A discrete cuckoo search algorithm for travelling salesman problem

Yongquan Zhou*, Xinxin Ouyang and Jian Xie

College of Information Science and Engineering,
Guangxi University for Nationalities,
Nanning, Guangxi 530006, China
E-mail: yongquanzhou@126.com
E-mail: 18934716006@189.cn
E-mail: xj2jc@163.com
*Corresponding author

Abstract: In this paper, a novel discrete cuckoo search algorithm to solve the travelling salesman problem (TSP) is proposed. The algorithm is based on the continuous cuckoo search with brood parasitic behaviour and Lévy flight. The algorithm does not save the initial routes in the bulletin board, until the segment between each city and adjacent cities partially are reversed. In each generation, an each cuckoo searches a new nest and abandons the old one to decrease the TSP route. In order to accelerate the discrete cuckoo search algorithm convergence speed, the proposed algorithm applies learning operator, 'A' operator and 3-opt operator to the bulletin board. The numerical experiment results show that the proposed algorithm can find the global optimal solution with rapid convergences and stable generation.

Keywords: discrete cuckoo search algorithm; metaheuristic; 3-opt; travelling salesman problem.

Reference to this paper should be made as follows: Zhou, Y., Ouyang, X. and Xie, J. (2014) 'A discrete cuckoo search algorithm for travelling salesman problem', *Int. J. Collaborative Intelligence*, Vol. 1, No. 1, pp.68–84.

Biographical notes: Yongquan Zhou received his MS in Computer Science from Lanzhou University, Lanzhou, China, in 1993 and PhD in Computation Intelligence from the Xiandian University, Xi'an, China, in 2006. He is currently a Professor in Guangxi University for Nationalities. His research interests include computation intelligence, neural networks, and intelligence information processing, etc. He has published one book, and more than 150 research papers in journals.

Xinxin Ouyang's research interests include computation intelligence and swarm intelligence algorithm.

Jian Xie's research interests include computation intelligence and swarm intelligence algorithm.

1 Introduction

Cuckoo search (CS) algorithm was proposed by Yang and Deb in 2009 (Yang and Deb, 2009; Yang, 2010). It was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds (of other species). CS gradually aroused scholars' close attention, and has been applied to solve other problems with great success. For example, the applications of CS into engineering optimisation problems have shown its promising efficiency (Yang and Deb, 2010). A promising discrete cuckoo search (DCS) algorithm is recently proposed to solve nurse scheduling problem (Tein and Ramli, 2010). An efficient computation approach based on CS has been proposed for data fusion in wireless sensor networks (Dhivya and Sundarambal, 2011). A new quantum-inspired CS was proposed to solve knapsack problems (Layeb, 2012). A multi-objective CS method has been formulated to deal with multi-criteria optimisation problems (Yang and Deb, 2011). A modification of the standard cuckoo search was made by Walton et al. with the aim to speed up convergence (Walton et al., 2011; Zheng and Zhou, 2012). More recent studies suggest that CS can outperform other algorithms in milling applications (Yildiz, 2012), manufacturing scheduling (Burnwal and Deb, 2012), and boundary value problems (Noghrehabadi et al., 2011).

Metaheuristics are used for combinatorial optimisation in which an optimal solution is sought over a discrete search-space. A classical example is the travelling salesman problem (TSP), which is also a classical NP-hard problem in the field of combinatorial optimisation (Chen and Wang, 2007). TSP can be describe as follows: given a complete undirected graph $G = (V, A)$, V being the vertex set and A being the arc set, with nonnegative costs associated with its arcs, find a minimum cost circuit in G passing through each vertex exactly once (Yildiz, 2012). The TSP can easily be solved by enumeration method when the number of cities is few, however, the TSP is very hard to solve under an acceptable computational time when the number of cities is very large.

Although several exact algorithms (such as dynamic programming, branch-and bound and linear programming) have shown its promising efficiency for solving TSP, the metaheuristics or heuristics shown its potential promising performance as well. There are many metaheuristics or heuristics proposed to solve the TSP, for example, ant colony optimisation (ACO) (Ji et al., 2010), glowworm swarm optimisation (GSO) (Zhou and Huang, 2012)] and others (Yan et al., 2007; Ouaarab et al., 2013; Jati et al., 2021). These techniques are known to be efficient in the search of a space solution providing optimal values. In order to extend the application of CS, this article proposed a DCS algorithm to solve the TSP.

The rest of this article is organised as follows. The CS and related notion are described in Sections 2. The DCS algorithm for TSP is described detailed in Section 3. The experimental results of the DCS and comparisons with other previous algorithms are shown in Section 4. In the last section, we conclude this paper and point out some future work in Section 5.

2 Basic operator and algorithm

2.1 Basic operator

- *Inver-over operator (*IO*)*: The IO is presented by Tao and Michalewicz in 1999, this operator is used to solve TSP as well (Guo and Michalewicz, 1998). The IO includes two operator, inverse and crossover, and the IO is illustrated as follows:
    - *Inversion operator*: Firstly, two cities C and C' are selected randomly in individual S, and then these cities between the successor of C and C' (include C') are inversed. For example, individual $S = (1, 2, 3, 4, 5, 6, 7, 8)$, $C = 2$ and $C' = 6$, after inverse operator, a new individual is (1, 2, 6, 5, 4, 3, 7, 8).
    - *Crossover operator:* Firstly, a city C is selected randomly in individual S, and the successor of C in another individual S' is recorded as C'. Secondly, these cities between the successor of C and C' (include C') are inversed in individual S. If the city C' and city C are neighbouring, the inversion operator is cancelled. For example, individual $S = (1, 2, 3, 4, 5, 6, 7, 8)$, $C = 2$, another individual $S' = (1, 3, 5, 6, 8, 4, 2, 7)$, according to this operator, $C' = 7$, the substring 3–7 in individual S is inversed, the new individual is (1, 2, 7, 6, 5, 4, 3, 8).

