will be less and a lot of inequalities will be necessary. In particular, if $i=4$, the LOHP^{4}s will be easy computed with one four vertices and three lines inequality. The principle of the four vertices and three lines inequality is illustrated in Fig. 1.

Given the OHC and HCs are composed of n vertices, the OHC is shown in Fig. 1(a) and one HC is shown in Fig. 1(b). An arbitrary LOHP^4 in the OHC is noted as $(v_{i-1}, v_i, v_j, v_{j+1})$ $(2\leqslant i\leqslant n, 1\leqslant j\leqslant n-1)$. After the two middle vertices in the LOHP^4 are exchanged, it become the LHP$^4=(v_{i-1}, v_j, v_i, v_{j+1})$ in the HC in Fig. 1(b). Given the other two paths LOHP^{n-4}s except the LOHP^4 and LHP^4 in Fig. 1(a and b) are identical, the length of the other two paths is equal and it is represented as L_{rest} in Fig. 1(a and b). The length of the LOHP^4 is computed as $l_{(i-1)\times i}+l_{i\times j}+l_{j\times(j+1)}$ and the length of LHP^4 is $l_{(i-1)\times j}+l_{j\times i}+l_{i\times(j+1)}$. It is known that the OHC in Fig. 1(a) is shorter than the HC in Fig. 1(b). Therefore, the four vertices and three lines inequality (5) holds for the symmetrical TSP.

$$l_{(i-1)\times i}+l_{i\times j}+l_{j\times(j+1)}\leqslant l_{(i-1)\times j}+l_{j\times i}+l_{i\times(j+1)} \tag{5}$$

where $l_{i\times j}$ is the length of edge $e_{i\times j}$ linking the vertices v_i and v_j $(2\leqslant i\leqslant n, 1\leqslant j\leqslant n-1)$.

Table 3
The first optimization strategy.

Step	The pseudo codes of the first local optimization strategy
1	Given one HC with n different vertices
2	For($i=1$; $i<n-2$; i++)
3	Select an LHP$^4=(v_i, v_{i+1}, v_{i+2}, v_{i+3})$ in the HC
4	Change the LHP^4 into the LOHP^4 according to the four vertices and three lines inequality
5	End and output the results

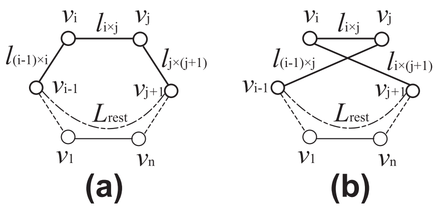

Fig. 1. The illustration of four vertices and three lines inequality.

The four vertices and three lines inequality is the constraints of the LOHP^{4}s, and all the LOHP^{4}s in the OHC conform to it. Given the OHC is composed of n LOHP^{4}s, the computation model (1) is tightened as model (6) when the constraint is added.

$$\left.\begin{array}{l} L_{\min}=\min L(\text{HC})=\frac{1}{3}\min(\sum_{i=1}^{n}L_i(\text{LOHP}_i^4)) \\ \text{Subject to } L_i(\text{LOHP}_i^4)\leqslant L_i'(\text{LHP}_i^4) \end{array}\right\} \tag{6}$$

where $L_i(\text{LOHP}_i^4)$ is the length of the ith LOHP^4 and $L_i'(\text{LHP}_i^4)$ is the length of the ith LHP^4. $LOHP_i^4=(v_i, v_{i+1}, v_{i+2}, v_{i+3}), LHP_i^4=(v_i, v_{i+2}, v_{i+1}, v_{i+3})$. $(v_{n+1}, v_{n+2}, v_{n+3})=(v_1, v_2, v_3)$ if $i\geqslant n-2$. After the LHP^{4}s in the HCs are changed into the LOHP^{4}s with the four vertices and three lines inequality, the shorter HCs will be obtained.

4. Experimental results and analysis

In order to test the performance of the HGA with the two local optimization strategies, the algorithm is coded with C++ language and implemented on the computer with a processor 2.3 GHz and inner memory 2G. The traditional GA without the two optimization strategies is coded for comparisons. In addition, the traditional GA plus the second optimization strategy (GA + 2nd) is also tested for comparisons. The results show that the HGA nearly always produces the best results under the same preconditions. In our experiments, the symmetrical Euclidean TSP instances are tried. The TSP instances are downloaded from the TSPLIB (http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/). For different scale of TSP, the parameters of the GA are adjusted slightly for reducing the computation time. The parameters of the HGA are set as those in Table 4 for different scale of TSP instances. With the increment of the scale of TSP, population size S_P is assigned bigger and bigger but less than 50. The probabilities of selection P_s and crossover P_c do not change much. For medium and large size of TSP, the probability of mutation P_m becomes big to enhance the diversity of the population and find a good approximation. It mentions that the values of the parameters for different scale of TSP are not the most suitable. We change the parameters slightly and find the outputs do not change much. For large scale of TSP with over 200 cities, the second local optimization algorithm is called $N_{\max}/m$ times in the computation process. To accelerate the converge of the HGA, the value m becomes bigger with the increment of the scale of TSP. In these experiments, m is equal to 5 or 10.

