Discrete Cuckoo Search Algorithm for the Traveling Salesman Problem

Aziz Ouaarab[a,*], Belaïd Ahiod[a], Xin-She Yang[b]

[a]GSCM-LRIT, Faculté des Sciences, Université Mohamed V-Agdal, B.P. 1014 Rabat, Morocco
[b]Department of Engineering, University of Cambridge Trumpinton Street, Cambridge CB2 1PZ, UK

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), a combinatorial optimisation problem classified as NP-hard. CS is a metaheuristic search algorithm which was recently developed by Xin-She Yang and Suash Deb in 2009, inspired by the cuckoo bird breeding behaviour. This new algorithm has proved to be very effective in solving continuous optimisation problems. We now extend and improve the CS by reconstructing the population of the solutions 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 benchmark instances 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 Metaheuristic, Cuckoo Search, Lévy flights, Combinatorial Optimisation, Traveling Salesman Problem.

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 efficient by any known algorithms in an acceptable amount of time. 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, search for every combination is extremely computationally expansive and unrealistic. The most famous example is probably

*Corresponding author. Tel.: +212 6 41 300 802
Email address: `aziz.ouaarab@gmail.com` (Aziz Ouaarab)

the traveling salesman problem (TSP) in which a salesperson intends to visit a number of cities exactly once, and returning to its starting point, while minimizing the total distance traveled 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 [1], whose the 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 [2, 3], such as, drilling of printed circuit boards, x-ray crystallography, computer wiring, vehicle routing and scheduling, and 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 problem 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 the metaheuristics [4, 5]. 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. One of the advantages is that metaheuristics, though simple, can solve complex problems, and can thus be adapted to solve any particular problem from the fields of operations research, engineering or artificial intelligence, wherever it is needed to optimize numerical functions, and in systems where a large number of parameters are simultaneously involved.

Metaheuristic algorithms use search strategies to explore only promising part of the search space, but most effectively. These methods begin with one or more solutions, and then they examine, step by step a sequence of solutions to reach, or hope to approach, the optimal solution. 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 [5], and particle swarm optimisation (PSO) [6]. Bee colonies optimisation (BCO) [7], monkey search algorithm (MS) [8], harmony search algorithm (HS) [9], firefly algorithm (FA) [10], intelligent water drops (IWD) [11], and cuckoo search [12] are among the recently proposed metaheuristics. Most of these metaheuristics are nature-inspired, mimicking successful feature in biological, physical or sociological systems. The success of these methods to solve various problems such as the combinatorial optimisation problems depends on many factors [13]: ease of implementation, ability to consider specific constraints that arise in practical applications and the high quality of the solutions they produce. However, some algorithms can produce better solutions to 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 [14].

Many of metaheuristic algorithms have been applied to solve TSP by various researchers such as, SA [15], TS [16], GA [17], ACO [18], Discrete Particle Swarm Optimisation (DPSO) [19], Genetic Simulated Annealing Ant Colony System with Particle Swarm Optimisation Techniques (GSA-ACS-PSOT) [20] and Fuzzy Particle Swarm Optimisation with Simulated Annealing and Neighborhood Information (PSO-NIC-SA) [21]. This paper introduces an improvement and adaptation of the cuckoo search algorithm (CS) to solve the symmetric TSP. CS is a metaheuristic search algorithm which is recently developed by Yang and Deb in 2009 [12, 22]. 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 that 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) [22].

This paper is organized as follows: Section 2 first briefly introduces basic 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 [23]. Finally, Section 6 concludes with some discussions.

2. Cuckoo Search Algorithm (CS)

2.1. Basic CS

Cuckoos are fascinating, medium sized birds. An interesting feature of cuckoos is that some species engage the so-called brood parasitism. Female cuckoos lay eggs in the nests of another species to ensure the breeding of the eggs and then let the host birds to raise young cuckoo chicks. To increase the probability of having a new cuckoo and reduce the probability of abandonment of eggs by the host birds, cuckoos (female, male and young) use several strategies [24].

In the standard Cuckoo Search algorithm (CS) [12], a cuckoo searches a new nest via Lévy flights. Lévy flights, named by the French mathematician Paul Lévy, represent a model of random walks characterized by its step length which obeys a power-law distribution. Several scientific studies have shown that the search for prey by hunters follows typically the same characteristics of Lévy flights. Lévy flights are widely used in optimisation and in many fields of science [12, 25, 26].