  IO combines features of inversion (or mutation) and crossover. Experimental results reported in Layeb (2012) indicate clearly that the IO is significantly better than random inversion. This kind of IO can easily construct the access path that does not exist in the intersectant edges, and this operator makes the convergence rate of algorithm is fast in the early iterations.

2.2 Cuckoo search algorithm

2.2.1 Brood parasitism of cuckoo species

CS was inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds (of other species). Some host birds can engage in direct conflict with the intruding cuckoos. For example, if a host bird discovers the eggs are not their own, it will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. Some cuckoo species such as the new world brood-parasitic Tapera have evolved in such a way that female parasitic cuckoos are often very specialised in the mimicry in colours and pattern of the eggs of a few chosen host species.

2.2.2 CS algorithm

According to the cuckoo's behaviour, Yang and Deb proposed three idealised rules (Yang and Deb, 2009; Yang, 2010):

1 each cuckoo lays one egg at a time, and dump its egg in randomly chosen nest

2 the best nests with high quality of eggs (optimal fitness) will carry over to the next generations

3 the number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $p_\alpha \in (0, 1)$.

Individual: the position of a nest. It also can be viewed as the position of egg.

Population: the position of whole nests in a search space.

Based on the above rules, basic CS can be described as follows:

Step 1 Initialise objective function or fitness function

Step 2 Generate initial population: n host nests $x_i(i = 1, 2, \cdots, n)$

Step 3 While termination criterion is met, go to Step 9

Step 4 Get a cuckoo i randomly by Lévy flight and evaluate its quality (fitness) F_i

Step 5 Choose a nest j among n, if $F_i > F_j$, replace j by the new solution

Step 6 A fraction (p_α) of worse nests are abandoned and new ones are built; the new solutions are evaluated, and the best solutions (or nests with quality solutions) are kept

Step 7 Rank the solutions and find the current best

Step 8 The iteration variable is updated, and go to Step 3

Step 9 Output experimental results.

The key of basic CS is to get a new nest (or laid a new egg) through Lévy flight. It makes the CS is simple and efficient to employ Lévy flight.

2.3 Lévy flight

Lévy flight is presented by Paul Pierre Lévy. Lévy flight is a random walk in which the step-lengths have a probability distribution that is heavy-tailed (Yang, 2010). Step-lengths $L(s)$ are drawn from the Lévy distribution, often in terms of a simple power-law formula

$$L(s) \sim |s|^{-\lambda}, 1 < \lambda \leq 3,$$

where $1 < \lambda \leq 3$ is an index.

In CS, a new position $x^{(t+1)}$ is updated by

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus L(\mathrm{s}),$$

where $\alpha > 0$ is the step size which should be related to the scales of the problem of interests. In most cases, we can use $\alpha = 1$, $\oplus$ means entry-wise multiplications, $L(s)$ drawn from a Lévy distribution. The step length $L(s)$ can be calculated by

$$L(\mathrm{s}) = \frac{u}{|v|^{\frac{1}{\beta}}},$$

where u and v are drawn from normal distributions, that is $u \sim N(0, \sigma_u^2),\ \ v \sim N(0, \sigma_v^2)$,

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma[(1+\beta)/2]\beta 2^{(\beta-1)/2}} \right\}^{1/\beta}, \sigma_v = 1,$$

here $\Gamma(z)$ is Gaussian function

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt.$$

In the case when $z = n$ is an integer, we have $\Gamma(z) = (n - 1)!$.

3 DCS algorithm for TSP

Many applications of CS in different optimisation problems have shown its promising efficiency. For example, for both spring design and welded beam design problems, CS obtained better solutions than existing solutions in literature (Yang and Deb, 2010). However, many combinational optimisation problems are discrete problems. In order to apply CS into TSP, we propose a DCS algorithm based on basic CS and several good strategies.

3.1 Definitions

Assume the number of the current city is C, the distance from city C to other cities is $A = [d_{C,1}, d_{C,2}, \ldots, d_{C,m}]$, where $d_{C,C} = +\infty$, m is the number of cities. And $p = u/|v|^{-\lambda}$, $1 < \lambda \leq 3$. In different instance, the value of λ is different, $p \in [0, +\infty)$.

1 while $p < 1$, if $d_{C,D} = \min(A)$, the nearest city D is the next city to be visited

2 while $p \geq 1$, $dist = d_{C,D} \times p$, $B = \{j |\ d_{C,j} \leq dist,\ 1 \leq j \leq m\}$ and $d_{C,E} = \max(d_{C,j}), j \in B$

where $dist$ denotes the flight distance of a cuckoo. Set B represents a set of cities that the flight distance is effective, $d_{C,E}$ represents that city E is the farthest city from city C; city E is a candidate city among the candidate set B.

In the process of solving TSP, the probability that each city is visited from city C is p, $p = [p_{c,1}, p_{c,2}, \cdots, p_{c,m}]$. The next visiting city is either the nearest city D or the farthest city E.

3.2 Basic notion for solving TSP

Individual: A tour that a cuckoo visits all cities.

Population: All tours that n cuckoos visit all cities.

Flight: The move that cuckoo flight from a nest to another nest. The selective methods of next nest refer to the description in Subsection 3.1.

Population initialisation: Generate n nests (cities) randomly.

Fitness function: In this article, the length of a whole route is regarded as fitness.

Discard: The current nest is discarded according to the quality of individual, and new nest is generated by Lévy flight.

Bulletin board: Record the optimal n tours obtained by cuckoos.

