Hindawi
Mathematical Problems in Engineering
Volume 2020, Article ID 1398595, 8 pages
https://doi.org/10.1155/2020/1398595

Research Article

Multipopulation Genetic Algorithm Based on GPU for Solving TSP Problem

Boqun Wang,[1,2] **Hailong Zhang**,[1,2,3] **Jun Nie**,[4] **Jie Wang**,[1] **Xinchen Ye**,[1] **Toktonur Ergesh**,[1] **Meng Zhang**,[1,2] **Jia Li**,[1] **and Wanqiong Wang**[1]

[1]*Xinjiang Astronomical Observatory, Chinese Academy of Sciences, Urumqi 830011, China*
[2]*University of Chinese Academy of Sciences, Beijing 100049, China*
[3]*Key Laboratory of Radio Astronomy, Chinese Academy of Sciences, Nanjing 210008, China*
[4]*Institute of Advanced Technology, University of Science and Technology of China, Hefei 230088, China*

Correspondence should be addressed to Hailong Zhang; zhanghailong@xao.ac.cn

Received 29 February 2020; Revised 28 May 2020; Accepted 23 June 2020; Published 28 August 2020

Academic Editor: Purushothaman Damodaran

A GPU-based Multigroup Genetic Algorithm was proposed, which parallelized the traditional genetic algorithm with a coarse-grained architecture island model. The original population is divided into several subpopulations to simulate different living environments, thus increasing species richness. For each subpopulation, different mutation rates were adopted, and the crossover results were optimized by combining the crossover method based on distance. The adaptive mutation strategy based on the number of generations was adopted to prevent the algorithm from falling into the local optimal solution. An elite strategy was adopted for outstanding individuals to retain their superior genes. The algorithm was implemented with CUDA/C, combined with the powerful parallel computing capabilities of GPUs, which greatly improved the computing efficiency. It provided a new solution to the TSP problem.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the essential problems in computer science. The mathematical description is as follows. Given a set of cities $\{c_1, c_2, c_3, \ldots, c_n\}$, the distance between every two cities $\{c_i, c_j\}$ is $d\{c_i, c_j\}$, and the problem requires a shortest sequence s to make total distance y minimal [1], and y is defined by

$$y = \sum_{i=1}^{n-1} d\left(c_{s(i)}, c_{s(i+1)}\right) + d\left(c_{s(1)}, c_{s(n)}\right). \quad (1)$$

This problem has been identified as an NPH problem, and it is difficult to find the optimal solution for each instance. At present, heuristic algorithms are used to solve most TSP.

Genetic Algorithm (GA) is a method to find the optimal solution by simulating the natural evolution process. The principle of GA is simple, operability is strong, and it is excellent for global searching, so it is widely used in solving TSP. However, GA has some defects, such as easily falling into local optimal solutions and long search time.

Compute Unified Device Architecture (CUDA) is a parallel programming model launched by NVIDIA, which runs upon the Graphics Processing Unit (GPU) [2]. With CUDA, developers accelerate their projects by running the sequential part of the program within CPU and the parallel part on GPU. Each NVIDIA GPU has thousands of CUDA cores, which can launch thousands of threads for numerical calculations that will significantly improve the efficiency of the algorithm.

Bao Lin presented an improved hybrid GA to solve the two-dimensional Euclidean TSP, in which the crossover operator is enhanced with a local search [3]. Jain V proposed a new genetic crossover operator using a greedy approach [4]. Based on the traditional GA, Yu et al. proposed an algorithm that introduces the greedy method into species initialization [5]. AF El-Samak used Affinity Propagation

Clustering Technique (AP) to optimize the performance of the GA for solving TSP [6]. Although previous studies have improved the traditional genetic algorithm, when the population size increases, the time consumed becomes an important factor affecting the efficiency of the algorithm, so it is necessary to parallelize the genetic algorithm to reduce the cost time of the genetic algorithm.

Chen S [7] and O'Neil [8] both proposed GPU-based parallel GA. However, they limited the initial population size to a small range, which was not conducive to improve population diversity and weakened the GA global search capability. In order to increase the diversity of the population, the initial population size should be increased as much as possible.

Here, we propose a coarse-grained parallel GA based on the island model, which increases the initial population size and divides the large-scale population into multiple subpopulations. The subpopulations simultaneously perform genetic operations such as distance crossing and adaptive mutation. Because this algorithm reduces running time when guaranteeing the accuracy of the results, it provides a feasible method to solve TSP.

