A Novel Discrete Particle Swarm Optimization to Solve Traveling Salesman Problem

Wen-liang Zhong, Jun Zhang, *Member, IEEE* and Wei-neng Chen

***Abstract*—Particle Swarm Optimization (PSO), which simulates the unpredictable flight of a bird flock, is one of the intelligent computation algorithms. PSO is well-known to solve the continuous problems, yet by proper modification, it can also be applied to discrete problems, such as the classical test model: Traveling Salesman Problem (TSP). In this paper, a novel discrete PSO call C3DPSO for TSP, with modified update formulas and a new parameter c_3 (called mutation factor, to help to keep the balance between exploitation and exploration), is proposed. In the new algorithm, the particle is not a permutation of numbers but a set of edges, which is different from most other algorithms for TSP. However, it still keeps the most important characteristics of PSO that the whole swarm is guided by *pbest* and *gbest*. According to some benchmarks in TSP lib, it is proved that the proposed PSO works well even with 200 cities.**

***Index Terms*—Traveling Salesmen Problem (TSP), Swarm Intelligent, Particle Swarm Optimization (PSO), Mutation Factor**

I. Introduction

A. Traveling Salesman Problem

Traveling salesman problem (TSP) is a well-known NP-complete problem as a benchmark for new algorithms. This problem can be simply described as follow: Suppose that there is one salesman who wants to visit n cities, and his object is finding out the shortest Hamilton cycle through which he can visit all the cities once and only once, and finally return to the start one. It's obvious that the computation cost of exhausting permutations for TSP is $O(n!)$. By now, many studies on TSP has been presented, such as genetic algorithms (GA)[1], simulated annealing (SA)[2], neural network(NN)[3] and ant colony system (ACO)[4], etc.

B. Particle Swarm Optimization

Particle Swarm Optimization (PSO), as a new evolutionary algorithms was first proposed by Eberhart and Kennedy [5][6] in1995. It use a simplified model of the social system of flying birds, in which each bird (called particle) will search the whole space guided both by its previous best position (*pbest*) and the best position of the swarm (*gbest*). Due to its high performance and simple implementation, PSO has been successfully used to optimize a wide range of continuous problems [7]~[10]. Meanwhile, some researchers have tried to extend PSO to discrete areas and gotten some achievements [11][12][13]. However, in solving TSP, PSO has seldom showed its magic power. In 2000, Maurice Clerc [14] first tried to adapt PSO to resolve TSP. It was feasible but not very efficient. In [15][16][17], some Chinese scholars also proposed several different discrete PSO for TSP. However, only a few results with benchmarks less than 100 cities are given in these papers, and most of these algorithms required considerable computation cost.

In this paper, a novel discrete PSO called C3DPSO with less computation cost and higher precision is proposed. In the algorithm, a particle represents a set of edges, which will make up a whole Hamilton cycle. So the update formulas should be modified to fit this change. Ulteriorly, to prevent the algorithm trapped in a local optimality, a new parameter c_3, called mutation factor, is introduced.

Following the introduction, the basic PSO algorithm and the novel discrete PSO are illustrated in part II & III. In part IV, analysis how the new parameter c_3 improve discrete PSO is presented. Part 5 displays result of experiments, and the last part draws a conclusion.

II. *Basic Particle Swarm Optimization Algorithm*

In the basic PSO model, a particle swarm stands for a bird flock and searches the solution in a D-dimension real space. Each particle has its own position vector (a valid solution) and velocity vector (decides the particle's next action), which are both D-dimension. What's more, the fitness of each particle should be calculated by a special function to estimate the quality of a particle.

For more detail, the algorithm in pseudo code follows:

```
1)  Randomly initialize the whole swarm
2)  For(i = 0；  i<size of swarm; i++)
3)      Evaluate  f (xi)
4)  While(termination criterion is not met){
5)      For(i = 0；  i<size of swarm; i++){
6)          if( f (xi)>f (pbesti) )   pbesti = xi ;
7)          if (f (xi)>f (gbest) )   gbest = xi;
8)          Update(xi , vi);
9)          Evaluating  f (xi)
10)     }
11) }
```

In line 3), $f(x_i)$ is the fitness function to estimate the quality of solution. And the line 8) is the most important part of PSO. The particles update as follow:

This work was supported in part by NSF of China Project No.60573066，and NSF of Guangdong Project No. 5003346

Wen-liang Zhong, Jun ZHANG and Wei-neng Chen are with Department of Computer Science, SUN Yat-sen University,Guangzhou, China.

