Applied Soft Computing 11 (2011) 3680–3689

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search

Xiutang Geng[a,*], Zhihua Chen[b], Wei Yang[a], Deqian Shi[a], Kai Zhao[a]

[a] No. 202 Institute of China Ordnange Industry, Xianyang 712099, China
[b] Key Laboratory of Image Processing and Intelligent Control, Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China

ARTICLE INFO

Article history:
Received 4 July 2009
Received in revised form 28 July 2010
Accepted 30 January 2011
Available online 25 February 2011

Keywords:
Simulated annealing
Greedy search
Traveling salesman problem
Adaptive
Combinatorial optimization
Heuristic

ABSTRACT

The traveling salesman problem (TSP) is a classical problem in discrete or combinatorial optimization and belongs to the NP-complete classes, which means that it may be require an infeasible processing time to be solved by an exhaustive search method, and therefore less expensive heuristics in respect to the processing time are commonly used in order to obtain satisfactory solutions in short running time. This paper proposes an effective local search algorithm based on simulated annealing and greedy search techniques to solve the TSP. In order to obtain more accuracy solutions, the proposed algorithm based on the standard simulated annealing algorithm adopts the combination of three kinds of mutations with different probabilities during its search. Then greedy search technique is used to speed up the convergence rate of the proposed algorithm. Finally, parameters such as cool coefficient of the temperature, the times of greedy search, and the times of compulsive accept and the probability of accept a new solution, are adaptive according to the size of the TSP instances. As a result, experimental results show that the proposed algorithm provides better compromise between CPU time and accuracy among some recent algorithms for the TSP.

1. Introduction

The TSP is a typical example of a very hard combinatorial optimization problem, and can be understood as a search for the shortest closed tour that visits each city once and only once. As is well known, the TSP belongs to a family of NP-complete problems [27] whose computational complexity rises exponentially by increasing the number of cities. For many current TSP applications such as data association, vehicle routing [21,30], data transmission in computer networks [25], scheduling, drilling of printed circuits boards [26], analysis of the structure of crystals, clustering of data arrays, assignment of routes for planes of a specified fleet, image processing and pattern recognition [2], and transportation and logistics, finding suboptimal solutions with a reasonable cost may be more advantageous, and therefore the great interest in efficient heuristics to solve the TSP.

The TSP has received considerable attention over twenty years and various methods based on deterministic or probabilistic heuristics have been proposed to solve the known problem. The methods include branch-and-bound [9], particle swarm optimization [21,22,32], neural network [6,23,29], tabu search [11], ant colony optimization [5,10,20,34], genetic algorithms [3,13,36], self-organizing maps [1], simulated annealing [18,19,24,33], elastic net [7,8,37], and Lagrangean relaxation [38].

Among the above methods, the branch-and-bound approach is an exact algorithm, which is only adapted the TSP with small scale. The particle swarm optimization algorithm has been successfully applied in solving the TSP with the problem size is ranged from 51 to 76, and been successfully applied in solving the vehicle routing problem and the probabilistic TSP. The neural network algorithms converge significantly faster than other algorithms do with neglecting the accuracy of the solution. The tabu search algorithm's convergence rate is fast while its effective is relatively poor. The ant colony optimization algorithm has been mainly used to solve small-scale TSP, fixed destination multi-depot multiple TSP and TSP with time windows. The genetic algorithm has been utilized for TSP with other approaches such as immunity and other heuristics. The self-organizing map strategy has been applied in solving TSP with the problem size is ranged from 51 to 442. It is difficult for the standard simulated annealing algorithm to improve its quality-time trade-off, and therefore simulated annealing technique is usually combined with other heuristics to solve the TSP. The elastic net approach has been considered to solve the large TSP, and the validity needs to be improved.

Simulated annealing is a new approach to the approximate solution of different combinational optimization problems. It was

* Corresponding author. Tel.: +86 29 33787914.
E-mail addresses: gxt1028@163.com (X. Geng), chenzhihua@mail.hust.edu.cn (Z. Chen).

1568-4946/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2011.01.039

Table 1
The results that VI, BI and BR are performed on benchmark instance *rat*_783.

No.	Mutation mode	Error (%)	CPU time (s)
1	Mutation VI	7.04	88.50
2	Mutation BI	48.72	121.96
3	Mutation BR	2.31	72.62
4	Mutation VI and BI	16.38	104.19
5	Mutation VI and BR	2.09	81.55
6	Mutation BI and BR	7.50	96.29
7	Mutation VI, BI and BR	2.09	75.06

originally proposed by Kirkpatrick, Gelatt and Vecchi [17], and Cerny [4], who reported promising results based on sketchy experiments. For simulated annealing technique, since then there has been an immense outpouring of papers such as graph partitioning [15], graph coloring [16], determine the thickness of a graph [28], logic programming [31], and machine scheduling [14]. Differ from other heuristics, simulated annealing in essence is a method for improving local optimization, and it needs less memory space. However, simulated annealing algorithm generally requires longer computational time to find a better solution than other heuristics such as neural network algorithms.

In our early work [35], an effective two-stage simulated annealing (Two-stage SA) algorithm was present for the TSP. For the Two-stage SA algorithm, in the first stage, a simple simulated annealing algorithm is proposed to obtain some appropriate solutions or closed tours. In the second stage, an effective simulated annealing algorithm is present to obtain solutions with good quality based on the solutions or closed tours obtained by the simple simulated annealing. Finally, the Two-stage SA algorithm is performed on 23 TSP benchmark instances with scale from 51 to 783. The numerical results show the proposed algorithm in [35] is difficulty in solving the TSP benchmark instances with scale exceeded 1000 cities based on CPU time.

Based on the reason, this paper presents a new meta-heuristics approach called adaptive simulated annealing algorithm with greedy search to solve TSP, and the original purpose of this paper is to look for more effective mutations, more rational greedy search strategy and more universal adaptive parameter control of the proposed algorithm to achieve better results and a faster convergence in solving TSP with large-scale cities from 51 to 85,900.

The rest of this paper is organized as follows. Section 2 describes the TSP and designs its mathematical formulation. The mutation strategy, greedy search strategy and adaptive parameter control strategy of the proposed algorithm based on simulated annealing and greedy search are proposed in Section 3, respectively. Section 4 selects some recent published competitive algorithms that have been used to solve TSP and gives the comparisons of experimental results between our algorithm and the recent algorithms. This paper is concluded in Section 5.

2. The traveling salesman problem

The TSP is one which has commanded much attention of mathematicians and computer scientists specifically because it is so easy to describe and so difficult to solve. The problem can simply be stated as: a search for the shortest closed tour that visits each city once and only once. A complete historical development of this and related problems can be seen in [12]. The importance of TSP is that it is representative of a larger class of problems known as NP-complete. Specifically, if one can find an efficient algorithm that will be guaranteed to find the optimal solution in a polynomial number of steps for the TSP, then the efficient algorithm could be found for all other problems in the NP-complete class. To date, however, no one has found a polynomial-time algorithm for TSP.

