A Discrete Fireworks Algorithm for Solving Large-Scale Travel Salesman Problem

Haoran Luo
Peking University
Beijing, China
Email: luohaoran@pku.edu.cn

Weidi Xu
Key Laboratory of Machine Perception (MOE)
Peking University, Beijing, China
Email: wead_hsu@pku.edu.cn

Ying Tan
Key Laboratory of Machine Perception (MOE)
Peking University, Beijing, China
Email: ytan@pku.edu.cn

Faculty of Design, Kyushu University
Fukuoka, Japan
Email: y.tan.736@m.kyushu-u.ac.jp

***Abstract*—Fireworks algorithm (FWA) is a newly proposed swarm intelligence optimization method. It simulates the fireworks explosion process to search for the best location of sparks and has demonstrated good performance in many continuous optimization problems. In this paper, we apply FWA to the travel salesman problem (TSP), a classical discrete optimization problem. We propose a discrete fireworks algorithm for TSP by combining the general framework of FWA and current ideas for solving the TSP. We call it DFWA-TSP. In DFWA-TSP, 2-opt and 3-opt edge exchange heuristic are implemented as the basic explosion operation in FWA. An adaptive strategy is designed to decide the explosion amplitude. A particular mutation method based on insertion is also used to cover the shortage of edge exchange and a new selection method based on the quality of fireworks is adopted to pick up good fireworks efficiently. Various experiments on both TSPLIB and synthetic data have been made to compare the performance of our algorithm with current heuristic methods for TSP, such as genetic algorithm and ant colony system algorithm. We conclude that our algorithm out-performs these algorithms, especially on large-scale cases.**

***Index Terms*—Discrete Fireworks Algorithm, Travel Salesman Problem, K-opt Heuristic**

I. Introduction

FWA is a newly proposed swarm intelligence algorithm [1]–[4]. It has shown good performance in many optimization problems [5]–[8]. FWA simulates the fireworks explosion process to search for the optimum point. At the beginning the algorithm, a certain number of fireworks are generated at pre-given locations, each of which stands for a possible solution. Then for each firework, explosion strength and amplitude is calculated; it explodes and generates sparks. In the FWA, explosion process is the key step that new possible solutions, the sparks, are generated based on the original fireworks. The strength pre-calculated stands for how many sparks each firework can generate and the amplitude stands for the maximal level the sparks can vary from the firework. After the explosion, certain selection strategy will be applied. Locations of fireworks for next iteration are picked up from locations of current fireworks and sparks. Finally, algorithm goes into a new explosion process. The iteration will continue until the given termination criterion is satisfied. Besides, some particular mutation methods may also be inserted after the explosion so as to generate certain kind of sparks from each firework. This method can be used to enhance the diversity of sparks.

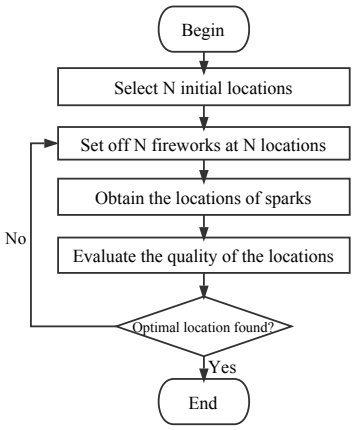

Fig. 1. Flow graph of FWA

TSP is a classical problem in combinatorial optimization. It aims to find the minimal length of the tour that visits every city exactly once. It can be defined mathematically as follows:

Definition 1: Given a complete graph $K_n (n \geq 3)$ and edge length: $E(K_n) \rightarrow R_+$, find a Hamiltonian circle T such that $\sum_{e \in E(T)} c(e)$ is minimal.

TSP is of great importance on both theory and application [9]. It is also always in hot research and has been viewed as a touchstone for discrete swarm intelligence algorithm.

In this paper, we apply FWA to TSP and propose a discrete fireworks algorithm DFWA-TSP for TSP. A character of TSP (and many other discrete optimization problems) is that it lacks good continuity, which means a small change of solution (the order of cities in the TSP) may have a huge influence of the result (the total tour length in the TSP). As a result, it forces us to consider how to make use of the character of the optimization problem apart from just applying the general framework of FWA. Considering the outstanding performance of edges exchange for TSP, we exploit 2-opt and 3-opt as

978-1-5090-6017-7/18/$31.00 ©2018 IEEE

the basic operation for generating sparks in the fireworks explosion. Besides, a particular mutation method based on insertion method is also used to generate a different kind of sparks from the explosion. We also use the adaptive strategy to control the explosion amplitude, which will be decided by the previous progress of best tour length rather than simple calculation based on the current quality and locations of fireworks. Special selection method based on the quality of fireworks(sparks) is used because of the huge feasible region of TSP. We show the numerical experiment results of DFWA-TSP, genetic algorithm and any colony system algorithm on various data.