Initial circuit construction: A cuckoo visits all cities from its nest, and returns to its nest. Each move use Flight operation. After the initial circuit (tour) is constructed, the initial circuit needs to improve by partial inversion operator. This partial inversion operator can be described as follows: assume current city is *C*, the next city is the nearest city *D*, if cities *C* and *D* are non-adjacent, then the segment between the farthest city *E* and the nearest city *D* (includes *D*) carry out inversion operator, after that, the next city of *C* perform same partial inversion operator. The tour generated by the last city performs the inversion operator is added to bulletin board, and the last city is regarded as new nest. Figure 1 shows the flow chart of initial circuit construction.

Figure 1 Flow chart of initial circuit construction (see online version for colours)

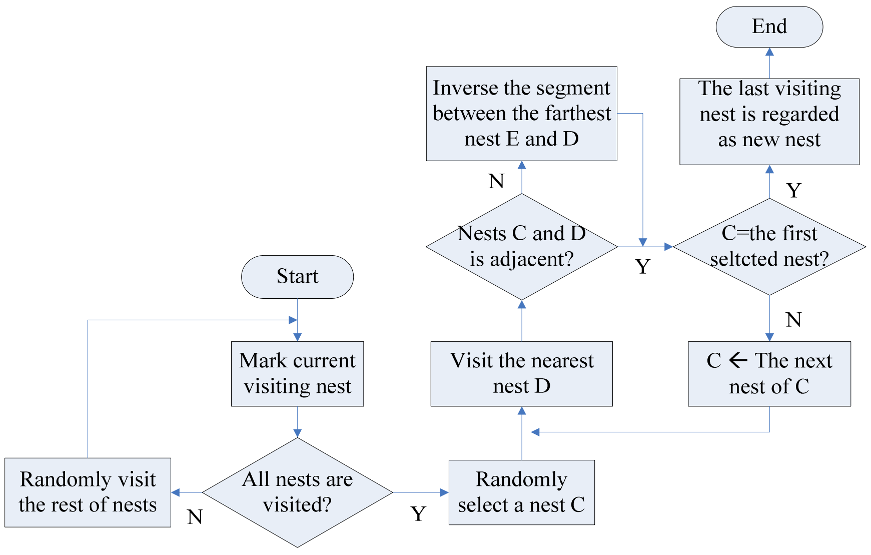
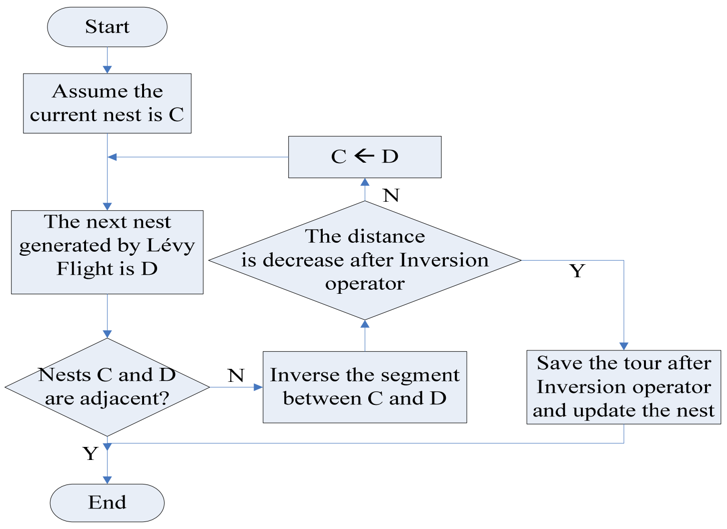

End
Inverse the segment between the farthest nest E and D
The last visiting nest is regarded as new nest
N
Y
Nests C and D is adjacent?
C=the first seltcted nest?
Y
Start
N
Mark current visiting nest
Visit the nearest nest D
C ← The next nest of C
Randomly visit the rest of nests
All nests are visited?
Randomly select a nest C
N
Y

Figure 2 Flow chart of new nest generation (see online version for colours)


Start
Assume the current nest is C
C ← D
N
The next nest generated by Lévy Flight is D
The distance is decrease after Inversion operator
Y
Nests C and D are adjacent?
N
Inverse the segment between C and D
Save the tour after Inversion operator and update the nest
Y
End

New individual generation: Assume the current cuckoo nest is *C*; the next visiting nest is *D* after Lévy flight. If the nests *C* and *D* are adjacent, then, a new individual is generated, otherwise, the individual needs to perform the inversion operator repeatedly until the total length of the tour decreases. Figure 2 shows the flow chart of new individual generation.

Learning operator: The learning operator is similar to the crossover of IO. The learning operator acts on two individuals in the bulletin board at the same time. Assume there are two individuals S_1 and S_2. If a substring in the S_2 is better than S_1, copy it to S_1 and change the bad of S_1. If this substring is worse, copy corresponding substring from S_1 to S_2 and mend the bad of S_2. They study well from each other and change bad.

For example, assume that

$$S_1 = (1,2,3,4,5,6,7,8),$$

$$S_2 = (1,3,5,6,8,4,2,7),$$

where S_1 and S_2 are choose randomly from the Bulletin Board. Suppose the current city $C = 2$, the operator search for the city S_2, which is next to 2, so the substring is '2, 7'. Thus the operator reverse the segment starts after city 2 and terminates after city 7 in the S_1. Consequently, a new individual is

$$S_1' = (1,2,7,6,5,4,3,8),$$

If the fitness of S_1' is less than S_1 $(F_{S_1'} < F_{S_1})$, S_1' would be saved. If not, it means that the substring '2, 7' +'3, 8' is worse than '2, 3' + '7, 8'. So the corresponding substring is '7, 8'. Therefore the segment for inversion in the S_2 starts after city '7' and terminates after city '8'. But '8' is in front of '7', the operator reverse the segment starts before '8' and ends before '7'. As a result, a new solution is produced

$$S_2' = (1,3,5,6,2,4,8,7).$$

If the fitness of S_2' is less than S_2, S_2' would be saved. The learning operator make $C = 1, 2, \cdots, m$ in turn, For each value, the inversion caused by 'study well' and 'change bad' has been applied several times in the process of executing.

