An Improved Adaptive Genetic Algorithm for Mobile Robot Path Planning Analogous to the Ordered Clustered TSP

Junjie Jiang
School of Mechanical and Automobile Engineering South China University of Technology
Guangzhou, China
304796768@qq.com

Xifan Yao
School of Mechanical and Automobile Engineering South China University of Technology
Guangzhou, China
mexfyao@scut.edu.cn

Erfu Yang
Department of Design, Manufacturing and Engineering Management University of Strathclyde
Glasgow, U.K
erfu.yang@strath.ac.uk

Jorn Mehnen
Department of Design, Manufacturing and Engineering Management University of Strathclyde
Glasgow, U.K
jorn.mehnen@strath.ac.uk

Hongnian Yu
School of Engineering and The Built Environment Edinburgh Napier University
Edinburgh, U.K
H.Yu@napier.ac.uk

***Abstract*—The material transportation planning with a mobile robot can be regarded as the ordered clustered traveling salesman problem. To solve such problems with different priorities at stations, an improved adaptive genetic simulated annealing algorithm is proposed. Firstly, the priority matrix is defined according to station priorities. Based on standard genetic algorithm, the generating strategy of the initial population is improved to prevent the emergence of non-feasible solutions, and an improved adaptive operator is introduced to improve the population ability for escaping local optimal solutions and avoid premature phenomena. Moreover, to speed up the convergence of the proposed algorithm, the simulated annealing strategy is utilized in mutation operations. The experimental results indicate that the proposed algorithm has the characteristics of strong ability to avoid local optima and the faster convergence speed.**

Keywords—clustered traveling salesman problem, genetic algorithm, simulated annealing, path planning, mobile robot

I. INTRODUCTION

Nowadays, mobile robots such as Automated Guided Vehicles (AGVs) are increasingly playing the role of material transportation in large factories [1]. Since AGVs were introduced in the 1950s, over the past decades, today's AGV guidance technology has evolved from electromagnetic guidance into laser and visual guidance. Such breakthrough makes AGVs no longer be limited by magnetic track, and therefore have more freedom and flexibility to move in the factory floors. As a result, AGV path planning problems have arisen and become a research hotspot, which attracts many researchers' interest in scientific research.

AGV/Mobile robot-path planning aims at optimizing one or more indicators (e.g., path length, elapsed time, cost) to quickly find an optimal route among many alternative non-collision paths [2]. A common problem is planning the shortest route, which connects each station, so that AGVs can supply material for each station along the road during the movement. In a factory, the order of each station's demand for materials strictly depends on the process corresponding to the station, so it is more reasonable for AGVs to transport materials according to the process. Thus, stations should be assigned different priorities and AGV should traverse each station according to its priority. Such a considered problem is called the ordered Clustered Traveling Salesman Problem (OCTSP), which is a NP-Hard problem. The problem can be defined as follows: let $G = (V, E)$ be a complete undirected graph with vertex set V and edge set E. The vertex set $V = \{v_1, v_2, \dots, v_n\}$, except the starting vertex v_1, is partitioned into m prespecified clusters $V_1, V_2, \dots, V_m$. The number of vertices in the clusters is $n_1, n_2, \dots, n_m$, respectively. $C = [c_{ij}]$ is the cost matrix, representing travel costs, distances, or travel times, defined on the edge set $E = \{(v_i, v_j): v_i, v_j \in V, i \neq j\}$. The objective of OCTSP is to determine the least cost Hamiltonian tour, which starts from v_1 and visits all vertices of any cluster V_k contiguously in the order $V_1, V_2, \dots, V_m$.

We seek approximate solution using optimization algorithm for OCTSP. To solving the traveling salesman problem (TSP) and related problems, well-known algorithms are utilized, such as genetic algorithm (GA) [3], tabu search (TS) [4], ant colony optimization (ACO) [5], simulated annealing (SA) [6], particle swarm optimization (PSO) [7], firefly algorithm (FA) [8] and bacterial evolutionary algorithm (BEA) [9]. Amongst these algorithms, GA is found to be the best algorithms for the TSP and its variations. GA is a relatively simple and practical algorithm, which imitates the mechanisms of genetics including selection, crossover and mutation operators. It initially operates on individuals from a randomly generated population to gradually improve the fitness of individuals, and eventually gets the best individual and find the optimal solution to the problem. However, GA has the slow convergence speed and easily falls into local optimal solutions in practical use. The principle of SA is randomly searching in the search space, iterating for several times, and gradually converging to the optimal solution by setting the initial temperature, final temperature, annealing temperature function, Markov chain length, etc. Its greatest advantage is that the global optimal solution is more likely to be obtained, and its convergence speed is faster than GA. Moreover, SA has strong robustness. Since GA is easy to integrate with other algorithms, the

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

combination of GA and SA can help to obtain the advantages of the both algorithms. Therefore, we propose an improved adaptive genetic simulated annealing algorithm (IAGSAA), which is based on standard genetic algorithm (SGA), and improved by generating initial population to avoid non-feasible solutions, and in which the improved adaptive operator is introduced to improve the ability of the algorithm to jump out from local optimal solutions and resistance power to the destruction of excellent individuals in each generation at the same time. Moreover, the simulated annealing mutation strategy is introduced into the GA mutation operation to improve the convergence speed of the proposed algorithm.

The remainder of this paper is organized as follows. Section II presents research work on the clustered traveling salesman problem (CTSP) and other related problems. Section III describes the mathematical model and the proposed algorithm for the OCTSP. The simulation results and analysis are presented in Section IV. And a summary conclusion is given in Section V.