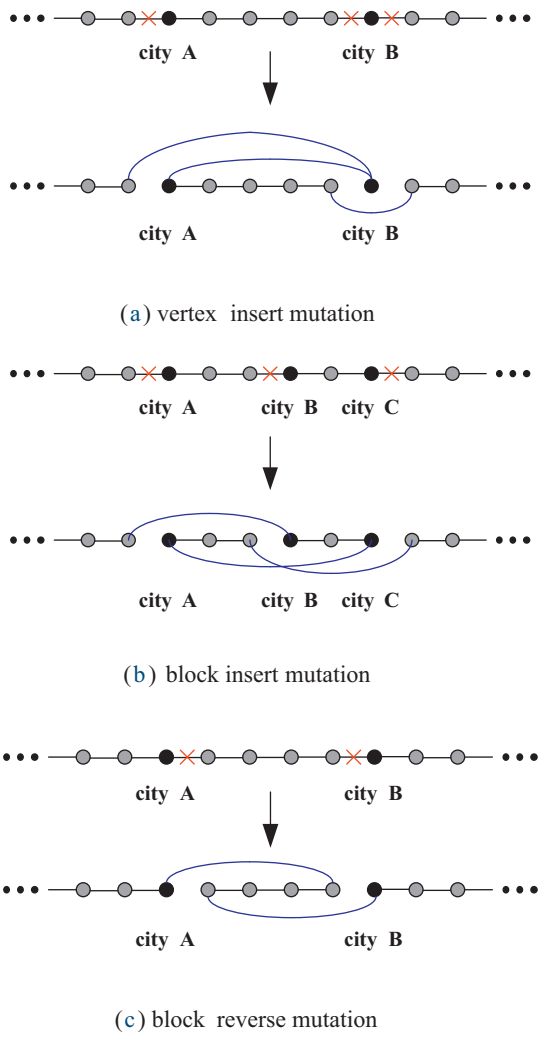


Fig. 1. The three mutations (a) vertex insert mutation (b) block insert mutation (c) block reverse mutation.

The first step to solve TSP must be to find a good mathematical formulation. In the case of TSP, the mathematical structure is a graph where each city is denoted by a point and lines are drawn connecting every two nodes. Associated with every line is an edge, distance or cost. When the salesman can get from every city to other cities directly, then the graph is said to be complete. A round-trip of the cities corresponds to some subset of the lines, and is called a tour or a Hamiltonian cycle in graph theory. The

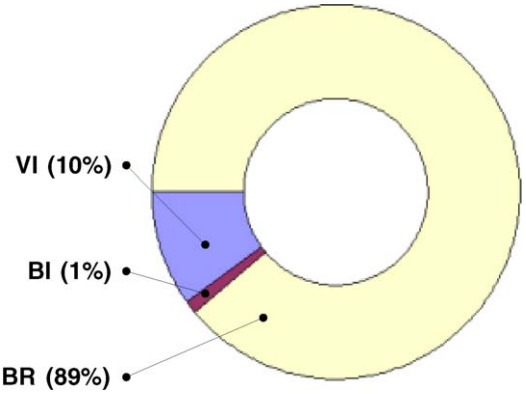


Fig. 2. The ratios of using VI, BI and BR.

Table 2
Scaling coefficients for adjusting the CPU time of the counterpart algorithms [1,6,23,29,32,34,36–38] with respect to the CPU time of the ASA-GS.

Counterpart algorithms	Processing system	Program language	Scaling coefficients
MGSOM [1]	NA	NA	NA
Memetic-SOM [6]	2.0 GHz	Java	0.71
RABNET-TSP [23]	3.0 GHz	MATLAB	1.06
CONN [29]	1.4 GHz	C++	0.49
Dis-PSO [32]	2.0 GHz	NA	0.71
ACOMAC-DNN [34]	NA	NA	NA
GCGA [36]	2.8 GHz	C++	0.99
Improved-EN [37]	1.8 GHz	NA	0.64
ELC-LR [38]	1.8 GHz	C++	0.66
Our algorithm (ASA-GS)	2.8 GHz	C++	1.00

length of a tour is the sum of the lengths of the lines in the round-trip.

In this paper, symmetric TSP (TSP) is considered. To formulate TSP, one notes that the direction traversed is immaterial, so that $L_{i,j} = L_{j,i}$, where $L_{i,j}$ denotes the edge, distance or cost between city i and city j. Since direction does not now matter, one can consider the graph where there is only one edge between every two nodes. To find a closed tour in this graph, one must select an orderly sequence of the n cities such that a feasible tour is contained. Hence, the TSP can be formulated as:

$$\min F(c) = \sum_{i=1}^{n-1} L_{c_i,c_j} + L_{c_1,c_n} \tag{1}$$

$$\begin{cases} c = (c_1, c_2, \cdots, c_n) \text{ and if } i \neq j \text{ then } c_i \neq c_j \\ c_i \in \{1, 2, \cdots, n\} \end{cases} \tag{2}$$

3. The adaptive simulated annealing algorithm with greedy search for the TSP

On the one hand, effective mutation strategy in the local search is crucial for some heuristics algorithms to solve the NP-complete problems such as TSP. On the other hand, the combination of some mutation strategies is more advantageous than one mutation strategy. In addition, different mutation strategies usually take different effective for the same problem. As a result, three effective mutation strategies such as vertex insert mutation (VI), block insert mutation (BI) and block reverse mutation (BR) are used with different frequencies in solving TSP in this paper, and the three mutations are described in Fig. 1(a)–(c), respectively. In order to validate the effectiveness of the three mutations, our algorithm is performed on the TSP benchmark instance *rat*_783 with different condition, and the results can be seen in Table 1.

In Table 1, the columns 'NO.', 'Mutation mode', 'Error', and 'CPU time' denote the sequence number of different run, the combination of the three mutations, the percent difference of the average closed tour, and the average CPU time for each trials, respectively.

As shown in Table 1, mutation BR obtains best solution with shortest CPU time, while mutation BI obtains worst solution with longest CPU time. Take the combination of the three mutations into account, VI and BR obtain best solution as mutations VI, BI and BR, while the latter takes shortest CPU time. To sum up the above arguments, our algorithm is more effective when the three mutations are used with different probability that mutation BR occupies the most part while mutation BI occupies the smallest part. As a result, as illustrated in Fig. 2, the ratios of VI, BI and BR in the proposed algorithm are designed for 10%, 1% and 89%, respectively.

The standard simulated annealing algorithm can find a better solution at the cost of vast time. As a result, faster convergence strategy such as greedy search technique is considered in order to achieve a reasonable trade-off between computation time, solution quality, and complexity of implementation. Basing on the standard

Table 3
Parameters used in the ASA-GS.