'A' operator: To optimise the traversal path in the shape of an 'A' word, as shown in Figure 3. E.g., in path (1, 2, 3, 4, 5, 6, 7, 8), we chose A = 2, A visit next the city is B = 3, B the next city is C = 4. The cuckoo after a flight, a city close to point B, suppose city for D = 7, D the next point to as E = 8. If AB + BC + DE > AC + DB + BE, then the B deleted from the original position, add to between D and E. After operation the path for (1, 2, 4, 5, 6, 7, 3, 8).

There is also another kind of special situation, city A and E coincidence. Assuming the cuckoo produced after a flight near the B city, then E = 2, and city A coincidence. In Figure 4, if BC + AD > AC + BD, then delete BC and AD, and add AC and BD.

3-opt: 3-opt analysis involves deleting three edges in a tour, reconnecting the tour in all other possible ways, and then evaluating each reconnection method to find the optimum one. This process is then repeated for a different set of three edges. An instance is used to illustrate the 3-opt in Figure 4, three edges (a, b), (c, d) and (e, f) are deleted randomly in tour *S*, the remaining segment are S_1, S_2 and S_3. If there are two reconnections, and the left reconnection is the best tour among all reconnections, this tour will replace the original tour.

Figure 3 Schematic diagram of ‘A’ operator

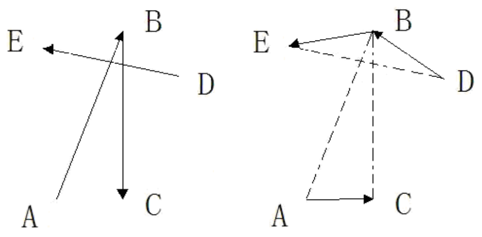


Figure 4 Schematic diagram of special ‘A’ operator

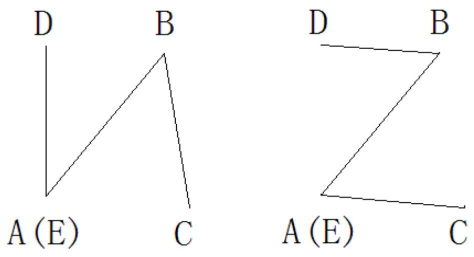
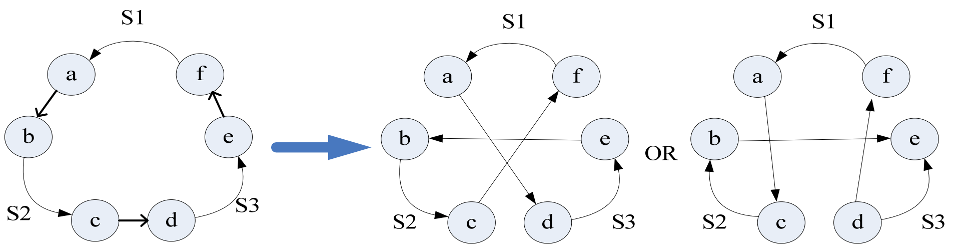


Figure 5 Schematic diagram of 3-opt (see online version for colours)



3.3 *DCS algorithm for TSP*

The complete step of DCS algorithm for TSP is described as follows:

Step 1 Initialise fitness and distance matrix.

Step 2 Generate initial population: *n* host nests $x_i (i = 1, 2, \cdots, n)$.

Step 3 Initial circuit is constructed by flight.

Step 4 The initial circuit is improved by partial inversion; the best tour is saved into bulletin board.

Step 5 When termination criterion is met, go to Step 14.

Step 6 Generate new individual by Lévy flight and evaluate its quality (fitness) F_i.

Step 7 Update the optimal tour on the bulletin board by learning operator.

Step 8 Update the optimal tour on the bulletin board by 'A' operator.

Step 9 Update the optimal tour on the bulletin board by 3-opt.

Step 10 If the new tour is better than original tour, the original tour is replaced by this new tour on bulletin board.

Step 11 The worse nest is discarded with probability $p_\alpha \in (0, 1)$, and then a new nest is generated by Lévy flight.

Step 12 Rank the tours on bulletin board and find out the current optimal tour.

Step 13 The iteration variable is updated, and go to Step 5.

Step 14 Output results.

4 Simulation experiment

Hardware environment: INSPIRON 530, 2.2GHZ, 1GB memory; Software environment: Windows XP SP3, MATLAB 2008a.

4.1 Parameter setting

Population size *ps* is set as 100, discard probability $p_\alpha = 0.5$, the parameter β in Lévy distribution is set as different value according to different instances. In general, $1 < \beta < 3$. The repeat not stop until the optimal tour on Bulletin Board is not updated.

4.2 Result analysis and comparison

In order to evaluate the performance of statistic experimental results, the evaluation criteria adopt deviation ratio (DR) and variance.

$$DR = \frac{fit - TSPLIB\_optimal}{TSPLIB\_optimal} \times 100\%;$$

$$Variance = \sum_{i=1}^{n} (fit_i - fit_{avg})^2 \Big/ fit_{avg};$$

where *fit* is the fitness, that is the length of best tour in DCS, *TSPLIB_optimal* is the optimal reference value in web TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/), fit_{avg} is the average fitness, and n is the number of independent experiment.

All instances take from TSPLIB, and each instance runs independently 20 times. Tables 1 and 2 show the experimental result, where the *m* is the number of cities, TSPLIB optimal is the optimal solutions in web TSPLIB, *β* is a parameter in Lévy flight, Best is our results, *W* is the number of iterations when the optimal is found, and average iteration is the average of iterations (when the algorithm terminates) in 20 times independent run.