2. Genetic Algorithm

2.1. Traditional GA. GA is a heuristic algorithm based on Darwin's theory of evolution. Its key thought is natural selection: individuals with higher fitness in a population can survive and reproduce the next generation. Evolution usually starts with a randomly generated population of individuals and is an iterative process. In each generation, the fitness of each individual in the population is evaluated. Individuals with high fitness from the current population are selected, and the genome of each individual is modified (crossed and mutated) to form a new generation. Then, a new generation of candidate solutions will be used in the next generation of the algorithm. Generally, the algorithm terminates when it reaches the maximum number of generations or overall level satisfactory fitness. Five major components in the GA initial population, fitness, selection, crossover, and mutation are explained as follows.

Initial population: generate randomly and allow the entire range of possible solutions. The larger the population size, the higher the species richness.

Fitness: judge the individual's ability to adapt to the environment. The greater the fitness, the higher the chance of survival. In TSP, the fitness is often set as the reciprocal of the individual path length.

Selection: select a pair of individuals with higher fitness from the population as parents of the next generation.

Crossover: this is the most important step in GA. Parents choose some points on their genes to exchange to produce offspring.

Mutation: the genes of the offspring may be subject to other influences and cause mutations. This step is used to simulate random mutations of chromosomes.

2.2. Coarse-Grained Parallelism Based on the Island Model. At present, mainstream parallel GA has four types of models: the master-slave model, island model, domain model, and hybrid model [7]. The island model is also known as a distributed model or a coarse-grained model. Its execution process is shown in Figure 1. First, a large population is initialized, and then the population is divided into several subpopulations 1, 2, ..., N. Second, subpopulations independently perform selection, crossover, and mutation. Third, some individuals in each subpopulation migrate to other subpopulations. Finally, when the specified number of generations is reached, the population is screened to find the optimal solution. We developed an algorithm based on the island model; divide individuals into several subpopulations. Then, load the subpopulations onto the GPU and create N threads, each thread is responsible for the genetic operations of a subpopulation. After reaching the specified number of generations, we search and output the optimal solution in all groups.

3. Algorithm Implementation

3.1. Selection. Selection is the process of selecting the fittest for the current population, intending to inheriting genes with higher fitness to future generations. Traditional selection algorithms include roulette algorithm and tournament algorithm. However, these algorithms require synchronization between threads, which is not conducive to massive parallelism. Here, we use a selection strategy based on fitness. The specific steps of this operation are as follows, and Figure 2 shows a specific example.

Steps 1: let the number of individuals in each subpopulation be M and set a selection threshold to an integer p, and $0 \le p \le [M/2]$

Steps 2: sort each subpopulation according to fitness from large to small

Steps 3: replace the $M - p + 1 \sim M$ values with the $1 \sim p$ values in each subpopulation

3.2. Crossover. Crossover is the process that two individuals exchange some of their genes according to a certain method to form a new individual. It is the most critical operation in GA, which determines the genes of individual offspring and is the key to search the global optimal solution. We use a decimal Pc in the range [0,1] to control whether or not two individuals' cross called the crossover rate. The specific operation is as follows: generate a crossover rate Pc_i for each subpopulation i by using formula (2), and combine the individuals within the population in pairs to serve as the parents of the next generation. A random number pr in the range [0,1] is generated for each pair of parents. If $pr \le Pc_i$, the parent does not cross, and if $pr > Pc_i$, cross is generated to generate offspring. Figure 3 shows a specific example.

The size of the crossover rate Pc_i is very important. If the crossover rate is too large, the genetic pattern is more likely to be destroyed so that the individual structure with high fitness is quickly destroyed; if the crossover rate is too low, the search process will be slow, even stagnant. We have

