A Method to Search Traveling Salesman Problem Backbones Based on GA and Frequency Graph

Yong Wang*, Changxin Geng, Yiwen Wu

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

* Corresponding author. Tel.: +8613611023249; email: yongwang@ncepu.edu.cn

Manuscript submitted March 7, 2019; accepted May 20, 2019.

doi: 10.17706/jcp.14.6.426-437

Abstract: The traveling salesman problem (TSP) is a well-known NP-hard problem in combinatorial optimization. The search space of the optimal Hamiltonian cycle (OHC) is increasing exponentially according to the scale n . In order to reduce the search space of TSP, we first convert the complete graph K^n into the frequency graph (FG) with the approximate optimal n -vertex paths computed with an improved GA. The GA is improved with a four-vertex-three-line inequality to compute the approximate paths of TSP. Then, TSP backbones with $2n$ edges are derived from the FG. The experimental results show that the TSP backbones include at least 95% edges in the OHC. Based on the backbones, the search space of the OHC or approximations is greatly reduced.

Key words: TSP, Backbone, frequency graph, GA.

1. Introduction

As is well-known, traveling salesman problem (TSP) is a famous NP-hard problem [1], which has attracted a lot of attention from researchers in various fields such as mathematics, computer science, bioinformatics, operations research, supply chain, etc. Since TSP represents a class of problems in the field of combinatorial optimization, the exploration of its accurate or approximate solution has always been an important direction in the design of algorithm [2]. There are two main kinds of algorithms for solving TSP in applications. One is the exact algorithms including the dynamic programming, branch and bound algorithm, cutting plane, etc. [3]. The other is the intelligent optimization algorithms, such as the GA, ACO, SA, PSO, etc. [4].

Finding the optimal solution for a small-scale TSP is simple. However, it is very difficult, most of the times impossible, for the big scale of TSP whose search space of the optimal Hamiltonian cycle (OHC) is very large. Because of the constraint conditions, we can solve the big scale TSP problem by integer programming and dynamic programming, but strong computer equipment is needed and the computation time is too long. It has been reported that Concorde package resolved the big scale TSP with 85900 cities [5]. The task was completed using 128 networked computer systems and consumed 1.2 years. The computation time is equal to 136 years of computing tasks using a computer.

For symmetrical TSP, The time (or space) complexity of the graph search algorithm is $O((n-1)!)$. It is hard to find the OHC by traversing all the Hamiltonian cycles (HCs) for big scale of TSP. There are no exact methods that can find the global optimal solution for NP-hard problems in polynomial time. Therefore, some scholars turn to the study of the exact solution or approximate solution for the TSP on the sparse

graphs [6]. In a sparse graph, the search space of the original TSP can be significantly less than that in complete graphs. We call the sparse graphs containing the edges of OHC the TSP backbones.

In this research direction, Boese introduced the conception of backbones into the field of TSP in 1995 [7]. In 1998, Monasson discussed the backbones algorithm for the satisfiability problem (SAT) [8]. In 2003, Schneider found that the intersections of the local optimal solutions are similar to the TSP backbones [9]. Thus, the intersections of the local optimal solutions are regarded as the approximate backbones. Five kinds of local optimal algorithms including the random 2-opt, fast opt, fast 3-opt, LK and LSMC are tried by Professor Boese for the 532 vertices scale of TSP [10]. It is found that the intersections of the local optimal solutions computed by these algorithms occupy more than 80% edges in the globally optimal solution.

The previous backbones algorithms are used to solve TSP in recent years. The general backbones algorithms consist of two stages. In the first stage, the backbones are found from the complete graph. The ideal TSP backbones should contain as the least number of edges as possible but contain as many edges as possible in the OHC. In the second stage, the backbones algorithm is to find the optimal solution or approximations by finding the edge in the OHC and the approximations. The backbones algorithms are based on the big hole theory owing to Blum [11]. The theory has been proven by many experiments that there are many common edges between the global optimal solution and the local optimal solutions computed by the algorithms.

Based on big hole theory, several approximate solutions are computed with various algorithms. Then the edges among these approximations are enumerated. The frequency of an edge is the number of approximations containing this edge. Some of the intersected edges are preserved according to the frequency of the edges. After the enumeration of every edge from the approximations of TSP, the frequent graph (FG) is formed. In the FG, we can see the frequency of each edge in the different approximate solutions. A certain number of larger frequency edges are selected from FG, we can get the TSP backbones including the OHC or approximations. This is the basic idea of backbones principle based on FG. Since the backbones include small quantities of edges, the search space in the backbones is greatly reduced. The key problem is to compute a better backbone containing the OHC edges as many as possible.