Jun ZHANG is corresponding author,e-mail: junzhang@ieee.org.

$$v_{id}^{k+1} = w * v_{id}^{k} + c_1 rand() * (pbest_{id}^{k} - x_{id}^{k}) + c_2 rand() * (gbest_{d}^{k} - x_{id}^{k}) \quad ①$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad ②$$

Here i indicates the number of particle and d is for the number of dimension. In formula □, w is called inertia weight and decide how much the pre-velocity will affected the new one. c_1 and c_2 are constant value called learning factors, which decide the degree of affection of $pbest_i$ and $gbest$. And $rand()$ devote to a random number between 0 and 1.

III. A Novel Discrete Particle Swarm Optimization for Traveling Salesman Problem

Now basic PSO will be partly modified for TSP. Suppose there are N cities to visit. First, x_i should represent a set of edges. So it's not a common vector in N-dimensional space and some definitions must be introduced to modify the update formulas.

Definition 1: $X(x, y)$ represents an edge(x, y) connected city x and y, with chosen probability X. When updating, a random number R between 0 and 1 will be given, and if $R <= X$, this edge will be chosen while the opposite if $R > X$. As only symmetric TSP is discussed in this paper, (x, y) makes no difference from (y, x);

Definition 2: Velocity v is a set of elements as $X(x, y)$, such as $\{A(a,b),B(a, c),C(f, g),\ldots\ldots\}$;

Definition 3: A substation between two position vectors is defined as a set of edges which exist in x_1 but not in x_2, and the probability of 100% is added to these edges, to make it uniform as velocity. For example, x_1 = {(1, 2),(2,3),(3, 4),(4, 5),(5,1)}, x_2 = {(1, 2),(2, 4),(4, 3),(3, 5),(5,1)}, then $x_1 - x_2$ = {1(2,3), 1(4,5)}.

Definition 4: A multiplication between a real number and a set of edges is defined like this: $A*\{B(2,3), C(3,5)\} = \{A*B(2,3), A*C(3,5)\}$, where B and C indicates the probability of corresponding edges;

Definition 5: The result of $v_1 + v_2$ is a set of $v_1 \square v_2$. In particularly, each city can appear at most 4 times in one velocity, if overflows, some edges may be discarded to prevent the velocity from a tremendous set. And the same edges can exist in one velocity, like (a,b) in follow example 2.

e.g.1: {A(a,b),B(e, f)}+{C(h, i)}={A(a,b),B(e, f),C(h, i)},

e.g.2:{A (a, b), B (a, c), C (a, d)}+{D(a,b),E(a, f)}={A(a,b),B(a, c)},C(a, d),D(a,b)}

Definition 6: The new parameter c_3 is introduced here and the reason why c_3 is much necessary is given in part IV. The formula ② is converted to formula ④ and contains three steps. First, choose edges depending on their corresponding probabilities in v_i to construct x_i'. Second, select edges from x_i according to the probability to complete the x_i'. Third, if the first two steps can't make a whole Hamilton cycle, add the absent edges with nearest principle. Specially, in the first two step, if a chosen edge will make x_i' illegal (for instance, no city can exist in more than three edges), it will be discarded. Thus it is obvious that edges in front of set vi get more opportunity to be selected because of its priority position. So we change the order of formulas to make algorithm guided by $gbest$ rather than $pbest$ and pre-velocity. The new update formulas are like this:

$$v_{id}^{k+1} = c_2 rand() * (gbest_{d}^{k} - x_{id}^{k}) + c_1 rand() * (pbest_{id}^{k} - X_{id}^{k}) + w * v_{id}^{k} \quad ③$$

$$x_{id}^{k+1} = v_{id}^{k+1} \oplus c_3 rand() * x_{id}^{k} \quad ④$$

Here if c_1, c_2 ,c_3 is bigger than 1, the probability of some edges after multiplied a random number may be larger than 100%. According to *Definition 1*, the probability would be limit to 100% forcedly in this case.