Parameters	Initial value	Meaning
N	–	The number of the cities
OPT	–	The optimal tour length
$t_{initial}$	1000.0	The initial temperature
t_{cool}	See formula (5)	The cool coefficient of the temperature
t_{greedy}	See formula (5)	The times of greedy search
t_{end}	0.005 or 0.0025	The end temperature
$t_{current}$	$t_{current} = t_{current} \times t_{cool}$	The temperature of the current state
t_v	See formula (5)	The times of compulsive accept

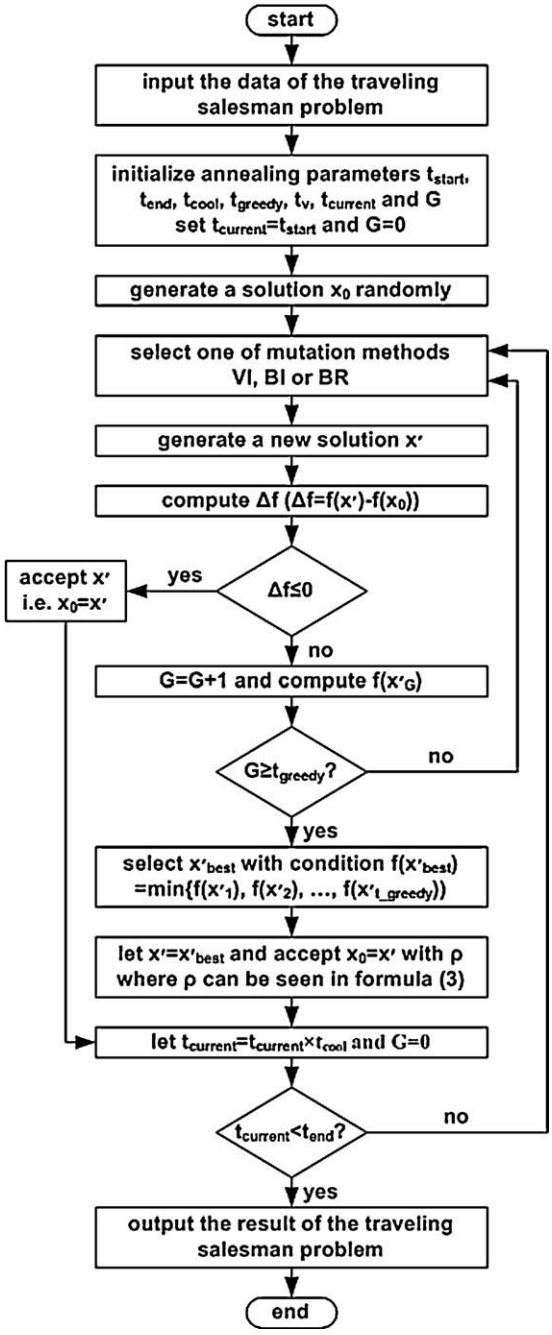


Fig. 3. The flow diagram of the ASA-GS.

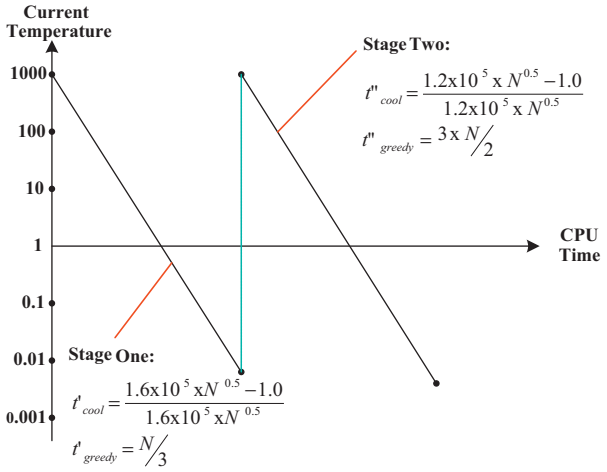


Fig. 4. The two-stage adaptive local search strategy.

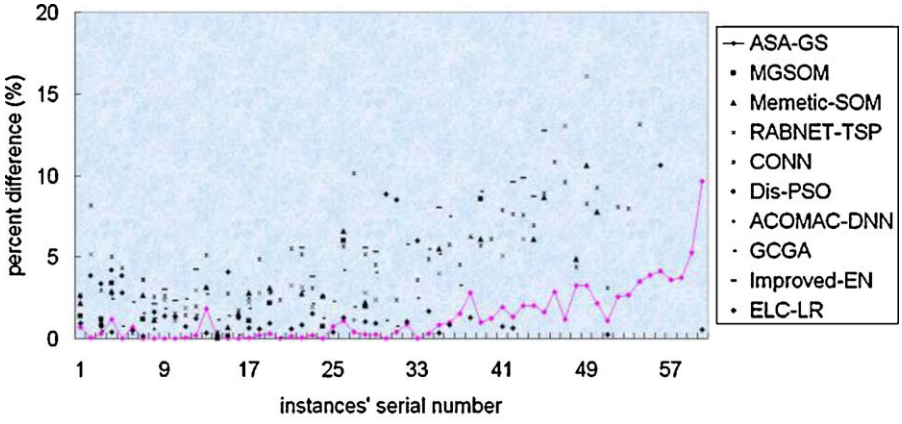
simulated annealing algorithm, the concept of the greedy search is that the old solution c^{old} is replaced by a new neighbor solution c^{new_1} when $F(c^{new\_1}) < F(c^{old})$ according to formula (1), and then turn to the next step. Otherwise the next new neighbor solution c^{new_2} is generated, and the old solution c^{old} is replaced by the new neighbor solution c^{new_2} when $F(c^{new\_2}) < F(c^{old})$, and then turn to the next step. As far as the $t_{greedy}th$ new neighbor solution $c^{new\_t_{greedy}}$ is generated. Finally, the old solution c^{old} is replaced by the best new solution c^{best} among the t_{greedy} neighbor solutions with probability ρ.

$$\rho = e^{-(F(c^{best}) - F(c^{old})/t_{current}) \times (10.0 \times N/OPT)} \tag{3}$$

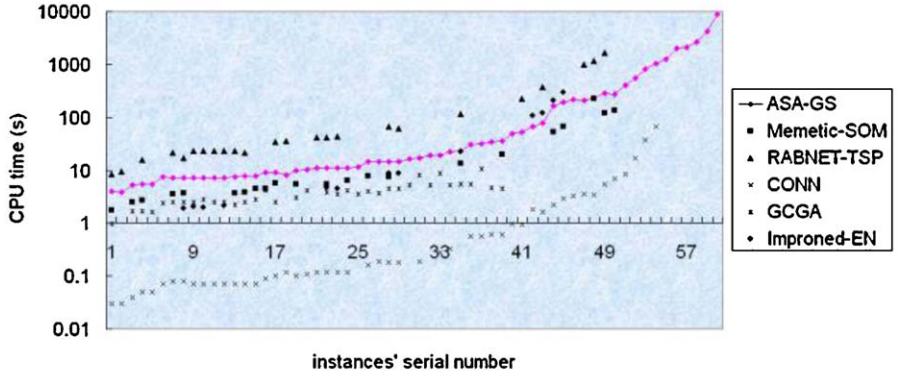
Then the procedure turns to the next step. The parameters $t_{current}$, N and OPT in formula (3) denote the temperature of the current state, the number of the cities and the optimal tour length. Apparently, the greedy search will vary from weak to strength when m varies from small to big, where $F(c^{best})$ can be seen in formula (4) and $m = t_{greedy}$.

$$F(c^{best}) = \min\left\{F(c^{new\_1}), F(c^{new\_2}), \ldots, F(c^{new\_m})\right\} \tag{4}$$

To solve TSP, the value assignments of the parameters such as cool coefficient of the temperature t_{cool}, the times of greedy search t_{greedy}, the times t_v of compulsive accept, and the probability ρ of accept a new solution, are difficult when different scale TSP instances are considered. For example, a quick cool coefficient is fit for small-scale TSP instances and a hot initial temperature is fit for the TSP instances with a big optimal tour length. t_v is useful in avoiding local minima and ultimately finding the desired solution, and the ρ is adjusted according to the optimal tour length in order that our algorithm is adaptive for cities scale from 51 to 85900. In order to obtain a better solution in solving the TSP with broad scale, the value assignments of the parameters such as t_{cool}, t_{greedy}, t_v and ρ are dynamic, i.e. the values of the parameters will change along with the amount of the cities and the optimal tour lengths.



