Neural Comput & Applic (2014) 24:1659–1669
DOI 10.1007/s00521-013-1402-2

ORIGINAL ARTICLE

Discrete cuckoo search algorithm for the travelling salesman problem

Aziz Ouaarab · Belaïd Ahiod · Xin-She Yang

Received: 24 November 2012 / Accepted: 28 March 2013 / Published online: 11 April 2013

Abstract In this paper, we present an improved and discrete version of the Cuckoo Search (CS) algorithm to solve the famous traveling salesman problem (TSP), an NP-hard combinatorial optimisation problem. CS is a metaheuristic search algorithm which was recently developed by Xin-She Yang and Suash Deb in 2009, inspired by the breeding behaviour of cuckoos. This new algorithm has proved to be very effective in solving continuous optimisation problems. We now extend and improve CS by reconstructing its population and introducing a new category of cuckoos so that it can solve combinatorial problems as well as continuous problems. The performance of the proposed discrete cuckoo search (DCS) is tested against a set of benchmarks of symmetric TSP from the well-known TSPLIB library. The results of the tests show that DCS is superior to some other metaheuristics.

Keywords Nature-inspired metaheuristics · Cuckoo search · Lévy flights · Combinatorial optimisation · Traveling salesman problem

A. Ouaarab (✉) · B. Ahiod
LRIT, Associated Unit to the CNRST(URAC) no 29, Mohammed V-Agdal University, B.P. 1014 Rabat, Morocco
e-mail: aziz.ouaarab@gmail.com

B. Ahiod
e-mail: ahiod@fsr.ac.ma

X.-S. Yang
School of Science and Technology, Middlesex University, The Burroughs, London NW4 4BT, UK
e-mail: xy227@cam.ac.uk

1 Introduction

Optimisation problems, either single-objective or multi-objective, are generally difficult to solve. Many of them are said to be NP-hard and cannot be solved efficiently by any known algorithm in a practically acceptable time scale. In fact, many seemingly simple problems are very difficult to solve because the number of combinations increases exponentially with the size of the problem of interest; searching for every possible combination is extremely computationally expansive and unrealistic. The most famous example is probably the traveling salesman problem (TSP) in which a salesperson intends to visit a number of cities exactly once, and returning to the starting point, while minimizing the total distance travelled or the overall cost of the trip. TSP is one of the most widely studied problems in combinatorial optimisation. It belongs to the class of NP-hard optimisation problems [2], whose computational complexity increases exponentially with the number of cities. It is often used for testing optimisation algorithms. In addition, The TSP and its variants have several important applications [19, 25], such as, drilling of printed circuit boards, X-ray crystallography, computer wiring, vehicle routing and scheduling, control of robots, among others. Therefore, solving this class of problems is both of academic interest and practical importance, and consequently, it has been an important topic of active research.

No efficient algorithm exists for the TSP and all its relevant variants or problems of the same class. The need to quickly find good (not necessarily optimal) solutions to these problems has led to the development of various approximation algorithms such as metaheuristics [3, 14]. On the other hand, metaheuristic algorithms have demonstrated their potential and effectiveness in solving a wide variety of optimisation problems and have many advantages over traditional algorithms. Two of the advantages are simplicity and flexibility. Metaheuristics are usually simple to implement, but they often can solve complex problems and can thus be adapted to solve many real-world optimization problems, from the fields of operations research, engineering to artificial intelligence [10, 11, 12, 36]. In addition, these algorithms are very flexible, and they can deal with problems with diverse objective function properties, either continuous, or discrete, or mixed. Such flexibility also enables them to be applied to deal a large number of parameters simultaneously.

Metaheuristic algorithms use search strategies to explore the search space more effectively, often focusing on some promising regions of the search space. These methods begin with a set of initial solutions or an initial population, and then, they examine step by step a sequence of solutions to reach, or hope to approach, the optimal solution to the problem of the interest. Among the most popular metaheuristics, we can have a long list, to name a few, genetic algorithms (GA), Tabu search (TS), simulated annealing (SA), ant colonies optimisation (ACO) which are presented in Glover and Kochenberger [14], and particle swarm optimisation (PSO)[16]. Bee colonies optimisation (BCO) [30], monkey search algorithm (MS) [22], harmony search algorithm (HS) [13], firefly algorithm (FA) [32], intelligent water drops (IWD) [26], bat-inspired algorithm (BA) [33], cuckoo search (CS) [34], and krill herd (KH) [9] are among the recently proposed metaheuristics. Most of these metaheuristics are nature inspired, mimicking the successful features of the underlying biological, physical, or sociological systems. The success of these methods in solving various problems such as combinatorial optimisation problems comes from a relative ease of implementation, a good adaptation to practical applications and proper consideration of their constraints, and producing high quality solutions [29]. However, some algorithms can produce better solutions to some particular problems than others. Therefore, there is no specific algorithm to solve all optimisation problems. So the development of new metaheuristics remains a great challenge, especially for tough NP-hard problems [31].

Many of metaheuristic algorithms have been applied to solve TSP by various researchers, such as SA [4], TS [20], GA [17], ACO [8], Discrete particle swarm optimisation (DPSO) [27], genetic –simulated annealing ant colony system with particle swarm optimisation techniques (GSA-ACS-PSOT) [6] and fuzzy particle swarm optimisation with simulated annealing and neighbourhood information (PSO-NIC-SA) [1]. This paper introduces a new variant of cuckoo search (CS) so as to improve and to efficiently solve the symmetric TSP. CS is a metaheuristic search algorithm which is recently developed by Yang and Deb in 2009 [34, 35]. This novel algorithm has been shown to be very effective in solving continuous optimisation problems. Inspired by the obligate brood parasitic behaviour of some cuckoo species, combined with Lévy flights which describe the foraging patterns adopted by many animals and insects, CS is a good example of nature-inspired metaheuristics. CS is also characterized by the reduced number of parameters and provided effective results for multimodal functions in comparison with both genetic algorithms (GA) and particle swarm optimisation (PSO) [35].