II. Related Work

Chisman [10], who first introduced CTSP, applied branch and bound approach to solve the above problem exactly after transforming CTSP to TSP. However, results obtained were not good. Thereafter, exact and approximation algorithms were developed to find optimal solutions. Jongens and Volgenant [11] proposed an exact algorithm, which was an extensive adaptation of an existing TSP algorithm based on the 1-tree relaxation combined with a new multiplier in the Lagrange approach, and finally a heuristic was developed to find exact solutions. Laporte, et al [4] further proposed a TS heuristic that combined with genetic diversification for the problem with clusters visited in a prespecified order, but it required more computational time. Anily, et al [12] adapted Christofides' Heuristic to develop an approximation algorithm of 5/3 performance ratio for the OCTSP. Ding, et al [13] proposed a two-level GA (TLGA) for solving CTSP. In the lower level, GA finds the shortest Hamiltonian cycle for each cluster. In the higher level, an improved GA is designed to determine an edge that will be deleted from the shortest Hamiltonian cycle for each cluster and the visiting sequence of all clusters with the objective of the shortest path length. Results demonstrate that the TLGA for large TSPs is more efficient than GA. To obtain exact optimal solution to OCTSP, Ahmed [14] developed a lexisearch algorithm (LSA), which was tested with some small sized asymmetric and symmetric instances, and exact solutions were obtained efficiently, but the algorithm is not efficient for large sized instances. Moreover, a hybrid genetic algorithm (HGA) [15] using sequential constructive crossover, 2-opt search, a local search and an immigration method was proposed for OCTSP, which was efficient in producing high quality of solution for the benchmark instances and better than LSA. Mestria [16] proposed a new hybrid heuristic algorithm, i.e., VNRDGILS algorithm, based on iterated metaheuristics, which used several variable neighborhood structures that selected in a random order. Using local search operators and diversification with a constructive heuristic and a perturbation method helps to intensify the search. It's reported that the VNRDGILS algorithm is more competitive in computational time and capable to obtain better results than the hybrid heuristic that only uses variable neighborhood descent.

III. Design of the Proposed Algorithm

A. Mathematical Model construction

Analogous to OCTSP, we assume that the total number of stations in the factory is n, the number of priorities, i.e., clusters is m, and the corresponding number of stations in each cluster is $n_1, n_2, \dots, n_m$, so the equation can be described as follows:

$$n = \sum_{i=1}^{m} n_i \ (n_1 = 1) \tag{1}$$

It is assumed that the smaller the station priority value is, the higher the station priority is. So, AGV must firstly traverse station v_1 in priority 1, then stations in priorities 2,3,... ,m, respectively. Therefore, a feasible travel path is described as: $P = (v_{1,1}, v_{2,1}, \dots, v_{2,n_2}, v_{3,1}, \dots, v_{3,n_3}, \dots, v_{m,1}, \dots, v_{m,n_m}, v_{1,1})$. We assume that the cost $c_{ij} = d(v_i, v_j)$. Then, the path length function of AGV traversing all the stations is defined as follows:

$$f(P) = \sum_{i=2}^{m} \sum_{j=1}^{n_i - 1} d(v_{i,j}, v_{i,j+1}) + \sum_{i=1}^{m-1} d(v_{i,n_i}, v_{i+1,1}) + d(v_{m,n_m}, v_{1,1}) \tag{2}$$

where, $v_{i,j}$ is the station number, representing the jth station in priority i; and $d(v_{i,j}, v_{i,j+1})$ is the distance between station $v_{i,j}$ and station $v_{i,j+1}$. The optimal solution should make (2) obtain the minimum value.

B. Defining Priority Matrix

We define the priority matrix as follows:

$$G = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} & a_{m-1,n} \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n-1} & a_{m,n} \end{bmatrix} \tag{3}$$

where G is a $m \times n$ matrix, representing totally n stations divided into m priorities. The value of each element in matrix G is 0 or 1, whose element $a_{i,j}$ is defined as follows:

$$a_{i,j} = \begin{cases} 0 & \text{if station } v_j \notin \text{priority } i \\ 1 & \text{if station } v_j \in \text{priority } i \end{cases} \tag{4}$$

so, in each column of matrix G, only one element is 1, and the rest are all 0. Since only station v_1 is in priority 1, $a_{1,1} = 1$ and

$$a_{i,1} = 0 \quad (2 \le i \le m) \tag{5}$$

C. Preliminary Sorting

Before sorting, the travel path is as same as the station number sequence. Suppose that the n-dimensional vector corresponding to the travel is $t = [1 \quad 2 \quad \cdots \quad n]$ and priority i contains n_i stations. We can preliminarily sort stations numbered 1 to n according to the priority matrix, sorting steps are described as follows:

Step 1: Define m zero vectors $t_i = [0 \quad 0 \quad \cdots \quad 0](i = 1,2,\dots,m)$, so the dimension of t_i is n_i.

Step 2: Successively examine elements of jth column of matrix G, if $a_{i,j} = 1$, then assign the jth element of vector t to vector t_i.

Step 3: Define a new n-dimensional vector t_{new} ($t_{new} = [t_1\ t_2\ \dots\ t_m]$), so the order of elements of vector t_{new} is preliminarily sorted by the station priorities.