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 [12]: 1) Each cuckoo will lay one egg at a time, and selects a nest randomly; 2) The best nest with egg of highest quality can pass onto the new generations; 3) The number of host nests is fixed, and the egg laid by the bird can be discovered by the host bird with a probability $p_a \in [0,1]$.

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
```

When generating a new solution $x_i^{(t+1)}$ for a cuckoo i, a Lévy flight is performed using the following equation:

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

where α ($\alpha > 0$) is the step size associated with the scale of the problem. 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

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

which has an infinite variance with an infinite mean [12]. 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 generic [12].

There is still some room for improvement in CS, both in terms of source of inspiration and in terms 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 this cuckoo. The proposed improvement considers a cuckoo as a first control-level of the intensification and diversification, and since such a cuckoo is an individual of a population, we can qualify the population as the second control-level. This second level can be restructured by adding a category of cuckoos smarter and more efficient in their research.

Studies show that cuckoos can also engage a kind of surveillance on nests likely to be a host [24]. 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 (in this case it looks for a new nest much better for the egg). The goal is to strengthen intensive research 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 this new category of cuckoos, represented in the population by a portion ”p_c” of the best solutions.

In improved CS algorithm (**Algorithm 2**), the population is structured in three types of cuckoos:

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

We can say 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 trapping in local optima in the case of TSP. This allows the adaptation of CS to TSP more control over the intensification and diversification with a few parameters. The adaptation of CS for solving symmetric TSP is described in the section 4.

3. The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) 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

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:      Intelligent cuckoos evolution with a fraction ($p_c$)
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
```

to find a tour (i.e. 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:

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

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 is now a Hamiltonian cycle and an optimal TSP tour is a shortest Hamiltonian cycle.

4. Solving the TSP with CS

The main idea of our work is that 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 to be trapped in a local optimum. This is easily strengthened by improved CS that requires the displacement by zones and not by solutions, which minimizes 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 re-interpretation of terminology used in the basic CS. CS and its inspiration source are structured around 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 consider that a cuckoo lays a single egg in one nest, we can gives eggs the following properties:

- An egg in a nest is a solution adopted by one individual in the population;
- An egg of the cuckoos is a new solution candidate for a place 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 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 only contains a single egg, for simplicity.

4.3. The 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. The 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 a position of nest is simply the change the actual values of the coordinates. We can notice, 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 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 from a solution to another neighbourhood.

The coordinates of cities are fixed coordinates of the visited cities, however, the visiting order between the cities can be changing.

4.4.1. Moving in the search space

Since the coordinates of cities are fixed, the movements are based on the cities visiting order. 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 (discrete CS), perturbations used to change the order of visited cities are 2-opt move [27] and double-bridge move [28]. 2-opt move is used for small perturbations and large perturbations are made by double-bridge move. The 2-opt move, as shown in **Figure** 1 removes two edges from a tour (solution or Hamiltonian cycle) and reconnects the two paths created. The double-bridge move cuts four edges and introduces four new ones as shown in **Figure** 2.

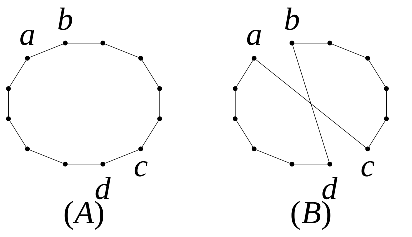

Figure 1: 2-opt move. (*A*) initial tour. (*B*) the tour created by the 2-opt move (the edges (a, b) and (c, d) are removed, while the edges (a, c) and (b, d) are added).

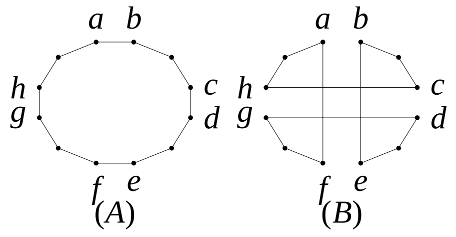

Figure 2: Double-bridge move. (*A*) initial tour. (*B*) the tour created by the double-bridge move (the edges (a, b), (c, d), (e, f) and (g, h) are replaced by the edges (a, f), (c, h), (e, b) and (g, d) respectively).

4.4.2. 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, 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.