This paper is organized as follows: Sect. 2 first briefly introduces the standard CS, and then describes the improvement carried out on the source of inspiration of CS. Section 3 introduces briefly the TSP. Section 4 describes the discrete CS to solve symmetric TSP. Section 5 presents in detail the results of numerical experiments on a set of benchmarks of the so-called symmetric TSP from the TSPLIB library [24]. Finally, Sect. 6 concludes with some discussions.

2 Cuckoo search algorithm (CS)

2.1 Basic CS

Among many interesting feature of cuckoo species, a striking feature of cuckoos is that some species engage the so-called brood parasitism. Female cuckoos lay eggs in the nests of another species to let host birds to hatch and brood young cuckoo chicks. To increase the probability of having a new cuckoo and reduce the probability of abandoning eggs by the host birds, cuckoos (female, male, and young) use several strategies [23].

In the standard Cuckoo Search algorithm (CS) [34], a cuckoo searches for a new nest via Lévy flights. Lévy flights, named by the French mathematician Paul Lévy, represent a model of random walks characterized by their step lengths which obey a power-law distribution. Several scientific studies have shown that the search for preys by hunters follows typically the same characteristics of Lévy flights. Lévy flights are widely used in optimisation and in many fields of sciences [5, 28, 34].

Algorithm 1 Cuckoo Search

```
1:  Objective function $f(x), x = (x_1, \ldots, x_d)^T$
2:  Generate initial population of $n$ host nests $x_i$ $(i = 1, \ldots, n)$
3:  while ($t$ <MaxGeneration) or (stop criterion) do
4:      Get a cuckoo randomly by Lévy flights
5:      Evaluate its quality/fitness $F_i$
6:      Choose a nest among $n$ (say, $j$) randomly
7:      if $(F_i > F_j)$ then
8:          replace $j$ by the new solution
9:      end if
10:     A fraction ($p_a$) of worse nests are abandoned and new ones are built
11:     Keep the best solutions (or nests with quality solutions)
12:     Rank the solutions and find the current best
13: end while
14: Postprocess results and visualization
```

CS is a metaheuristic search algorithm which was recently developed by Xin-She Yang and Suash Deb in 2009, initially designed for solving multimodal functions. CS as shown in Algorithm 1 is summarized around the following ideal rules [34]: (1) Each cuckoo lays one egg at a time and selects a nest randomly; (2) The best nest with the highest quality egg can pass onto the new generations; (3) The number of host nests is fixed, and the egg laid by a cuckoo can be discovered by the host bird with a probability $p_a \in [0,1]$.

The following Eq. (1) is used to produce a new solution $x_i^{(t+1)}$, for a cuckoo i, by a Lévy flight:

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

where α ($\alpha > 0$) is the step size. In the general case, α should be associated with the scales of the problem of interest, though $\alpha = O(1)$ can be used in many cases. The step length follows the Lévy distribution

$$\mathrm{Levy}(s, \lambda) \sim s^{-\lambda}, \quad (1 < \lambda \leq 3) \quad (2)$$

which has an infinite variance with an infinite mean [34]. Here, s is step size drawn from a Lévy distribution.

2.2 Improved CS

CS succeeded in proving its superior performance, compared with PSO and GA (to solve multimodal functions) with its better strategy in exploring the solution space. This strategy is enhanced by Lévy flights, which has an important role in controlling the balance between intensification and diversification. And the reduced number of parameters enables CS to be more versatile [34].

There is still some room for improvement in CS, both in terms of the inspiration source and of the algorithm itself. The strength of CS is the way how to exploit and explore the solution space by a cuckoo. This cuckoo can have some 'intelligence' so as to find much better solutions. So we can control the intensification and diversification through the cuckoo's mobility in the search space. The proposed improvement considers a cuckoo as a first level of controlling intensification and diversification, and since such a cuckoo is an individual of a population, we can qualify the population as a second level of control, which can be restructured by adding a new category of cuckoos smarter and more efficient in their search.

Studies show that cuckoos can also engage a kind of surveillance on nests likely to be a host [23]. This behaviour can serve as an inspiration to create a new category of cuckoos that have the ability to change the host nest during incubation to avoid abandonment of eggs. These cuckoos use mechanisms before and after brooding such as the observation of the host nest to decide if the nest is the best choice or not (so, it looks for a new nest much better for the egg). In this case, we can talk about a kind of local search performed by a fraction of cuckoos around current solutions.

For simplicity, we can divide the mechanism adopted by this new fraction of cuckoos in our proposed approach into two main steps: (1) a cuckoo, initially moves by Lévy flights towards a new solution (which represents a new area); (2) from the current solution, the cuckoo in the same area seeks a new, better solution. According to these two steps, we can see that the search mechanism with a new fraction p_c can be directly introduced in the standard algorithm of CS. So the population of improved CS algorithm (Algorithm 2) can be structured in terms of three types of cuckoos:

1. A cuckoo, seeking (from the best position) areas which may contain new solutions that are much better than the solution of an individual can be randomly selected in the population;
2. A fraction p_a of cuckoos seek new solutions far from the best solution;
3. A fraction p_c of cuckoos search for solutions from the current position and try to improve them. They move from one region to another via Lévy flights to locate the best solution in each region without being trapped in a local optimum.

The goal of this improvement is to strengthen intensive search around the best solutions of the population, and at the same time, randomization should be properly used to explore new areas using Lévy flights. Thus, an extension to the standard CS is the addition of a method that handles the fraction 'p_c' of smart cuckoos.

We can expect that the new category of the cuckoo makes it possible for CS to perform more efficiently with fewer iterations. It gives better resistance against any potential traps and stagnation in local optima in the case of TSP. This allows to the adaptation of CS to TSP more control over the intensification and diversification with fewer parameters. The adaptation of CS for solving symmetric TSP is described in the Sect. 4.