D. Fitness Function

We normalize the path length corresponding to the individuals in each generation of population and the normalized value is used to represent the fitness value of the individual. Then, the fitness function can be defined as follows:

$$f_i = (maxl - l_i)/(maxl - minl + a) \tag{6}$$

where l_i represents the path length corresponding to the ith individual; $maxl$ and $minl$ respectively represent the longest and shortest paths in this generation of population; and the parameter a is an extremely small positive number, which prevents the denominator of (6) from being 0 later in the iteration. In order to minimize the impact of a on the calculation of fitness, it should be ensured that the value of a is 4 or 5 orders of magnitude smaller than that of $(maxl - minl)$. Later in the iteration, the value of $(maxl - minl)$ usually has an order of magnitude, unless it is 0. Therefore, we set $a = 0.0001$ in the proposed algorithm. The shorter the individual's corresponding path is, the larger the fitness value is and the higher the survival probability is. The fitness value calculated from (6) is ranged in (0, 1).

E. Generating Partitioned Initial Populations

In GA, the initial population is randomly generated. While, the generation criteria of the initial population will be adjusted in OCTSP. As we have divided vector t_{new} into m subvectors, the priorities of stations corresponding to the elements in different subvectors are different. The elements in each subvector are respectively and randomly sorted to generate the initial population for ensuring that the generated initial population still satisfies the priority requirement. Thus, the generated initial population is partitioned and the generating process is shown in Fig. 1, in which there are 11 stations that are divided into four priorities.

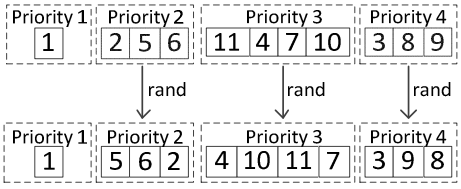


Fig. 1. Generating partitioned initial population.

F. Genetic Operator Design

a) Selection Operator

The individuals of larger fitness values will be retained, while those of smaller fitness values will be discarded in each generation. The larger the individual fitness value is, the higher the probability of being retained is. Suppose that the fitness values of 1-10 individuals in a generation are 0.4, 0.27, 0.53, 0.82, 0.68, 1, 0.45, 0.73, 0.96, and 0.14, respectively, individuals 4, 5, 6, 8, 9 will be retained and survived while the rest will be eliminated if the generated random number is 0.6.

b) Crossover and Mutation Operators

The crossover operation, which can improve the searching ability of the population, is to match chromosomes randomly and exchange some genes with a certain crossover probability p_c. In this paper, we adopt partitioned Partial Mapped Crossover (PMX), to ensure that the crossed gene positions belong to the same priority. The operation process is described as follows:

Step1: Randomly select two paternal chromosomes A and B.

Step2: Generate a random number m' from 2 to m and randomly find two adjacent gene positions in the m'th subvector, cross chromosomes A and B at the selected gene positions.

Step3: Modify the gene values outside the crossed gene positions according to the mapping relation of the crossed gene values in order that the same gene values do not appear on one chromosome. Therefore, generate two chromosomes A1 and B1.

As shown in Fig. 2, we use the chromosome segmentation in the previous section to illustrate the crossover process. Suppose $m' = 3$, the crossed gene positions are 2 and 3.

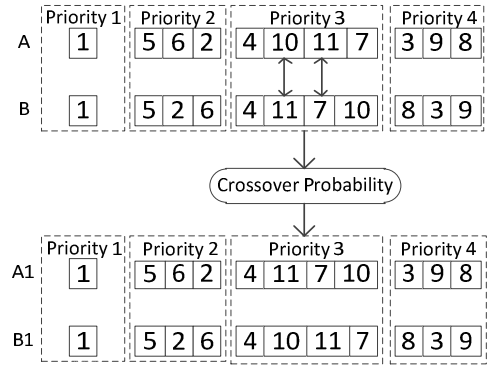


Fig. 2. Partitioned partially mapped crossover.

The mutation operation, which can improve population diversity and avoid premature phenomena in some degree, is to change one or more gene values of individuals in a population with a certain mutation probability p_m. As mentioned above, SA is better than GA in the convergence speed. In this paper, we take the treatment strategy of the deteriorating solution in SA to deal with the new individual generated after mutation operation in GA. Therefore, an improved mutation strategy with Simulated Annealing Mutation (SAM) is utilized. The improved strategy is a combination of partitioned two-point exchange mutation, insertion mutation and inversion mutation. The operation process is described as follows:

Step1: Generate a random number x from 0 to 1, if $x \leq 0.33$, then use exchange mutation; if $0.33 < x < 0.67$, then insertion mutation; if $x \geq 0.67$, then inversion mutation.

Step2: Select the m'th segment of chromosomes A1 and B1 at random, to generate two different random numbers $i_{m'1}$ and $i_{m'2}$ ($i_{m'1}, i_{m'2} \leq j_{m'}$, where $j_{m'}$ is the total number of genes on the m'th segment of chromosomes) on the m'th segment of chromosome, then perform mutation operation based on the crossover operation. Fig. 3 shows the mutation process illustration (suppose $i_{m'1} = 1$, $i_{m'2} = 4$ and $m' = 3$).

Step3: Calculate the fitness values of the two individuals before and after the mutation respectively. The mutation is accepted if the fitness value becomes larger. Otherwise, whether the mutation accepted or not is determined by a certain annealing probability p_t.