The rest of the paper is organized as follows. In section II, we introduce related works of algorithms for TSP and current discrete fireworks algorithms. In section III, we describe our algorithm DFWA-TSP in detail. In section IV, we demonstrate the experiment results and our corresponding discussion. In section V, we give our conclusions.

II. Related Work

General TSP is known to be NP-hard and even its special case, Euclidean TSP, in which all the cities are points in R^n and edge length is the Euclidean distance, is NP-hard too [10]. Therefore, algorithms that can find suboptimal solutions in rational time have always been investigated. A large number of algorithms have been developed [11]. We list several important and typical ones here. The greedy algorithm starts from a random city and goes the nearest unvisited city each time. Christofides' algorithm [12] makes use of the minimal spanning tree and minimum-weight perfect matching to gain approximate solution. Lin-Kernighan heuristics [13] exploits k-opt heuristic method first. It tries to exchange several edges in the tour to gain a better solution. We use this operation in our explosion process. In [14], a polynomial time approximation scheme for Euclidean TSP in fixed dimensions is proposed. Swarm intelligence algorithm, which models the behaviour of real animals or human, have also been applied to solve TSP. There have been some good results, such as genetic algorithm [15]–[17], ant colony system [18], [19].

While FWA is originally raised for the continuous optimization problem, there has been already several successful attempts of applying FWA to discrete optimization problems. In [20], FWA is applied as an efficient method to solve network reconfiguration problem, which is formulated as a complex combinatorial optimization problem. In [21], discrete fireworks algorithm for single machine scheduling problem is proposed. In [22], a discrete modified fireworks algorithm is raised for community detection in complex systems. In [23], a binary enhanced fireworks algorithm for maximum satisfiability problem is designed. For the tourist trip design problem, [24] applies the FWA by selecting the definition of distance. For the multi-resource range scheduling problem, [25] propose a modified fireworks algorithm.

In [26], a discrete fireworks algorithm is proposed for aircraft mission planning. It should be noted that in this paper aircraft mission planning is converted into TSP. FWA is applied, by designing a particular distance between two routes and using the reverse operator. In [27], DFWA is proposed for TSP. It first combines the k-opt heuristic with FWA and raises mutation method based on insertion operation. Its performance in small and medium cases is quite good, however, it can not compete with other swarm intelligence algorithms in large-scale cases. The DFWA-TSP in this paper can be viewed as an improvement of DFWA. We modify how to combines the k-opt heuristic. Besides, new initial location selection and improved adaptive strategy also promote its performance, especially in large-scale cases.

III. Discrete Fireworks Algorithm

Our DFWA-TSP follows the general principle of FWA. It also simulates fireworks explosion to search the best tour of TSP. Generally, it consists of 4 key parts:

1) **Fireworks Initialization** In this step, we choose the initial locations of the fireworks, which means that we pick up a certain number of tours that visit each city exactly once.
2) **Explosion** In this step, fireworks explode and generate explosion sparks. In the explosion, two parameters need to be decided first: strength and amplitude. Strength stands for the number of sparks this firework generates. The larger the strength is, the more fireworks it generates. Amplitude stands for the possible distance (difference) between a firework and the sparks it generates. The larger the amplitude is, the bigger the difference can be and the broader it is to investigate the feasible region. After the calculation of these two parameters, explosion begins. 2-opt and 3-opt heuristic method are applied to generate sparks(new solutions). Different from the traditional k-opt, there is a probability that we will accept a worse solution.
3) **Mutation** In this step, we make use of the insertion method to generate more sparks, as a supplement to edge exchange. For every firework, it will generate given number of mutation sparks.
4) **Selection** In this step, we decide the locations of fireworks in the next iteration by picking up them from current fireworks and sparks. All fireworks and sparks will be taken into consideration. The optimal one (which has the shortest tour length in this iteration) will always be kept. Then, the probability of picking up others will be calculated. The algorithm randomly chooses from them according to this probability. After selection, termination criterion is checked whether satisfied or not. If yes, the algorithm terminates and returns the optimal tour. Otherwise, the fireworks and sparks selected in this step will form the fireworks in the next iteration and algorithm will go to step 2 Explosion.

Below is the detail description of each part.

A. Fireworks Initialization

Suppose that N fireworks are needed in the iteration, we initialize each of the fireworks as follows. For every firework,

Algorithm 1 DFWA-TSP(N, C_e, C_m, m, M)

```
Note: N is the number of fireworks. C_e is the parameter of explosion, C_m is the parameter of mutation. m, M are the parameter of explosion amplitude.
FireworksInitialization(N)
for k = 1 to maxIteration do
    calculate explosion strength and amplitude
    for i = 1 to N do
        Explosion(i,C_e)
    end for
    for i = 1 to N do
        Mutation(i,C_m)
    end for
    Selection(N)
end for
```

we pick up a random city as the starting point. Then, the greedy algorithm is applied, which continues visiting the nearest unvisited city until a Hamiltonian circle is established. Generally, the greedy algorithm can efficiently generate much better initial tours than random permutation and therefore saves lots of time.