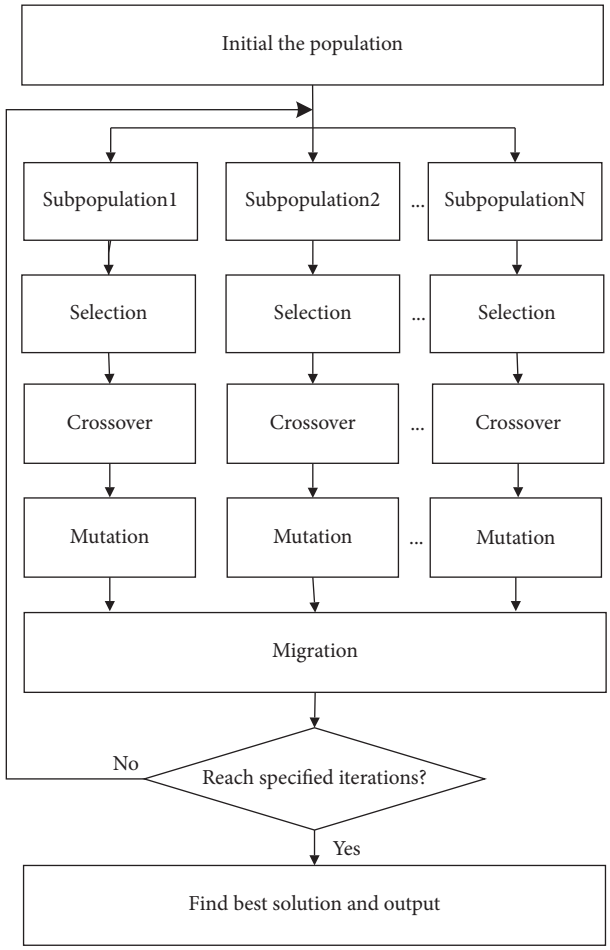


Figure 1: Coarse-grained parallelism based on the island model.

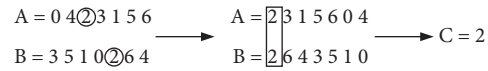
assigned different crossover rates to each subpopulation within a specific range to simulate evolution in different environments. The crossover rate Pc_i of the subpopulation i can be expressed by

$$Pc_i = \alpha \frac{i}{N} + \beta. \tag{2}$$

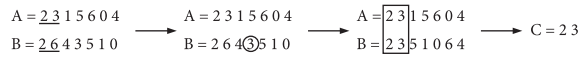
Formula (2): let the total number of subpopulations be N, i represents the number of the current population, $1 \le i \le N$, where α and β are control coefficients, to ensure that Pc_i changes between $[\alpha/N + \beta, \alpha + \beta]$.

The parents who are ready to cross are determined, and then the specific method of crossover becomes the focus of research. Single-point crossover and two-point crossover are traditional crossover algorithms. Inspired by Tang [9], we use the crossover method based on individual distance, which is named distance crossover algorithm. Specific examples are given to describe the algorithm. Take 7 cities for example, number them from 0 to 6, and the distance between every two cities is shown in Table 1. A pair of parents A and B generate offspring C and D, and the generating steps are as follows:

Step 1: assume that parent $A = 0\ 4\ 2\ 3\ 1\ 5\ 6$ and $B = 3\ 5\ 1\ 0\ 2\ 6\ 4$

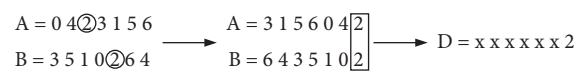


Step 2: determine the head of C. Choose a city randomly. Here, select City 2 and move City 2 and the cities after it to the head of the sequence. City 2 is called a "determined city," which is the area framed by a rectangle in the figure. In this way, the head of C is 2



Step 3: determine the second city of C. For the A and B sequences generated by step 1, compare the size of $d(2, 3)$ and $d(2, 6)$ according to the distance table, and we can get $d(2, 3) < d(2, 6)$. According to the principle of small distance, the second city of C is determined as 3. It is equivalent to A providing a second city for C; then, sequence A remains unchanged, moving the city after the city 3 in sequence B to the back of the "determined city," and the "determined city" at this time is 2, 3, which is the area framed by the rectangle.

Step 4: repeat step 3 continuously to get sequence C



Step 5: the generation of child D is similar to C, except that the order of the cities has changed. After selecting city 2 in step 1, move city 2 and the city in front of it to the end of the sequence so that the tail of D is 2

$$C = 2\ 3\ 1\ 0\ 6\ 4\ 5$$
$$D = 0\ 3\ 1\ 5\ 6\ 4\ 2$$

Step 6: the D sequence is obtained according to the distance. The specific method is similar to step 3, and the arrangement of D is determined from tail to head. Finally, we get the order of offspring C and D.