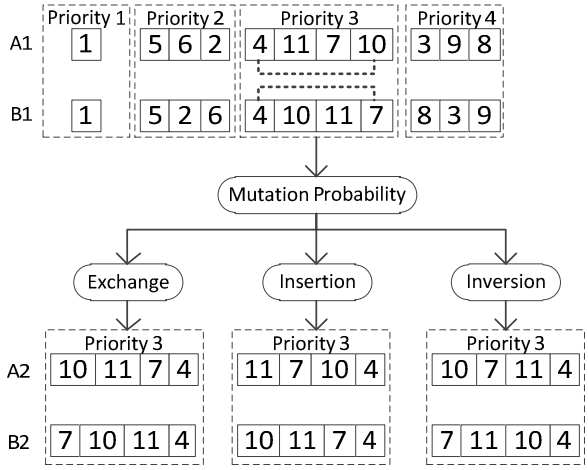


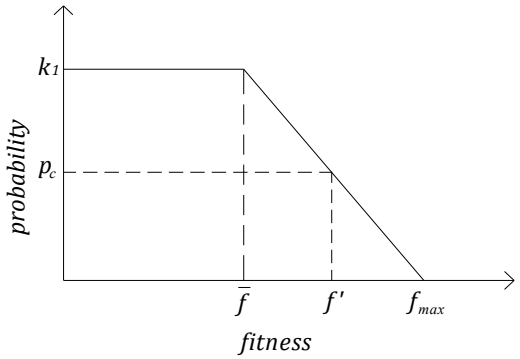
Fig. 3. Partitioned mutation process.

The crossover probability p_c and mutation probability p_m will directly affect the performance of the algorithm. In SGA, the values of p_c and p_m are fixed. The average fitness value of the population can be rapidly increased at the initial stage of the algorithm. However the better individuals will be destroyed at the later stage of the algorithm, leading to premature phenomena. In this paper, the values of crossover and mutation probabilities will change adaptively. Adapted genetic algorithm (AGA) originally was proposed by Srinivas and Patnaik [17] with the aim of increasing the crossover and mutation probabilities later in the iteration, so that the population can jump out of the local optimal solutions. The formulae for calculating the adaptive crossover and mutation probabilities given in [17] are as follows:

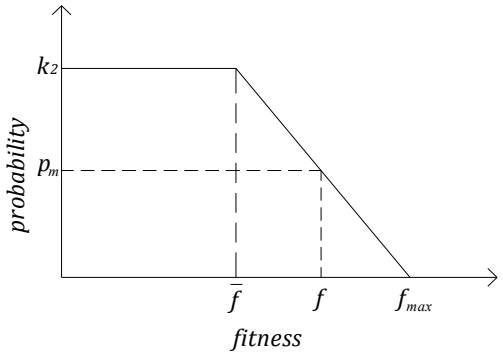
$$p_c = \begin{cases} \dfrac{k_1(f_{max} - f')}{f_{max} - \bar{f}}, f' \geq \bar{f} \\ k_3 \qquad\qquad , f' \leq \bar{f} \end{cases} \tag{7}$$

$$p_m = \begin{cases} \dfrac{k_2(f_{max} - f)}{f_{max} - \bar{f}}, f \geq \bar{f} \\ k_4 \qquad\qquad , f \leq \bar{f} \end{cases} \tag{8}$$

where f_{max} represents the maximum fitness value of the population; $\bar{f}$ represents the average fitness value of the population; f' is the larger of the fitness values of the individuals to be crossed; f is the fitness values of the individuals to be mutated; $k_1, k_2, k_3$and k_4 are parameters. The adjustment curves of the crossover and mutation probability corresponding to (7) and (8) are shown in Fig. 4.



(a) Adjustment curve of crossover probability.



(b) Adjustment curve of mutation probability.

Fig. 4. Adjustment curves of crossover and mutation probabilities in AGA.

According to Fig. 4, if the fitness value of the individual is smaller than the average value, higher crossover and mutation probabilities are used; and if the fitness value of the individual is larger than the average value, lower crossover and mutation probabilities are used. The probabilities vary with the change of the fitness values of individuals. However, the crossover and mutation probabilities of the optimal individual in the population calculated from (7) and (8) are 0. At the initial stage of the algorithm, even the individual with the largest fitness value is generally not the global optimal solution. Therefore, if the individual genes cannot be changed, inversely, they are retained so many that the algorithm is still likely to get stuck at the local optimum. To overcome this shortcoming, many improved formulae have been used to calculate the adaptive crossover and mutation probabilities. In addition, when there are more individuals, whose fitness values are near the average fitness values, in the population, they have advantages in the population evolution because the individual genes are such similar that poor effect of the subsequent evolution is resulted in. The crossover and mutation probabilities of the individuals, whose fitness values are near the maximum fitness value, are such different that some better individuals are more likely to be destroyed because of the relatively high crossover and mutation probabilities. To solve the problem, the adaptive adjustment curve in $\bar{f}$ and f_{max} should be flattened out. We adopt nonlinear adjustment as follows:

$$pc = \begin{cases} \sqrt{2}(pc_1 - pc_2) + pc_2 - (pc_1 - pc_2) \times sin\left(\dfrac{(f' - \bar{f}) \times \pi}{(f_{max} - \bar{f}) \times 2}\right), f' \geq \dfrac{\bar{f} + f_{max}}{2} \\ pc_2 + (pc_1 - pc_2) \times cos\left(\dfrac{(f' - \bar{f}) \times \pi}{(f_{max} - \bar{f}) \times 2}\right) \quad , \bar{f} \leq f' \leq \dfrac{\bar{f} + f_{max}}{2} \\ pc_1 \quad , f' \leq \bar{f} \end{cases} \tag{9}$$

$$pm = \begin{cases} \sqrt{2}(pm_1 - pm_2) + pm_2 - (pm_1 - pm_2) \times sin\left(\dfrac{(f - \bar{f}) \times \pi}{(f_{max} - \bar{f}) \times 2}\right), f \geq \dfrac{\bar{f} + f_{max}}{2} \\ pm_2 + (pm_1 - pm_2) \times cos\left(\dfrac{(f - \bar{f}) \times \pi}{(f_{max} - \bar{f}) \times 2}\right) \quad , \bar{f} \leq f \leq \dfrac{\bar{f} + f_{max}}{2} \\ pm_1 \quad , f \leq \bar{f} \end{cases} \tag{10}$$