B. Explosion

Explosion is the core part of DFWA-TSP, mainly in which new solutions are generated. In this part, three things will be done: calculation of explosion strength, calculation of explosion amplitude, explosion(sparks generation).

1) Calculation of Explosion Strength: As in the other FWA, good fireworks(those with short tour length) will generally generate more sparks, while bad fireworks(those with long tour length) will generate fewer sparks. This method guarantees that we put the main computing resource in exploring the promising part of the feasible region. In DFWA-TSP, the sparks number of each firework will be calculated as follows. Suppose the tour length of firework i is L_i, the longest tour length of all the fireworks is L_w, the sparks number of firework i will be

$$N_i = round(C_e \times \frac{L_w - L_i}{\sum_i (L_w - L_i) + \epsilon}) \tag{1}$$

where C_e is the parameter to control the total number of explosion sparks, ϵ is a const small number that avoids the denominator to be 0 and round is the function mapping every number to the nearest integer.

2) Calculation of Explosion Amplitude: Due to the characteristic of discrete problems, explosion amplitude needs careful consideration. In essence, it stands for the horizon of investigating the feasible region from current locations. Therefore, we take it to be the parameter of controlling the probability of accepting a worse solution in explosion. It is adjusted by adaptive strategy. If the shortest tour length decreases quickly in recent iterations, which means that we are at a good part of the feasible region, we should focus on the local area. On the contrary, if the shortest tour length does not make progress for lots of iterations, which means that we are at a bad part of the feasible region, we should broaden our horizon. A variable θ is kept in the algorithm for this purpose. It is initialized to be $\theta_0 = \frac{M+m}{2}$ and will be updated in each iteration as follows:

$$\theta_{k+1} = \begin{cases} \max(\theta_k/1.1, m) & L_k^\star \geq L_{k-1}^\star \\ \min(\theta_k \times 1.1, M) & L_k^\star < L_{k-1}^\star \end{cases} \tag{2}$$

where $L_k^\star$ stands for the shortest tour length among all fireworks and sparks in iteration k. The effect of θ will be described in the following explosion step.

3) Explosion: As mentioned in section I, we use 2-opt and 3-opt as the basic operation for explosion.

2-opt select 2 edges each time and tries to re-connect the adjacent vertexes. It should be noted that there is only one way to re-connect the vertexes so that the result is still a Hamiltonian circle and different from the original one.

3-opt select 3 edges each time and tries to re-connect the adjacent vertexes. There can be several ways to re-connect them.

In this step, for firework i, N_i sparks will be generated. When generating each spark of this firework, one of 2-opt and 3-opt will be tried. More exactly, if $\theta_k > \sqrt{M * m}$, 2-opt method will be used, otherwise, 3-opt method will be used. When using 2-opt method, one edge is selected randomly at first. Then all other edges will be given a permutation randomly. Edges will be picked one by one and be exchanged with the selected edge. If a better solution is achieved, we accept this new solution as a spark. Otherwise, we accept it with a probability

$$Pr = \exp(-\frac{L_{new}}{L_{old}} \times \theta_k) \tag{3}$$

where L_{new} is the length of the new result and L_{old} is the length of the old tour, θ_k is the one parameter described above.

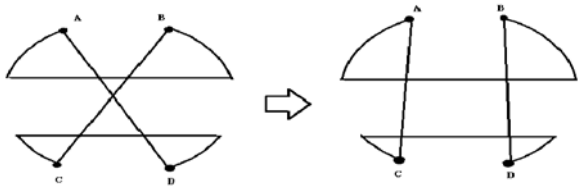

Fig. 2. An example of 2-opt heuristic

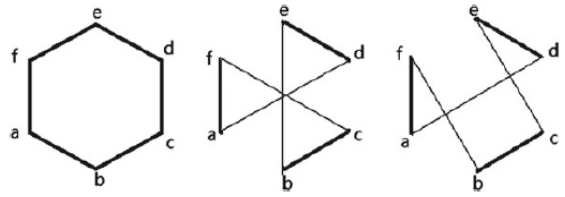

Fig. 3. An example of 3-opt heuristic(left one:original tour, right two: tours possibly generated)

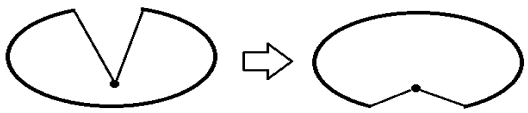

Fig. 4. A common condition that edge exchange cannot improve efficiently

If rejected, we will continue to pick up the next edge. If no solution is accepted after the last edge is tried, we will randomly exchange the edge selected with another edge and accept this solution whatever. For the 3-opt the only difference is that two edges will be selected first and every possible re-connection of three edges will be tried in certain order. It should be noticed that that larger θ_k is, the bigger chance we will accept a worse solution, which means that we keep a wider horizon to the feasible region.