Algorithm 2 Improved Cuckoo Search

```
1:  Objective function $f(x), x = (x_1, \ldots, x_d)^T$
2:  Generate initial population of $n$ host nests $x_i$ $(i = 1, \ldots, n)$
3:  while ($t$ <MaxGeneration) or (stop criterion) do
4:     Start searching with a fraction ($p_c$) of smart cuckoos
5:     Get a cuckoo randomly by Lévy flights
6:     Evaluate its quality/fitness $F_i$
7:     Choose a nest among $n$ (say, $j$) randomly
8:     if ($F_i > F_j$) then
9:        replace $j$ by the new solution;
10:    end if
11:    A fraction ($p_a$) of worse nests are abandoned and new ones are built;
12:    Keep the best solutions (or nests with quality solutions);
13:    Rank the solutions and find the current best
14: end while
15: Postprocess results and visualization
```

3 The traveling salesman problem

The Traveling salesman problem (TSP) [15, 18] is defined by N cities and distance matrix $D = (d_{ij})_{N \times N}$ which gives distances between all pairs of cities. In TSP, the objective is to find a tour (i.e., a closed path) which visits each city exactly once and has the minimum length. A tour can be represented as a cyclic permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(N))$ of cities from 1 to N if $\pi(i)$ is interpreted to be the city visited in step $i, i = 1, \ldots, N$. The cost of a permutation (tour) is defined as:

$$f(\pi) = \sum_{i=1}^{N-1} d_{\pi(i)\pi(i+1)} + d_{\pi(N)\pi(1)} \quad (3)$$

If the distances satisfies $d_{ij} = d_{ji}$ for $1 \leq i, j \leq N$, this case is the symmetric TSP.

TSP can be modelled as a weighted graph. The vertices of the graph correspond to cities and the graph's edges correspond to connections between cities,; the weight of an edge is the corresponding connections distance. A TSP tour now becomes a hamiltonian cycle and an optimal TSP tour is the shortest hamiltonian cycle.

4 Solving the TSP with CS

The main idea of our present work is that the improved CS seeks good solutions found using local search in areas specified by Lévy flights. We can say that both approaches, improved CS and local search, constitute a single entity in finding solutions of good quality. The weakness of local search is that it is likely to be trapped in a local optimum. This can easily be strengthened by using our improved CS that requires the displacements by zones and not by solutions, which should minimize the probability of falling into local optima.

One of the objectives in extending CS to solve the traveling salesman problem (TSP) is to keep its main advantages and integrate these advantages into the discrete version of improved CS. The process of adapting CS to TSP focuses mainly on the reinterpretation of terminology used in the basic CS. CS and its inspiration sources can be structured and explained in the following five main elements: egg, nest, objective function, search space, and Lévy flights. These key elements can have important meanings for combinatorial problems.

4.1 The egg

If we assume that a cuckoo lays a single egg in one nest, we can give eggs the following properties:

– An egg in a nest is a solution represented by one individual in the population;
– An egg of the cuckoos is a new solution candidate for a place/location in the population.

We can say that an egg is the equivalent of a hamiltonian cycle. Here, we neglect the need to take a departure city for all circuits and also the direction of the tour taken by the salesman.

4.2 The nest

In CS, the following features can be imposed concerning a nest:

– The number of nests is fixed;
– A nest is an individual of the population and the number of nests is equal to the size of the population;
– An abandoned nest involves the replacement of an individual of the population by a new one.

By the projection of these features on TSP, we can say that a nest is shown as an individual in the population with its own hamiltonian cycle. Obviously, a nest can have multiple eggs for future extensions. In the present paper, each nest contains only one egg, for simplicity.

4.3 Objective function

Each solution in the search space is associated with a numeric objective value. So the quality of a solution is proportional to the value of the objective function. In CS, a nest egg of better quality will lead to new generations. This means that the quality of a cuckoo's egg is directly related to its ability to give a new cuckoo. For the traveling salesman problem, the quality of a solution is related to the length of the hamiltonian cycle. The best solution is the one with the shortest hamiltonian cycle.

4.4 Search space

In the case of two dimensions, the search space represents the positions of potential nests. These positions are $(x, y) \in \mathbb{R} \times \mathbb{R}$. To change the position of a nest, we only have to modify the actual values of its coordinates. It is obvious that moving nests or locations of the nests does not impose real constraints. This is the case in most continuous optimisation problems, which can be considered as an advantage that avoids many technical obstacles such as the representation of the coordinates in the solution space of TSP, especially in the mechanism for moving a solution from one neighbourhood to another. The coordinates of cities are fixed coordinates of the visited cities; however, the visiting order between the cities can be changed.

4.4.1 Moving in the search space

Since the coordinates of cities are fixed, the movements are based on the order of visited cities. There are several methods, operators, or perturbations that generate a new solution from another existing solution by changing the order of visited cities.

In the adaptation of CS to TSP, we have a discrete CS where perturbations used to change the order of visited cities are 2-opt moves [7] and double-bridge moves [21]. The 2-opt move method is used for small perturbations, and large perturbations are made by double-bridge move. A 2-opt move, as shown in Fig. 1, removes two edges from a tour (solution or Hamiltonian cycle) and reconnects the two paths created. A double-bridge move cuts four edges and introduces four new ones as shown in Fig. 2.

4.4.2 The neighbourhood

In continuous problems, the meaning of neighbourhood is obvious. However, for combinatorial problems, the notion of neighbourhood requires that the neighbour of a given solution must be generated by the smallest perturbation. This perturbation must make the minimum changes on the solution. This leads to the 2-opt move, because, for a new solution, and the minimum number of non-contiguous edges that we can delete is two. So, 2-opt move is a good candidate for this type of perturbation.