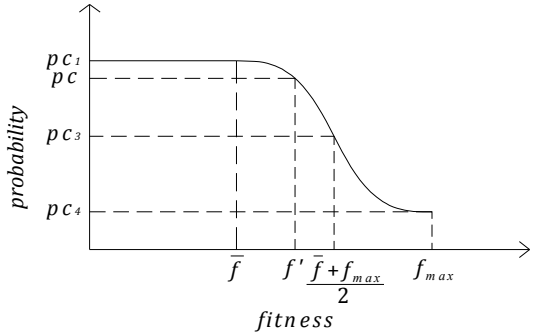
where f_{max}, $\bar{f}$ and f' have the same meanings as in (7) and (8); f is the smaller of the mutated individual fitness values; pc_1, pc_2, pm_1 and pm_2 are parameters. Respectively, pc_1 and pm_1 determine the maximum crossover and mutation probabilities; pc_1 and pc_2 codetermine the minimum crossover probability; pm_1 and pm_2 codetermine the minimum mutation probability. The mutation probability calculated from (10) can further protect the better individuals generated by the crossover operation. The improved adjustment curves of the crossover and mutation probabilities are shown in Fig. 5, where:

$$pc_3 = \sqrt{2}pc_1/2 + (2 - \sqrt{2})pc_2/2 \tag{11}$$

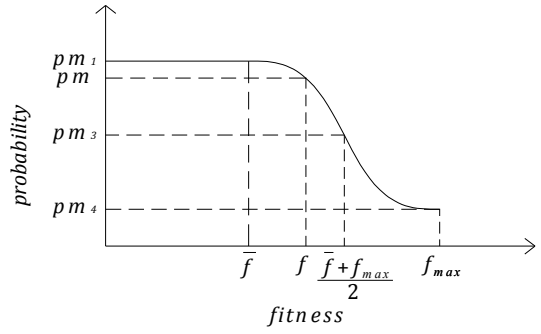
$$pc_4 = \left(\sqrt{2} - 1\right)pc_1 + \left(2 - \sqrt{2}\right)pc_2 \tag{12}$$

$$pm_3 = \sqrt{2}pm_1/2 + (2 - \sqrt{2})pm_2/2 \tag{13}$$

$$pm_4 = \left(\sqrt{2} - 1\right)pm_1 + \left(2 - \sqrt{2}\right)pm_2 \tag{14}$$



(a) Improved adjustment curve of crossover probability.



(b) Improved adjustment curve of mutation probability.

Fig. 5. Adjustment curves of crossover and mutation probabilities for this study.

The calculation formula of annealing probability pt and the annealing temperature function are as follows:

$$p_t = exp((f_{new} - f_{old})/T) \tag{15}$$

$$T(k+1) = K \times T(k) \tag{16}$$

where f_{old} and f_{new} are fitness values before and after individual mutation; T is a temperature parameter that varies with the number of iterations k, so $T(0)$ is the initial temperature; and K is the temperature attenuation parameter.

G. Terminal Condition

We set K_{max} as the maximum number of iterations. If $k \geq K_{max}$, then the iteration will be terminated and the optimal result will be output.

H. Improved algorithm flowchart

The whole algorithm flowchart mainly includes inputting the priority matrix, setting parameters, preliminary sorting, as well as selection, crossover and mutation operations in the improved genetic algorithm, as shown in Fig. 6.

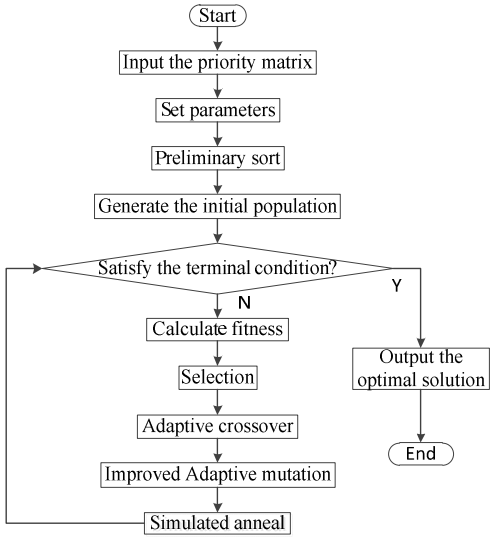


Fig. 6. The flowchart of the proposed algorithm.

IV. Simulation Experiments and Result Analysis

In order to verify the effectiveness of proposed IAGSAA in this paper, TSPLIB [18] instances are utilized for simulation experiments. Four algorithms (LSA [14], HGA [15], SGA and

IAGSAA) are compared for some symmetric instances. SGA and IAGSAA are run 30 times in MATLAB R2014a. Set 200 as the population quantity and 1500 as the maximum number of iterations. In SGA, p_c and p_m are set to 0.8 and 0.1, respectively. In IAGSAA, the initial temperature $T(0)$ is 0.31 and the temperature attenuation parameter K is 0.995. The values of other parameters in IAGSAA are shown in TABLE I.