As shown as Table 1, instances Att48 and Pr107 always can find out the optimal solution, for instances Pr76 and Pr124, DCS can find out the optimal solution with more than 90% success rates. Furthermore, DCS found out the optimal solution with more than 80% success rates for other instances in Table 1. The results which success rate is less than 80% are listed in Table 2.

From Table 2, for instances when the number of cities is less than 500, the DR is less than 2%, the average DR also less than 2% for most of instances, and the variance is small. For instances when the number of cities is more than 500, the DR is greater than 2%, but the variance is small as well. From Tables 1 and 2, we can find out the number of iterations is increasing while the number of cities, however, the increasing rate is less. In general, the convergence rate is fast, the quality of solutions is good, and the proposed DCS is stable for different TSP instances.

The DR of instances Pr144, U159 and Tsp225 is less than 0 in Table 2, the reason is that the solutions obtained by DCS is better than the optimal solutions summarised in TSPLIB. Figures 6 to 8 show the optimal tour and corresponding convergence curve. '·' denotes a city, '*' denotes the host nest.

Table 1 Some results obtained by DCS (success rates greater than 80%)

Instance	*m*	*TSPLIB optimal*	*DCS*			
			β	*Best*	*W*	*Average iteration*
Att48	48	33,523	2	33,523.7085	10	20.5
Eil51	51	426	2	428.8718	6	27.0
Berlin52	52	7,542	2	7,544.3659	10	15.9
St70	70	675	2	677.1096	10	26.4
Pr76	76	108,159	1.5	108,159.438	9	32.9
KroA100	100	21,282	1.5	21,285.4432	5	28.5
KroB100	100	22,141	1.5	22,139.0746	4	36.3
KroC100	100	20,749	2	20,750.7625	5	34.0
KroD100	100	21,294	2.5	21,294.2908	3	37.6
KroE100	100	22,068	2.5	22,068.7587	2	40.3
Rd100	100	7,910	1.5	7,910.3962	3	34.6
Lin105	105	14,379	2	14,382.9959	10	23.0
Pr107	107	44,303	2	44,301.6837	10	16.9
Pr124	124	59,030	2	59,030.7537	2	29.8

Table 2 Some results obtained by DCS (success rates less than 80%)

Instance	*m*	*TSPLIB optimal*	*DCS*						
			β	*Best*	*DR_Best (%)*	*Mean*	*DR_AVG (%)*	*Average iteration*	*Variance*
Eil76	76	538	1.5	544.3691	1.1838	547.8538	1.8316	25.9	0.0117
Rat99	99	1,211	1.5	1,219.2438	0.6761	1,224.6611	1.1281	40.8	0.0115
Bier127	127	118,282	2	118,500.4047	0.1846	118,835.9028	0.4683	40.5	0.2599
Ch130	130	6,110	2.5	6,126.9009	0.2766	6,164.6290	0.8941	38.8	0.0477
Pr136	136	96,772	1.5	96,800.8147	0.0298	97,632.9652	0.8897	49.4	2.1633
Chn144	144	30,353	2	30,386.3700	0.1099	30,497.6849	0.4767	43	0.3235
Pr144	144	58,537	1.5	58,535.2218	–0.0030	58,572.1665	0.0601	36.5	0.0250
KroA150	150	26,524	1.5	26,598.4800	0.2808	26,791.1208	1.0071	39.1	0.5382
KroB150	150	26,130	1.5	26,227.1311	0.3717	26,302.6741	0.6608	52.7	0.2526
Ch150	150	6,528	2	6,549.5533	0.3302	6,552.2649	0.3717	33.4	0.0002
Pr152	152	73,682	2.5	73,683.6406	0.0022	73,830.1391	0.2011	35.2	0.2205
U159	159	42,080	2	42,075.6700	–0.0103	42,177.8680	0.2326	29.2	0.2989
Rat195	195	2,323	2	2,345.3719	0.9631	2,357.3082	1.4769	34.7	0.0256
D198	198	15,780	2.5	15,847.9251	0.4305	15,909.5497	0.8210	49.3	0.0886
KroA200	200	29,368	2	29,489.37259	0.4133	29,533.1373	0.5623	41.5	0.0388
KroB200	200	29,437	1.2	29,687.2493	0.8501	29,854.6447	1.4188	54.5	0.2341
Ts225	225	126,643	2	126,760.6813	0.0929	126,968.9027	0.2573	31.3	0.1445

Table 2 Some results obtained by DCS (success rates less than 80%) (continued)

Instance	*m*	*TSPLIB optimal*	*DCS*						
			β	*Best*	*DR_Best (%)*	*Mean*	*DR_AVG (%)*	*Average iteration*	*Variance*
Tsp225	225	3,916	2	3,902.7394	–0.3386	3,933.7630	0.4536	48.7	0.0428
Pr226	226	80,369	1.5	80,440.0356	0.0884	80,735.1776	0.4556	35.7	0.6061
Pr264	264	49,135	2	49,144.6958	0.0197	49,265.2268	0.2650	35.8	0.7240
A280	280	2,579	2	2,596.4976	0.6785	2,614.8731	1.3910	33.5	0.0474
Pr299	299	48,191	1.5	48,907.4274	1.4866	49,178.9552	2.0501	54.3	0.6927
Lin318	318	42,029	2	42,603.1760	1.3661	42,861.7373	1.9813	53.2	0.8801
Rd400	400	15,281	2	15,543.1118	1.7153	15,612.5177	2.1695	57.8	0.1170
Fl417	417	11,861	1.5	11,949.8625	0.7492	11,990.0567	1.0881	47	0.0792
Pr439	439	107,217	2	108,136.8121	0.8579	108,910.0541	1.5791	40.4	1.6785
Pcb442	442	50,778	2	51,568.4634	1.5567	51,765.8888	1.9455	39.3	0.2402
Att532	532	87,550	2.25	88,686.5156	1.2981	89,039.4465	1.7013	50.6	0.3618
Rat575	575	6,773	2	6,957.3482	2.7218	7,007.5401	3.4629	55.4	0.0641
U724	724	41,910	2	42,860.8954	2.2689	43,084.7660	2.8031	56	0.2812
Pr1002	1,002	259,045	2	266,031.4591	2.6970	26,7812.9399	3.3847	66.5	2.7453
Fl1400	1,400	20,127	2.25	20,605.6503	2.3782	20,725.6266	2.9742	63.4	0.2560
Fl3795	3,795	28,772	2	29,567.9261	2.7663	29,854.0745	3.7609	83	2.3596