There are many methods to compute TSP backbones. The previous methods generally compute the shortest distance of several vertices firstly, then the frequency of edges in these approximate solutions is computed. At last, the heuristic rules are used to compute the TSP backbones. these kinds of methods are characterized by fast speed but poor accuracy. If more vertices are selected to calculate the shortest distance, the complexity will be increased and the accuracy will be further improved. As the number of vertices increases, it is difficult to solve the shortest distance between them, so heuristic intelligent algorithm is needed.

Among the algorithms to compute the approximations, genetic algorithm (GA) is one of the most effective meta-heuristic methods, which is widely used in optimization problems [12]. GA simulates the natural selection of Darwinian evolution and the biological evolution of the genetic mechanism. GA was first proposed by professor JH Holland in 1975 [13]. The implicit parallelism of the algorithm promotes the global optimization performance [14]. According to the probability rules, the search direction can be adjusted automatically to the optimal solution. Compared with the traditional methods for TSP, GA has the good ability of convergence based on the evolutionary rules. Compared with the other heuristics, when the accuracy of solutions is required, it needs the smaller computing time and has the higher robustness.

Although GA has the good capacity of finding the optimal solution for global search and inherent parallelism, the local search ability is poor, which leads to the relatively time consuming and low efficiency in the late computation process. In practice, GA is apt to be premature and stagnates at the local optimum. The method of selection should not only preserve the good individuals but also maintain the diversity of the

group. This problem is usually confronted when we use the GA to resolve the complex problems.

Because the large amount of time is consumed to calculate the better approximations, the above methods use a limitative number of approximations to compute the TSP backbones. Unlike previous methods, a large number of approximate optimal n -vertex paths are computed with the improved GA. The GA is improved by incorporating the four-vertex-three-line inequality to compute the better approximate optimal n -vertex paths. Then, the frequency of edges is enumerated from these approximate optimal n -vertex paths. At last, the TSP backbones are computed based on the edge frequency.

The rest of the paper is organized as follows. The frequency graph is introduced in section 2. The conception of optimal k -vertex paths (OP^k) is given and the number of which is formulated. Section 3 describes the improved GA for computing the approximate optimal n -vertex paths of TSP. In section 4, we computed the frequency graph by approximate optimal n -vertex paths and the backbones are computed based on the frequency graph. In section 5, we shall do experiments to compute the backbones for some TSP instances in the TSPLIB. In the end, the conclusions are drawn as well as the future research directions are given.

2. The Frequency Graph for TSP

A TSP graph is generally represented as $G = (V, E)$. The vertex set is $V = (v_1, v_2, v_3, \dots, v_i, \dots, v_n)$ and the edge set is $E = (e_{1 \times 2}, e_{1 \times 3}, \dots, e_{(n-1) \times n})$. v_i ($1 \leq i \leq n$) represents the vertex and e_{ij} ($1 \leq i, j \leq n$) is the edge that links the two vertices v_i and v_j . A sparse graph G can be represented by an adjacent matrix $A(G) = \{a_{ij}\}$ ($1 \leq i, j \leq n$). If an edge e_{ij} is in the sparse graph, the element $a_{ij} = 1$. Otherwise, $a_{ij} = 0$. We devote to computing the sparse graph including as many as edges in OHC.

We firstly explain the relationships between the HCs and the k -vertex paths. For n -vertex TSP, an HC is represented as $HC = (v_1, v_2, v_3, \dots, v_i, \dots, v_n, v_1)$ whose two terminal vertices are identical. It is obviously that the HC is made up of some k -vertex paths (P). A k -vertex path can be represented as $P^k = (v_1, v_2, \dots, v_{k-1}, v_k)$ ($2 \leq k < n$). There are two terminal vertices at both ends and the other vertices are in the middle. The number of vertices in the P^k is noted with the superscript k . The P^k s in the OHC are different from most of the other P^k s. In general, they are much shorter than most of the other P^k s. We call the P^k s in the OHC as the optimal k -vertex paths (OP). Any one OP^k is represented as $OP^k = (v_1, v_2, \dots, v_{k-1}, v_k)$. For any one OP^k in the OHC, its two terminal vertices are determined. Furthermore, the OP^k is shorter than the other P^k s containing the same set of vertices and the two identical terminal vertices. Given a set of k vertices in a weight graph, there are $k \times (k-1)/2$ OP^k s if their two terminal vertices are determined. The weight of OP^k is minimal compared with the other P^k s with the same vertices and two terminal vertices.