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 intensified search around a solution, followed by big steps in the long run. And according to [12], 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 the TSP taken from the publicly available electronic library TSPLIB of TSP problems [23]. 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 [23], all these TSP instances belong to the Euclidean distance type. A TSP instance provides some 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) [20] and Discrete Particle Swarm Optimisation (DPSO) [19]). Notice that in [20], the authors have compared their proposed method with others metaheuristic algorithms for solving TSP in literature.

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

We have implemented that 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 needed to reach this solution. The parameter settings used in the experiments are shown in **Table 1**. In each case study, 30 independent runs of algorithm with the selected parameters are carried out. **Figure 3** shows that the maximum number of iterations (*MaxGeneration*) can be set equal to 500 for both the improved and the basic DCS.

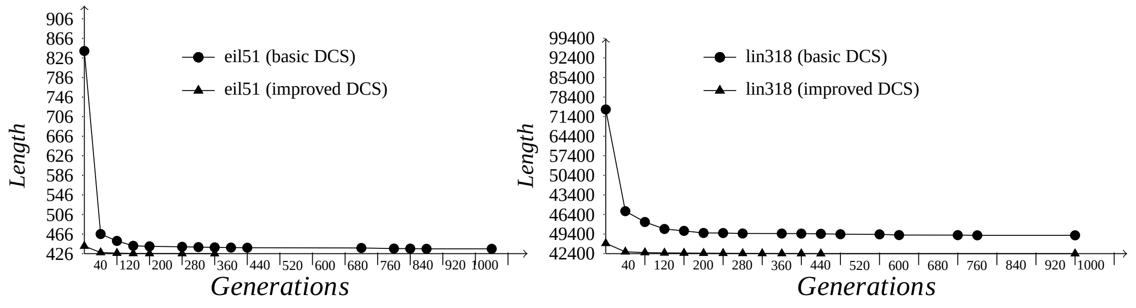

Figure 3: Average length of best solution of 10 runs for eil51(opt=426) and lin318(opt=42029).

Table 2 and **Table 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:

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

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

instance		Basic DCS						Improved DCS					
	opt	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	7542	**7542**	7836.4	8356	3.90	**0.00**	2.24	**7542**	**7542**	**7542**	**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	108159	108469	111225.6	116278	2.83	0.28	4.22	**108159**	**108159**	**108159**	**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	21282	21515	22419.96	23444	5.34	1.09	6.53	**21282**	**21282**	**21282**	**0.00**	**0.00**	2.70
kroB100	22141	22335	23417.06	25177	5.76	0.87	6.69	**22141**	22141.53	22157	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	118282	120858	127832.23	159788	8.07	2.17	9.96	**118282**	118359.63	118730	0.06	**0.00**	25.50
ch130	6110	6309	6661.23	7883	9.02	3.25	10.60	**6110**	6135.96	6174	0.42	**0.00**	23.12
ch150	6528	6913	7234.9	8064	10.82	5.89	18.06	**6528**	6549.90	6611	0.33	**0.00**	27.74
kroA150	26524	27328	28928.83	32786	9.06	3.03	14.24	**26524**	26569.26	26767	0.17	**0.00**	31.23
kroA200	29368	30641	32896.03	37993	12.01	4.33	23.43	29382	29446.66	29886	0.26	0.04	62.08
lin318	42029	44278	49623.96	69326	18.07	5.35	72.55	42125	42434.73	42890	0.96	0.22	156.17