Figure 6 Optimal tour and convergence curve of Pr144 (total length is 58535.2218, iteration is 28) (see online version for colours)

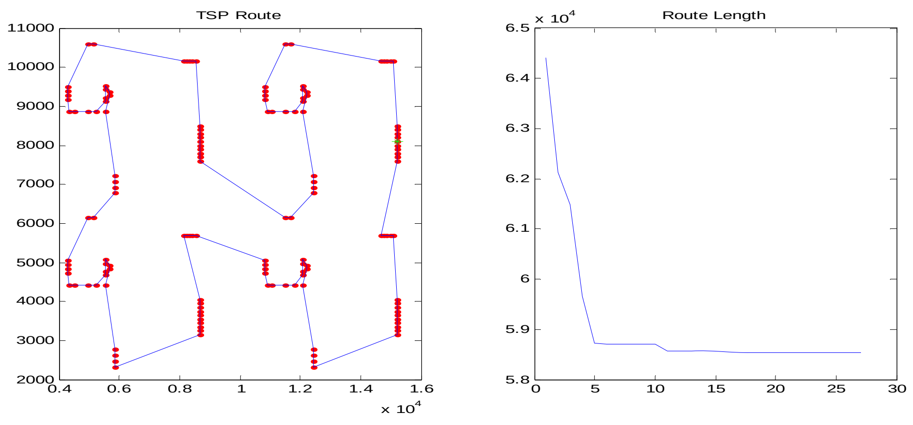


Figure 7 Optimal tour and convergence curve of U159 (total length is 42075.6700, iteration is 32) (see online version for colours)

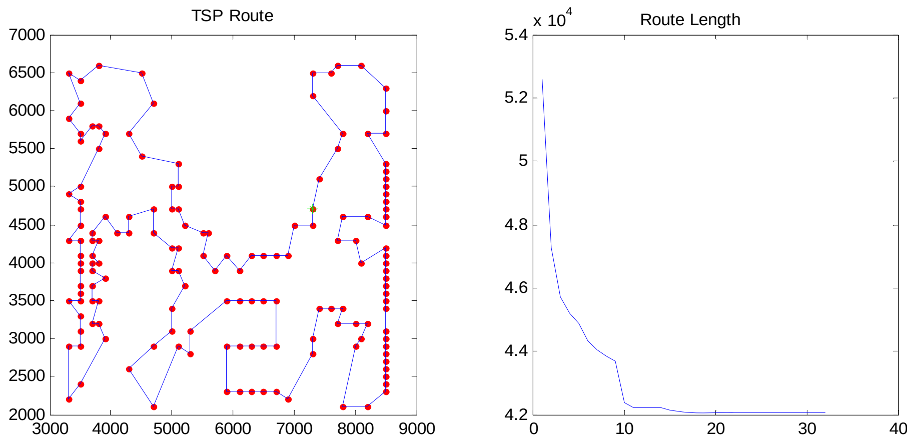
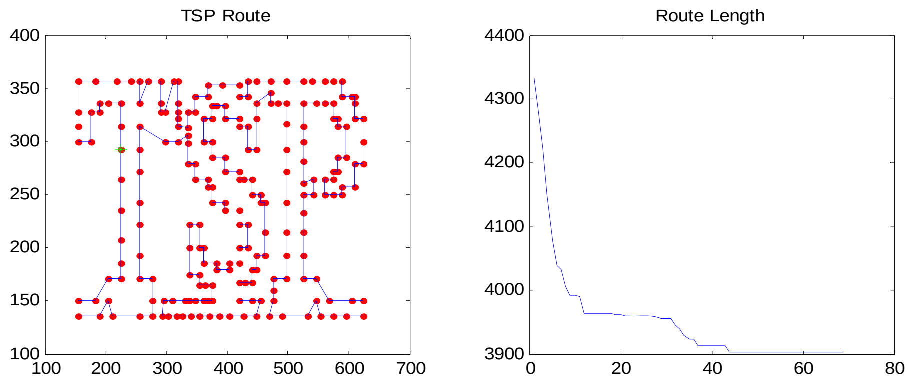


Figure 8 Optimal tour and convergence curve of Tsp225 (total length is 3902.7394, iteration is 69) (see online version for colours)



In order to compare the performance of DCS further, Table 3 shows some optimal results of other metaheuristic, '-' represents no records in corresponding literature. The italics text represents the best result among comparative results.