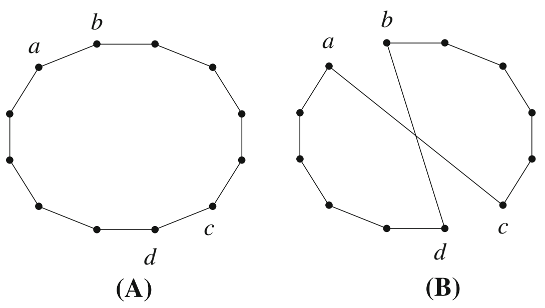

Fig. 1 2-opt move. **a** Initial tour. **b** The tour created by 2-opt move [the edges (*a*, *b*) and (*c*, *d*) are removed, while the edges (*a*, *c*) and (*b*, *d*) are added]

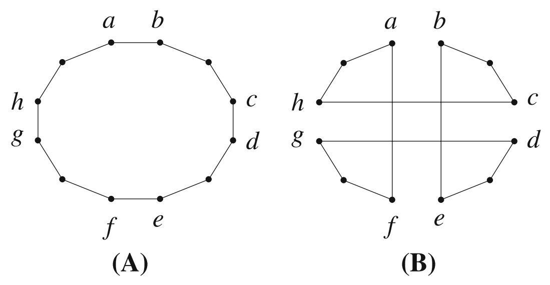

Fig. 2 Double-bridge move. **a** Initial tour. **b** The tour created by double-bridge move [the edges (*a*, *b*), (*c*, *d*), (*e*, *f*) and (*g*, *h*) are replaced by the edges (*a*, *f*), (*c*, *h*), (*e*, *b*) and (*g*, *d*), respectively]

4.4.3 The step

The step of a movement is the distance between two solutions. It is based on the space topology and the concept of neighbourhood. The step length is proportional to the number of successive 2-opt moves on a solution. A big step is represented by a double-bridge move.

4.5 Lévy flights

Lévy flights have as a characteristic of an intensive search around a solution, followed by occasional big steps in the long run. According to Yang and Deb [34], in some optimisation problems, the search for a new best solution is more efficient via Lévy flights. In order to improve the quality of search, we will associate the step length to the value generated by Lévy flights as outlined in the standard CS.

5 Experimental results

The basic and the improved discrete cuckoo search (DCS) algorithms proposed are tested on some instances (benchmarks) of TSP taken from the publicly available electronic library TSPLIB of TSP problems [24]. Most of the instances included in TSPLIB have already been solved in the literature and their optimality results can be used to compare algorithms. Forty-one instances are considered with sizes ranging from 51 to 1379 cities. In Reinelt [24], all these TSP instances belong to the Euclidean distance type. A TSP instance provides cities with their coordinates. The numerical value in the name of an instance represents the number of provided cities, e.g., the instance named eil51 has 51 cities.

A comparison between both algorithms, the basic DCS, and the improved DCS, is firstly carried out. Then, the improved DCS algorithm is compared with some other recent methods (genetic simulated annealing ant colony system with particle swarm optimisation techniques (GSA-ACS-PSOT) [6] and discrete particle swarm optimisation (DPSO) [27]). Notice that in Chen and Chien [6], the authors have compared their proposed method with others metaheuristic algorithms for solving TSP in literature.

We have implemented basic/standard and improved DCS algorithms using Java under 32 bit Vista operating system. Experiments are conducted on a laptop with Intel(R) Core^{TM} 2 Duo 2.00 GHz CPU, and 3 GB of RAM. The values of parameters of the proposed algorithm are selected, based on some preliminary trials. The selected parameters in both algorithms (basic and improved DCS) are those values that gave the best results concerning both the solution quality and the computational time. The parameter settings used in the experiments are shown in Table 1. In each case study, 30 independent runs of the algorithms with these parameters are carried out. Figure 3 shows that the maximum number of iterations (*MaxGeneration*) can be set to 500 for both algorithms.

Tables 2 and 3 summarize the experiments results, where the first column shows the name of the instance, the column '**opt**' shows the optimal solution length taken from the TSPLIB, the column '**best**' shows the length of the best solution found by each algorithm, the column '**average**' gives the average solution length of the 30 independent runs of each algorithm, the column '**worst**' shows the length of the worst solution found by each algorithm, the column '**PDav(%)**' denotes the percentage deviation of the average solution length over the optimal solution length of 30 runs, the column '**PDbest(%)**' gives the percentage deviation of the best solution length over the optimal solution length of 30 runs, and the column '**time**' shows the average time in seconds for the 30 runs. The percentage deviation of a solution to the best known solution (or optimal solution if known) is given by the formula:

$$\text{PDsolution}(\%) = \frac{\text{solutionlength} - \text{bestknownsolutionlength}}{\text{best known solution length}} \times 100 \tag{4}$$

Table 1 Parameter settings for both algorithms, basic and improved DCS

Parameter	Value	Meaning
n	20	Population size
p_a	0.2	Portion of bad solutions
p_c	0.6	Portion of intelligent cuckoos (only for the improved DCS)
MaxGeneration	500	Maximum number of iterations

In Table 2, the experimental results of the comparison between basic DCS and improved DCS algorithms are given. It can be seen from this Table and Figs. 3 and 4 that the improved DCS is superior to basic DCS regarding to both PDav(%) and PDbest(%). The improved DCS gets the smallest values for the fourteen TSP instances. The high performance of the improved DCS in relation to the basic DCS may be due to the improvement applied on the basic DCS by the new category of cuckoos which have an efficient method of generating new solutions by moving from one area to another to find one of the best solutions for each area.