Fig. 1. The illustration of OP^k in a weighted graph.

In Fig. 1, an OP^k is illustrate by a simple weighted graph (WG) with 5 vertices A–E where $k=5$. The weight of the edge is the number on the side. There are $5! / 2$ P^5 s composed of the five vertices and there are 10 OP^5 s for symmetric TSP. If we take the vertices A and E as the two terminal vertices, there are six P^5 s and the shortest P^5 is the OP^5 for vertices A and E. In Fig. 1, the weights of the six paths (A, B, C, D, E), (A, B, D, C, E), (A, C, B, D, E), (A, D, B, C, E), (A, C, D, B, E), (A, D, C, B, E) are 9 ,11, 6, 12, 12 and 16, respectively. Obviously, the path (A, C, B, D, E) is the OP^5 whereas the rest of the five paths are general P^5 s. Since the $OP^5 = (A, C, B, D, E)$ is shortest, it has the potential to be included in the OHC.

The number of P^k s is computed as equation (1) for symmetrical TSP.

$$N_{P^k} = \frac{k!}{2} C_n^k \quad (1)$$

The number of OP^k s is computed as formula (2).

$$N_{OP^k} = \frac{k \times (k - 1)}{2} C_n^k \quad (2)$$

where C_n^k in equation (2) is the combination number in the case that k vertices are selected from n vertices. The number of OP^k is much smaller than the number of P^k s. The number of OP^k s rises first if $k \leq n/2+1$ for even number n and $k \leq (n \pm 1)/2+1$ for odd number n . Then, it decreases according to k . There is a maximal number when $k=n/2+1$ for even number n ($k= (n \pm 1)/2+1$ for odd number n). For symmetrical TSP, the total number of OP^k is computed as formula (3).

$$N = \sum_{k=2}^n N_{OP^k} = \frac{1}{2} \sum_{k=2}^n k \times (k - 1) C_n^k = n(n - 1)2^{n-3} \quad (3)$$

As the number of vertices is creasing, the total number of OP^k increases exponentially according to n . The P^k s excluding from the OP^k s are the non- OP^k s (NOP). They will become shorter if we exchange their intermediate vertices. It is clear that the number of NOPs is much bigger than that of the OP^k s once k is bigger than 4. As we know, the NOPs do not belong to OHC.

Given a number of OP^k s, the OP^k s have the potential to form OHC as well as the edges in OP^k s. However, there is no doubt that not all edges are belonging to the OHC. Therefore, we need to find a way to distinguish the edges in OHC from the other edges. There is an applicable method to screen out them based on the frequency of the edges.

The OHC is made up of some edges. We pay more attention to the frequency of edges computed with OP^k s. An FG is computed by a set of P^k s or OP^k s. The frequencies of edges are much distinctive due to the selection of OP^k s. The FG is computed as follows [6]. Given a set of P^k s (or OP^k s) and $k>2$, the frequency of each edge is computed from them. The frequency of edge is the number of the P^k s (or OP^k s) containing this edge.

The edges of OHC are concerned. Given a set of OP^k s with more than 2 vertices, the frequencies of the edges are enumerated and an FG is computed. In an FG, the numbers on the edges are their frequencies enumerated from the set of OP^k s. If the FG is computed with the OP^k s, the edges with the big frequency are regarded. Based on the frequency of edges, most of the useless edges will be ignored as one chooses the OHC edges from the FG. In this paper, these selected edges are taken as part of the TSP backbones, in which there's a big proportion of OHC edges.

Therefore, the big frequency edges are taken to be the OHC candidate edges. In an ideal case, the edges

included in all of the OP^k s belong to OHC and the frequencies of them are maximal. The sparse graph includes all the OHC edges which is also the TSP backbones. The approximate solution or accurate solution can be found in the sparse graph. However, not all edges of big frequency belong to OHC due to selection of the various OP^k s. Therefore, we aim to compute the sparse graphs that contain the most number of OHC edges and are tried to preserve the least number of useless edges.