Further more, $pbest_i$ and $gbest$ easily become the same after iteration. So the $pbest_i$ won't be replaced by x_i with same length of $gbest$, in order to keep diversity. This is different from basic PSO.

Finally, the fitness function can be simple defined as the length of Hamilton cycle, and the smaller, the better.

Here is an example of update process, supposing w=0.5, c_1 = 1, c_2 =0.4, c_3 = 1. The upper three graphs of Figure 1 show $pbest_i$, $gbest$ and x_i before updating. And the lower ones indicate how to build the new x_i' follow the three steps.

Before update:

$pbest_i$: {(1, 2),(2, 3),(3, 4),(4, 5),(5, 6),(6, 7),(7, 8),(8, 9),(9, 10),(10,1)} (see the first graph of figure1)

$gbest$: {(1, 3),(3, 2),(2,5),(5, 6),(6, 7),(7, 8),(8, 9),(9, 10),(10, 4),(4,1)} (see the second graph of figure1)

x_i : {(1, 2),(2, 4),(4, 5),(5, 6),(6, 7),(7, 8),(8,10),(10, 9),(9, 3),(3,1)} (see the third graph of figure1)

v_i : {0.3(2, 4), 0.4(3,5),0.6(7,9) }

Update process:

$gbest$ - x_i = {1(2,3),1(2,5),1(8,9),1(10,4),1(4,1)},suppose $rand()$= 0.5;

$pbest_i$ - x_i = {1(2,3),1(3,4),1(8,9),1(10,1)} , suppose $rand()$=0.6, then

v_i'={0.2(2,3),0.2(2,5),0.2(8,9),0.2(10,4),0.2(4,1),0.6(2,3),0.6(3,4),0.6(8,9),0.6(10,1), 0.15(2,4),0.2(3,5),0.3(7,9)}

$v_i' \oplus c_3 rand() * x_i$, suppose $rand()$= 0.7

step 1: select edges from v_i' depend on probability, and x_i'= *{(2,3), (2,5), (3,4), (8,9), (10,1),(7,9)}*

step 2: add edges from x_i depend on probability (c_3*0.7 here), so x_i'={(2,3),(2,5),(3,4), (8,9), (10,1),(7,9),*(8,10),(6,7)* }

step 3: add the absent edges with nearest principle, then x_i' ={ (2,3), (2,5), (3,4), (8,9),(10,1),(7,9),(8,10),(6,7),*(5,6), (1,4)*}

Italics letter indicates edges which join in x_i' at current step. So in the end, x_i' = {(1, 4),(4, 3),(3, 2),(2, 5),(5, 6),(6, 7),(7, 9),(9, 8),(8, 10),(10,1)}

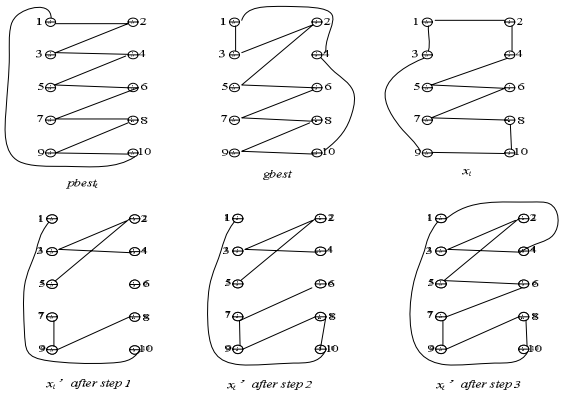

Fig. 1. An whole example for the update of a particle i

IV. ANALYSIS OF MUTATION FACTOR, C_3

Figure 2 shows that how many edges come from the 3 steps in benchmark KroA100 without c_3, that is say, c_3*rand() is set to 100% and the other parameters are set as description in part V. As a result, edges will be chosen from x_i in step 2 as mush as possible. In figure 2, the middle line indicates that the number of edges from v_i' decrease with the increasing of iteration, while the ones from x_i is opposite, as the top line shows. The reason is that $pbest_i$, $gbest$ and x_i become very similar in the later phase (see in figure 3), so new edges added to v_i' become fewer and fewer according to formula ③ in part III. Ulteriorly, as the probabilities are 100%, edges in x_i will be surely chosen to x_i' if it is possible (see *definition 6* in part III). So, after step 1 & 2, x_i' is very likely to x_i, and few new edges are brought into x_i' in step 3, as showed by the lowest line in figure 2. It will probably make the swarm premature and stop searching a better path. The experiment results in part V also demonstrate this.