Table 4
The parameters of the hybrid GA.

Scale of TSP	$N_{\max}$	S_P	P_s	P_c	P_m
$s\leqslant 130$	50	20	0.6	0.6	0.1
$130<s\leqslant 200$	50	30	0.6	0.6	0.1
$200<s\leqslant 300$	100	50	0.8	0.8	0.15
$300<s\leqslant 500$	100	50	0.8	0.6	0.15

Note: $N_{\max}$ – computational iterations, S_P – population size, P_s – probability of selection, P_c – probability of crossover, P_m – probability of mutation.

For a TSP instance, the three algorithms, i.e., the HGA, the GA + 2nd (GA with the second optimization strategy) and the GA, are used to compute the approximate OHCs. For TSP with less than 200 cities, each algorithm is executed 10 times and 10 results are computed. For the other TSP instances, each of the algorithms is executed 5 times to reduce the computation time.

The experimental results with the three algorithms are shown in Table 5. The length of the computation results may be a little bigger than the given OHC in the TSPLIB due to the computation with the floating numbers. The given length of the OHC of some TSP instances is noted with integers in the TSPLIB.

The average length of the results is computed as *AL* and the shortest length of the results is selected. In the column of Best length in Table 5, the numbers behind the slash "/" represent the number of the best approximations found in the 10 or 5 experiments. If the identical best approximation is found with the three algorithms in the 10 or 5 experiments, it is clear that the times of the HGA are the most. The relative error (*RE*) of the average length to the given OHC is defined as $RE = (AL - OL)/OL \times 100\%$, where *OL* represents the length of the given OHC. In view of the *RE*s in Table 5, it also finds that the HGA nearly always find the better approximations than the GA + 2nd and GA do under the same preconditions. Among the 25 small scale of TSP with less than 200 cities, the HGA finds the 20 OHCs in the experiments. For each of the TSP instance, the HGA finds the OHC more than once in the 10 or 5 experiments. The GA + 2nd finds the 4 OHCs for these TSP instances. The GA does not find any OHC for them. For the TSP with more than 200 cities, the HGA seldom finds the OHCs, but it always finds the better approximations than the GA + 2nd and GA do. If the second optimization strategy is executed more times, the better approximations will be found.

To illustrate the performance of the HGA, GA + 2nd and GA for TSP, we select the best approximations of some TSP instances searched with the three algorithms and compare the changes of their length in the optimization process. The changes of their length are lined and shown in Figs. 2–7. The length of the initial randomized HCs is included in Fig. 2. It is much longer than the approximate OHC or the OHC. In general, the randomized HC is tens or hundreds times longer than the approximate OHC or OHC. It is also found that the length of the initial HCs is reduced to a great extent after the first computation cycle of the HGA and GA + 2nd. In view of the datum, it is known that the length of the first generated HCs and the final approximate OHCs or OHCs have the same magnitude in numbers. It says that the two optimization strategies play an important role to improve the

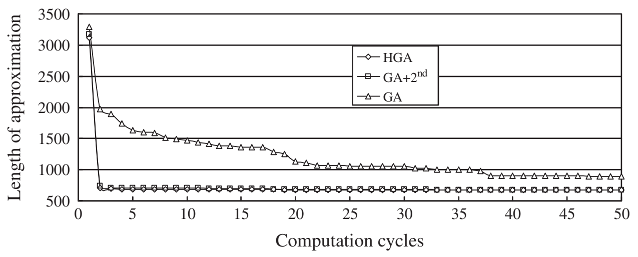

Fig. 2. The change of the approximate OHCs of St70 with three algorithms.

Table 5
The experimental results with the three algorithms.