Table 3 Comparison of results

Instance	*m*	*TSPLIB optimal*	*ACS (Ji et al., 2010)*	*ACOMGR (Ji et al., 2010)*	*AGSO (Zhou and Huang, 2012)*	*FEA (Yan et al., 2007)*	*DCS*
Eil51	51	426	-	-	*428.87*	-	*428.87*
Berlin52	52	7,542	-	-	*7,544.37*	-	*7,544.37*
StT70	70	675	-	-	*677.11*	678.60	*677.11*
Eil76	76	538	-	-	-	545.39	*544.37*
KroA100	100	21,282	-	-	*21,285.44*	*21,285.44*	*21,285.44*
Rd100	100	7,910	-	-	-	*7,910.40*	*7,910.40*
Lin105	105	14,379	-	-	*14,383.00*	-	*14,383.00*
Pr107	107	44,303	44,647.6	44,346.2	44,337.36	-	*44,301.68*
Pr124	124	59,030	60,702.4	59,091.80	*59,030.74*	-	*59,030.74*
Pr136	136	96,772	99,324.6	96,916.50	97,684.42	*96,770.92*	96,800.81
Pr144	144	58,537	-	-	-	*58,535.22*	*58,535.22*
Pr152	152	73,682	77,969.70	73,881.70	*73,683.64*	-	*73,683.64*
Pr226	226	80,369	85,254.50	80,442.40	80,422.98	-	*80,440.04*
A280	280	2,579	-	-	-	2,856.77	*2,613.12*
Fl1400	1,400	20,127	22,085.6	20,689.4	-	-	*20,605.65*
Fl3795	3,795	28,772	31,926.3	29,723.4	-	-	*29,710.98*

From Table 3, except for instance Pr136, the solutions obtained by DCS is better than ACS and ACOMGR among the 16 instances; although the solution of a part of instances is identical comparing with DCS and AGSO, DCS is superior to AGSO for instances Pr107, Pr136 and Pr226. Analogously, DCS is superior to FEA for instances StT70, Eil76 and A280. In general, the quality of solutions obtained by DCS is good. Figures 9 to 10 is the optimal tour of instances Fl1400 and Fl3795, these instances include more than 1000 cities.

Figure 9 Optimal tour of Fl1400 (total length is 20605.65) (see online version for colours)

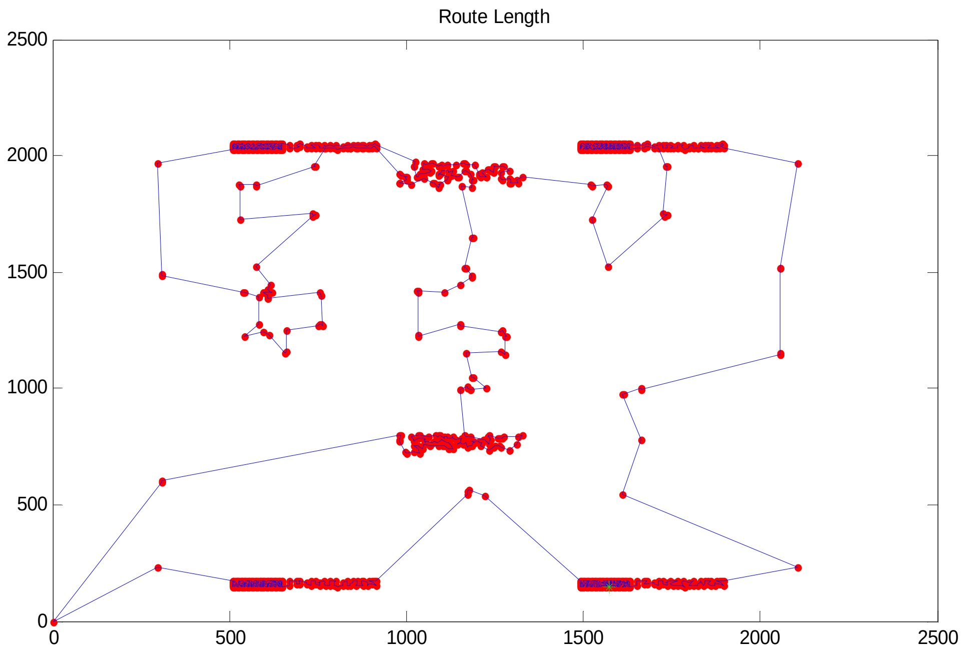
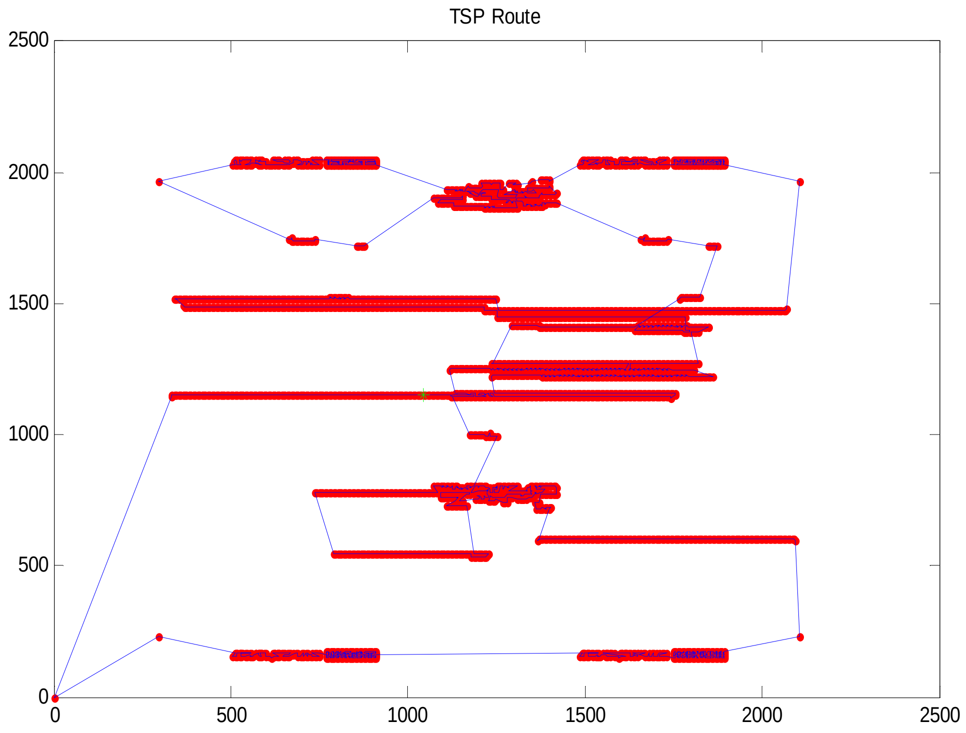