Table 3 presents the computational results of improved DCS algorithm on 41 TSPLIB instances. The column 'SD' denotes the standard deviation which takes the value 0.00 shown in bold when all solutions found have the same length over the 30 runs, while the column '$C_{1\,\%}/C_{\text{opt}}$' gives the number of solutions that are within 1 % optimality (over 30 runs)/the number of the optimal solutions. With respect to PDbest(%), we can say that 90.24 % of the values of PDbest(%) are less than 0.5 %, which means that the best solution found, of the 30 trials, approximates less than 0.5 % of the best known solution, while the value of 0.00 shown in bold in column PDav(%) indicates that all solutions found on the 30 trials have the same length of the best known solution. Numerical values presented in Table 3 show that the improved DCS can indeed provide good solutions in reasonable time.

In Tables 4 and 5, the experimental results of the improved DCS algorithm are compared with the both methods GSA-ACS-PSOT and DPSO. The results of these two methods are directly summarized from original papers 6, 27]. It can be seen clearly from Tables 4 and 5 that DCS outperforms the other two algorithms (GSA-ACS-PSOT and DPSO) in solving all the eighteen/five tested TSP instances. The proposed DCS algorithm obtains fifteen/five best solutions while GSA-ACS-PSOT/DPSO only obtains eleven/two best solutions among eighteen/five TSP instances. Moreover, we find that the average of SDs/PDav(%)s is equal to 161.55/3.54 for the GSA-ACS-PSOT/

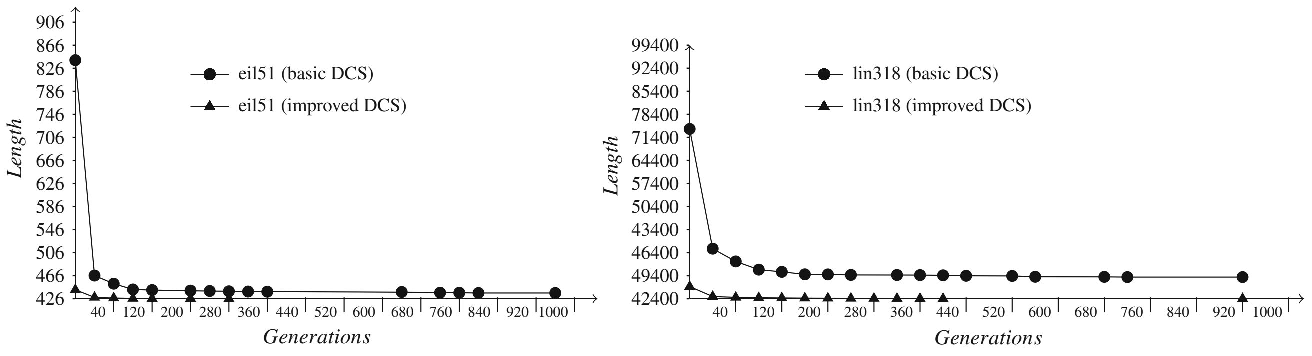

Fig. 3 Average length of the best solutions of 10 runs for eil51(opt = 426) and lin318(opt = 42029)

Table 2 Comparison of both algorithms, the basic DCS and the improved DCS on 14 TSP benchmark instances from TSPLIB

Instance	Opt	Basic DCS						Improved DCS					
		Best	Average	Worst	PDav(%)	PDbest(%)	Time(s)	Best	Average	Worst	PDav(%)	PDbest(%)	Time(s)
eil51	426	**426**	439	459	3.05	**0.00**	2.05	**426**	**426**	**426**	**0.00**	**0.00**	1.16
berlin52	7,542	**7,542**	7,836.4	8,356	3.90	**0.00**	2.24	**7,542**	**7,542**	**7,542**	**0.00**	**0.00**	0.09
st70	675	**675**	696.9	725	3.24	**0.00**	3.54	**675**	**675**	**675**	**0.00**	**0.00**	1.56
pr76	108,159	108,469	111,225.6	11,6278	2.83	0.28	4.22	**108,159**	**10,8159**	**108,159**	**0.00**	**0.00**	4.73
eil76	538	544	565.7	599	5.14	1.11	4.27	**538**	538.03	539	0.00	**0.00**	6.54
kroA100	21,282	21,515	22,419.96	23,444	5.34	1.09	6.53	**21,282**	**21,282**	**21,282**	**0.00**	**0.00**	2.70
kroB100	22,141	22,335	23,417.06	25,177	5.76	0.87	6.69	**22,141**	22,141.53	22,157	0.00	**0.00**	8.74
eil101	629	648	669.4	699	6.42	3.02	7.59	**629**	630.43	633	0.22	**0.00**	18.7
bier127	118,282	120,858	127,832.23	159,788	8.07	2.17	9.96	**118,282**	118,359.63	118,730	0.06	**0.00**	25.50
ch130	6,110	6,309	6,661.23	7,883	9.02	3.25	10.60	**6,110**	6,135.96	6,174	0.42	**0.00**	23.12
ch150	6,528	6,913	7,234.9	8,064	10.82	5.89	18.06	**6,528**	6,549.90	6,611	0.33	**0.00**	27.74
kroA150	26,524	27,328	28,928.83	32,786	9.06	3.03	14.24	**26,524**	26,569.26	26,767	0.17	**0.00**	31.23
kroA200	29,368	30,641	32,896.03	37,993	12.01	4.33	23.43	29,382	29,446.66	29,886	0.26	0.04	62.08
lin318	42,029	44,278	49,623.96	6,9326	18.07	5.35	72.55	42,125	42,434.73	42,890	0.96	0.22	156.17

Table 3 Computational results of Improved DCS algorithm for 41 TSP benchmark instances for TSPLIB