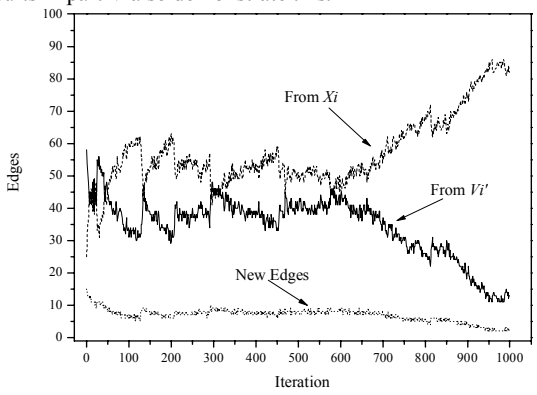

Fig. 2. Distribution of edges in KroA100 in 3 steps without c3.

To avoid premature, parameter c_3 is introduced to step 2. c_3, multiplied by a random number between 0 and 1, is the probability of an edge in x_i being selected. That is said, not all the legal edges in x_i will be chosen in step 2. So, x_i' after the first two steps is probably lack of some edges, and new edges will be added in step 3. Figure 4 illustrates that about 10% new edges are added to x_i', like mutation. And figure 5 shows that x_i always keep different from *gbest* at least 3%~5% rate, which indicates x_i maintaining to search the solution space. So c_3 is also called mutation factor and it will help to keep balance between exploration and exploitation of this algorithm.

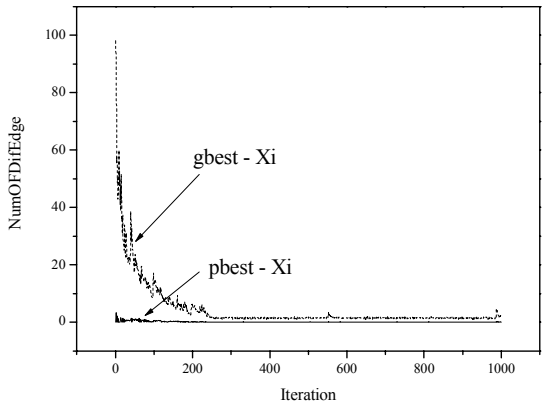

Fig. 3. Average number of different edges among *gbest*, $pbest_i$ and x_i in algorithm without c_3 in KroA100

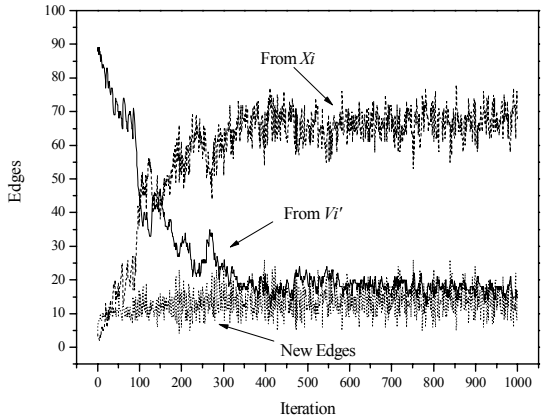

Fig. 4. Distribution of edges in KroA100 in 3 steps with c3.

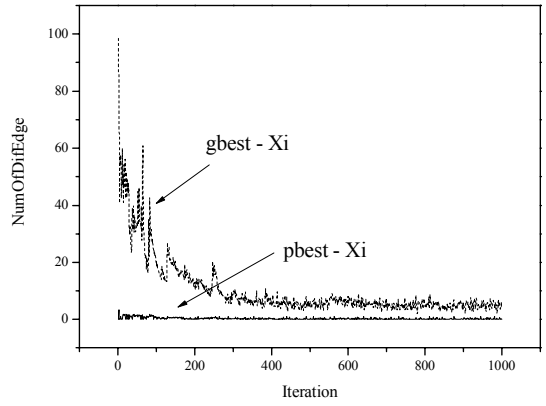

Fig. 5. Average number of different edges among *gbest*, $pbest_i$ and x_i in algorithm with c_3 in KroA100