Name	Average length			Best length			Given OHC	*RE* (%)		
	HGA	GA + 2nd	GA	HGA	GA + 2nd	GA		HGA	GA + 2nd	GA
Eil51	429.19	430.88	507.51	428.87/2	428.98/2	482.39/1	429.98	0	0.21	18.03
Berlin52	7544.37	7544.37	9345.56	7544.37/10	7544.37/10	8623.71/1	7544.36	0	0.00	23.87
St70	677.39	683.20	1029.05	677.11/9	677.44/1	897.14/1	678.59	0	0.68	51.65
Eil76	546.06	551.85	694.12	544.37/4	547.39/1	658.83/1	545.38	0.12	1.19	27.27
Pr76	108255.94	108902.83	144647.04	108159.42/4	108323.17/1	135547.52/1	108159.43	0.09	0.69	33.74
Rat99	1221.95	1228.92	1728.01	1219.24/5	1219.86/1	1557.26/1	1211	0.90	1.48	42.69
KroA100	21312.45	21464.54	32162.60	21285.44/6	21373.68/1	29236.87/1	21285.43	0.13	0.84	51.10
KorC100	20812.22	20871.62	33005.31	20750.76/4	20754.47/1	29896.13/1	20750.76	0.30	0.58	59.06
kroD100	21344.67	21576.13	30545.90	21294.29/4	21388.50/1	28076.39/1	21294.28	0.24	1.32	43.45
Rd100	7913.48	7951.78	9744.02	7910.39/6	7915.83/1	9313.30/1	7910.39	0.04	0.52	23.18
Eil101	644.82	654.62	1045.26	640.21/3	644.54/1	990.33/1	642.30	0.39	1.92	62.74
Lin105	14422.89	14456.24	26721.37	14382.99/4	14383.00/1	24080.21/1	14382.99	0.28	0.51	85.78
Pr107	44341.67	44974.54	72091.41	44301.68/5	44574.99 /1	60975.00/1	44,303	0.09	1.52	62.72
Pr124	59094.13	59419.36	87186.1	59030.73	59188.79/1	79843.20/1	59,030	0.11	0.66	47.70
Ch130	6130.277	6171.294	7679.926	6110.72/3	6110.72/1	7240.593/1	6110.86	0.32	0.99	25.68
Pr136	97019.291	97300.53	148015.61	96785.852/2	97890.23/1	135865.36/1	96,772	0.26	0.55	52.95
Pr144	58535.22	58547.32	134923.8	58535.22/10	58535.22/6	114987.7/1	58,537	0	0.02	130.49
kroA150	26597.78	26784.31	46603.59	26524.86/2	26564.13/1	42674.04/1	26,524	0.28	0.98	75.70
kroB150	26335.85	26639.16	50081.31	26127.35/1	26421.08/1	47374.45/1	26,130	0.79	1.95	91.66
Ch150	6557.6961	6572.3938	11908.542	6530.90/1	6535.01/1	10552.53/1	6532.28	0.39	0.61	82.30
Pr152	73765.70	73833.32	117124.96	73683.63/6	73683.63/3	115410.25/1	73,682	0.11	0.21	58.96
Rat195	2356.02	2362.99	3118.04	2347.24/1	2352.90/1	3017.86/1	2323	1.42	1.72	34.22
D198	15963.28	16080.41	21814.92	15896.95/1	16037.28/2	20274.21/2	15,780	1.16	1.90	38.24
kroA200	29458.809	29594.361	60944.6547	29369.40/1	29444.22/1	55840.44/1	29,368	0.31	0.77	107.52
kroB200	29583.38	30005.19	58413.682	29450.50/1	29714.22/1	53379.75/1	29,437	0.50	1.93	98.44
Tsp225	3892.88	3947.43	6309.01	3878.66/1	3932.70/1	6172.65/1	3859.00	0.88	2.29	63.49
Ts225	128295.65	128714.92	237032.71	128141.92/1	128601.30/1	205254.86/1	126,643	1.30	1.64	87.17
Pr226	80534.39	80924.68	128144.74	80436.04/2	80370.25/1	125477.30/1	80,369	0.21	0.69	59.45
Pr264	49163.26	49353.75	74703.81	49151.22/2	49236.43/1	72739.70/1	49,135	0.06	0.45	52.04
A280	2676.14	2716.80	3332.29	2659.76/1	2682.24/1	3159.63/1	2586.76	3.46	5.03	28.82
Pr299	49757.66	50528.18	79490.83	49462.43/1	50092.98/1	77423.12/1	48,191	3.25	4.85	64.95
Lin318	42877.24	44426.78	76022.39	42624.34/2	44294.75/1	74202.45/1	42,029	2.02	5.71	80.88
Linhp318	43247.07	44448.37	77374.48	43050.69/1	44156.65/1	67530.77/1	41,345	4.60	7.51	87.14
Rd400	16143.96	16457.68	35300.14	16049.59/1	16325.62/1	33114.27/1	15,281	5.65	7.70	131.01
Pr439	111209.97	112200.35	150859.01	110171.34/1	110826.13/1	146861.94	107,217	3.72	4.65	40.70
Pcb442	53016.16	54049.00	76545.25	52874.33/1	53848.04/1	74381.84/1	50783.55	4.40	6.43	50.73

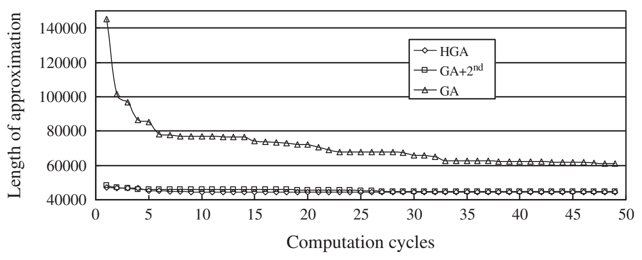

Fig. 3. The change of the approximate OHCs of pr107 with the three algorithms.