It is worth noting that the distance crossover method does not completely guarantee that the offspring are superior to the parents. In the process of generating offspring, the distance from the last city to the first city is not considered. If this distance is very large, the offspring may be worse than the parents. Therefore, this method can only guarantee the superiority of the offspring with a high probability. Without special circumstances, this crossover method accelerates the convergence very well.t

3.3. Mutation. The mutation is an auxiliary method of GA to generate new individuals. It improves the local search capability of the algorithm and is also a powerful guarantee for achieving population diversity. We introduced the mutation rate *Pmu* in the range [0,1] to control the number of mutated individuals; generate a random number *pmr* in the range [0,1] for each individual, and then compare it with *Pmu*. If $pmr \le Pmu$, the individual mutates, otherwise stays unchanged. The specific mutation way is the two points' method, which randomly selects two points on the sequence of individuals and exchanges them. Figure 4 shows a specific example.

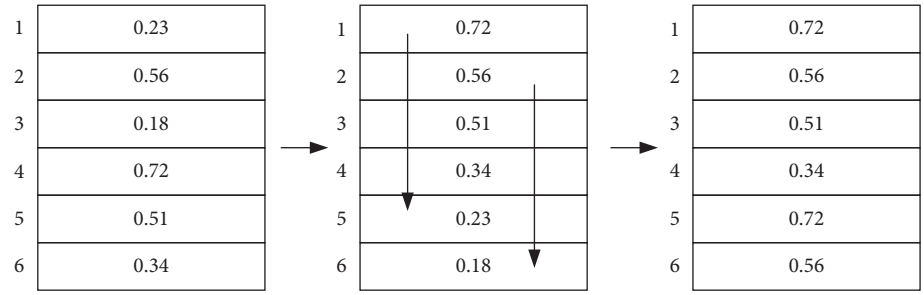


Figure 2: Selection: taking a certain subpopulation as an example, the number of individuals in the subpopulation $M = 6$, the fitness of each individual is displayed in a rectangular box. Set the selection threshold $p = 2$, first sort following the degree of fitness from large to small. Then, replace the individuals with numbers 5 and 6 with the individuals with numbers 1 and 2 to get a new subpopulation.

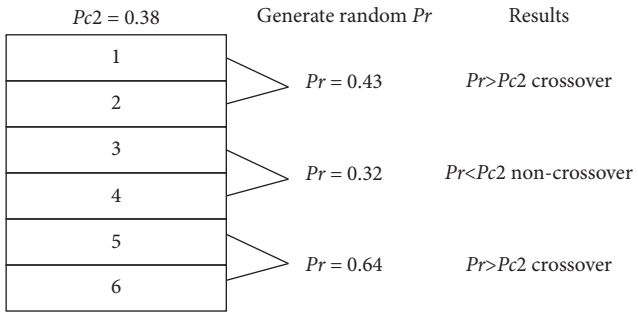


Figure 3: Cross determination method. The subpopulation number is 2 and $Pc_2 = 0.38$. First, pair individuals to generate 3 pairs of parents, and then generate 3 random numbers pr for these 3 pairs of parents. The $pr = 0.43$ of the first pair of parents is greater than Pc_2, so this pair of parents performs a crossover operation. The $pr = 0.32$ of the second pair of parents is less than Pc_2, so no crossover is performed. The $pr = 0.64$ of the third pair of parents is greater than Pc_2, so the crossover operation is performed.

Traditional GA usually use a global Pmu, that is, every individual uses the same Pmu, which has drawbacks; the diversity of the population in the early stages of evolution is good enough that a large mutation rate is not needed, while the diversity in the late stage of evolution is reduced and a large mutation rate is required to produce excellent individuals. We use an adaptive mutation method that Pmu increases with the number of generations. This method has been proved to be effective in [10]. The specific steps are as follows:

Step 1: set the maximum mutation rate Pmu_max.

Step 2: calculate the mutation rate of this generation according to the formula $Pmu(t) = (t/\text{MAXGEN})\ Pmu\_\text{max}$. The mutation rate corresponding to the tth generation is $Pmu(t)$, and MAXGEN is the maximum number of generations.

The mutation operation improves the local search ability of the GA and promotes the result to converge to the optimal solution, while the adaptive mutation algorithm ensures the diversity of species in the later stage of evolution and prevents the algorithm from falling into the local optimal solution.

3.4. Migration. In nature, the same species distributed in different environments often migrate with each other. This behavior is called population migration.

Table 1: The distance between each two cities.