C. Mutation

In [27], it is found that there is a condition that the 2-opt and 3-opt cannot improve the quality of solutions efficiently. It happens when there exists one city next to two other cities but these three cities are not adjacent in the tour, as shown in figure 4

In order to overcome this problem, we continue to use the following mutation method based on insertion. Each firework will also generate C_m mutations sparks in one iteration, apart from the explosion sparks. To generate each spark, one city is chosen randomly, then edges that are not adjacent will be given a random permutation. The algorithm will continue to check whether there is an edge that if the city chosen is inserted between the two cities of this edge, the total length can decrease. If the total length does decrease, this new solution will be accepted as a mutation spark. If not, different from [27], we have the probability to accept a worse solution and the probability is exactly the same as the one in explosion step. The algorithm will continue to try to insert the city selected into the next edge. If no solution is accepted finally, we will randomly pick up an edge that is not adjacent, insert this city and accept it as a mutation spark whatever.

D. Selection

Different from the traditional FWA on continuous problems, where selection strategy is mainly based on the distance between each firework and the optimal firework, we directly calculate the probability according to the length of the tour. All the fireworks and sparks are taken into consideration. The best firework(spark) will always be kept. This guarantees that the best tour length won't increase after new iterations. The probability of other fireworks(sparks) is calculated as follows. For firework(spark) i

$$P_i = \frac{1}{C'} \times \frac{1}{(L_i - L^\star)^2 + \epsilon} \tag{4}$$

where $C^{'} = \sum_i \frac{1}{(l_i - l^\star)^2 + \epsilon}$, and L_i and $L^\star$ stands for the total length of firework(spark) i and the best firework(spark) separately, ϵ is a const small number that avoids the denominator to be 0. We choose this method mainly because the feasible region for TSP is huge, and this method can pick up good solutions quickly.

IV. Experiments and Discussions

In this section, we demonstrate the experiments details and results.

We compare the performance of DFWA-TSP with genetic algorithm [15] and ant colony system algorithm [18]. The genetic algorithm is also initialed by greedy solutions too in our experiment. We also list the results given by greedy algorithm as a reference. The greedy algorithm we use here will try to begin from every city and build a greedy solution and finally return the shortest one among all these solutions.

We implement algorithms mentioned above in Matlab. In the following test, N, C_e, C_m, m, M is set to be 5,8,3,0.5,30 separately. We test our algorithms on TSPLIB [28] data as well as on randomly generated data.

A. TSPLIB data

We first demonstrate the results of TSPLIB data. TSPLIB is a library of sample instances for the TSP from various sources and of various types. When generating the results, the data for DFWA-TSP and genetic algorithm are recorded every 50 and 5000 iterations separately for the readability.

B. Randomly Generated Data

We also test DFWA-TSP on randomly generated large-scale data. Every city is uniformly randomly located in $[0, 1000] \times [0, 1000]$ and the distance is just Euclidean distance. Similarly, the data for DFWA-TSP and genetic algorithm are recorded every 50 and 5000 iterations separately.

C. Discussions

Based on the results above, we can conclude that DFWA-TSP has outstanding performance compared to the other two algorithms, especially on large-scale cases. The main cause should be as follows. Firstly, beneficial from the extendibility of FWA, we can naturally combine the efficient k-opt heuristic for TSP with the general framework of FWA, which makes the investigation of DFWA-TSP quite effective. Besides, although DFWA-TSP starts from greedy solutions, the explosion and mutation method make it own the ability to escape from them and make progress quickly. Therefore, greedy solutions can indeed serve as good initial locations.

We also make a short comparison between DFWA-TSP and other two algorithms.

1) Compared with Ant Colony System: Ant colony system has to contain a long process of pheromone accumulation, which can be time-consuming especially in the large-scale case. Instead, DFWA-TSP simply stores the information by the locations of fireworks. This character markedly increases the time efficiency. Besides, the explosion strategy of DFWA-TSP (edge exchange) is relatively simple compared to ant colony system, in which every ant has to pick up the entire tour. This character also helps DFWA-TSP save massive time.