V. *Experiment Results*

A. Related Settings of the Experiment

Six classic TSP benchmarks from TSP lib have been tested by using the novel discrete PSO and ACO [18] (for comparison) algorithm, which are Burma14, Eil51, Berlin52, Eil76, KroA100 and KroA200. Because there are only a few researchers who have optimized TSP with PSO, and almost all of their benchmarks are smaller than 100 cities. So Burma14, Eil51, Berlin52 and Eil76 are adapted to compare with the other discrete PSO, and KroA100, 200 are used to proved the efficiency of our algorithm in solving larger TSP.

Each test has run for 50 times. And the computer system for test is consisted of Celeron 2.26GHz and 256 MB memory with Windows XP.

In C3DPSO, parameters w, c_1, c_2, c_3, n, *iter* are set to 0.6, 1.5, 2, 2, 30, N*10, respectively. n and N are the size of the swarm and the number of cities, respectively. *iter* represent number of iteration.

In ACO, α=0.1,ρ=0.1,β=2,$q0$=0.9, n=10, as recommended in [18]. And *iter* is set to N*30, so the numbers of computing the length of Hamilton cycle in the two algorithms are the same.

B. Results

The average and best result, relative error, running time and percentage of success computed by the two algorithms are showed in table 2. In this table, "average" is the average length of 50 running, while the "best" is the best one. "Error" is ("average" – "optima" / "optima"), where the "optima" is given by TSP Lib. And "Time" is the average cost of 50 running, estimated by milliseconds. "Success" here is defined as *sucTimes/runningTimes*. And "*sucTimes*" is the number of Hamilton cycles whose lengths are shorter than "*limitation*". $limitation = ((floor(N/50)+1)*0.01+1)*optimal$.

Here "*floor*(*x*)" means the largest integer smaller than *x*.

For example, in Eil76, limitation = (*floor*(76/50) + 1)*0.01+1)* 538 = 1.02*538=549.

For there are seldom results of PSO in solving TSP presented, only those in [16] are used for comparison in table 2, which was made by PSO-TS, PSO-TS-2opt, PSO-TS-CO and PSO-TS-CO-2opt. The reason to choose it is that it gives more and harder benchmarks than other papers we have found out.

In table 2, results made by discrete PSO proposed in this paper without c_3, that is say c_3*rand() in formula ④ is always 100%, are presented first. It proved that when N <= 76, the algorithm performs not bad, with relative error smaller than 20%. But its precision decreases rapidly with growing N.

C3DPSO are much more effective in all benchmarks than other algorithms, as showed in table 2, whose relative error are low than 5% even in KroA100 and KroA200. Figure 6, 7 & 8 denote the convergence of the proposed PSO with and without c_3 in Berlin52, Eil76 and KroA200. They make it evident the mutation factor c_3 play an important role to enhance the searching ability of the algorithm. Compared to other discrete PSO done in [16], C3PSO improves greatly either in precision or computational cost. In particularly, there may be some mistakes in Burma14 give by [16], because the lengths are much shorter than those given by TSPLib.

In table 2, the boldfaced and italic numbers denote the winner between C3DPSO and ACO. It's obvious that C3DPSO makes advantages in most benchmarks with the same computational cost. Specially, the running times of C3DPSO increase slower than ACO. That is, C3DPSO will perform better and better than ACO as N becomes bigger and bigger.

TABLE 2
EXPERIMENT RESULTS ARE SHOWED IN THE TABLE. BOLD AND ITALIC NUMBERS ARE THE BEST ONES IN THE COLUMN.