	City 0	City 1	City 2	City 3	City 4	City 5	City 6
City 0	0	1	4	1	5	3	2
City 1	1	0	3	2	4	2	6
City 2	4	3	0	1	2	5	3
City 3	1	2	1	0	5	4	2
City 4	5	4	2	5	0	3	1
City 5	3	2	5	4	3	0	2
City 6	2	6	3	2	1	2	0

Communicating with each other enriches the gene pool of species and promotes the evolution of species. We use an exchange method based on the migration rate, with migration interval K, migration rate ER, and migration number E. The specific steps for migration are as follows:

Step 1: determine the size of migration interval K, migration rateER, and migration number E. Among them, K represents the number of generations between two migrations; let MAXGEN be the maximum number of generations, then $1 \le k \le \text{MAXGEN}$. ER represents the probability of whether migration is successful, $0 \le \text{ER} \le 1$. E refers to the number of individuals in each subpopulation participating in migration; let M be the number of individuals in each subpopulation, then $1 \le E \le M$.

Step 2: when the number of generations reaches nK, $1 \le n \le (\text{MAXGEN}/K)$, generate a random number Eran in range of [0,1], and if $Er \le \text{ER}$, migrate individuals from this generation

Step 3: the specific migration method is to migrate the $1 \sim E$ individuals of each subpopulation to the adjacent subpopulation and replace the $M - E + 1 \sim M$ individuals of the adjacent subpopulation. Figure 5 shows a specific example.

Communication between populations is very important. It not only realizes the genetic communication between the populations but also eliminates the difference solutions, and remains the optimal and suboptimal solutions. Through migration, the overall fitness of the population is further improved, and the global search speed is accelerated.

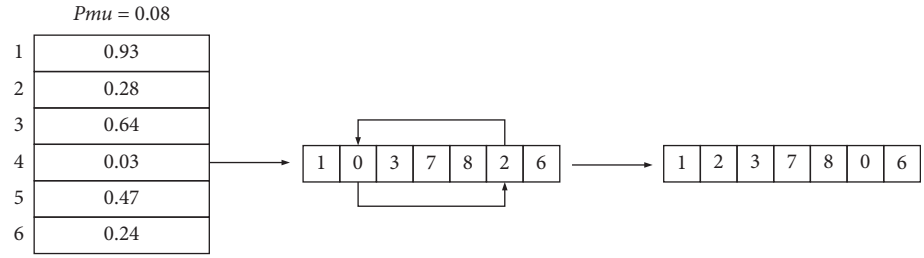


Figure 4: Mutation: taking a subpopulation with 6 individuals as an example, let $Pmu = 0.08$. First, a random decimal pmr in range [0, 1) was generated for each individual in the subpopulation. The example showed that $pmr < Pmu$ of individual No. 4 and $pmr > Pmu$ for others, so mutation operation was only performed on No. 4 individual. Randomly select two points within the sequence No. 4, where the second and sixth points are selected, and the points are swapped to obtain a new genotype.

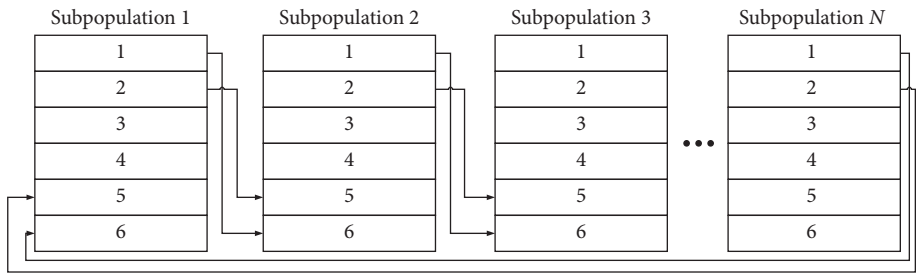


Figure 5: Migration: the initial population is divided into N subpopulations and the number of individuals in each subpopulation $M = 6$. Set the migration number $E = 2$, that is, 2 individuals in each subpopulation participate in migration. The specific method is to replace the fifth and sixth individuals of the adjacent subpopulation with the first and second individuals of each subpopulation, respectively. Subpopulation 1 migrates to subpopulation 2, subpopulation 2 migrates to subpopulation 3, and so on until subpopulation N migrated to subpopulation 1.