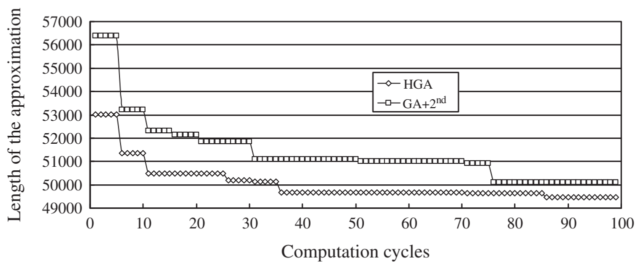

Fig. 4. The change of the approximate OHCs of pr299 with the two algorithms.

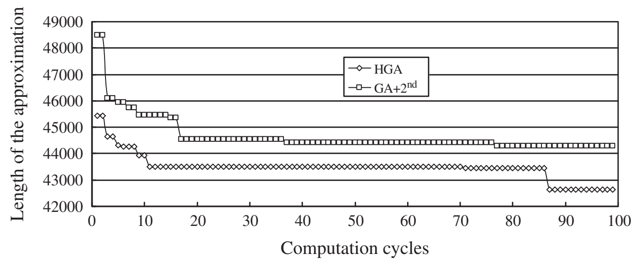

Fig. 5. The change of the approximate OHCs of lin318 with the two algorithms.

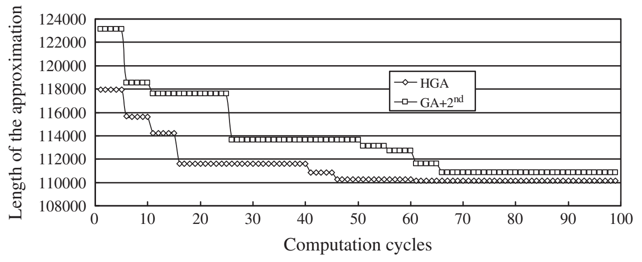

Fig. 6. The change of the approximate OHCs of pr439 with the two algorithms.

performance of GA. The second optimization strategy promotes the GA to find the approximations quickly and the first optimization strategy drives the GA to find the better approximations until the OHC is detected. The first and second optimization strategies are executed in a structural program and their time complexity is $O(n)$ and $O(n^3)$, respectively. To show the change of the approximate OHCs clearly in the computation process, the initial randomized HCs are omitted in Figs. 3–7. In Fig. 3, it finds that the final approximation searched with the GA is much larger than those found with the HGA and the GA + 2nd. For the other TSP instances, the approximations searched with the HGA and the GA + 2nd are also much smaller than those searched with the GA. To compare the performance of the HGA and the GA + 2nd intensively, the change of the approximations searched with the two algorithms

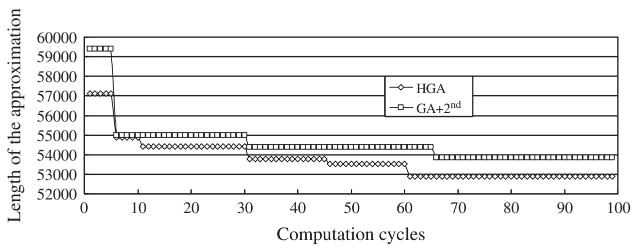

Fig. 7. The change of the approximate OHCs of pcb442 with the two algorithms.

are shown in Figs. 4–7. It finds that the HGA almost always converge to the better approximations than the GA + 2nd does. For the big scale of TSP with more than 225 cities, the HGA nearly always finds the shorter approximations in the same computation cycle. In addition, the convergence speed of the HGA is faster than that of the GA + 2nd in most cases. The changing lines also demonstrate that most of the approximate OHCs evolve to the OHCs or approximate OHCs before the $N_{\max}$ computational iterations. For large scale of TSP instances with more than 300 cities, the OHCs are not found within the 100 computational iterations. The approximate OHCs are still evolving to the shorter approximate OHCs although the terminal condition is met.

In view of above results, we know that the HGA has good performance for TSP. To reveal the performance of HGA in a further step, we adjust the parameters of HGA and do the next experiments for some TSP instances. For each TSP instance, the HGA algorithm is executed once. The parameters are given in Table 6. The maximal computation cycle $N_{\max}$ is added as well as the population size S_P.

The TSP instances with the given OHC are computed first and compared with the given OHC. The computation results are shown in Table 7. The floating numbers are used to compute the results and they are a little different from the results in the TSPLIB. The TSP instances without OHCs are also computed and the results are shown in Table 8 (the given length in the TSPLIB are integers). For TSP with more than 225 cities, the second strategy is executed $N_{\max}/5$ times to reduce the computation time. The iterative times of convergence are recorded and listed in Tables 7 and 8. In view of the datum, the initial HCs evolve to the approximate OHCs or OHCs before the $N_{\max}$ computational iterations. After the $N_{\max}$ iterative times are completed, the computation time are computed and given in Tables 7 and 8. The relative errors are also computed and shown in Tables 7 and 8. Comparing the results in Tables 5, 7 and 8, it knows that that the HGA find the better approximations with a bigger size of population and more times of computation in general.