(a) Comparing our algorithm with the recent nine algorithms with percent difference for above instances



(b) Comparing our algorithm with the recent five algorithms in [6], [23], [29], [36] and [37] with CPU time for above instances

Fig. 5. Summary of the compares between our algorithm and the above algorithms with percent difference and CPU time.

Table 4
Results of our algorithm (ASA-GS) for 60 TSP benchmark instances from TSPLIB.

No.	Instances	Scale	*OPT*	Best	Average	Worst	Error (%)	CPU time (s)
1	*eil_51*	51	426	428.872	428.872	428.872	0.67	3.91
2	*berlin_52*	52	7542	7544.37	7544.37	7544.37	0.03	3.83
3	*st_70*	70	675	677.11	677.11	677.11	0.31	5.15
4	*eil_76*	76	538	544.369	544.369	544.369	1.18	5.50
5	*pr_76*	76	108159	***108159***	***108159***	***108159***	0.00	5.49
6	*rat_99*	99	1211	1219.24	1219.49	1219.86	0.70	7.34
7	*rd_100*	100	7910	7910.4	7910.4	7910.4	0.00	7.13
8	*kroA_100*	100	21282	21285.4	21285.4	21285.4	0.01	7.14
9	*kroB_100*	100	22141	***22139.1***	***22139.1***	***22139.1***	–	7.14
10	*kroC_100*	100	20749	20750.8	20750.8	20750.8	0.00	7.14
11	*kroD_100*	100	21294	21294.3	21301.0	21311.2	0.03	7.13
12	*kroE_100*	100	22068	22106.3	22112.3	22123.7	0.20	7.14
13	*eil_101*	101	629	640.212	640.515	640.926	1.83	7.42
14	*lin_105*	105	14379	14383	14383	14383	0.02	7.68
15	*pr_107*	107	44303	***44301.7***	***44301.7***	***44301.7***	–	7.78
16	*pr_124*	124	59030	59030.7	59030.7	59030.7	0.00	9.01
17	*bier_127*	127	118282	118294	118349	118418	0.05	9.20
18	*ch_130*	130	6110	6110.72	6121.15	6158.57	0.18	8.00
19	*pr_136*	136	96772	96966.3	97078.9	97186.0	0.31	9.86
20	*pr_144*	144	58537	***58535.2***	58545.6	58587.1	0.01	10.28
21	*ch_150*	150	6528	6530.9	6539.8	6553.1	0.16	10.91
22	*kroA_150*	150	26524	26524.9	26538.6	26564.2	0.05	10.90
23	*kroB_150*	150	26130	26140.7	26178.1	26223.2	0.18	10.90
24	*pr_152*	152	73682	73683.6	73694.7	73739.1	0.01	10.85
25	*u_159*	159	42080	42392.9	42398.9	42408	0.75	11.49
26	*rat_195*	195	2323	2345.22	2348.05	2352.38	1.07	14.37
27	*d_198*	198	15780	15830.6	15845.4	15877.8	0.41	14.60
28	*kroA_200*	200	29368	29411.5	29438.4	29461.6	0.23	14.26
29	*kroB_200*	200	29437	29504.2	29513.1	29520.8	0.25	14.24
30	*ts_225*	225	126643	126646	126646	126646	0.00	16.05
31	*pr_226*	226	80369	80542.1	80687.4	80808.3	0.39	16.70
32	*gil_262*	262	2378	2393.64	2398.61	2404.96	0.86	19.43
33	*pr_264*	264	49135	***49135***	49138.9	49142.4	0.00	19.09
34	*pr_299*	299	48191	48269.2	48326.4	48351.4	0.28	21.94
35	*lin_318*	318	42029	42306.7	42383.7	42563.8	0.84	23.35
36	*rd_400*	400	15281	15350.7	15429.8	15498.0	0.97	30.40
37	*fl_417*	417	11861	11940.4	12043.8	12121.0	1.54	32.02
38	*pr_439*	439	107217	110020	110226	110474	2.80	34.92
39	*pcb_442*	442	50778	51063.9	51269.2	51365.0	0.96	35.75
40	*u_574*	574	36905	37232.3	37369.8	37465.3	1.25	48.47
41	*rat_575*	575	6773	6872.11	6904.82	6923.47	1.94	52.10
42	*u_724*	724	41910	42274.7	42470.4	42803.0	1.33	66.83
43	*rat_783*	783	8806	8954.36	8982.19	9013.86	2.00	78.90
44	*pr_1002*	1002	259045	263512	264274	264922	2.01	164.42
45	*pcb_1173*	1173	56892	57760.6	57820.5	57932.2	1.63	193.08
46	*d_1291*	1291	50801	51751.2	52252.3	52798.9	2.85	214.64
47	*rl_1323*	1323	270199	271964	273444	275315	1.20	210.16
48	*fl_1400*	1400	20127	20647.4	20782.2	20956.8	3.25	232.02
49	*d_1655*	1655	62128	63635.9	64155.9	64509.9	3.26	281.88
50	*vm_1748*	1748	336556	342437	343911	344812	2.18	276.98
51	*u_2319*	2319	234256	236356	236744	237077	1.06	410.97
52	*pcb_3038*	3038	137694	140742	141242	142062	2.57	554.28
53	*fnl_4461*	4461	182566	186956	187409	187928	2.65	830.90
54	*rl_5934*	5934	556045	572380	575437	580017	3.48	1043.95
55	*pla_7397*	7397	23260728	24101920	24166453	24195636	3.89	1245.22
56	*usa_13509*	13509	19982859	20799629	20811106	20827805	4.14	2016.05
57	*brd_14051*	14051	469385	485622	486762	486197	3.58	2080.50
58	*d_18512*	18512	645238	668104	669445	670812	3.75	2593.97
59	*pla_33810*	33810	66048945	69391254	69533166	69693399	5.27	4199.88
60	*pla_85900*	85900	142382641	155290611	156083025	156631417	9.62	8855.13
Total average for ASA-GS				1.33	435.92			

As a result, the above four parameters are formulized in formula (5). Where α and β are constants, and can be seen in Table 3.

$$\begin{cases} t_{cool} = \dfrac{(\alpha \times N^{0.5} - 1.0)}{(\alpha \times N^{0.5})} \\ t_{greedy} = \beta \times N \\ t_v = \dfrac{N}{10} \\ \rho = e^{-(F(c^{best}) - F(c^{old})/t_{current}) \times (10.0 \times N/OPT)} \end{cases} \quad (5)$$

The proposed adaptive simulated annealing algorithm with greedy search (ASA-GA) is shown as a flow diagram in Fig. 3. It involves the above three mutations, greedy search and adaptive parameter control strategies, which are used to improve the effectiveness of the standard simulated annealing algorithm.