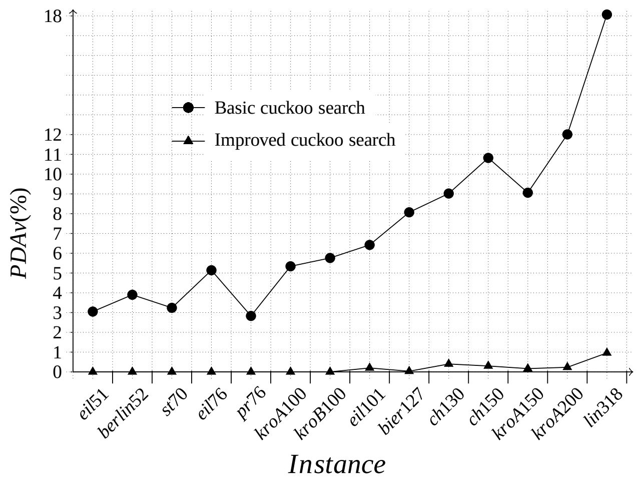

Figure 4: PDAv(%)(over 30 runs) for 14 TSPLIB instances

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

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	**7542**	**7542**	**7542**	**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	108159	**108159**	**108159**	**108159**	**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	21282	**21282**	**21282**	**21282**	0.00	**0.00**	**0.00**	**30/30**	2.70
kroB100	22141	**22141**	22157	22141.53	2.87	**0.00**	**0.00**	30/29	8.74
kroC100	20749	**20749**	**20749**	**20749**	**0.00**	**0.00**	**0.00**	**30/30**	3.36
kroD100	21294	**21294**	21389	21304.33	21.79	0.04	**0.00**	30/19	8.35
kroE100	22068	**22068**	22121	2281.26	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	14379	**14379**	**14379**	**14379**	**0.00**	**0.00**	**0.00**	**30/30**	5.01
pr107	44303	**44303**	44358	44307.06	12.90	0.00	**0.00**	30/27	12.89
pr124	59030	**59030**	**59030**	**59030**	**0.00**	**0.00**	**0.00**	**30/30**	3.36
bier127	118282	**118282**	118730	118359.63	12.73	0.06	**0.00**	30/18	25.50
ch130	6110	**6110**	6174	6135.96	21.24	0.42	**0.00**	28/7	23.12
pr136	96772	96790	97318	97009.26	134.43	0.24	0.01	30/0	35.82
pr144	58537	**58537**	**58537**	**58537**	**0.00**	**0.00**	**0.00**	**30/30**	2.96
ch150	6528	**6528**	6611	6549.9	20.51	0.33	**0.00**	29/10	27.74
kroA150	26524	**26524**	26767	26569.26	56.26	0.17	**0.00**	30/7	31.23
kroB150	26130	**26130**	26229	26159.3	34.72	0.11	**0.00**	30/5	33.01
pr152	73682	**73682**	**73682**	**73682**	**0.00**	**0.00**	**0.00**	**30/30**	14.86
rat195	2323	2324	2357	2341.86	8.49	0.81	0.04	20/0	57.25
d198	15780	15781	15852	15807.66	17.02	0.17	0.00	30/0	59.95
kroA200	29368	29382	29886	29446.66	95.68	0.26	0.04	29/0	62.08
kroB200	29437	29448	29819	29542.49	92.17	0.29	0.03	28/0	64.06
ts225	126643	**126643**	126810	126659.23	44.59	0.01	**0.00**	30/26	47.51
tsp225	3916	**3916**	3997	3958.76	20.73	1.09	**0.00**	9/1	76.16
pr226	80369	**80369**	80620	80386.66	60.31	0.02	**0.00**	30/19	50.00
gil262	2378	2382	2418	2394.5	9.56	0.68	0.16	22/0	102.39
pr264	49135	**49135**	49692	49257.5	159.98	0.24	**0.00**	28/13	82.93
a280	2579	**2579**	2623	2592.33	11.86	0.51	**0.00**	25/4	115.57
pr299	48191	48207	48753	48470.53	131.79	0.58	0.03	27/0	138.20
lin318	42029	42125	42890	42434.73	185.43	0.96	0.22	15/0	156.17
rd400	15281	15447	15704	15533.73	60.56	1.65	1.08	0/0	264.94
fl417	11861	11873	11975	11910.53	20.45	0.41	0.10	30/0	274.59
pr439	107217	107447	109013	107960.5	438.15	0.69	0.21	22/0	308.75
rat575	6773	6896	7039	6956.73	35.74	2.71	1.81	0/0	506.67
rat783	8806	9043	9171	9109.26	38.09	3.44	2.69	0/0	968.66
pr1002	259045	266508	271660	268630.03	1126.86	3.70	2.88	0/0	1662.61
nrw1379	56638	58951	59837	59349.53	213.89	4.78	4.08	0/0	3160.47

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 '$C_{1\%}/C_{opt}$' 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 which is that of the best known solution. The results presented in **Table 3** show that improved DCS can provide good results in reasonable time.