For the small scale of TSP instances with less than 225 cities, the second local optimization strategy is executed in every computation cycles. When the scale of TSP instance is above 225, the second local optimization algorithm is applied in $N_{\max}/5$ times. Therefore, the computation time of larger scale of TSP instances may be shorter than that used to compute the smaller scale of TSP instances.

In view of the results in Tables 7 and 8, the OHCs of most of the TSP instances with less than 200 cities are found. For some

Table 6
The parameters of the HGA.

Scale of TSP	$N_{\max}$	S_P	P_s	P_c	P_m
$s \leqslant 100$	100	20	0.8	0.6	0.1
$100 < s \leqslant 200$	200	100	0.8	0.6	0.1 or 0.2
$200 < s \leqslant 300$	200	100	0.6	0.4	0.1
$300 < s \leqslant 500$	200	50	0.6	0.4	0.1

Table 7
The experimental results with given OHCs in the TSPLIB.

Name	Results	The *i*th cycle of convergence	Times (ms)	Given OHC	*RE* (%)
Eil51	428.87	51	31,454	429.98	0
Berlin52	7544.36	9	33,485	7544.36	0
St70	677.10	40	86,125	678.59	0
Eil76	544.36	14	123,500	545.38	0
Pr76	108159.42	33	110,781	108159.43	0
Kroa100	21285.435	16	299,328	21285.439	0
Rd100	7910.39	12	282,062	7910.39	0
KorC100	20750.75	71	281,328	20750.76	0
kroD100	21294.28	56	289,344	21294.28	0
Eil101	640.21	30	544,266	642.30	0
Lin105	14382.99	37	626,562	14382.994	0
Ch130	6110.71	60	1,457,906	6110.86	0
Ch150	6530.89	46	2,532,344	6532.28	0
Tsp225	3865.97	159	12,343,107	3859.00	0.180
A280	2587.80	194	13,267,734	2586.76	0.040
Pcb442	52376.26	161	39,345,313	50783.55	3.136

Table 8
The experimental results with unknown OHCs in the TSPLIB.

Name	Results	The *i*th cycle of convergence	Times (ms)	Given OHC	*RE* (%)
Rat99	1219.24	30	264,516	1211	0.68
Rd100	7910.39	14	282,062	7910	0
Pr107	44301.67	77	790,030	44,303	0
Pr124	59030.72	90	1,232,891	59,030	0
Bier127	118293.44	94	1,318,530	118,282	0.0096
Pr136	96780.45	161	1,695,762	96,772	0.0087
Pr144	58535.21	12	2,162,281	58,537	0
KroA150	26524.85	170	2,598,446	26,524	0
KroB150	26127.34	183	26,13,016	26,130	0
Pr152	73683.62	70	2,726,578	73,682	0
U159	42075.65	53	3,013,846	42,080	0
Rat195	2340.36	147	7,201,968	2323	0.747
D198	15888.98	81	7,558,093	15,780	0.690
KroA200	29369.39	103	7,893,625	29,368	0.0047
KroB200	29557.35	196	7,443,150	29,437	0.408
ts225	126645.94	62	12,343,203	126,643	0.0023
Pr226	80370.23	200	12,728,640	80,369	0.0153
Pr264	49196.99	104	23,829,609	49,135	0.123
Pr299	48480.56	198	4,392,817	48,191	0.600
Lin318	42914.03	183	5,534,562	42,029	2.105
Rd400	15852.74	178	13,317,859	15,281	3.741
Pr439	109249.66	114	17,688,906	107,217	1.895

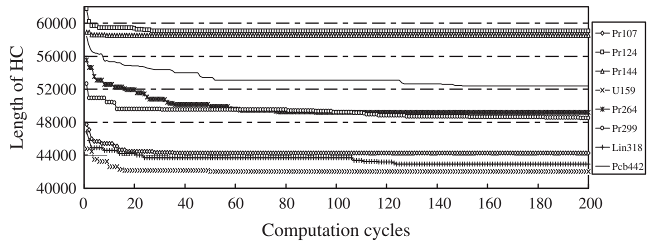

Fig. 8. The change of the length of HC of Pr107, Pr124, U159, Pr264, Pr299, Lin318 and Pcb442.

instances, such as the Eil51, St70, Eil76, Eil101, Ch150, Pr107, Pr144, KroB150 and U159, the results computed with the HGA are shorter than those computed with the given OHCs or length in the TSPLIB. The rounding methods used by the TSPLIB should be taken into consideration to evaluate the performance the HGA algorithm.

When the TSP scale is above 225, the results deviate from the given OHCs because the second local optimization strategy is executed with fewer times. Otherwise, the computation time will be lengthened. For example, the approximate OHC of Pr439 cannot be computed within 17,688,906 ms when the second local optimization strategy is run in each computation cycle.