TABLE I. PARAMETER SETTING

Algorithm	Parameter	Value
IAGSAA	pc_1	0.8
	pc_2	0.4
	pm_1	0.1
	pm_2	0.001

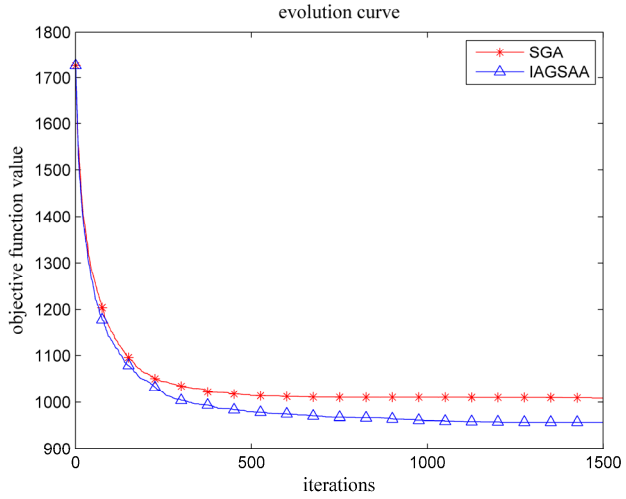
TABLE II shows the comparative study between LSA, HGA, SGA and IAGSAA. The first column is the instance name, the second column represents the clusters i.e., priorities. For example, an 11-vertex instance with two clusters (5, 5) means $V_1 = \{2,3,4,5,6\}$, $V_2 = \{7,8,9,10,11\}$, and V_1 is followed by V_2. Optimal solution value (*Opt*) is the best solution in the available literature. *Best* means best solution value obtained with each algorithm and *Avg* represents average solution value in 30 runs with SGA and IAGSAA. The calculation formulae of the percentage of the error (E (%)) and the relative error (RE (%)) are as follows:

$$E(\%) = \frac{Best - Opt}{Opt} \times 100\% \tag{17}$$

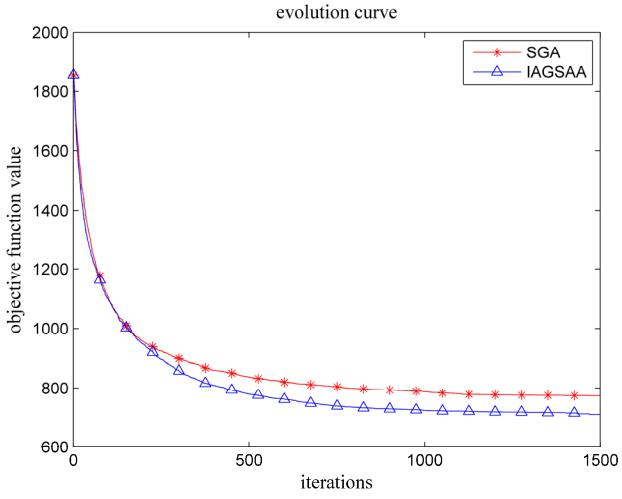
$$RE(\%) = \frac{Avg - Best}{Best} \times 100\% \tag{18}$$