3. FG computed with the Improved GA

In this section, the general GA is improved with the four-point three-line inequality to compute the OP^n s or approximate OP^n s to compute the FG. Since computing the OP^n s is as hard as computing the OHC, the GA is improved with a four-vertex-three-line inequality to compute the OP^n s or approximate OP^n s. The approximate OP^n s also have many intersections of edges with the OHC if the approximate OP^n s approach the OHC. The frequency of the OHC edges will be much bigger than most of the other edges if they are computed with the approximate OP^n s. The procedure of the improved GA to compute the OP^n s or approximate OP^n s is shown in Fig. 2. Besides the five operations of the chromosome coding, crossover, variation, evaluation and selection in the traditional GA, the four-point three-line local optimization strategy is also included.

Fig. 2. The improved GA to compute FG.

The implementation of each operation in the improved GA is described as follows.

Step 1: Chromosome Encoding. The vertices are noted as integers. Each digit represents a vertex. A group of sequences containing the vertices are generated randomly and they are taken as the population of GA. Each sequence represents a Hamiltonian path of TSP. The number of iterations (N) and the number of the population are given at this stage.

Step 2: Fitness Evaluation. The objective function of TSP is the distance of the travelling tour. The reciprocal of a path length L is considered as the fitness function, i.e., $1/L$. The greater the fitness value of the individual path is, the shorter the individual path is and the better the path is. In GA, a better path (also

as the chromosome) is more likely to be selected to the next generation.

Step 3: Selection. Selection is the process of generating new population from a previous generation of individuals with strong vitality. According to fitness function, individuals (paths) with bigger fitness values are more likely to go into the next generation. The process is the imitation of natural selection. In this paper, the method of truncation selection is adopted to ensure the global convergence of the GA and the individual diversity of the population. Here the elitist strategy will be considered. In this process, the best path in the population is always guaranteed to pass to the next generation.

Step 4: Crossover. The so-called crossover operation means that two pairs of paths exchange some parts of their genes (path fragments) in order to form two new individuals. The paths exchange their segments with each other according to the cross probability. Two individuals in population are chosen at random firstly. Secondly, the exchanged partial paths in the two individuals are selected with the random method. Thirdly, the two individuals exchange the selected partial paths according to crossover probability to generating two new individuals.

Step 5: Mutation. The basic content of the mutation operation is to change some of the genes of the individual in the population. First, the mutation probability is determined according to the mutation probability. The vertices in different path positions are chosen at random. Then the vertices change their positions to form a new path.

Step 6: Local optimization. In the GA, after cross, mutation, and the evaluation of paths, there are still non-optimal local paths in the whole path. If we use the local optimization methods, these long local paths can become shorter so as to compute the better paths. To enhance the traditional GA, the four-point-three-line inequality is as the local optimization to compute better paths after the evaluation stage. The principle of the local search algorithm is illustrated in Fig. 3.

There are two paths (a) and (b) in Fig. 3. The path (a) contains the local path $(P_{i-1}, P_i, P_j, P_{j+1})$ whereas the path (b) includes the local path $(P_{i-1}, P_i, P_j, P_{j+1})$. Each local path is composed of four vertices and three lines where $l_{i \times j}$ is the length of edge $e_{i \times j}$ between the vertices P_i and P_j ($2 \leq i \leq n, 1 \leq j \leq n-1$). Except for the two local paths, we assume the paths (a) and (b) contain the other identical local path and the length is noted as L_{rest} . Provided that the path (b) is the optimal solution it's obvious that the inequality $l_{(i-1) \times j} + l_{i \times j} + l_{j \times (j+1)} \geq l_{(i-1) \times i} + l_{i \times j} + l_{j \times (j+1)}$ holds. This is the four-point-three-line inequality. According to the inequality, we can exchange the positions of vertices P_i and P_j in path (a) to compute the better path (b).

Fig. 3. The principle of four-vertex- three-line inequality.

According to this principle, we can convert all the local paths containing 4 consecutive vertices into the shorter local paths so that the better paths will be computed. The four-point-three-line inequality is easy to

implement and the computation time is $O(n)$, which will improve the ability of GA to find the better results. From the experimental results, it is found that the improved GA improves the convergent speed and reduces the computational time. Moreover, the generated paths are more accurate for computing the frequency graph. The algorithm of the improved GA is given in Table 1 and Table 2.

The procedure of the improved GA with local search algorithm is presented in Algorithm 1.