3.5. Elite Retention. The elite individuals of the parent generation are retained to prevent from participating in genetic operations of the next generation, and then these elites replace the poorer solutions in the offspring. The elite strategy prevents optimal individuals from being destroyed due to hybridization and accelerates the convergence speed of the GA. The elite retention strategy is as follows: set a retention value SN, the number of individuals in each subpopulation is M, then $1 \leq \mathrm{SN} \leq M$. First, sort each subpopulation according to fitness from small to large. Then, select SN individuals at the tail and send them directly to the next generation without selecting, crossing, or mutating.

4. Experimental Results and Analysis

4.1. Comparison of Acceleration Effect. First, the acceleration effect of the GPU was tested, and the traditional Simple Genetic Algorithm (SGA) and Multipopulation Genetic Algorithm (MPGA) were used to calculate the chn31 problem and the running time was recorded. Since the evolution of GA is continuous, that is, the traits of the offspring depend on the genes of the parent, so the acceleration effect is more obvious in a single generation. In this experiment, the number of evolution generation is 500, and the average running time for each generation is obtained after the total running time is obtained. The experimental environments are shown in Table 2, and the experimental results are shown in Table 3. Figures 6 and 7 represent the time-population scale relationship diagram and the population scale-speed up diagram, respectively. In the experiment, each thread is responsible for a population, the number of individuals in each population is 20, the selection threshold p is 5, the migration interval k is 10, the migration rate ER is 0.5, the migration number $E = 3$, the maximum mutation rate $Pmu\_\max = 0.2$, and the elite reserve value SN = 10.

Table 2: Experimental equipment.

Hardware	Device type	Storage (GB)	Frequency
CPU	Intel core i7-4720HQ	8	2.6 GHZ
GPU	NVIDIA GeForce GTX 960M	4	1.1 GHZ

From Table 3, in the case of fixed generations, as the population size continues to increase, the results obtained are closer to the optimal solution. Figures 6 and 7 show that when the population size is less than 1000, the speed of the GPU is not as fast as the CPU because data replication and GPU-CPU communication cost more time. While the population increases to 1500 and more, the running time of the CPU keeps increasing linearly, and the running time of the GPU is relatively stable, which accelerates the speed-up ratio to 200 when the population size is 20,000. As a result, the efficiency of MPGA with a large-scale population is significantly higher than SGA.

Figure 8 shows the curve of MPGA and SGA evolution under chn31 for 500 generations, the horizontal axis represents the number of generations, and the vertical axis represents the shortest path length. The solid and dashed lines represent the evolution of SGA and MPGA,

Table 3: chn31 solution results for SGA and MPGA with different population sizes.

Population size	SGA time (ms)	SGA result	MPGA time (ms)	MPGA result	Speed-up ratio
40	0.136	23308	16.56	17594	0.00821256
100	0.446	22456	21.64	16292	0.020609982
500	6.078	19358	32.28	15539	0.188289963
1000	20.79	18365	34.26	15476	0.606830123
1500	48.238	16873	35.26	15385	1.368065797
2000	80.394	16844	35.9	15467	2.239387187
5000	518.98	16797	36.3	15379	14.2969697
10000	2107.83	16599	36.88	15377	57.15374187
15000	4714.64	16161	43.24	15377	109.0342276
20000	8824.48	16161	43.88	15377	201.1048314

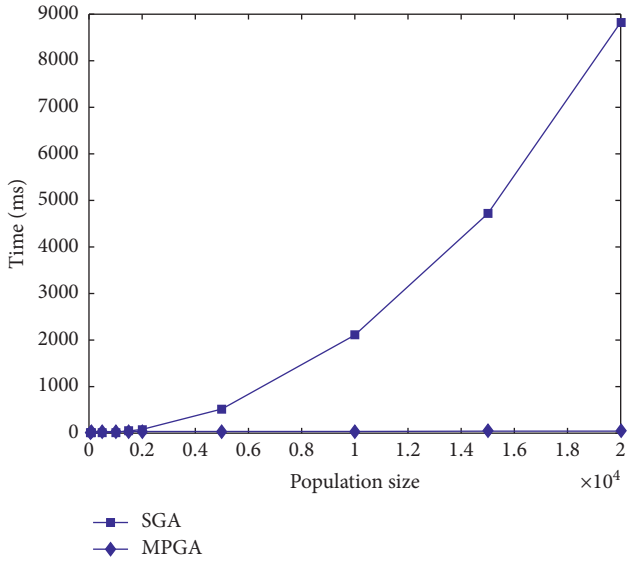


Figure 6: Population size-time relation.