		Burma14	Eil51	Berlin52	Eil76	KroA100	KroA200
PSO without C_3	Average	3342.88	473.34	8730.18	626.94	27725.4	41204.5
	Best	3323	443	7724	566	24343	36464
	Error (%)	0.60	11.11	15.75	16.53	30.27	40.30
	Suc (%)	86	0	0	0	0	0
	Time(ms)	223	3878	3885	11250	21114	149340
C3DPSO	Average	3324.04	***433.64***	***7598.76***	***551.72***	***21689.30***	***30374.30***
	Best	3323	427	7542	***540***	***21296***	***29563***
	Error (%)	***0.03***	***1.793***	***0.753***	***2.550***	***1.914***	***3.427***
	Suc (%)	***100***	60	***74***	***36***	***84***	***82***
	Time(ms)	267	***4060***	***4116***	***11594***	***23950***	***198551***
PSO-TS	Average	33.69	582.501	8100.105	588.712	--------	--------
	Error (%)	9.12	35.47	7.37	9.98	--------	--------
	Time(ms)	1000	30000	120000	60000	--------	--------
PSO-TS-2opt	Average	30.88	459.273	7938.041	564.523	--------	--------
	Error (%)	0	6.81	5.22	5.46	--------	--------
	Time(ms)	1000	30000	120000	60000	--------	--------
PSO-TS-CO	Average	33.94	500.231	8323.432	604.126	--------	--------
	Error (%)	10	16.34	10.33	12.86	--------	--------
	Time(ms)	1000	30000	120000	60000	--------	--------
PSO-TS-CO-2opt	Average	30.88	440.900	7704.242	560.712	--------	--------
	Error (%)	0	2.54	2.12	4.75	--------	--------
	Time(ms)	1000	30000	120000	60000	--------	--------
ACO	Average	3329.68	434.14	7694.92	560.72	21821.16	31057.02
	Best	3323	***426***	7542	543	21341	30083
	Error (%)	0.20	1.91	2.03	4.22	2.53	5.75
	Suc (%)	98	***66***	56	6	64	36
	Time(ms)	***107***	5765	5897	18229	41642	338981
Optimal		3323	426	7542	538	21282	29368
Limitation		3356	435	7693	549	21920	30836

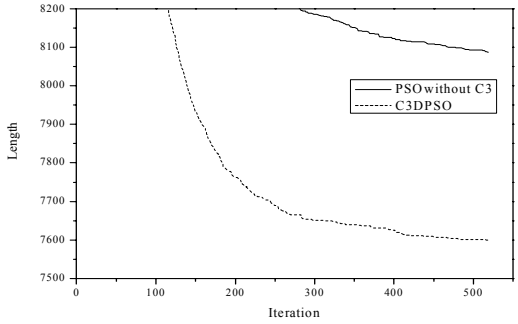

Fig. 6. Convergence of the algorithm in Benlin52

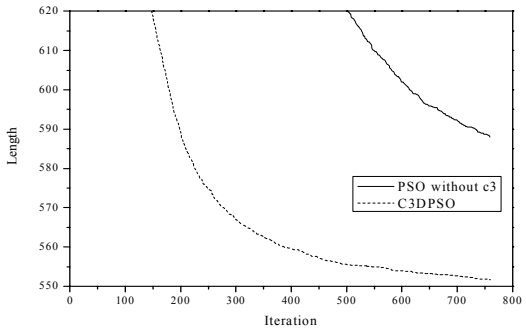

Fig. 7. Convergence of the algorithm in Eil76

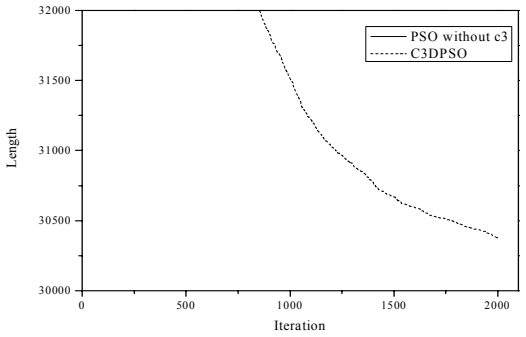

Fig. 8. Convergence of the algorithm in KroA200

VI. CONCLUSION

In this paper, a novel discrete PSO called C3DPSO to approach TSP is proposed. It was proved that the algorithm is much more effective than previous work done by [14]~[17]. With mutation factor c_3, some new edges are brought into the solutions to keep balance between exploitation and exploration. Further work should focus on improving the algorithm, such as how to set effective parameters, and how to adjust the parameters dynamically, and so on.

REFERENCES

[1] Shubhra.Sankar.Ray, Sanghanitra Bandyopadhyay, Sankar.K.Pal, "New Operators of Genetic Algorithms for Traveling Salesman Problem". In 17th International Conference on Pattern Recognition.2004:497-500.