Table 1. The Improved GA

Algorithm 1: improved GA for finding optimal or near-optimal OP ⁿ s
01: Input the initialization parameters, population number P, iteration number N.
02: Generate the initialize population, initialize $i = 1$
03: for $i \leq N$ do
04: Compute fitness of each chromosome and evaluation the population.
05: Perform the selection operation to choose a set of good individuals.
06: Perform crossover operation to generate hybrid individuals.
07: Perform mutation operation to generate mutated individuals.
08: Perform four-vertex-three-line inequality to compute better individuals.
09: End for
10: Output the solutions.

In summary, the FG is computed with the OPⁿs or approximate OPⁿs computed with the improved GA. The procedure is given in Table 2.

Firstly, two end vertices are chosen as the end vertices of a Pⁿs. The approximate optimal OPⁿs between the two vertices is computed by the improved GA. For n-vertex scale of TSP, not all pairs of vertices are used. Instead, we choose the n pairs of them to compute n OPⁿs or approximate OPⁿs, which greatly reduces the amount of calculation time.

Secondly, the frequency of each edge is enumerated from the n the OPⁿs or approximate OPⁿs. After the FG is obtained, the 2n edges with the biggest frequency are drawn from FG to construct a sparse graph as the backbones of the TSP.

Table 2. The Algorithm to Compute FG

Algorithm 2: computing FG based on P ^k s
01: While (termination criteria not satisfied) do
02: Generate two terminal vertices
03: Search the approximate OP ⁿ s based on improved GA
04: End while
05: Compute the frequency of every edge
06: Select the candidate edges to form an FG
07: Output the FG

4. TSP Backbones from FG

Backbone is a conception from statistical physics. Simply, backbone is the same part or structure as the optimal solution. Backbone is a very effective concept for NP-hard problems in combinatorial optimization. It is of great significance to measure the difficulty of hard problem. Slaney studied the relationship between backbones and the difficulty or cost of search for resolving hard problems. He found that the backbones were usually used to develop heuristic algorithms for resolving the combinatorial optimization problems [15].

For TSP, the backbone contains most of the OHC edges. The size of a backbone is the number of edges in a

sparse graph. The size significantly affects the difficulty of TSP. The larger the size is, the more difficult the algorithm is to compute the global optimal solution of TSP.

Since the backbones include a small number of edges, the search space of TSP is greatly reduced. Getting a better backbone for TSP becomes the research hot spot in recent years [16]. A good backbone generally contains the OHC edges as most as possible and the other edges as least as possible. Since it is usually difficult to obtain the OHC, many researchers try to compute backbones by indirect methods such as limit crossing [17]. However, the drawback is that it takes a long time and the method is not applicable in practice for big scale of TSP.

Given a TSP instance, we aim to save the OHC edges with a small computation cost. Here the FG is taken as the basis to figure out backbones of TSP. In an FG, the edges are sorted according to their frequencies from big to small values. Then the edges with big frequency are preserved as the OHC candidate edges to form a TSP backbone. After an FG is computed, this method is efficient. For example, the sorting algorithm requires $O(n^2 \log n)$ time. If k^n edges with big frequency are maintained, the time complexity is $O(n)$. Thus, the computation of TSP backbone needs $O(n^2 \log n)$ computation time. Since the FG is computed with the approximate OP^n s, the OHC will have big frequency and most of them will be preserved in the backbone.

In Fig. 4. (a), a 10-vertex TSP is used to illustrate the performance of FG computed with approximate OP^n s and conclude the backbone. The ten points are selected from the 31 main cities in China. The black lines represent the edges in complete graph and the green dash lines are the edges in OHC. Firstly, 10 pairs of terminal vertices are selected at random. Secondly, the approximate OP^n s is computed based on the improved GA. In the improved GA, the number of population is 30, the number of iterations is 200, the fitness value is 2, and the probability of crossover and mutation is both 0.4. The frequencies of the edges are enumerated from the 10 approximate OP^n s. The FG is computed with these approximate OP^n s. Ten edges with the highest frequency are selected from the FG to form the backbone as Fig. 4. (b). The number on an edge is the frequency in FG. The frequencies of the other edges are smaller than those of the ten edges in Fig. 4. (b). For evaluating the performance of the FG, we compute the OHC with the exact branch and bound method. It found that the backbone was just the OHC of the 10-vertex instance. The simple example displays the approximate OP^n s really have many intersected edges with the OHC. Therefore, it is feasible to compute a backbone of TSP from the FG computed with the approximate OP^n s.