4. Numerical results

In order to confirm the effectiveness of the ASA-GS for TSP, our algorithm was run on 60 benchmark instances with five trials, and

Table 5
Comparison ASA-GS with ACOMAC-DNN [34], MGSOM [1], Dis-PSO [32], and ELC-LR [38], respectively.

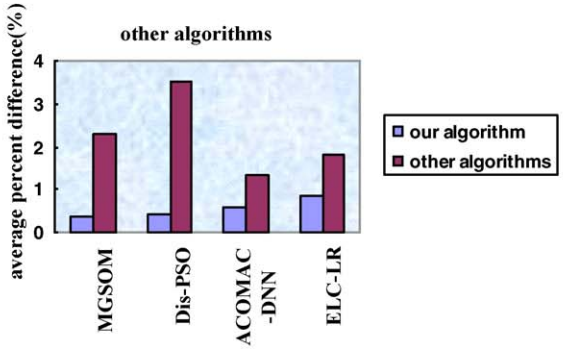
Comparison ASA-GS with ACOMAC-DNN [34], MGSOM [1], Dis-PSO [32], and ELC-LR [38], respectively

Correspond to ACOMAC-DNN		
No.	ASA-GS	ACOMAC-DNN
1	0.67	0.94
4	1.18	2.71
8	0.01	0.59
27	0.41	1.11
Av.	**0.56**	**1.34**

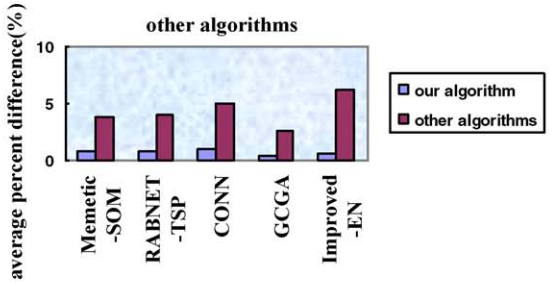
Correspond to MGSOM		
No.	ASA-GS	MGSOM
1	0.67	1.39
3	0.31	1.18
4	1.18	3.38
7	0.00	1.17
14	0.02	0.02
15	0.00	0.17
17	0.05	1.09
19	0.31	2.15
24	0.01	0.74
26	1.07	5.98
28	0.23	1.97
39	0.96	8.57
Av.	**0.40**	**2.32**

Correspond to Dis-PSO		
No.	ASA-GS	Dis-PSO
1	0.67	2.57
2	0.03	3.84
3	0.31	3.34
4	1.18	4.16
5	0.00	3.81
Av.	**0.44**	**3.54**

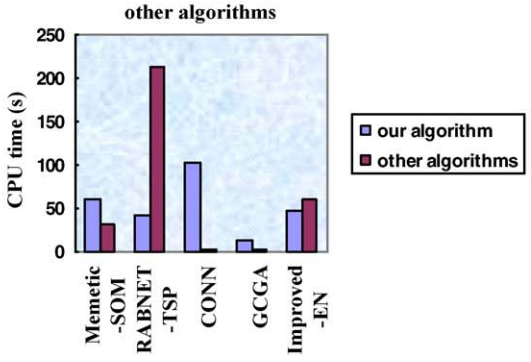
Correspond to ELC-LR		
No.	ASA-GS	ELC-LR
1	0.67	0.94
3	0.31	0.74
4	1.18	0.37
5	0.00	2.82
6	0.70	0.50
7	0.00	0.16
8	0.01	1.63
9	0.00	1.39
10	0.00	1.34
11	0.03	0.72
12	0.20	1.22
13	1.83	0.32
14	0.02	0.06
15	0.00	4.09
16	0.00	1.30
17	0.05	0.66
18	0.18	0.59
19	0.31	0.95
21	0.16	0.61
22	0.05	0.85
23	0.18	1.53
24	0.01	2.58
25	0.75	0.37
26	1.07	1.29
28	0.23	1.04
29	0.25	0.93
30	0.00	8.84
31	0.39	8.49
32	0.86	1.01
33	0.00	6.01
34	0.28	1.68
35	0.84	0.32
36	0.97	0.83
38	2.80	1.30
41	1.94	0.74
42	1.33	0.64
51	1.06	0.25
56	4.14	10.61
60	9.62	0.56
Av.	**0.83**	**1.80**



(a) Comparing our algorithm with algorithms in [1], [32], [34] and [38] with average percent difference for above instances



(b) Comparing our algorithm with algorithms in [6], [23], [29], [36] and [37] with average percent difference for above instances



(c) Comparing our algorithm with algorithms in [6], [23], [29], [36] and [37] with average CPU time for above instances

Fig. 6. Summary of the compares between ASA-GS and the recent algorithms with percent difference and CPU time.

was carried out in C++ and run on a 2.83 GHz PC with 2GB of RAM running the Windows XP operating system (Table 2). To compare the ASA-GS with the recent algorithms [1,6,23,29,32,34,36–38] in term of CPU time, we scale the CPU time of each algorithm by an appropriate scaling coefficient related to their processing systems, the CPU time and their scaling coefficients are shown in Table 2.

4.1. Related work

The various approximate approaches have been applied in a lot of papers over twenty years, though there were not too many benchmark results for TSP ten years ago. As a result, nine recent algorithms that can be seen in Table 2 are cited and compared with the ASA-GS in this paper. The nine algorithms involve an ant colony technique (ACOMAC-DNN), a self-organizing maps method (MGSOM), a discrete particle swarm optimization algorithm (Dis-PSO), a Lagrangean relaxation approach (ELC-LR), a self-organizing neural network (RABNET-TSP), a memetic neural network (Memetic-SOM), constructive-optimizer neural network technique (CONN), a genetic algorithm (GCGA) and an improved elastic net method (Improved-EN).

4.2. Parameters

In this subsection, all parameters used in the ASA-GS are described in Table 3. The parameters are obtained by testing the ASA-GS on 60 TSP benchmark instances. As shown in Table 3 and formula (5), N denotes the amount of the cities, OPT denotes the optimal tour length, the symbol '-' denotes the parameter is a constant. The parameters $t_{initial}$, t_{cool}, t_{greedy} and t_{end} denote the initial temperature, cool coefficient of the temperature, the times of greedy search and end temperature, respectively. The parameters $t_{current}$ and t_v denote the temperature of the current state and the times of compulsive accept, respectively. As a rule, the hot $t_{initial}$, low t_{end}, and slow t_{cool} are necessary for finding a good solution at the cost of a lot of computation time. Hence, in order to speed

Table 6
Comparison ASA-GS with RABNET-TSP [23], Memetic-SOM [6], and CONN [29], respectively.