[2] Chi-Hwa Song, Kyunghee Lee, Won Don Lee, "Extended simulated annealing for augmented TSP and multi-salesmen TSP Neural Networks", 2003. Proceedings of the International Joint Conference on Volume 3, 20-24 July 2003 Page(s):2340 – 2343

[3] Gee. A.H, Prager. R.W "Limitations of neural networks for solving traveling salesman problems", Neural Networks, IEEE Transactions on Volume 6, Issue 1, Jan. 1995 Page(s):280–282

[4] Macro Dorigo, Vittorio Maniezzo and Alberto Colorni, "Ant system: optimization by a colony of cooperating agents," IEEE Trans. on systems man, and cybernetics - part B: cybernetics, vol. 26, 1996, pp. 29-41.

[5] Kennedy J, and Eberhart R.C, "Particle swarm optimization", Proceeding of the 1995 IEEE International Conference on Neural Networks, Perth, Australia, pp. IV: 1942-1948, 1995.

[6] Eberhan, R. C. and Kennedy, J. "A new optimizer using particle swarm theory", Proceedings of the Sixth International Symposium on Micro machine and Human Science, Nagoya, Japan. pp. 3943, 1995

[7] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm adaptation, " in Evolutionary Programming VII, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds.Berlin, Germany: Springer-Verlag, 1997, pp. 591–600.

[8] P. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences, " in Evolutionary Programming VII, V.W.Porto, N. Saravanan, D. Waagen, and A.E.Eiben, Eds. Berlin, Germany: Springer-Verlag, 1998, pp. 601–610.

[9] Shi Y, Eberhart R C, "A modified particle swarm optimizer"，IEEE International Conference of Evolutionary Computation , Anchorage , Alaska , May 1998

[10] Shi Y, Eberhart R C. "Fuzzy adaptive particle swarm optimization ". Proceedings of the Congress on Evolutionary Computation. Seoul , Korea , 2001

[11] Kennedy J, Eberhart R C. "A discrete binary version of the particle swarm algorithm", Proceedings of the World Multiconference on Systemics , Cybernetics and Informatics . Piscataway , NJ : IEEE Service Center , 1997. 4104～4109

[12] Chandrasekaran. S, Ponnambalam.S.G., Suresh. R.K., Vijayakumar. N., "A Hybrid Discrete Particle Swarm Optimization Algorithm to Solve Flow Shop Scheduling Problems", Cybernetics and Intelligent Systems, 2006 IEEE Conference on June 2006 Page(s):1 – 6

[13] Quan-Ke Pan, Tasgetiren. M.F, Yun-Chia Liang, "A Discrete Particle Swarm Optimization Algorithm for Single Machine Total Earliness and Tardiness Problem with a Common Due Date", Evolutionary Computation, 2006. CEC 2006. IEEE Congress on 16-21 July 2006 Page(s):3281 - 3288

[14] Maurice Clerc, "Discrete Particle Swarm Optimization Illustrated by the Traveling Salesman Problem", http://www.mauriceclerc.net, 29 February 2000

[15] Cuiru Wang, Jiangwei Zhang, Jing Yang, Chaoju Hu, Jun Liu, "A Modified Particle Swarm Optimization Algorithm and its Application For Solving Traveling Salesman Problem"，Neural Networks and Brain, 2005. ICNN&B '05. International Conference on Volume 2, 13-15 Oct. 2005 Page(s):689 - 694

[16] Wei Pang, Kang-Ping Wang etc, "Modified Particle Swarm Optimization based on Space Transformation for Solving Traveling Salesmen Problem"，Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, 26-29 August 2004 P2342~2346

[17] Wei Pang, Kang-Ping Wang, etc "Fuzzy Discrete Particles Swarm Optimization for Solving Traveling Salesman Problem" Proceeding of the Fourth International Conference on Computer and Information Technology 2004, P796 – 800

[18] Marco Dorigo, Luca Maria Gambardella, "Ant Colony System: A Cooperative Leaning Approach to the Traveling Salesman Problem", IEEE Trans. On Evolutionary Computation VOL 1, No 1, April 1997