TABLE II. A COMPARATIVE STUDY BETWEEN LSA, HGA, SGA AND IAGSAA FOR SOME SYMMETRIC TSPLIB INSTANCES

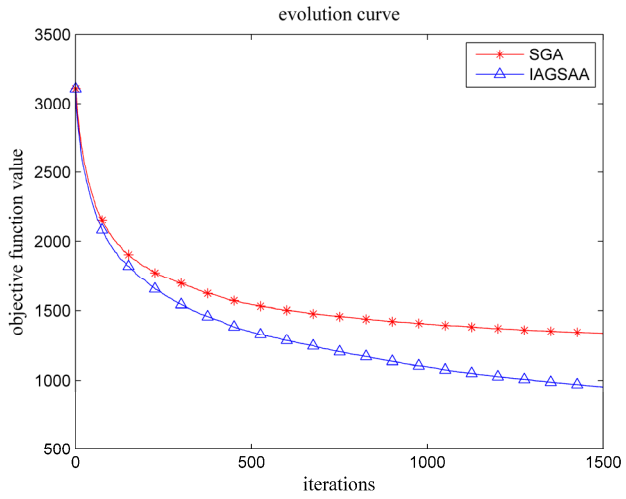
Instance	Clusters	*Opt*	LSA		HGA		SGA				IAGSAA			
			Best	***E***(%)	***Best***	***E***(%)	***Best***	***Avg***	***E***(%)	***RE***(%)	***Best***	***Avg***	***E***(%)	***RE***(%)
burma14	(6,7)	3621	3621	0.00	3621	0.00	3621	3621.00	0.00	0.00	3621	3621.00	0.00	0.00
ulysses16	(7, 8)	7303	7303	0.00	7303	0.00	7191	7191.00	-1.53	0.00	**7191**	**7191.00**	**-1.53**	**0.00**
gr17	(8, 8)	2517	2517	0.00	2517	0.00	2517	2549.40	0.00	1.29	2517	2517.00	0.00	0.00
gr21	(10, 10)	3465	3465	0.00	3465	0.00	3465	3613.30	0.00	4.28	3465	3467.00	0.00	0.06
ulysses22	(10, 11)	8190	8190	0.00	8190	0.00	8078	8275.60	-1.37	2.45	**8078**	**8091.17**	**-1.37**	**0.16**
gr24	(11, 12)	1558	1558	0.00	1558	0.00	1558	1670.77	0.00	7.24	1558	1561.63	0.00	0.23
fri26	(12, 13)	957	957	0.00	957	0.00	957	1008.17	0.00	5.35	957	957.00	0.00	0.00
bayg29	(14, 14)	2144	2144	0.00	2144	0.00	2192	2319.73	2.24	5.83	2144	2187.27	0.00	2.02
	(9, 9, 10)	2408	2408	0.00	2408	0.00	2408	2540.63	0.00	5.51	2408	2436.63	0.00	1.19
bays29	(14, 14)	2702	2702	0.00	2702	0.00	2702	2886.63	0.00	6.83	2702	2743.47	0.00	1.53
	(9, 9, 10)	2991	2991	0.00	2991	0.00	2991	3146.67	0.00	5.20	2991	3016.80	0.00	0.86
dantzig42	(20, 21)	699	699	0.00	699	0.00	770	919.80	10.16	19.45	699	714.33	0.00	2.19
	(13, 14, 14)	699	699	0.00	699	0.00	699	773.87	0.00	10.71	699	708.97	0.00	1.43
	(10, 10, 10, 11)	699	699	0.00	699	0.00	699	720.50	0.00	3.08	699	703.60	0.00	0.66
swiss42	(20, 21)	1605	1605	0.00	1605	0.00	1735	1863.77	8.10	7.42	1605	1646.70	0.00	2.60
	(13, 14, 14)	1919	1919	0.00	1919	0.00	1957	2079.37	1.98	6.25	1919	1954.83	0.00	1.87
	(10, 10, 10, 11)	1944	1944	0.00	1944	0.00	1964	2085.63	1.03	6.19	1944	1957.43	0.00	0.69
gr48	(23, 24)	6433	6656	3.47	**6433**	**0.00**	7351	8155.23	14.27	10.94	**6433**	**6595.33**	**0.00**	**2.52**
	(15, 16, 16)	7466	7466	0.00	7466	0.00	7870	8666.70	5.41	10.12	7466	7587.73	0.00	1.63
	(11, 12, 12, 12)	8554	8554	0.00	8554	0.00	8900	9619.40	4.04	8.08	8554	8660.50	0.00	1.25
eil51	(25, 25)	564	570	1.06	**564**	**0.00**	608	665.57	7.80	9.47	**564**	**581.17**	**0.00**	**3.04**
	(16, 17, 17)	681	689	1.17	**681**	**0.00**	720	769.87	5.73	6.93	**681**	**696.70**	**0.00**	**2.31**
	(12, 12, 13, 13)	714	714	0.00	714	0.00	735	794.50	2.94	8.10	714	725.17	0.00	1.56
berlin52	(25, 26)	10422	NA	NA	10422	0.00	11233	12718.83	7.78	13.23	10422	10933.50	0.00	4.91
st70	(34, 35)	916	NA	NA	916	0.00	1225	1336.77	33.73	9.12	916	952.80	0.00	4.02
eil76	(37, 38)	721	NA	NA	721	0.00	892	1010.63	23.72	13.30	721	752.57	0.00	4.38
rat99	(49, 49)	1346	NA	NA	**1346**	**0.00**	1999	2170.53	48.51	8.58	1398	1455.43	3.87	4.11
kroA100	(24, 25, 25, 25)	45733	NA	NA	45733	0.00	60380	66941.33	32.03	10.87	45733	48146.13	0.00	5.28
Average	—	—	—	0.25	—	0.00	—	—	7.38	7.35	—	—	0.03	1.80



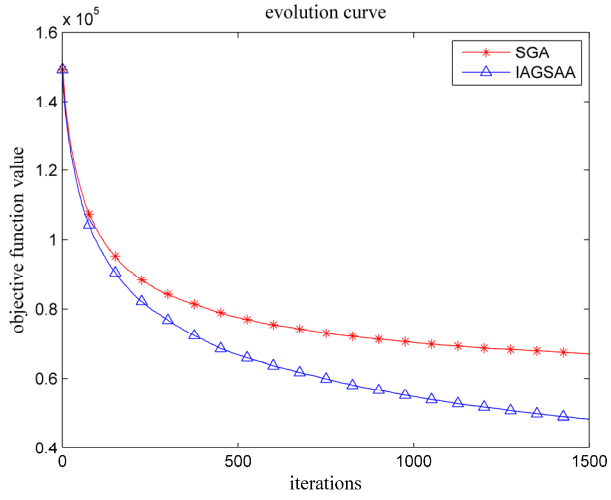
(a) Average evolution curves for fri26 (22, 23).



(b) Average evolution curves for dantzig42 (13, 14, 14).



(c) Average evolution curves for st70 (34, 35).



(d) Average evolution curves for kroA100 (24, 25, 25, 25).

Fig. 7. Average evolution curves in 30 runs for several TSPLIB instances.

Fig. 7 shows the average evolution curves in 30 runs with SGA and IAGSAA for several TSPLIB instances. According to the data in TABLE II, our IAGSAA can almost obtain the optimal solutions for all benchmark instances listed in the table except rat99 (49, 49). For gr48 (23, 24), eil51 (25, 25) and eil51 (16, 17, 17), HGA and IAGSAA can obtain the optimal solution. For ulysses16 (7, 8) and ulysses22 (10, 11), our IAGSAA can even find a better solution than LSA and HGA. Moreover, the relative errors of the IAGSAA are not greater than 5.28% and the average relative error is only 1.80%, which means the solution quality is good and the proposed algorithm is effective. Whereas, the average error and relative error of the SGA are 7.38% and 7.35% respectively, indicating that solutions for some instances obtained by the SGA are much worse than optimal solutions and the solution quality is relatively low. As shown in Fig. 7, the IAGSAA has a faster convergence speed than the SGA. To sum up, the IAGSAA proposed in this paper has good effectiveness, which is able to avoid premature phenomena, as well as accelerate the convergence speed. However, the relative error is increasing inevitably with the scale of the problem.