Table 4: Comparison of experimental results of Improved DCS with **GSA-ACS-PSOT** [20]

instance		GSA-ACS-PSOT			Improved DCS		
	opt	best	average	SD	best	average	SD
eil51	426	427	427.27	0.45	**426**	**426**	**0.00**
berlin52	7542	**7542**	**7542.00**	**0.00**	**7542**	**7542**	**0.00**
eil76	538	**538**	540.20	2.94	**538**	538.03	0.17
kroA100	21282	**21282**	21370.47	123.36	**21282**	**21282**	**0.00**
kroB100	22141	**22141**	22282.87	183.99	**22141**	22141.53	2.87
kroC100	20749	**20749**	20878.97	158.64	**20749**	**20749**	**0.00**
kroD100	21294	21309	21620.47	226.60	**21294**	21304.33	21.79
kroE100	22068	**22068**	22183.47	103.32	**22068**	2281.26	18.50
eil101	629	630	635.23	3.59	**629**	630.43	1.14
lin105	14379	**14379**	14406.37	37.28	**14379**	**14379**	**0.00**
bier127	118282	**118282**	119421.83	580.83	**118282**	118359.63	12.73
ch130	6110	6141	6205.63	43.70	**6110**	6135.96	21.24
ch150	6528	**6528**	6563.70	22.45	**6528**	6549.90	20.51
kroA150	26524	**26524**	26899.20	133.02	**26524**	26569.26	56.26
kroB150	26130	**26130**	26448.33	266.76	**26130**	26159.3	34.72
kroA200	29368	29383	29738.73	356.07	29382	29446.66	95.68
kroB200	29437	29541	30035.23	357.48	29448	29542.49	92.17
lin318	42029	42487	43002.09	307.51	42125	42434.73	185.43

In **Tables 4** and **5**, the experiments results of the improved DCS algorithm compared with the both methods GSA-ACS-PSOT and DPSO, are shown. The results of these two methods are directly summarizes from original papers [20] and [19]. 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/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 **Figure 5**, the lower curve that is associated with improved DCS algorithm is the better. This is explained basically by the

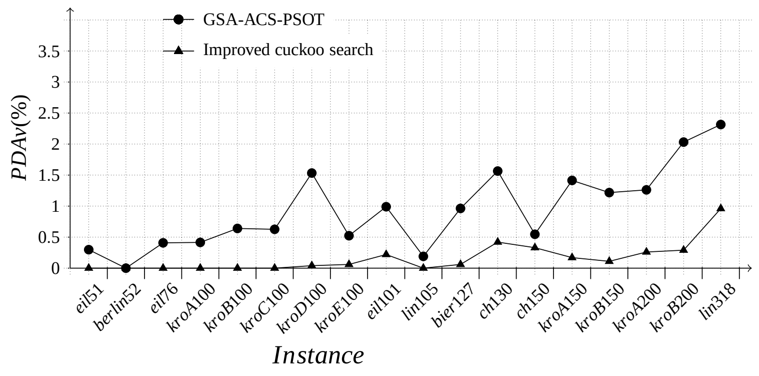

Figure 5: PDAv(%)(over 30 runs) for 18 TSPLIB instances

strengths of CS: a good balance between intensification and diversification, an intelligent use of Lévy flights and the reduced number of parameters. Its also explained by the structure improved of the population which uses a variety of cuckoos search methods. One of the advantage of DCS is a relative independence of cuckoos in their search from the best position. So, more likely to find good solutions in areas unexplored by metaheuristics which take the best position as a starting point to look for other much better.

Table 5: Comparison of experimental results of Improved DCS with **DPSO** [19].

instance		DPSO			Improved DCS		
	opt	best	worst	PDAv(%)	best	worst	PDAv(%)
eil51	426	427	452	2.57	**426**	**426**	**0.00**
berlin52	7542	**7542**	8362	3.84	**7542**	**7542.0**	**0.00**
st70	675	**675**	742	3.34	**675**	**675**	**0.00**
pr76	108159	108280	124365	3.81	**108159**	**108159**	**0.00**
eil76	538	546	579	4.16	**538**	539	0.00

6. Conclusion

In this paper, we have extended and improved the cuckoo search (CS) via Lévy flights by reconstructing the population and introducing a new cuckoo category which is more intelligent. Improved CS is adapted to solve the symmetric traveling salesman problem (TSP). This adaptation is based on the study of interpretation of the terminology used in CS and in its inspiration source. 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 [20] and DPSO [19]. The results of the comparison have shown that Discrete CS outperforms all other two 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 the new category of the cuckoo relative to the best solution in its search for a new solution can probably provide better strategy in generating new solutions than the simple use of the current solution, thus leading to a more efficient algorithm.

The use of local perturbations introduced in our proposed Discrete CS can provide more abstraction 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 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 efficiently solve very complex problems by algorithms with apparent simplicity. We also want to develop new algorithms that are truly intelligent. New algorithms should be more controllable and less complex, compared with contemporary metaheuristics. In the end, some truly efficient and intelligent algorithms may emerge to solve NP-hard problems in ever-increasing efficient way. At least, some seemingly intractable problems can be solved by new metaheuristic algorithms.