Comparison ASA-GS with RABNET-TSP [23], Memetic-SOM [6], and CONN [29], respectively

Correspond to RABNET-TSP				
No.	ASA-GS		RABNET-TSP	
	Error (%)	CPU time (s)	Error (%)	CPU time (s)
1	0.67	3.91	2.69	8.42
2	0.03	3.83	5.18	9.44
4	1.18	5.50	3.41	15.97
7	0.00	7.13	3.66	21.07
8	0.01	7.14	1.13	16.82
9	0.00	7.14	2.35	23.38
10	0.00	7.14	1.07	22.94
11	0.03	7.13	1.89	23.23
12	0.20	7.14	2.93	22.77
13	1.83	7.42	3.12	23.02
14	0.02	7.68	0.15	21.85
17	0.05	9.20	2.20	33.94
18	0.18	8.00	2.82	35.30
21	0.16	10.91	3.22	42.29
22	0.05	10.90	3.14	42.28
23	0.18	10.90	1.92	43.04
28	0.23	14.26	2.80	65.59
29	0.25	14.24	2.37	61.38
35	0.84	23.35	3.97	116.89
41	1.94	52.10	5.06	228.19
43	2.00	78.90	6.11	373.15
47	1.20	210.16	13.00	1003.10
48	3.25	232.02	4.88	1172.15
49	3.26	281.88	16.07	1663.41
Av.	**0.73**	**42.83**	**3.96**	**212.07**

Correspond to Memetic-SOM				
No.	ASA-GS		Memetic-SOM	
	Error (%)	CPU time (s)	Error (%)	CPU time (s)
1	0.67	3.91	2.14	1.78
3	0.31	5.15	0.99	2.52
4	1.18	5.50	2.88	2.74
7	0.00	7.13	2.65	3.58
8	0.01	7.14	1.14	3.74
13	1.83	7.42	3.15	3.63
14	0.02	7.68	0.34	3.88
15	0.00	7.78	0.67	4.41
16	0.00	9.01	1.52	4.50
17	0.05	9.20	2.78	5.70
19	0.31	9.86	3.10	5.40
22	0.05	10.90	2.73	5.41
24	0.01	10.85	2.60	6.34
26	1.07	14.37	6.59	7.66
28	0.23	14.26	2.20	7.55
35	0.84	23.35	5.51	13.23
39	0.96	35.75	6.08	19.51
44	2.01	164.42	6.11	53.72
45	1.63	193.08	8.66	66.29
48	3.25	232.02	4.86	222.35
49	3.26	281.88	10.62	118.13
50	2.18	276.98	7.74	137.17
Av.	**0.90**	**60.80**	**3.86**	**31.78**

Correspond to CONN				
No.	ASA-GS		CONN	
	Error (%)	CPU time (s)	Error (%)	CPU time (s)
1	0.67	3.91	2.58	0.03
2	0.03	3.83	8.18	0.03
3	0.31	5.15	2.96	0.04
4	1.18	5.50	5.02	0.05
5	0.00	5.49	4.34	0.05
6	0.70	7.34	0.33	0.07
7	0.00	7.13	3.59	0.08
8	0.01	7.14	2.57	0.08
9	0.00	7.14	2.60	0.07
10	0.00	7.14	1.53	0.07
11	0.03	7.13	1.42	0.07
12	0.20	7.14	1.97	0.07
13	1.83	7.42	5.09	0.07
14	0.02	7.68	0.38	0.07
15	0.00	7.78	2.77	0.07
16	0.00	9.01	1.74	0.09
17	0.05	9.20	2.45	0.10
18	0.18	8.00	4.88	0.12
19	0.31	9.86	2.79	0.10
20	0.01	10.28	2.34	0.11
21	0.16	10.91	5.50	0.12
22	0.05	10.90	5.17	0.12
23	0.18	10.90	3.09	0.12
24	0.01	10.85	0.79	0.12
26	1.07	14.37	5.64	0.16
27	0.41	14.60	10.1	0.19
28	0.23	14.26	5.16	0.18
29	0.25	14.24	4.50	0.18
31	0.39	16.70	2.35	0.19
33	0.00	19.09	3.58	0.28
34	0.28	21.94	4.85	0.32
36	0.97	30.40	5.77	0.57
37	1.54	32.02	4.54	0.57
38	2.80	34.92	6.24	0.60
39	0.96	35.75	5.72	0.61
40	1.25	48.47	6.11	0.96
41	1.94	52.10	7.84	0.95
42	1.33	66.83	7.61	1.82
43	2.00	78.90	7.59	1.60
44	2.01	164.42	6.94	2.18
45	1.63	193.08	8.90	2.96
46	2.85	214.64	10.80	3.30
47	1.20	210.16	9.60	3.55
48	3.25	232.02	4.38	3.40
49	3.26	281.88	8.28	5.50
50	2.18	276.98	9.23	7.00
51	1.06	410.97	3.08	8.35
52	2.57	554.28	8.04	17.20
53	2.65	830.90	7.94	36.65
54	3.48	1043.95	13.1	67.35
Av.	**0.94**	**101.89**	**5.03**	**3.37**

up the convergence rate of the ASA-GS, greedy search technique is considered and designed in this paper.

In this paper, our algorithm is carried out with a two-stage adaptive local search strategy based on the four parameters $t_{initial}$, t_{cool}, t_{greedy} and t_{end}, and the two-stage adaptive local search strategy is described in Fig. 4.

4.3. The ASA-GS performance on TSPLIB instances

In this subsection, the ASA-GS is performed on 60 TSP benchmark instances from TSPLIB with cities scale from 51 to 85900. The ASA-GS is executed on each TSP instances with five trials, and the results are listed in Table 4.

In Table 4, the columns 'Instances', 'Scale', '*OPT*', 'Best', 'Average', 'Worst', 'Error', and 'CPU time' denote the name of the TSP instance, the number of cities, the optimal tour length as reported in TSPLIB, the shortest closed tour among the five trials, the average closed tour among the five trials, the longest closed tour among the five trials, the percent difference of the average closed tour, and the average CPU time for each trials, respectively. As can be seen in Table 4, for the 60 TSP instances with our algorithm ASA-GS, the percent difference of the average closed tour and the average CPU time for all trials are 1.33 and 435.92 s, respectively. In addition, for TSP instances *pr*_76 and *pr*_264, our algorithm ASA-GS always can find the best known solutions 108159 and 49135. Particularly, for TSP instances *kroB*_100, *pr*_107 and *pr*_144, the best known solutions 22141, 44303 and 58537 are replaced by the new best known solutions 22139.1, 44301.7 and 58535.2. In Table 4, the symbol '—' denotes that the percent difference is negative, i.e. the length of the closed tour obtained by the proposed algorithm is shorter than the length of the best known closed tour.