V. CONCLUSION

This study divides stations in a factory into several priorities based on the process, which is represented by the priority matrix, and analogous to the OCTSP. Since solving the problem by SGA shows a slow convergence speed and premature phenomena, we have proposed the IAGSAA, in which the generation strategy of the initial population of the SGA is improved, and improved adaptive crossover and mutation, as well as SAM are introduced. Later in the iteration of the IAGSAA, the adaptive crossover and mutation can enrich the diversity of the population to find new search directions and jump out of local optimal solutions. SAM is very useful to accelerate the convergence speed of the algorithm. Simulation results have indicated that our IAGSAA is more efficient in producing high quality solutions than LSA and SGA for some benchmark instances, and has a faster convergence speed than SGA.

Since the environment in the factory is complex and dynamic, dynamic constraints of mobile robots' moving space will be taken into account in the further research, as well as the physical constraints from AGVs/Mobile robots. Moreover, the cooperative motion problem of multiple AGVs/Mobile robots also will be studied in depth.

ACKNOWLEDGMENT

This work supported by the National Natural Science Foundation of China (51675186), the National Natural Science Foundation of China and the Royal Society of Edinburgh (51911530245), and the Science and Technology Project of Guangdong Province (2018A030321002).

REFERENCES

[1] E. Liu, X. Yao, M. Liu and H. Jin, "AGV path planning based on improved grey wolf optimization algorithm and its implementation prototype platform," Commputer Integrated Manufacturing Systems, vol. 24, no. 11, pp. 2779-2791, November 2018.

[2] C. He, Y. Song, Q. Le, X. Lv, R. Liu and J. Chen, "Integrated Scheduling of Multiple AGVs and Machines in Flexible Job Shops," China Mechanical Engineering, vol. 30, no. 04, pp. 64-73, February 2019.

[3] Y. Yu, Y. Chen and T. Li, "Improved genetic algorithm for solving TSP," Control and Decision, vol. 29, no. 8, pp. 1483-1488, August 2014.

[4] G. Laporte, J. Y. Potvin and F. Quilleret, "A Tabu Search Heuristic using Genetic Diversification for the Clustered Traveling Salesman Problem," Journal of Heuristics, vol. 2, no. 3, pp. 187-200, January 1997.

[5] S. P. Tseng, C. W. Tsai, M. C. Chiang and C. S. Yang, "A fast ant colony optimization for traveling salesman problem," IEEE Congress on Evolutionary Computation (CEC), 2010.

[6] Q. He, Y. L. Wu and T. W. Xu, "Application of improved genetic simulated annealing algorithm in TSP optimization," Control and Decision, vol. 33, no. 02, pp. 219-225, February 2018.

[7] M. Rosendo and A. Pozo, "A hybrid Particle Swarm Optimization Algorithm for Combinatorial Optimization Problems," IEEE Congress on Evolutionary Computation (CEC), 2010.

[8] A. Jaradat, B. Matalkeh and W. Diabat, "Solving Traveling Salesman Problem using Firefly algorithm and K-means Clustering," IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), 2019.

[9] L. T. Koczy, P. Foldesi and B. Tuu-Szabo, "A Discrete Bacterial Memetic Evolutionary Algorithm for the Traveling Salesman Problem," IEEE Congress on Evolutionary Computation (CEC), 2016.

[10] J. A. Chisman, "The clustered traveling salesman problem," Computers and Operations Research, vol. 2, no. 2, pp. 115–119, September 1975.

[11] K. Jongens and T. Volgenant, "The symmetric clustered traveling salesman problem," European Journal of Operational Research, vol. 19, no. 1, pp. 68-75, February 1985.

[12] S. Anily, J. Bramel and A. Hertz, "A 5/3-approximation algorithm for the clustered traveling salesman tour and path problems," Operations Research Letters, vol. 24, no. 1, pp. 29–35, February 1999.

[13] C. Ding, Y. Cheng and M. He, "Two-level Genetic Algorithm for Clustered Traveling Salesman Problem with Application in Large-Scale TSPs," Tsinghua Science and Technology, vol. 12, no. 4, pp. 459–465, August 2007.

[14] Z. H. Ahmed, "An exact algorithm for the clustered travelling salesman problem," Opsearch, vol. 50, no. 2, pp. 215-228, April 2013.

[15] Z. H. Ahmed, "The Ordered Clustered Travelling Salesman Problem: A Hybrid Genetic Algorithm," The Scientific World Journal, vol. 2014, pp. 1-13, February 2014.

[16] M. Mestria, "New hybrid heuristic algorithm for the clustered traveling salesman problem," Computers & Industrial Engineering, vol. 116, pp. 1-12, February 2018.

[17] M. Srinivas and L. M. Patnaik, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms," IEEE Transactions on Systems, Man and Cybernetics, vol. 24, no. 4, pp. 656-667, May 1994.

[18] G. Reinelt, "TSPLIB—A Traveling Salesman Problem Library," ORSA Journal on Computing, vol. 3, no. 4, pp. 376-384, November 1991.