Figure 10 Optimal tour of Fl3795 (total length is 29710.98) (see online version for colours)



5 Conclusions

In this paper, a DCS algorithm is proposed to solve the TSP. The algorithm is based on the continuous CS with brood parasitic behaviour and Lévy flight. In every generation, each CS makes a new nest and abandons the old one to decrease the TSP route. In order to accelerate the convergence rate, the proposed algorithm applies learning operator, ‘A’ operator and 3-opt to the bulletin board. In general, the idea of new individual generation and learning operator derives from IO; however, there are some differences. The biggest difference is that the operators in the proposed algorithm applying DCS to search around the current city. In addition, the learning operator is more effective than original crossover operator. The numerical experiments results show that the proposed algorithm can find the global optimal solution with rapid convergences and stable generation. Moreover, the applications of CS into optimisation problems have shown its promising efficiency, we think that this methodology can easily be adapted to other variants of the TSP and even to other combinatorial optimisation problems.

Acknowledgements

This work is supported by National Science Foundation of China under Grant No. 61165015. Key Project of Guangxi Science Foundation under Grant No. 2012GXNSFDA053028, Key Project of Guangxi High School Science Foundation under Grant No. 20121ZD008, the Funded by Open Research Fund Program of Key Lab of Intelligent Perception and Image Understanding of Ministry of Education of China under Grant No.IPIU01201100 and the Innovation Project of Guangxi Graduate Education under Grant No. YCSZ2012063.

References

Burnwal, S. and Deb, S. (2012) ‘Scheduling optimization of flexible manufacturing system using cuckoo search-based approach’, *Int. J. Adv Manuf Technol*, pp.1–13.

Chen, Y. and Wang, Z. (2007) ‘Study on combinatorial optimization problem represented by TSP: recent research work and perspective’, *Computer Simulation*, Vol. 24, No. 6, pp.171–174, 247.

Dhivya, M. and Sundarambal, M. (2011) ‘Cuckoo search for data gathering in wireless sensor networks’, *Int. J. Mobile Communications*, Vol. 9, No. 6, pp.642–656.

Guo, T. and Michalewicz, Z. (1998) ‘Invor-Over Operator for the TSP’, *Proceedings of the 5th Parallel Problem Solving from Nature Conference*, Springer, Berlin, pp.1498–1520.

Jati, G.K., Manurung, H.M. and Suyanto, S. (2012) ‘Discrete cuckoo search for traveling salesman problem’, *Computing and Convergence Technology (ICCCT), 2012 7th International Conference on*, 3–5 December, pp.993, 997.

Ji, J.Z., Huang, Z., Liu, C.N. et al. (2010) ‘An ant colony algorithm based on multiple-grain representation for the traveling salesman problems’, *Journal of Computer Research and Development*, Vol. 47, No. 3, pp.434–444.

Layeb, A. (2012) ‘A novel quantum-inspired cuckoo search for knapsack problems’, *Int. J. Bio-inspired Computation*, Vol. 3, No. 5, pp.297–305.

Noghrehabadi, A., Ghalambaz, M. and Vosough, A. (2011) ‘A hybrid power series – cuckoo search optimization algorithm to electrostatic deflection of micro fixed-fixed actuators’, *Int. J. Multidisciplinary Science and Engineering*, Vol. 2, No. 4, pp.22–26.

Ouaarab, A., Ahiod, B. and Yang, Xin-She (2013) 'Discrete version of the cuckoo search algorithm for the traveling salesman problem', *Neural Comput & Applic*, 11 April, Published online.

Tein, L.H. and Ramli, R. (2010) 'Recent advancements of nurse scheduling models and a potential path', in *Proc. 6th IMT-GT Conference on Mathematics, Statistics and its Applications (ICMSA 2010)*, pp.395–409.

Walton, S., Hassan, O., Morgan, K. and Brown, M.R. (2011) *Modified Cuckoo Search: A New Gradient Free Optimisation Algorithm*, 30 June, Chaos, Solitons & Fractals.

Yan, X., Wu, Q. and Li, H. (2007) 'A fast evolutionary algorithm for traveling salesman problem', *Proceedings of the Third International Conference on Natural Computation*, IEEE Press.

Yang, X.S. and Deb, S. (2011) 'Multiobjective cuckoo search for design optimization', *Computers & Operations Research*, Vol. 10, No. 9, pp.1–9.

Yang, X-S. (2010) *Nature-Inspired Metaheuristic Algorithms*, 2nd ed,, Luniver Press.

Yang, X-S. and Deb, S. (2009) 'Cuckoo search via Lévy flights', *World Congress on Nature & Biologically Inspired Computing*, IEEE Publications, pp.210–214.

Yang, X-S. and Deb, S. (2010) 'Engineering optimisation by cuckoo search', *Int. J. Mathematical Modelling and Numerical Optimisation*, Vol. 1, No. 4, pp.330–343.

Yildiz, A.R. (2012) 'Cuckoo search algorithm for the selection of optimal machine parameters in milling operations', *Int. J. Adv. Manuf. Technol.*, Vol. 40, pp.617–623.

Zheng, H. and Zhou, Y. (2012) 'A novel cuckoo search optimization algorithm based on Gauss distribution', *J. Computational Information Systems*, Vol. 8, No. 10, pp.4193–4200.

Zhou, Y. and Huang, Z. (2012) 'Artificial glowworm swarm optimization algorithm for TSP', *Control and Decision*, Vol. 27, No. 12, pp.1816–1821.