4.4. Comparing with four new algorithms

In this subsection, we compared the ASA-GS with four recent algorithms such as ACOMAC-DNN [34], MGSOM [1], Dis-PSO [32], and ELC-LR [38]. The comparisons between our algorithm ASA-GS and the above four algorithms are just based on the percent difference of the CPU time. It can be seen from Table 5, corresponding to the four TSP instances, the average percent difference of the ACOMAC-DNN is 1.34 while our algorithm ASA-GS provides lower average percent difference 0.56. Corresponding to another 12 TSP benchmark instances, the average percent difference of the MGSOM is 2.32 while our algorithm ASA-GS provides a lower average percent difference 0.40. For Dis-PSO with another five TSP instances, the average percent difference of the Dis-PSO is 3.54 while our algorithm ASA-GS is 0.44.

Corresponding to another 39 TSP instances, the average percent difference of the ELC-LR is 1.80 while our algorithm ASA-GS provides lower average percent difference 0.83. In a word, the solutions obtained by our algorithm ASA-GS have lower average percent difference or better accuracy solutions than those obtained by the above four recent algorithms. In addition, in Table 5, the symbol 'Av.' denotes the average value of the percent difference for its corresponding column.

4.5. Comparing with neuronal network algorithms

In this subsection, we compared the ASA-GS with three recent competitive neural network algorithms such as RABNET-TSP [23], Memetic-SOM [6] and CONN [29]. It can be seen in Table 6, comparing with the RABNET-TSP for 24 TSP instances, the RABNET-TSP takes longer CPU time (212.07 s) than the CPU time (42.83 s) with our algorithm ASA-GS. At the same time, toward each of the 24 TSP instances, our algorithm ASA-GS can find better solutions (with an average percent difference 0.73) than the solutions obtained by the RABNET-TSP (with an average percent difference 3.96). As shown in Table 6, comparing with the Memetic-SOM, for another 22 TSP instances, the Memetic-SOM takes shorter CPU time (31.78 s) than the CPU time (60.80 s) with our algorithm ASA-GS. However, for each of the 22 TSP instances, our algorithm ASA-GS can find better solutions (with average percent difference 0.90) than the solutions obtained by the Memetic-SOM (with average percent difference 3.86). According to Table 6, comparing with the CONN, for another 50 TSP instances, the CONN takes shorter CPU time (3.37 s) than the CPU time (101.89 s) with our algorithm ASA-GS. However, for each of the 50 traveling salesman problems, our algorithm ASA-GS can find better solutions (with average percent difference 0.94) than the solutions obtained by the algorithm CONN (with average percent difference 5.03).

Table 7
Comparison ASA-GS with GCGA [36] and Improved-EN [37], respectively.

Correspond to GCGA				
No.	ASA-GS		GCGA	
	Error (%)	CPU time	Error (%)	CPU time
1	0.67	3.91	0.94	0.97
3	0.31	5.15	0.44	1.67
4	1.18	5.50	2.42	1.66
5	0.00	5.49	0.72	1.65
6	0.70	7.34	2.23	2.36
7	0.00	7.13	1.53	2.52
8	0.01	7.14	1.23	2.54
9	0.00	7.14	1.81	2.52
10	0.00	7.14	1.33	2.85
11	0.03	7.13	2.42	2.50
12	0.20	7.14	1.41	2.50
13	1.83	7.42	2.70	2.19
14	0.02	7.68	1.15	2.52
15	0.00	7.78	1.37	2.81
16	0.00	9.01	0.19	4.18
17	0.05	9.20	1.80	2.49
19	0.31	9.86	2.82	2.98
20	0.01	10.28	0.04	4.21
22	0.05	10.90	2.92	3.79
23	0.18	10.90	2.11	3.54
24	0.01	10.85	1.22	4.14
25	0.75	11.49	2.28	3.59
26	1.07	14.37	4.18	3.94
27	0.41	14.60	1.93	3.74
28	0.23	14.26	1.85	4.43
29	0.25	14.24	4.04	4.50
30	0.00	16.05	1.08	5.23
31	0.39	16.70	0.75	7.99
32	0.86	19.43	5.76	5.34
33	0.00	19.09	2.46	8.59
34	0.28	21.94	5.44	5.31
35	0.84	23.35	5.14	5.42
36	0.97	30.40	7.45	5.54
37	1.54	32.02	3.22	10.40
38	2.80	34.92	6.13	4.59
39	0.96	35.75	8.99	4.42
Av.	0.46	13.40	2.59	3.88

Correspond to Improved-EN				
No.	ASA-GS		Improved-EN	
	Error (%)	CPU time	Error (%)	CPU time
8	0.01	7.14	2.09	1.88
9	0.00	7.14	3.02	2.02
10	0.00	7.14	2.33	1.94
12	0.20	7.14	4.22	2.16
22	0.05	10.90	5.56	4.64
23	0.18	10.90	3.80	4.49
28	0.23	14.26	5.53	8.81
29	0.25	14.24	5.31	8.75
35	0.84	23.35	8.02	22.47
42	1.33	66.83	9.59	107.87
43	2.00	78.90	9.83	121.72
44	2.01	164.42	8.70	209.58
45	1.63	193.08	12.73	290.04
Av.	0.67	46.57	6.21	60.49

4.6. Comparing with other algorithms

In this subsection, we compared the ASA-GS with two recent competitive algorithms such as GCGA [36] and Improved-EN [37]. According to Table 7 comparing with the GCGA for 36 TSP instances, the GCGA takes shorter CPU time (3.88 s) than the CPU time (13.40 s) with our algorithm ASA-GS. However, toward

each of the 36 TSP instances, our algorithm ASA-GS can find better solutions (with average percent difference 0.46) than the solutions obtained by the GCGA (with average percent difference 2.59). It can be observed from in Table 7, comparing with the Improved-EN for another 13 TSP instances, the Memetic-SOM takes longer CPU time (60.49 s) than the CPU time (46.57 s) with our algorithm ASA-GS. At the same time, to each of the 13 TSP instances, our algorithm ASA-GS can find better solutions (with average percent difference 0.67) than the solutions obtained by the Improved-EN (with average percent difference 6.21).

4.7. Summary of the compares

In this subsection, we sum up the compares between our algorithm and the nine recent algorithms with percent difference and CPU time. It can be seen from Fig. 5(a), for most of TSP instances, our algorithm can find better solutions than the solutions that obtained by the recent nine algorithms. However, as shown in Fig. 5(b), for the three algorithms in [6], [29] and [36], the convergence rate of our algorithm is slower than that of the three algorithms. Particularly, comparing with the algorithm in [29], our algorithm's convergence rate needs to be improved greatly.

As shown in Fig. 6(a) and (b), our algorithm can find better average solutions than the average solutions that obtained by the recent nine algorithms. However, as illustrated in Fig. 6(c), for the three algorithms in [6], [29] and [36], the average convergence rate of our algorithm is slower than the solutions obtained by the three algorithms. Particularly, comparing our ASA-GS with the CONN [29], convergence rate of our algorithm is fairly poor.