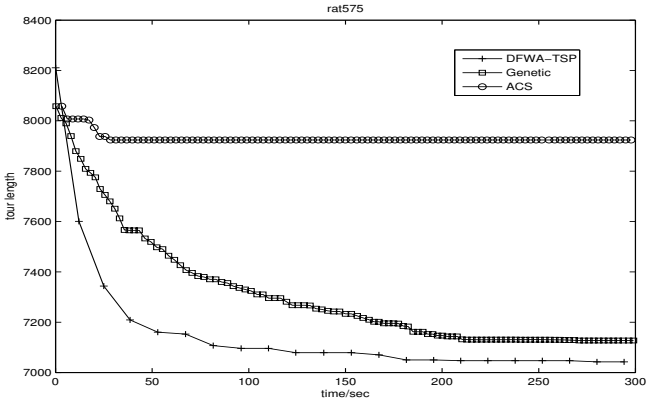

Fig. 5. rat573

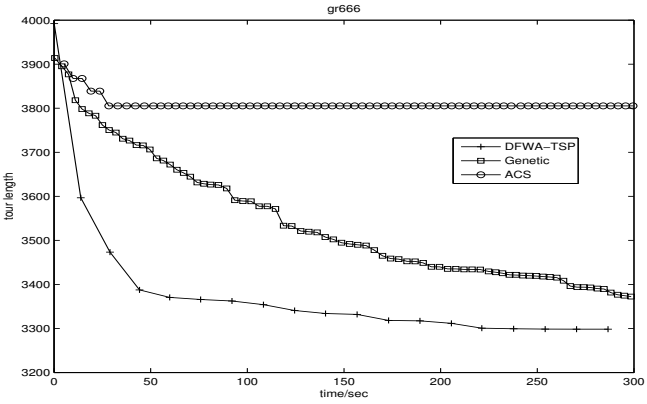

Fig. 6. gr666

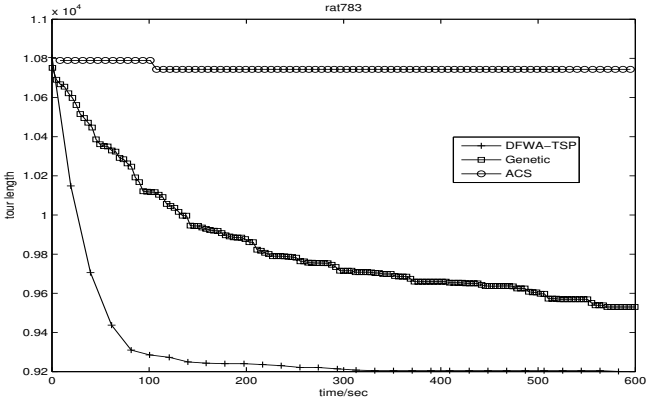

Fig. 7. rat783

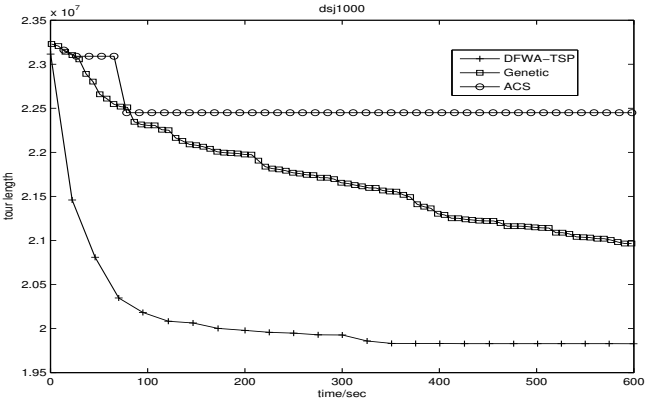

Fig. 8. dsj1000

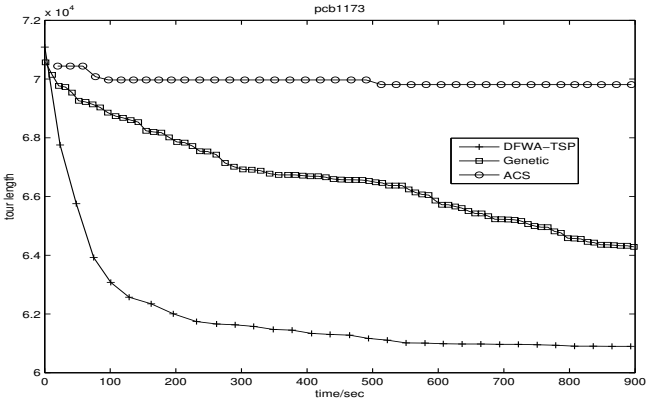

Fig. 9. pcb1173

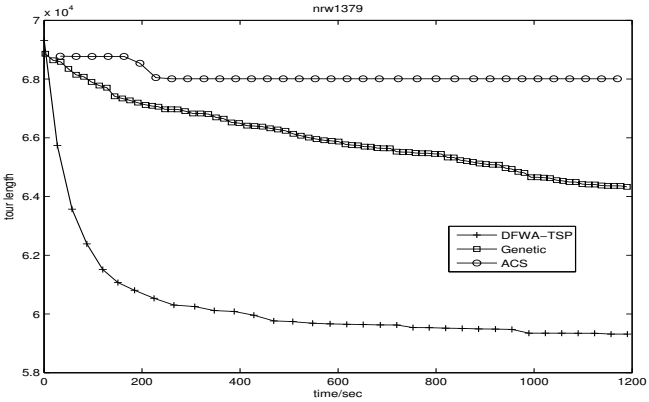

Fig. 10. nrw1379

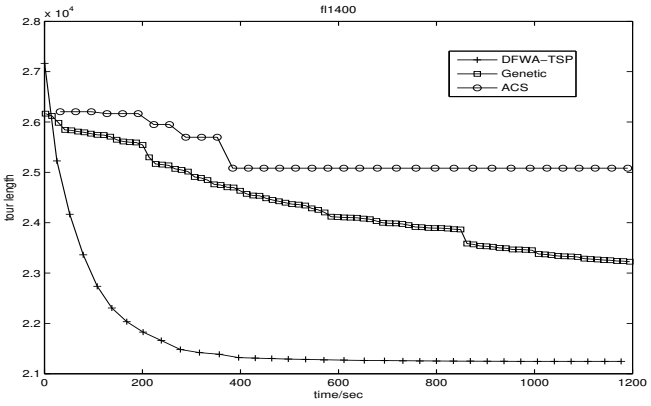

Fig. 11. fl1400

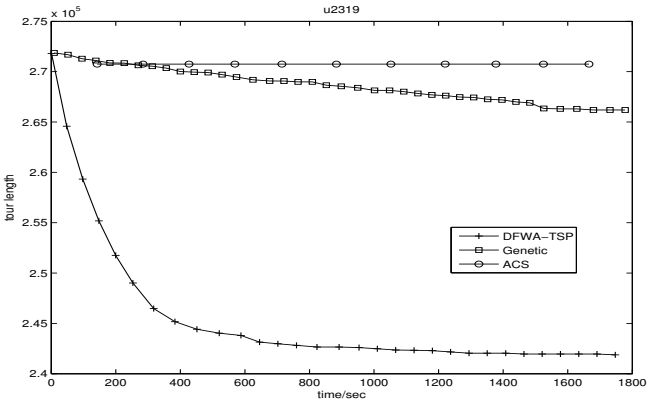

Fig. 12. u2319

TABLE I
COMPARISON ON TSPLIB DATA

	greedy	time	DFWA-TSP	Genetic	ACS
rat575	7928.031	600	7042.616	7099.299	7923.896
gr666	3835.215	900	3295.002	3293.816	3805.314
rat783	10708.953	900	9201.152	9487.500	10743.991
dsj1000	22449665.176	1200	19792718.524	20469872.595	22449665.176
pcb1173	69692.151	1200	60861.626	63745.073	69811.052