Fig. 4. (a) A 10-vertex TSP instance, (b) The backbone of the 10-vertex TSP.

For small scale TSP, the OP^n s is apt to compute. The OHC can be obtained by selecting the ten edges of the highest frequency. With the vertex number increasing, the number of OP^n s will increase rapidly. Brutal method is unable to compute the OP^n s. To compute a good TSP backbone, the approximate OP^n s are computed with the improved GA and they are used to compute the FG. As the approximate OP^n s have many intersections with the OHC, the backbones derived from the FG will contain many OHC edges. It mentions

that the backbone quality depends on the approximate OP^n s computed with the improved GA. The nearer the approximate OP^n s approach the OHC, the better the performance of the FG is and the smaller the search space of TSP is on the backbone.

5. Example and Result

Some Euclidean TSP instances are used to validate the method. These TSP instances are downloaded from TSPLIB. The backbones containing a small number of edges are computed and the search space of OHC is significantly reduced. For comparison, the OHC of these instances are computed with Concorde package online owing to Mittelmann (NEOS Server for Concorde). The OHC edges are used to display how many OHC edges are contained in the backbones. The computation program is coded with MATLAB and the program is run on a DELL computer with 2.33GHz processor and 4G memory.

For each TSP instance, two vertices are randomly selected firstly as the terminal vertices. Then an approximate OP^n is computed with the improved GA. In this way, total n OP^n s are computed for computing an FG. Then, based on the FG, $2n$ edges with the biggest frequencies are selected to compose the TSP backbones. The experimental results are shown in Table 3. For these TSP instances, the backbones contain nearly all of the OHC edges and only a few OHC edges are lost. The backbones contain more than 95% of the OHC edges for these TSP instances. It means the improved GA computes the good approximate OP^n s. The OHC edges have the bigger frequencies than most of the other as they are computed with these approximate OP^n s. As we maintain more edges with the big frequency, less OHC edges will be lost in the backbones.

To show the advantage of using approximate OP^n s to compute the backbones, the OP^4 s are used to compute the backbones for comparisons. For each TSP instance, 600,000 OP^4 s are chosen at random to compute the frequency of edges at first. Then the $2n$ edges with the biggest frequencies are chosen to compose a sparse graph. The results illustrated that the sparse graphs lose a big portion of the OHC edges, see Table 3. It says the approximate OP^n s are better as a means to compute the backbones than the OP^4 s for TSP.

Table 3. The Results of the Sparse Graphs

Instances	n	Edges	Ratio	Iterations	Edges not in $OHC(OP^n)$	Proportion	Edges not in $OHC(OP^4)$	Ratio
att48	48	96	100	300	1	2.08%	5	9.62%
berlin52	52	104	100	600	0	0.00%	2	2.86%
st70	70	140	100	600	0	0.00%	9	12.86%
pr76	76	152	150	600	2	2.63%	14	18.42%
eil76	76	152	150	800	1	1.32%	16	21.05%
kroC100	100	200	200	800	2	2.00%	18	18.00%
kroA100	100	200	200	800	5	5.00%	24	24.00%
eil101	101	202	150	800	4	3.96%	34	33.66%
pr124	124	248	200	800	4	3.23%	23	18.55%
ch150	150	300	200	1200	6	4.00%	48	32.00%
pr152	152	304	200	1200	5	3.29%	34	22.37%

Fig. 5. The backbones of kroC100, kroA100, eil101, ch150.

In Fig. 5, the backbones and the OHC of several typical examples are shown. The dotted green lines are the OHC of TSP and the solid black lines are edges in the backbones. One can find that the backbones include the majority edges in OHC. According to the experimental results, the TSP backbones can be selected by this method effectively, and the search space of OHC can be reduced. In the above method, we select $2n$ edges in the backbones. If more edges are selected in TSP backbones, the portion of the edges in OHC will be bigger.

6. Conclusions and Future Work

TSP backbones are useful to reduce the computation time of TSP algorithms. The FG is taken as the basis to derive the TSP backbones according to the frequency of edges. The frequency of the OHC edges is generally much bigger than that of most of the other edges in an FG owing to the approximate OP^n s. The GA is improved with a four-vertex-three-line inequality to compute the approximate OP^n s. The experimental results for several TSP instances show that the backbones contain more than 95% edges in the OHC and they just contain $2n$ edges. We also compute the TSP backbones with a lot of OP^4 s. The backbones computed with the approximate OP^n s are much better than the results computed based on OP^4 s. According to the backbones, the search space of TSP can be reduced greatly. In the near future, two problems will be researched. Firstly, the method will apply to the big scale and other types of TSP instances to generate their backbones. Secondly, the OHC or approximations of TSP will be studied in the backbones.