References

[1] S. Arora, Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems, Journal of the ACM (JACM) 45 (1998) 753–782.

[2] J. K. Lenstra, A. H. G. Rinnooy Kan, Some simple applications of the travelling salesman problem, Operational Research Quarterly (1975) 717–733.

[3] G. Reinelt, The traveling salesman: computational solutions for TSP applications, Springer-Verlag, 1994.

[4] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys (CSUR) 35 (2003) 268–308.

[5] F. Glover, G. A. Kochenberger, Handbook of metaheuristics, Springer, 2003.

[6] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Neural Networks, 1995. Proceedings., IEEE International Conference on, volume 4, IEEE, pp. 1942–1948.

[7] D. Teodorovic, P. Lucic, G. Markovic, M. D. Orco, Bee colony optimization: principles and applications, in: Neural Network Applications in Electrical Engineering, 2006. NEUREL 2006. 8th Seminar on, IEEE, pp. 151–156.

[8] A. Mucherino, O. Seref, Monkey search: a novel metaheuristic search for global optimization, in: Data Mining, Systems Analysis, and Optimization in Biomedicine(AIP Conference Proceedings Volume 953), volume 953, American Institute of Physics, 2 Huntington Quadrangle, Suite 1 NO 1, Melville, NY, 11747-4502, USA,, pp. 162–173.

[9] Z. W. Geem, J. H. Kim, et al., A new heuristic optimization algorithm: harmony search, Simulation 76 (2001) 60–68.

[10] X. S. Yang, Firefly algorithms for multimodal optimization, Stochastic algorithms: foundations and applications (2009) 169–178.

[11] H. Shah-Hosseini, The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm, International Journal of Bio-Inspired Computation 1 (2009) 71–79.

[12] X. S. Yang, S. Deb, Cuckoo search via lévy flights, in: Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on, IEEE, pp. 210–214.

[13] E. D. Taillard, L. M. Gambardella, M. Gendreau, J. Y. Potvin, Adaptive memory programming: A unified view of metaheuristics, European Journal of Operational Research 135 (2001) 1–16.

[14] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, Evolutionary Computation, IEEE Transactions on 1 (1997) 67–82.

[15] E. Bonomi, J. L. Lutton, The n-city travelling salesman problem: Statistical mechanics and the metropolis algorithm, SIAM review (1984) 551–568.

[16] M. Malek, M. Guruswamy, M. Pandya, H. Owens, Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem, Annals of Operations Research 21 (1989) 59–84.

[17] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, S. Dizdarevic, Genetic algorithms for the travelling salesman problem: A review of representations and operators, Artificial Intelligence Review 13 (1999) 129–170.

[18] M. Dorigo, L. M. Gambardella, et al., Ant colonies for the travelling salesman problem, BioSystems 43 (1997) 73–82.

[19] 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.

[20] S. M. Chen, C. Y. Chien, Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques, Expert Systems with Applications (2011).

[21] R. F. Abdel-Kader, Fuzzy Particle Swarm Optimization with Simulated Annealing and Neighborhood Information Communication for Solving TSP, International Journal of Advanced Computer Science and Applications 2 (2011).

[22] X. S. Yang, S. Deb, Engineering optimisation by cuckoo search, International Journal of Mathematical Modelling and Numerical Optimisation 1 (2010) 330–343.

[23] G. Reinelt, Tsplib—a traveling salesman problem library, ORSA Journal on Computing 3 (1991) 376–384.

[24] R. B. Payne, M. D. Sorenson, The cuckoos, volume 15, Oxford University Press, USA, 2005.

[25] M. F. Shlesinger, G. M. Zaslavsky, U. Frisch, Lévy flights and related topics in physics:(Nice, 27-30 June 1994), Springer, 1995.

[26] C. T. Brown, L. S. Liebovitch, R. Glendon, Lévy flights in dobe ju/'hoansi foraging patterns, Human Ecology 35 (2007) 129–138.

[27] G. A. Croes, A method for solving traveling-salesman problems, Operations Research (1958) 791–812.

[28] O. Martin, S. W. Otto, E. W. Felten, Large-step markov chains for the traveling salesman problem, Complex Systems 5 (1991) 299–326.