nrw1379	68326.872	1500	59268.243	63603.077	68009.193
fl1400	25709.273	1800	21237.496	22652.471	25083.725
u2319	268892.350	1800	241753.120	266056.895	270747.106

where time means the time limitation for the three swarm intelligence algorithms.

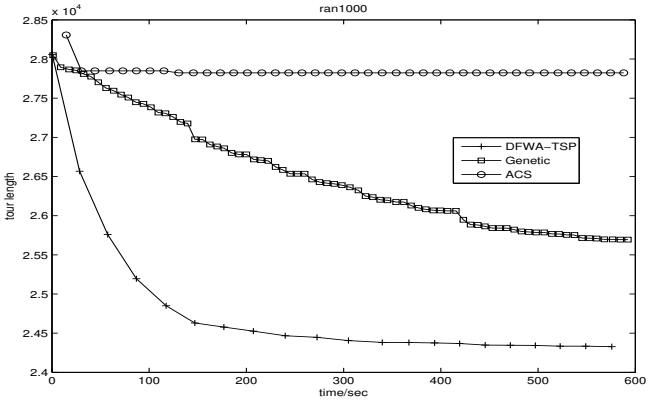

Fig. 13. random1000

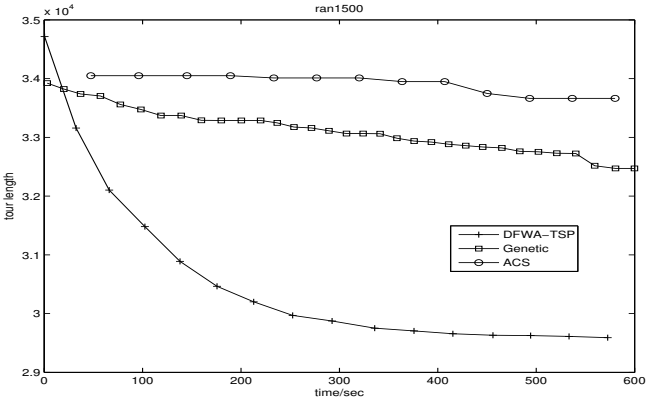

Fig. 14. random1500

TABLE II
COMPARISON ON RANDOMLY GENERATED DATA

cityNumber	greedy	time	DFWA-TSP	Genetic	ACS
1000	27783.929	600	24327.511	25674.926	27823.178
1500	33886.281	600	29578.157	32471.141	33663.991
2000	39050.267	900	34414.895	38334.551	38579.101
2500	43294.022	1000	38416.832	42957.004	43271.810
3000	48432.611	1200	42467.223	48167.555	48538.081
3500	52629.688	1200	46519.148	52608.546	52820.135

where time means the time limitation for the three swarm intelligence algorithms.