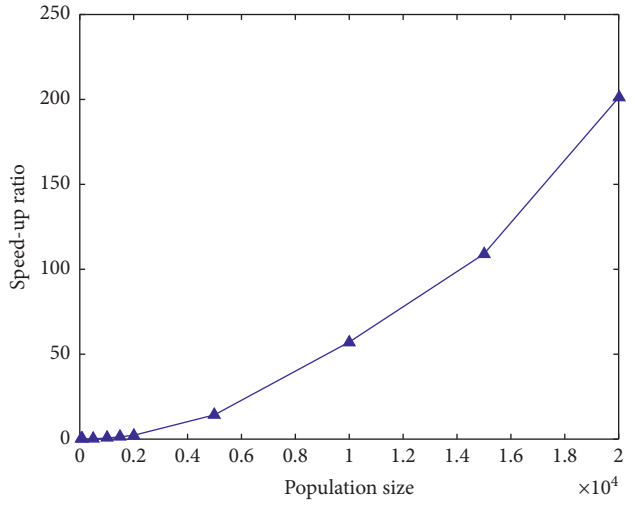


Figure 7: Population size-speed up ratio relation.

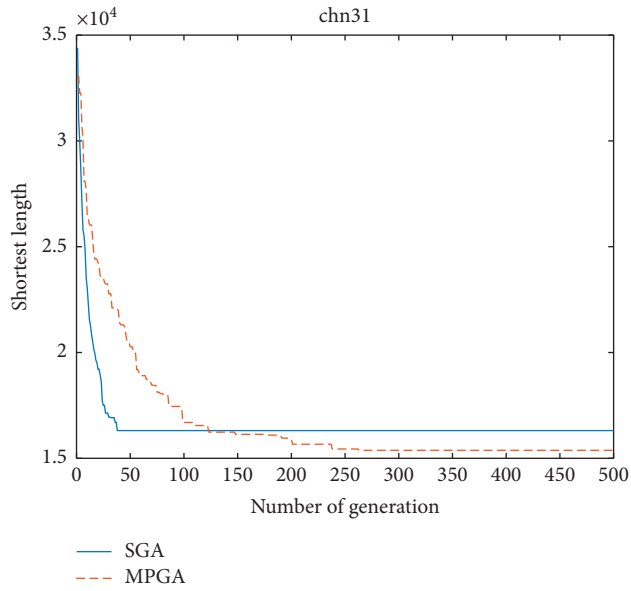


Figure 8: Evolution curve.

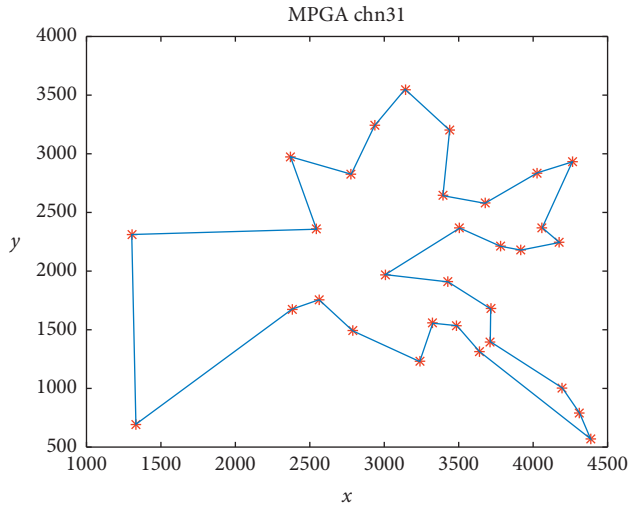


Figure 9: Result of MPGA chn31.

respectively. It can be seen from this that the SGA convergence rate is too fast, and it falls into a local optimal solution around 40 generations. The MPGA convergence rate is relatively gentle, and the optimal solution 15377 was found at 250 generations. This proves that MPGA has better convergence effect than SGA, and it is not easy to fall into the local optimal solution and has a strong global search ability. Figure 9 shows

Table 4: Partial TSPLIB results of SGA and MPGA.