We also illustrate the change of the approximate OHCs of some TSP instances in these experiments. They are lined and shown in Figs. 8–10. To show the change of the HCs clearly in the computation process, the initial randomized HCs are omitted in these figures. The changes of the HCs also demonstrate that most of the HCs evolve to the OHCs or approximate OHCs before the $N_{\max}$ computational iterations. For large scale of TSP instances with more than 300 cities, the OHCs are not found within 200 computational

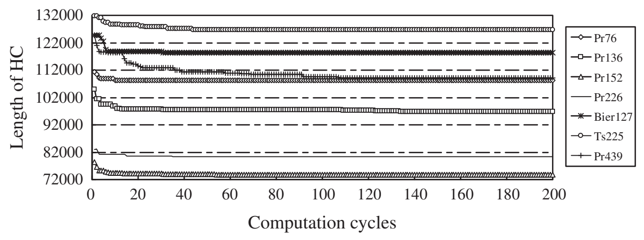

Fig. 9. The change of HC of Pr76, Pr136, Pr152, Pr226, Bier127, Ts225 and Pr439.

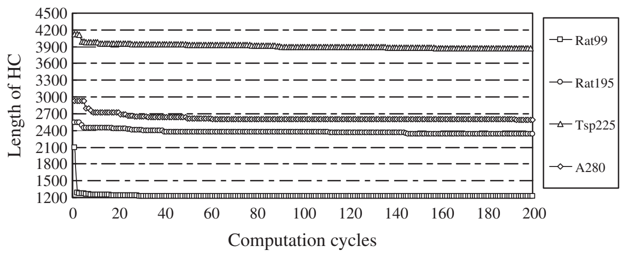

Fig. 10. The change of HC of Rat99, Rat195, Tsp225 and A280.

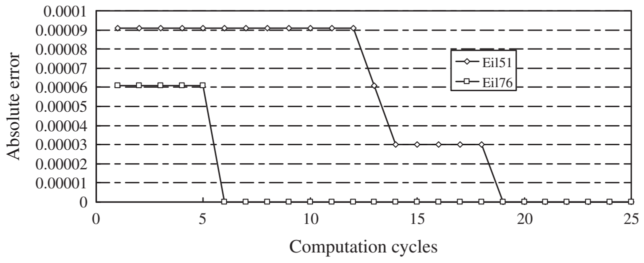

Fig. 11. The change of the absolute error from near OHC to OHC of Eil51 and Eil76.

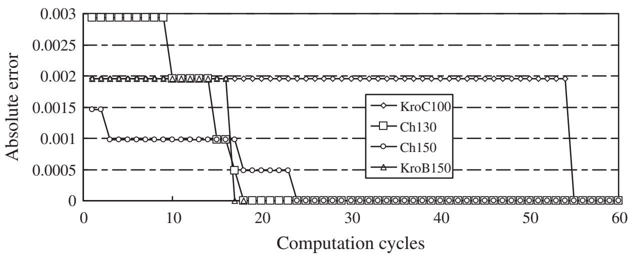

Fig. 12. The change of the absolute error from near OHC to OHC of KroC100, Ch130, Ch150 and KroB150.

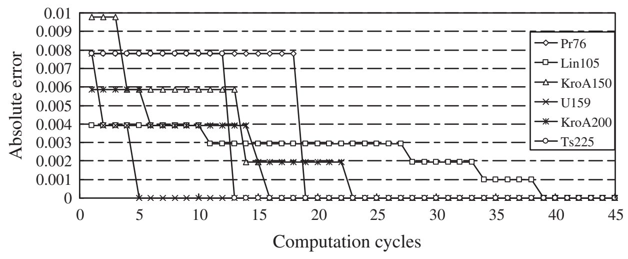

Fig. 13. The change of the absolute error from near OHC to OHC of Pr76, Lin105, KroA150, U159, KroA200 and Tsp225.

iterations. The approximate OHCs are still evolving to the shorter approximate OHCs although the terminal condition is met.

Considering the results of some instances, it is found that the integer values of the approximations are the same whereas the OHC and approximate OHC are different. The decimal parts of the OHC length is not important to most of the industrial applications but significant to verify if it is the OHC or not. To discover the minor distinction of the OHC and the approximate OHC, the integer values and two decimal values of the length of the results are maintained (the decimals are not rounded). The absolute error (AE) is defined as $AE = L(\text{AOHC}) - L(\text{OHC})$ to illuminate the minor distinction between the approximate OHC and the OHC, especially

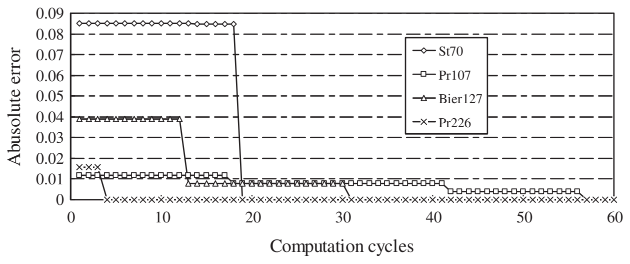

Fig. 14. The change of the absolute error from near OHC to OHC of St70, Pr107, Bier127 and Pr226.