Acknowledgment

The authors acknowledge the funds supported by the Fundamental Research Funds for the Central Universities (No.2018ZD09 and 2018MS039).

References

- [1] Mohiddinshaw, S., & Gurram, D. (2015). A note on computational approach to travelling sales man problem. *International Journal of Computer Applications*, 115(8), 975-8887.
- [2] Kuo, I. H., Horng, S. J., Kao, T. W., Lin, T. L., Lee, C. L., Chen, Y. H., Pan, Y. I., & Terano, T. (2010). A hybrid swarm intelligence algorithm for the travelling salesman problem. *Expert Systems*, 27(3), 166–179.
- [3] Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the TSP. *Operations Research*, 21(2), 498-516.
- [4] Pan, G., Li, K., Ouyang, A., & Li, K. (2016). Hybrid immune algorithm based on greedy algorithm and delete-cross operator for solving TSP. *Soft Computing*, 20(2), 555-566.
- [5] Lenstra, J. K., & Shmoys, D. B. (2009). Book review: The traveling salesman problem, a computational study (by D. L. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook). *Siam Review*, 799-801.
- [6] Wang, Y. (2015). An approximate method to compute a sparse graph for traveling salesman problem. *Expert Systems with Applications*, 42(12), 5150-5162.
- [7] Boese, K. D., Kahng, A. B., & Muddu, S. (1994). A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2), 101-113.
- [8] Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., & Troyansky, L. (1999). Determining computational complexity from characteristic 'phase transitions.' *Nature*, 400(6740), 133-137.
- [9] Schneider, J. (2003). Searching for backbones—A high-performance parallel algorithm for solving combinatorial optimization problems. *Future Generation Computer Systems*, 19(1), 121-131.
- [10] Boese, K. D., Kahng, A. B., & Tsay, R. S. (1994). Scan chain optimization: Heuristic and optimal solutions. *Internal Report UCLA Cs Dept.*
- [11] Blum, H. (1967). A transformation for extracting new descriptors of shape. *Models for the Preception of Speech & Visual Form*, 19, 362-380.
- [12] Holland, J. H. (1987). Genetic algorithms and classifier systems: Foundations and future directions. *Proceedings of International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*.
- [13] Holland, J. H. (1973). Erratum: Genetic algorithms and the optimal allocation of trials. *Siam Journal on Computing*, 2(2), 88-105.
- [14] Anderson-Cook, C. M. (1998). Practical genetic algorithms. *Publications of the American Statistical Association*, 100(471), 1099-1099.
- [15] Slaney, J., & Walsh, T. (2001). Backbones in optimization and approximation. *IN IJCAI-01*, 254-259.
- [16] Zhang, W., & Looks, M. (2005). A novel local search algorithm for the traveling salesman problem that exploits backbones. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*. Edinburgh, Scotland, UK
- [17] Zhang, W. X. (2004). Phase transitions and backbones of the asymmetric traveling salesman problem. *Journal of Artificial Intelligence Research*, 21, 471-497.

Yong Wang is an associate professor at the North China Electric Power University, Beijing. He obtained his Ph.D degree on manufacturing engineering of aeronautics and astronautics at Beihang University, Beijing, China, the master degree on vehicle engineering at Beihang University, Beijing, China, and the bachelor degree on manufacturing engineering of

aircraft at Nanjing University of Aeronautics and Astronautics, Nanjing, China. His research interests include combinatorial optimization, graph algorithms and applications in industrial engineering.

Changxin Geng was born in China in 1993. She received the B.S degree from North China Electric Power University (NCEPU), Beijing, China, in 2016. His main areas of research interest are combinatorial optimization, machines learning and intelligent algorithms. Mr. Geng is a member of the State Key Laboratory Of Alternate Electrical Power System With Renewable Energy Sources in NCEPU.

Yiwen Wu was born in China in 1994. She received the B.S. degree from North China Electric Power University, Beijing, China, in 2017. Her main areas of research interest are collection lines optimization, intelligent algorithms, and travelling salesman problem. Since September 2017. Miss Wu is studying for a MS degree at North China Electric Power University, Beijing, China.