Instance	Opt	Best	Worst	Average	SD	PDav(%)	PDbest(%)	$C_{1\%}/C_{opt}$	time
eil51	426	**426**	**426**	**426**	**0.00**	**0.00**	**0.00**	**30/30**	1.16
berlin52	7542	**7,542**	**7,542**	**7,542**	**0.00**	**0.00**	**0.00**	**30/30**	0.09
st70	675	**675**	**675**	**675**	**0.00**	**0.00**	**0.00**	**30/30**	1.56
pr76	108,159	**108,159**	**108,159**	**108,159**	**0.00**	**0.00**	**0.00**	**30/30**	4.73
eil76	538	**538**	539	538.03	0.17	0.00	**0.00**	30/29	6.54
kroA100	21,282	**21,282**	**21,282**	**21,282**	0.00	**0.00**	**0.00**	**30/30**	2.70
kroB100	22,141	**22,141**	22,157	22,141.53	2.87	**0.00**	**0.00**	30/29	8.74
kroC100	20,749	**20,749**	**20,749**	**20,749**	**0.00**	**0.00**	**0.00**	**30/30**	3.36
kroD100	21,294	**21,294**	21,389	21,304.33	21.79	0.04	**0.00**	30/19	8.35
kroE100	22,068	**22,068**	22,121	2,281.26	18.50	0.06	**0.00**	30/18	14.18
eil101	629	**629**	633	630.43	1.14	0.22	**0.00**	30/6	18.74
lin105	14,379	**14,379**	**14,379**	**14,379**	**0.00**	**0.00**	**0.00**	**30/30**	5.01
pr107	44,303	**44,303**	44,358	44,307.06	12.90	0.00	**0.00**	30/27	12.89
pr124	59,030	**59,030**	**59,030**	**59,030**	**0.00**	**0.00**	**0.00**	**30/30**	3.36
bier127	118,282	**118,282**	11,8730	118,359.63	12.73	0.06	**0.00**	30/18	25.50
ch130	6,110	**6,110**	6,174	6,135.96	21.24	0.42	**0.00**	28/7	23.12
pr136	96,772	96,790	97,318	97,009.26	134.43	0.24	0.01	30/0	35.82
pr144	58,537	**58,537**	**58,537**	**58,537**	**0.00**	**0.00**	**0.00**	**30/30**	2.96
ch150	6,528	**6,528**	6,611	6,549.9	20.51	0.33	**0.00**	29/10	27.74
kroA150	26,524	**26,524**	26,767	26,569.26	56.26	0.17	**0.00**	30/7	31.23
kroB150	26,130	**26,130**	26,229	26,159.3	34.72	0.11	**0.00**	30/5	33.01
pr152	73,682	**73,682**	**73,682**	**73,682**	**0.00**	**0.00**	**0.00**	**30/30**	14.86
rat195	2,323	2,324	2,357	2,341.86	8.49	0.81	0.04	20/0	57.25
d198	15,780	15,781	15,852	15,807.66	17.02	0.17	0.00	30/0	59.95
kroA200	29,368	29,382	29,886	29,446.66	95.68	0.26	0.04	29/0	62.08
kroB200	29,437	29,448	29,819	29,542.49	92.17	0.29	0.03	28/0	64.06
ts225	126,643	**126,643**	126,810	126,659.23	44.59	0.01	**0.00**	30/26	47.51
tsp225	3,916	**3,916**	3,997	3,958.76	20.73	1.09	**0.00**	9/1	76.16
pr226	80,369	**80,369**	80,620	80,386.66	60.31	0.02	**0.00**	30/19	50.00
gil262	2,378	2,382	2,418	2,394.5	9.56	0.68	0.16	22/0	102.39
pr264	49,135	**49,135**	49,692	49,257.5	159.98	0.24	**0.00**	28/13	82.93
a280	2,579	**2,579**	2,623	2,592.33	11.86	0.51	**0.00**	25/4	115.57
pr299	48,191	48,207	48,753	48,470.53	131.79	0.58	0.03	27/0	138.20
lin318	42,029	42,125	42,890	42,434.73	185.43	0.96	0.22	15/0	156.17
rd400	15,281	15,447	15,704	15,533.73	60.56	1.65	1.08	0/0	264.94
fl417	11,861	11,873	11,975	11,910.53	20.45	0.41	0.10	30/0	274.59
pr439	107,217	107,447	109,013	107,960.5	438.15	0.69	0.21	22/0	308.75
rat575	6,773	6,896	7,039	6,956.73	35.74	2.71	1.81	0/0	506.67
rat783	8,806	9,043	9,171	9,109.26	38.09	3.44	2.69	0/0	968.66
pr1002	259,045	266,508	271,660	268,630.03	1,126.86	3.70	2.88	0/0	1662.61
nrw1379	56,638	58,951	59,837	59,349.53	213.89	4.78	4.08	0/0	3160.47

DPSO algorithm in table 4/5, while the average of SDs/PDav(%)s of our proposed DCS algorithm is equal to 31.28/0.00. Figure 5 shows the PDav(%) of both algorithms improved DCS and GSA-ACS-PSOT for the eighteen different size instances. From Fig. 5, the lower curve which is associated with the improved DCS algorithm is better, in terms of solution quality. This can be explained basically by the strengths of CS: a good balance between intensification and diversification, an intelligent use of Lévy flights and the reduced number of parameters. It can also be explained by the improved structure of the population, which uses a variety of cuckoos for search and new solution generation. One of the advantages of the improved DCS is that it is relatively independent of cuckoos in the search process for the best position. So, it is more likely to find good solutions in areas unexplored by metaheuristics, which take the best position found so far as a starting point for other much better solutions.