Algorithm	MPGA				SGA			
TSPLIB	att48	beilin52	pr124	rat195	att48	beilin52	pr124	rat195
Optimal	33522	7542	59030	2323	33522	7542	59030	2323
Best	33522	7542	59030	2323	39680	9309	90514	3508
Worst	33715	7548	60156	2591	43634	10429	92701	3702
Mean	33612	7544	59864	2413	41446	9975	92264	3591
Time (ms)	82	157	3125	5014	2517	4278	30856	75829
Deviation	0.003	0.0002	0.014	0.039	0.25	0.32	0.56	0.55

the final path of MPGA chn31, and the horizontal and vertical axes show the coordinates of the city.

4.2. TSPLIB Test. To verify the ability of MPGA of dealing with large-scale TSP for a further step, some TSPLIB datasets are selected for testing and comparing with traditional SGA. Repeat 30 times for each dataset, with an evolution generation of 1000 and population size of 10,000. The results are shown in Table 4.

According to the test results, the MPGA is superior to the SGA in various indicators. As the scale of the problem increases, the SGA falls into the local optimal solution, and the result is very different from the optimal solution. The MPGA is stable with a small deviation from the optimal solution. Although GA is parallelized in articles [7, 8], the number of populations is limited to a small range, which is not conducive to the improvement of population diversity. Based on the island model, MPGA improves the selection, crossover, and mutation algorithms to ensure the accuracy of the results.

5. Conclusion

In this paper, the island model coarse-grained architecture is used to implement the parallelization of GA, the MPGA. Compared with other GA [3, 4] based on GPU, MPGA increases the population size and the richness of species. With the computing power of the GPU, the large-scale population is divided and ruled, which shortens the running time. Compared with serial SGA, the speed is improved by about 200 times. For the crossover step in GA, variable crossover probability and distance-based crossover method are proposed to improve the crossover efficiency and ensure the global search ability of the algorithm. For the mutation step in GA, a generation adaptive mutation method is proposed to prevent the result from falling into the local optimal solution. MPGA increases the population size on the premise of ensuring the accuracy and achieves the balance between diversity and speed, which provides a new idea for solving TSP.

Data Availability

The data used to support this study are available from the corresponding author upon request. The source code can be downloaded at http://data.xao.ac.cn/static/gatsp.zip.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) (11873082 and 11803080), National Key Research and Development Program of China (2018YFA0404704), Youth Innovation Promotion Association CAS, and program of the Light in China's Western Region (2019-XBQNXZ-B-018). Data resources are supported by China National Astronomical Data Center (NADC). This work was supported by Astronomical Big Data Joint Research Center, co-founded by National Astronomical Observatories, Chinese Academy of Sciences. The algorithms in this paper have applied Taurus High-Performance Computing Cluster of Xinjiang Astronomical Observatory, CAS during the testing process.

References

[1] F. Liu and G. Zeng, "Study of genetic algorithm with reinforcement learning to solve the TSP," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6995–7001, 2009.

[2] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional, Boston, MA, USA, 2010.

[3] B. Lin, X. Sun, and S. Salous, "Solving travelling salesman problem with an improved hybrid genetic algorithm," *Journal of Computer and Communications*, vol. 04, no. 15, pp. 98–106, 2016.

[4] V. Jain and J. S. Prasad, "An optimized algorithm for solving travelling salesman problem using greedy cross over operator," in *Proceedings of the International Conference on Computing for Sustainable Global Development*, pp. 5076–5079, New Delhi, India, March 2016.

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

[6] A. F. El-Samak and W. Ashour, "Optimization of traveling salesman problem using affinity propagation clustering and genetic algorithm," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 5, no. 4, pp. 239–245, 2015.

[7] S. Chen, S. Davis, H. Jiang et al., *CUDA-based Genetic Algorithm on Traveling Salesman Problem*, pp. 241–252, Springer, Berlin, Germany, 2011.

[8] M. A. O'Neil, D. Tamir, and M. Burtscher, "A parallel gpu version of the traveling salesman problem," in *Proceedings of the international conference on parallel and distributed*

processing techniques and applications (PDPTA), p. 1, Computer Engineering and Applied Computing (WorldComp), Las Vegas, NV, USA, July 2011.

[9] l x. Tang, "Improved genetic algorithm for travel salesman problem (TSP)," *Journal of Northeastern University*, vol. 20, no. 1, pp. 40–42, 1999.

[10] Q. Yue, *Coarse-grained Parallel Computing Performance of Genetic Algorithm and its Application*, Huazhong university of science and technology, Wuhan, China, 2008.