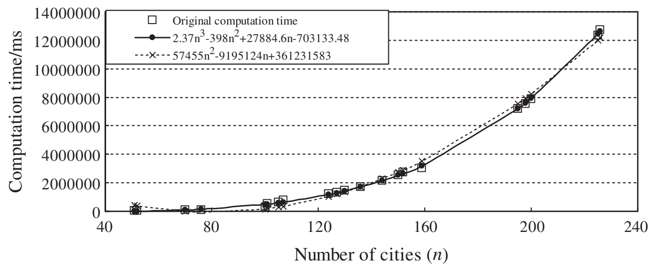

Fig. 15. The fitting curves to the original computation time.

for the approximate OHCs and the OHC of the length with the same integer values.

The length of the approximations is very large and the changes of the approximate OHCs illustrated in above Figs. 2–10 are rough. The minor distinctions between the approximate OHCs and the OHCs cannot be illuminated clearly, especially for the case that the integer values of their length are identical. To display the minor distinctions between the approximations and the OHCs in the computation process, the absolute errors are computed for some TSP instances and the changes of these absolute errors are shown in Figs. 11–14. Figs. 11–14 display the changes from the approximate OHCs to the OHCs. Despite owning the identical integer values of length, the approximate OHCs and the OHCs are distinctive and they cannot be considered as the same HC in principle.

In the last, the time complexity of the HGA is analyzed. The time complexity of the HGA is mainly determined by the computation time of two computation loops. The computation time of the outer loop is restricted by the computational iterations $N_{\max}$ and the time of the inner loop is mainly consumed by the second local optimization strategy. Therefore, the time complexity of the hybrid GA is approximately to $O(N_{\max} \times n^3)$. The computation time of the 20 TSP instances with less than 250 cities are computed and approximated with polynomial curves. It is found the cubic polynomial has a good fitness to the discrete values of computation time. The cubic and quadratic fitting curves with the original computation time are shown in Fig. 15. The fitting cubic polynomial of computation time is $T(n) = 2.37n^3 - 398n^2 + 27884.6n - 703133.48$(ms). The cubic fitting curve is nearer to the computation time than the quadratic fitting curve $T(n) = 574.553n^2 - 91951.24n + 3612315.83$.

5. Conclusions

The HGA is introduced with the two local optimization strategies for TSP. The four vertices and three lines inequality is used to convert the LHP[4]s into the LOHP[4]s in the HCs and the shorter HCs are obtained. The four vertices and three lines inequality is derived from the fact that the OHC is composed of the LOHPs. The tightened computation model for TSP is given based on the inequality. The LOHP[4] is convenient to compute and implement to generate an approximate OHC or the OHC. The second local optimization strategy is to reverse the LHPs with more than 2 vertices in the HCs to generate the shorter HCs. It is also convenient to apply whereas it is timely complex. The computation complexity of the two strategies is $O(n)$ and $O(n^3)$, respectively. The HGA is verified with the concrete TSP instances. The results show that it is competitive with the coordination of the two optimization strategies. For large scale of TSP instances, the computed results deviate from the OHC due to the fewer computational times of the second local optimization strategy. With personal computers, the two optimization strategies must be executed within a smaller number of computational cycles to reduce the computation time and the approximate solutions are generated.

It notes that the assignments of the parameters play an important role to affect the performance of HGA. In general, with bigger parameters listed in Table 6, the HGA converges to the better approximations. For example, the better solutions will be obtained with more computational times and population size. But for large scale of TSP instances, the computation time will be lengthened or the powerful computers will be necessary to generate the approximate OHC or OHC quickly. Comparing with the other algorithms (Bontoux et al., 2010; Liu and Zeng, 2009), the disadvantage of the algorithm is its time complexity to cause a long computation time. It will be improved in the future work.

With the floating numbers of cities' coordinates, the minor distinctions between the approximate OHC and the OHC are revealed although their integer values of length are identical. It is significant to discriminate which approximation is the actual OHC.

Acknowledgements

The work was supported by the NSFC (Grant No. 51205129). The work benefits from the facilities of National Key Laboratory of New Energy Power System and the Beijing Key Laboratory of New and Renewable Energy, North China Electric Power University, Beijing, China.