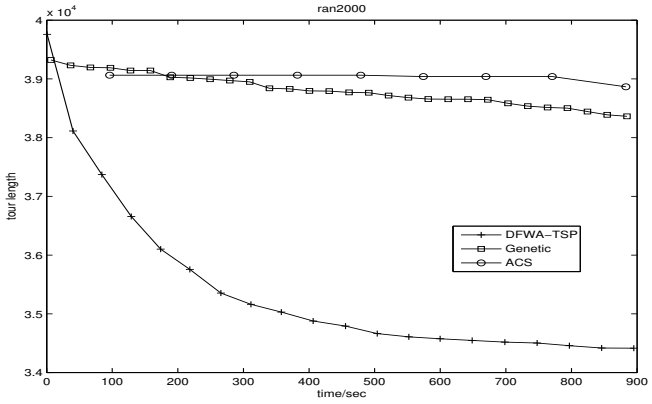

Fig. 15. random2000

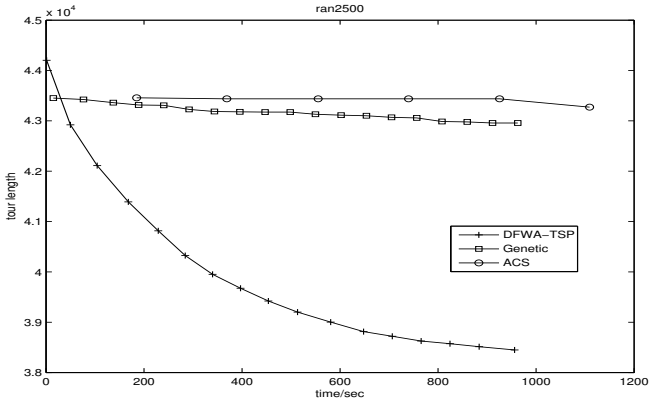

Fig. 16. random2500

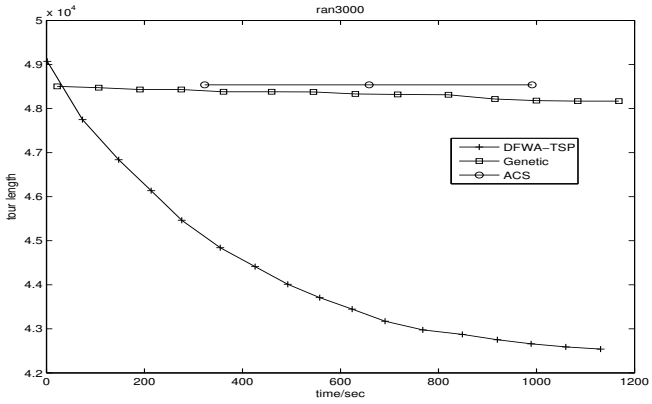

Fig. 17. random3000

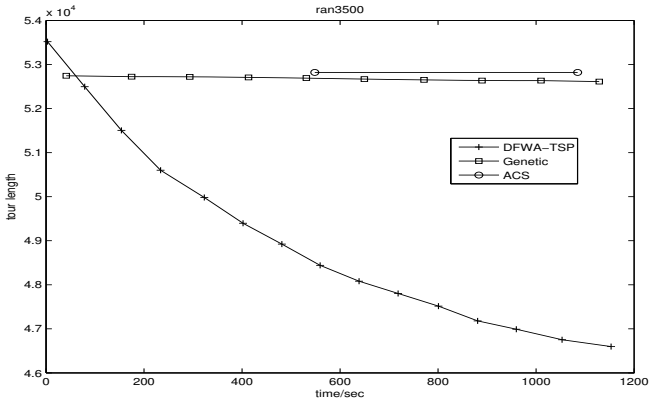

Fig. 18. random3500

2) Compared with Genetic Algorithm: Genetic algorithm mainly employs the crossover strategy to generate new solutions. It is somewhat random and not so efficient in improving the quality of solutions compared to edge exchange. As a result, DFWA-TSP typically needs fewer agents compared to the genetic algorithm.

V. Conclusions and Future Work

FWA and its variants have shown good applicants in lots of continuous optimization tasks. In this paper, we apply it to TSP and propose DFWA-TSP. 2-opt and 3-opt heuristic are selected as the basic operation. Adaptive strategy and mutation method are also used. Numerical experiments show that DFWA-TSP has outstanding performance, especially for large-scale TSP.

There are two possible ways to improve DFWA-TSP. The first is to exploit more adaptive strategies since it is hard to design a scheme suitable for all cases. Adaptive strategies can efficiently make use of the information stored by previous iterations. The second is to try to exchange more edges in the explosion. How to decide the number of edges in one explosion needs careful consideration, since generally speaking, exchanging more edges can be powerful nevertheless it needs much more computing resource.