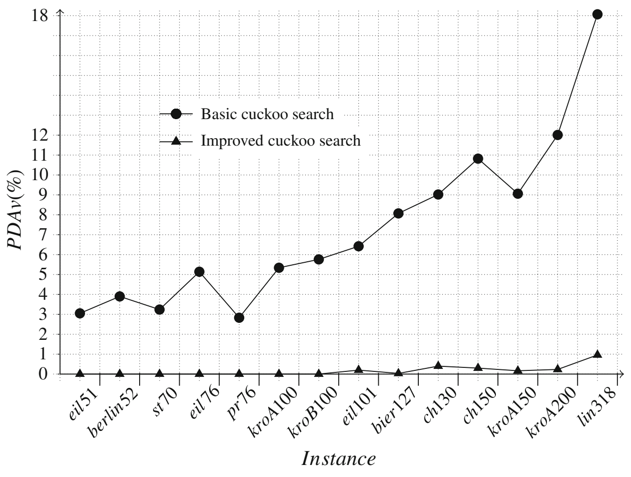

Fig. 4 PDAv(%)(over 30 runs) for 14 TSPLIB instances

6 Conclusion

In this paper, we have extended and improved the standard/basic cuckoo search (CS) via Lévy flights by reconstructing its population and introducing a new cuckoo category which is more intelligent. The improved CS is discretized and then used to solve the symmetric traveling salesman problem (TSP). This adaptation is based on a study of terminology interpretation used in CS and in its inspiration sources. Discrete CS (improved CS adapted to TSP) has been implemented and its performance has been tested on forty-one benchmark TSP instances. Its performance has been compared with GSA-ACS-PSOT [6] and DPSO [27]. The results of the comparison have shown that Discrete CS outperforms all two other methods for solving TSP. This can be explained by the management of intensification and diversification through Lévy flights and the structure of the population which contains a variety of cuckoos that use multiple research methods. Thus, the independence of our cuckoo new category, relatively to the best solution, in its search for a new solution may provide a better strategy in generating new solutions than the simple use of one current best solution, thus leading to a more efficient algorithm.

The use of local perturbations introduced in our proposed Discrete CS can provide more flexibility to solve other combinatorial optimisation problems. The goal of Discrete CS is to give good ideas for designing new generations of more efficient metaheuristic algorithms. Further

Table 4 Comparison of experimental results of the improved DCS with **GSA-ACS-PSOT** [6]

Instance	Opt	GSA-ACS-PSOT			Improved DCS		
		Best	Average	SD	Best	Average	SD
eil51	426	427	427.27	0.45	**426**	**426**	**0.00**
berlin52	7,542	**7,542**	**7,542.00**	**0.00**	**7,542**	**7,542**	**0.00**
eil76	538	**538**	540.20	2.94	**538**	538.03	0.17
kroA100	21,282	**21,282**	21,370.47	123.36	**21,282**	**21,282**	**0.00**
kroB100	22,141	**22,141**	22,282.87	183.99	**22,141**	22,141.53	2.87
kroC100	20,749	**20,749**	20,878.97	158.64	**20,749**	**20,749**	**0.00**
kroD100	21,294	21,309	21,620.47	226.60	**21,294**	21,304.33	21.79
kroE100	22,068	**22,068**	22,183.47	103.32	**22,068**	2,281.26	18.50
eil101	629	630	635.23	3.59	**629**	630.43	1.14
lin105	14,379	**14,379**	14,406.37	37.28	**14,379**	**14,379**	**0.00**
bier127	118,282	**118,282**	119,421.83	580.83	**118,282**	118,359.63	12.73
ch130	6,110	6,141	6,205.63	43.70	**6,110**	6,135.96	21.24
ch150	6,528	**6,528**	6,563.70	22.45	**6,528**	6,549.90	20.51
kroA150	26,524	**26,524**	26,899.20	133.02	**26,524**	26,569.26	56.26
kroB150	26,130	**26,130**	26,448.33	266.76	**26,130**	26,159.3	34.72
kroA200	29,368	29,383	29,738.73	356.07	29,382	29,446.66	95.68
kroB200	29,437	29,541	30,035.23	357.48	29,448	29,542.49	92.17
lin318	42,029	42,487	43,002.09	307.51	42,125	42,434.73	185.43

Table 5 Comparison of experimental results of the improved DCS with **DPSO** [27]

Instance	Opt	DPSO			Improved DCS		
		Best	Worst	PDAv(%)	Best	Worst	PDAv(%)
eil51	426	427	452	2.57	**426**	**426**	**0.00**
berlin52	7,542	**7,542**	8,362	3.84	**7542**	**7,542.0**	**0.00**
st70	675	**675**	742	3.34	**675**	**675**	**0.00**
pr76	108,159	108,280	124,365	3.81	**108,159**	**108,159**	**0.00**
eil76	538	546	579	4.16	**538**	539	0.00

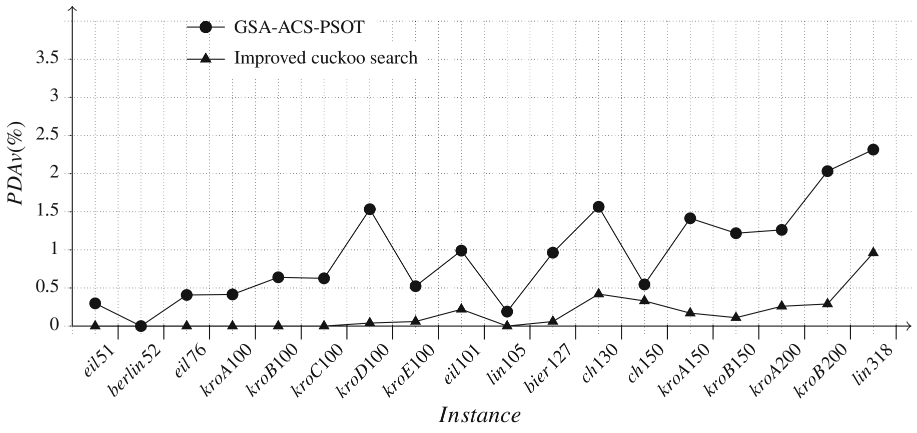

Fig. 5 PDAv(%)(over 30 runs) for 18 TSPLIB instances

studies can be fruitful if we can focus on the parametric studies and applications of DCS into other combinatorial problems such as scheduling and routing.