5. Conclusion

In this paper, an adaptive simulated annealing algorithm with greedy search (ASA-GS) is proposed to solve the TSP. The ASA-GS is based on the standard simulated annealing algorithm, and utilizes greedy search technique to speed up the convergence rate. And then, aim at the parameters such as cool coefficient of the temperature, the times of greedy search, and the times of compulsive accept and the probability of accept a new solution, adaptive parameters control of the ASA-GS is considered and designed to improve its quality-time trade-off. As a result, the ASA-GS with greedy search achieves a reasonable trade-off among computation time, solution quality, and complexity of implementation. Simulation results indicate that the ASA-GS performs well on a set of 60 TSP instances, and verify its validity. However, the ASA-GS takes longer CPU time than the three algorithms [6,29,36] for the TSP instances. So, it may be a subject of future work. The ASA-GS will be improved in order to obtain better solutions with shorter CPU time.

Acknowledgements

The authors thank the anonymous reviewers and Professor Rajkumar Roy for their constructive comments and suggestions. The research was supported in part by the National Natural Science Foundation of China (grant nos. 60803113, 60373089, 60674106, and 60533010), and the National High Technology Research and Development Program of China (grant nos. 2009AA012413).

References

[1] Y. Bai, W. Zhang, Z. Jin, A new self-organizing maps strategy for solving the traveling salesman problem, Chaos, Solitons and Fractals 28 (2006) 1082–1089.
[2] D. Banaszak, G.A. Dale, A.N. Watkins, J.D. Jordan, An optical technique for detecting fatigue cracks in aerospace structures, in: Proc. 18th ICIASF, 1999, pp. 1–7.
[3] R. Baraglia, J.I. Hidalgo, R. Perego, A hybrid heuristics for the traveling salesman problem, IEEE Transactions on Evolutionary Computation 5 (6) (2001) 613–622.
[4] V. Cerny, A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications 45 (1985) 1334–1380.
[5] C.B. Cheng, C.P. Mao, A modified ant colony system for solving the traveling salesman problem with time windows, Mathematical and Computer Modeling 46 (2007) 1225–1235.
[6] J.C. Créput, A. Koukam, A memetic neural network for the Euclidean traveling salesman problem, Neurocomputing 72 (2009) 1250–1264.
[7] R. Durbin, R. Szeliski, A. Yuille, An analysis of the elastic net approach to the traveling salesman problem, Neural Computation 1 (1989) 348–358.
[8] R. Durbin, D. Willshaw, An analogue approach to the traveling salesman problem using an elastic net approach, Nature 326 (1987) 689–691.
[9] G. Finke, A. Claus, E. Gunn, A two-commodity network flow approach to the traveling salesman problem, Congressus Numerantium 41 (1984) 167–178.
[10] S. Ghafurian, N. Javadian, An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesman problems, Applied Soft Computing 11 (1) (2011) 1256–1262.
[11] F. Glover, Artificial intelligence, heuristics frameworks and tabu search, Managerial & Decision Economics 11 (1990) 365–378.
[12] A.J. Hoffman, P. Wolfe, History, in: Lawler, Lenstra, Rinooy Kan, Shmoys (Eds.), The Traveling Salesman Problem, Wiley, 1985, pp. 1–16.
[13] L. Jiao, L. Wang, A novel genetic algorithm based on immunity, IEEE Transactions on Systems, Man and Cybernetics—Part A: Systems and Humans 30 (5) (2000) 826–830.
[14] F. Jin, S. Song, C. Wu, A simulated annealing algorithm for single machine scheduling problems with family setups, Computers & Operations Research 36 (7) (2009) 2133–2138.
[15] D.S. Johnson, C.R. Aragon, L.A. Mcgeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation, Part I, Graph partitioning, Operations Research 37 (1989) 865–892.
[16] D.S. Johnson, C.R. Aragon, L.A. Mcgeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation, Part II, Graph coloring and number partitioning, Operations Research 39 (3) (1991) 378–406.
[17] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[18] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, Configuration space analysis of traveling salesman problem, Journal Physique 46 (1985) 1277–1292.
[19] C.C. Lo, C.C. Hus, Annealing framework with learning memory, IEEE Transactions on Systems, Man, and Cybernetics— Part A Systems and Humans 28 (5) (1998) 1–13.
[20] M López-Ibáñez, C. Blum, Beam-ACO for the traveling salesman problem with time windows, Computers and Operations Research 37 (2010) 1570–1583.
[21] Y. Marinakis, M. Marinaki, G. Dounias, A hybrid particle swarm optimization algorithm for the vehicle routing problem, Engineering Applications of Artificial Intelligence 23 (2010) 463–472.
[22] Y. Marinakis, M. Marinaki, A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem, Computers and Operations Research 37 (2010) 432–442.
[23] T.A.S. Masutti, L.N. de Castro, A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem, Information Sciences 179 (10) (2009) 1454–1468.
[24] K. Meer, Simulated annealing versus metropolis for a TSP instance, Information Processing Letters 104 (6) (2007) 216–219.
[25] M.K. Mehmet Ali, F. Kamoun, Neural networks for shortest tour computation and routing in computer networks, IEEE Transactions on Neural Network 4 (5) (1993) 941–953.
[26] G.C. Onwubolu, M. Clerc, Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization, International Journal of Production Research 42 (3) (2004) 473–491.
[27] C.H. Papadimitriou, Euclidean traveling salesman problem is NP-complete, Theoretical Computer Science 4 (1978) 237–244.
[28] T. Poranen, A simulated annealing algorithm for determining the thickness of a graph, Information Sciences 172 (1–2) (2005) 155–172.
[29] M. Saadatmand-Tarzjan, M. Khademi, M.R. Akbarzadeh-T, H.A. Mghaddam, A novel constructive-optimizer neural network for the traveling salesman problem, IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 37 (4) (2007) 754–770.
[30] K. Savla, E. Frazzoli, F. Bullo, Traveling salesperson problems for the Dubins vehicle, IEEE Transactions on Automatic Control 53 (6) (2008) 1378–1391.
[31] M. Serrurier, H. Prade, Improving inductive logic programming by using simulated annealing, Information Sciences 178 (6) (2008) 1423–1441.
[32] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu, Q.X. Wang, Particle swarm optimization-based algorithms for TSP and generalized TSP, Information Processing Letters 103 (2007) 169–176.

[33] F. Tian, L. Wang, Chaotic simulated annealing with augmented Lagrange for solving combinatorial optimization problems, in: Proc. 26th Annual IECON, vol. 4, 2000, pp. 2722–2725.
[34] C.F. Tsai, C.W. Tsai, C.C. Tseng, A new hybrid heuristic approach for solving large traveling salesman problem, Information Sciences 166 (2004) 67–81.
[35] Z.C. Wang, X.T. Geng, Z.H. Shao, An effective simulated annealing algorithm for solving the traveling salesman problems, Journal of Computational and Theoretical Nanoscience 6 (2009) 1680–1686.
[36] J. Yang, C. Wu, H.P. Lee, Y. Liang, Solving traveling salesman problem using generalized chromosome genetic algorithm, Progress in Natural Science 18 (2008) 887–892.
[37] J. Yi, G. Yang, Z. Zhang, Z. Tang, An improved elastic net method for traveling salesman problem, Neurocomputing 72 (2009) 1329–1335.
[38] R. Zamani, S.K. Lau, Embedding learning capability in Lagrangean relaxation: an application to the traveling salesman problem, European Journal of Operational Research 201 (1) (2010) 82–88.