Acknowledgments

This work was supported by the Natural Science Foundation of China (NSFC) under grant no. 61673025 and 61375119 and Supported by Beijing Natural Science Foundation (4162029), and partially supported by National Key Basic Research Development Plan (973 Plan) Project of China under grant no. 2015CB352302.

References

[1] Y. Tan and Y. Zhu, "Fireworks algorithm for optimization," *Advances in swarm intelligence*, pp. 355–364, 2010.

[2] Y. Tan, *Fireworks algorithm: a novel swarm intelligence optimization method*. Springer, 2015.

[3] Y. Tan, C. Yu, S. Zheng, and K. Ding, "Introduction to fireworks algorithm," *International Journal of Swarm Intelligence Research (IJSIR)*, vol. 4, no. 4, pp. 39–70, 2013.

[4] J. Liu, S. Zheng, and Y. Tan, "Analysis on global convergence and time complexity of fireworks algorithm," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 3207–3213.

[5] S. Zheng, J. Li, A. Janecek, and Y. Tan, "A cooperative framework for fireworks algorithm," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 14, no. 1, pp. 27–41, 2017.

[6] C. Yu, L. Kelley, S. Zheng, and Y. Tan, "Fireworks algorithm with differential mutation for solving the cec 2014 competition problems," in *Evolutionary computation (CEC), 2014 IEEE congress on*. IEEE, 2014, pp. 3238–3245.

[7] J. Liu, S. Zheng, and Y. Tan, "The improvement on controlling exploration and exploitation of firework algorithm," in *International Conference in Swarm Intelligence*. Springer, 2013, pp. 11–23.

[8] Y. Zhou and Y. Tan, "Particle swarm optimization with triggered mutation and its implementation based on gpu," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 1–8.

[9] Traveling salesman problem. [Online]. Available: http://www.math.uwaterloo.ca/tsp/

[10] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical computer science*, vol. 4, no. 3, pp. 237–244, 1977.

[11] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[12] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, no. 6, pp. 1138–1162, 1970.

[13] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[14] S. Arora, "Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems," *Journal of the ACM (JACM)*, vol. 45, no. 5, pp. 753–782, 1998.

[15] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," in *Proceedings of the first International Conference on Genetic Algorithms and their Applications*. Lawrence Erlbaum, New Jersey (160-168), 1985, pp. 160–168.

[16] L. V. Snyder and M. S. Daskin, "A random-key genetic algorithm for the generalized traveling salesman problem," *European Journal of Operational Research*, vol. 174, no. 1, pp. 38–53, 2006.

[17] N. Ulder, E. Aarts, H.-J. Bandelt, P. van Laarhoven, and E. Pesch, "Genetic local search algorithms for the traveling salesman problem," *Parallel problem solving from nature*, pp. 109–116, 1991.

[18] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on evolutionary computation*, vol. 1, no. 1, pp. 53–66, 1997.

[19] T. Stützle and H. Hoos, "Max-min ant system and local search for the traveling salesman problem," in *IEEE International Conference on Evolutionary Computation (ICEC'97)*. Citeseer, 1997.

[20] A. M. Imran and M. Kowsalya, "A new power system reconfiguration scheme for power loss minimization and voltage profile enhancement using fireworks algorithm," *International Journal of Electrical Power & Energy Systems*, vol. 62, pp. 312–322, 2014.

[21] M. A. El Majdouli and A. A. El Imrani, "Discrete fireworks algorithm for single machine scheduling problems," *International Journal of Applied Metaheuristic Computing (IJAMC)*, vol. 7, no. 3, pp. 24–35, 2016.

[22] M. Guendouz, A. Amine, and R. M. Hamou, "A discrete modified fireworks algorithm for community detection in complex networks," *Applied Intelligence*, vol. 46, no. 2, pp. 373–385, 2017.

[23] H. M. Ali and D. C. Lee, "Solving the max-sat problem by binary enhanced fireworks algorithm," in *Innovative Computing Technology (INTECH), 2016 Sixth International Conference on*. IEEE, 2016, pp. 204–209.

[24] H. Ding, L. Ke, and Z. Geng, "Route planning in a new tourist recommender system: A fireworks algorithm based approach," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE, 2016, pp. 4022–4028.

[25] Z. Liu, Z. Feng, and L. Ke, "A modified fireworks algorithm for the multi-resource range scheduling problem," in *International Conference in Swarm Intelligence*. Springer, 2016, pp. 535–543.

[26] J.-J. Xue, Y. Wang, H. Li, and J.-y. Xiao, "Discrete fireworks algorithm for aircraft mission planning," in *International Conference in Swarm Intelligence*. Springer, 2016, pp. 544–551.

[27] Y. Tan, "Discrete firework algorithm for combinatorial optimization problem," in *Fireworks Algorithm*. Springer, 2015, pp. 209–226.

[28] G. Reinelt, "Tsplibła traveling salesman problem library," *ORSA journal on computing*, vol. 3, no. 4, pp. 376–384, 1991.