We want to mimic nature so that we can design new algorithms to solve very complex problems more efficiently. We also want to develop new algorithms that are truly intelligent. New algorithms should be more controllable and less complex, compared with contemporary metaheuristics. It can be expected that an intelligent algorithm should have ability to tune its algorithm-dependent parameters so as to optimize its performance automatically. In the end, some truly efficient and intelligent algorithms may emerge to solve NP-hard problems in an ever-increasing efficient way. At least, some seemingly intractable problems can be solved more efficiently by new metaheuristic algorithms.

References

1. Abdel-Kader RF (2011) Fuzzy particle swarm optimization with simulated annealing and neighbourhood information communication for solving TSP. Int J Adv Comput Sci Appl 2(5):15–21
2. Arora S (1998) Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. J ACM (JACM) 45(5):753–782
3. Blum C, Roli A (2003) Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surveys (CSUR) 35(3):268–308
4. Bonomi E, Lutton JL (1984) The n-city travelling salesman problem: statistical mechanics and the metropolis algorithm. SIAM Rev 26(4):551–568
5. Brown CT, Liebovitch LS, Glendon R (2007) Lévy flights in dobe ju/'hoansi foraging patterns. Hum Ecol 35(1):129–138
6. Chen SM, Chien CY (2011) Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. Expert Syst Appl 38(12):14439–14450
7. Croes GA (1958) A method for solving traveling-salesman problems. Oper Res 6(6):791–812
8. Dorigo M, Gambardella LM et al (1997) Ant colonies for the travelling salesman problem. BioSystems 43(2):73–82
9. Gandomi AH, Alavi AH (2012) Krill herd: a new bio-inspired optimization algorithm. Commun Nonlinear Sci Numer Simul 17(12):4831–4845
10. Gandomi AH, Yang XS, Alavi AH (2011) Mixed variable structural optimization using firefly algorithm. Comput Struct 89(23–24):2325–2336
11. Gandomi AH, Talatahari S, Yang XS, Deb S (2012) Design optimization of truss structures using cuckoo search algorithm. Struct Des Tall Special Build. doi:10.1002/tal.1033
12. Gandomi AH, Yang XS, Alavi AH (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 29(1):17–35
13. Geem ZW, Kim JH et al (2001) A new heuristic optimization algorithm: harmony search. Simulation 76(2):60–68
14. Glover F, Kochenberger GA (2003) Handbook of metaheuristics. Springer, New York
15. Gutin G, Punnen AP (2002) The traveling salesman problem and its variations, vol 12. Springer, New York
16. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Neural networks, 1995. Proceedings, IEEE international conference on, IEEE, vol4, pp 1942–1948

17. Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S (1999) Genetic algorithms for the travelling salesman problem: a review of representations and operators. Artif Intell Rev 13(2):129–170
18. Lawler EL, Lenstra JK, Kan AR, Shmoys DB (1985) The traveling salesman problem: a guided tour of combinatorial optimization, vol 3. Wiley, New York
19. Lenstra JK, Rinnooy Kan AHG (1975) Some simple applications of the travelling salesman problem. Oper Res Q 26(5): 717–733
20. Malek M, Guruswamy M, Pandya M, Owens H (1989) Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Ann Oper Res 21(1):59–84
21. Martin O, Otto SW, Felten EW (1991) Large-step markov chains for the traveling salesman problem. Complex Syst 5(3):299–326
22. Mucherino A, Seref O (2007) Monkey search: a novel metaheuristic search for global optimization. In: Data mining, systems analysis, and optimization in biomedicine(AIP conference proceedings, vol 953), American Institute of Physics, 2 Huntington Quadrangle, Suite 1 NO 1, Melville, NY, 11747-4502, USA, vol 953, pp 162–173
23. Payne RB, Sorenson MD (2005) The cuckoos, vol 15. Oxford University Press, USA
24. Reinelt G (1991) Tsplib—a traveling salesman problem library. ORSA J Comput 3(4):376–384
25. Reinelt G (1994) The traveling salesman: computational solutions for TSP applications, vol 15. Springer, New York
26. Shah-Hosseini H (2009) The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. Int J Bio-Inspired Comput 1(1):71–79
27. Shi XH, Liang YC, Lee HP, Lu C, Wang QX (2007) Particle swarm optimization-based algorithms for tsp and generalized tsp. Inf Process Lett 103(5):169–176
28. Shlesinger MF, Zaslavsky GM, Frisch U (1995) Lévy flights and related topics in physics:(Nice, 27–30 June 1994). Springer, New York
29. Taillard ED, Gambardella LM, Gendreau M, Potvin JY (2001) Adaptive memory programming: a unified view of metaheuristics. Eur J Oper Res 135(1):1–16
30. Teodorovic D, Lucic P, Markovic G, Orco MD (2006) Bee colony optimization: principles and applications. In: Neural network applications in electrical engineering, 2006. NEUREL 2006. 8th Seminar on, IEEE, pp 151–156
31. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. Evol Comput IEEE Trans 1(1):67–82
32. Yang XS (2009) Firefly algorithms for multimodal optimization. In: Stochastic algorithms: foundations and application, SAGA 2009. Lecture notes in computer sciences, vol 5792, pp 169–178
33. Yang XS (2010) A new metaheuristic bat-inspired algorithm. Nature inspired cooperative strategies for optimization (NICSO 2010), pp 65–74
34. Yang XS, Deb S (2009) Cuckoo search via lévy flights. In: Nature & biologically inspired computing, 2009. NaBIC 2009. World congress on, IEEE, pp 210–214
35. Yang XS, Deb S (2010) Engineering optimisation by cuckoo search. Int J Math Modell Numer Optim 1(4):330–343
36. Yang XS, Gandomi AH (2012) Bat algorithm: a novel approach for global engineering optimization. Eng Comput 29(5):464–483