References

Berman, P., & Karpinski, M. (2006). 8/7-Approximation algorithm for (1,2)-TSP. In *SODA'06, Miami, FL, 2006* (pp. 641–648).
Bontoux, B., Artigues, C., & Feillet, D. (2010). A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers & Operations Research, 37*(11), 1844–1852.
Chen, S. M., & Chien, C. Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert System with Applications, 38*(12), 14439–14450.
Chen, W. N., Zhang, J., Chung, H. S. H., Zhong, W. L., Wu, W. G., & Shi, Y. H. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on Evolutionary Computation, 14*(2), 278–300.
Choi, I. C., Kim, S. I., & Kim, H. S. (2003). A genetic algorithm with a mixed region search for the asymmetric traveling salesman problem. *Computers & Operations Research, 30*(5), 773–786.
Climer, S., & Zhang, W. X. (2006). Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence, 170*(8–9), 714–738.
Črepinšek, M., Liu, S. H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *Computing Surveys (CSUR), ACM, 45*(3). http://dx.doi.org/10.1145/2480741.2480752.
Deineko, V., Klinz, B., & Woeginger, G. (2006). Four point conditions and exponential neighborhoods for symmetric TSP. In *Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithm, SODA 2006, Miami, FL, 2006* (pp. 544–553).
Ding, C., Cheng, Y., & He, M. (2007). Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale TSPs. *Tsinghua Science and Technology, 12*(4), 459–465.
Douglas, B. W. (2006). *Introduction to graph theory* (Section ed.). Beijing: Pearson Education Asia Limited and China Machine Press, pp. 74–78.
Gen, M., & Cheng, R. (1997). *Genetic algorithms & engineering design*. New York: John Wiley & Sons, pp. 82–89.
Ghaziri, H., & Osman, I. H. (2003). A neural network algorithm for the traveling salesman problem with backhauls. *Computers & Industrial Engineering, 44*(2), 267–281.
Goldberg, D., & Lingle, R. (1985). Alleles, loci, and the travelling salesman problem. In *Proceedings of the first international conference on genetic algorithms and their applications, Pittsburgh, USA, July 24–26, 1985* (pp. 154–159).
Helsgaun, K. (xxxx). An effective implementation of the Lin–Kernighan traveling salesman heuristic. <http://www2.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/>.
Holland, J. H. (1975). *Adaptation in nature and artificial system*. Cambridge (MA): MIT Press.
Ise website (2013). <http://www.ise.ncsu.edu/fangroup/ga.html>.
Johnson, D. S., & McGeoch, L. A. (2004). *The traveling salesman problem and its variations, combinatorial optimization*. London: Springer Press, pp. 445–487.
Liu, Y. H. (2008). Diversified local search strategy under scatter search framework for the probabilistic traveling salesman problem. *European Journal of Operational Research, 191*(2), 332–346.
Liu, Y. H. (2010). Different initial solution generators in genetic algorithms for solving the probabilistic traveling salesman problem. *Applied Mathematics and Computation, 216*(1), 125–137.
Liu, F., & Zeng, G. Z. (2009). Study of genetic algorithm with reinforcement learning to solve the TSP. *Expert System with Applications, 36*(3), 6995–7001.
Majumdar, J., & Bhunia, A. K. (2011). Genetic algorithm for asymmetric traveling salesman problem with imprecise travel times. *Journal of Computational and Applied Mathematics, 235*(9), 3063–3078.
Michalewicz, Z. (1994). *Genetic algorithms + data structures = evolution programs*. Berlin: Springer-Verlag, pp. 209–237.
Oliveto, P. S., & Witt, C. (2012). On the analysis of the simple genetic algorithm. In *Proceedings of the fourteenth international conference on genetic and evolutionary computation conference, Philadelphia, PA, USA, July 7–11, 2012* (pp. 1341–1348).
Reeves, C. (1994). Genetic algorithms and neighborhood search. *Evolutionary Computing, AISB Workshop, Lecture Notes in Computer Science, 865*, 115–130.
Rodríguez, A., & Ruiz, R. (2012). The effect of the asymmetry of road transportation networks on the traveling salesman problem. *Computers & Operations Research, 39*(7), 1566–1576.
Schmitt, L. J., & Amini, M. M. (1998). Performance characteristics of alternative genetic algorithmic approaches to the traveling salesman problem using path representation: An empirical study. *European Journal of Operational Research, 108*(3), 551–570.
Sharma, A., & Mehta, A. (2013). Review paper of various selection methods in genetic algorithm. *International journal of Advanced Research in Computer Science and Software Engineering, 3*(7), 1476–1479.
Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: A survey. *IEEE Journal on Computer, 27*(6), 11–26.
Verhoeven, M. G. A., Aarts, E. H. L., & Swinkels, P. C. J. (1995). A parallel 2-opt algorithm for the traveling salesman problem. *Future Generation Computer Systems, ll*(2), 175–182.
Wang, Y., & Liu, J. H. (2010). Chaotic particle swarm optimization for assembly sequence planning. *Robotics and Computer-Integrated Manufacturing, 26*(2), 212–222.
wiazda, T. D. G. (2006). *Genetic algorithms reference: Crossover for single-objective numerical optimization problems*. Berlin: Springer Press.
Xing, L. N., Chen, Y. W., Yang, K. W., Hou, F., Shen, X. S., & Cai, H. P. (2008). A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric traveling salesman problem. *Engineering Applications of Artificial Intelligence, 21*(8), 1370–1380.
Yu, X., & Gen, M. (2010). *Introduction to evolutional algorithms*. London: Springer-Verlag, pp. 288–298, pp. 285–287.
Zhang, W. X., & Korf, R. E. (1996). A study of complexity transitions on the asymmkric traveling salesman problem. *Artificial Intelligence, 81*(1–2), 223–239.