Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm

Amira Gherboudj*, Abdesslem Layeb and Salim Chikhi

MISC Laboratory, Computer Science Department, Mentouri University Constantine, Route Ain El Bey, Constantine 25017, Algeria
E-mail: a.gherboudj@umc.edu.dz
E-mail: layeb@umc.edu.dz
E-mail: chikhi@umc.edu.dz
*Corresponding author

Abstract: Cuckoo search (CS) is one of the most recent population-based meta-heuristics. CS algorithm is based on the cuckoo’s behaviour and the mechanism of Lévy flights. Unfortunately, the standard CS algorithm is proposed only for continuous optimisation problems. In this paper, we propose a discrete binary cuckoo search (BCS) algorithm in order to deal with binary optimisation problems. To get binary solutions, we have used a sigmoid function similar to that used in the binary particle swarm optimisation algorithm. Computational results on some knapsack problem instances and multidimensional knapsack problem instances show the effectiveness of the proposed algorithm and its ability to achieve good quality solutions.

Keywords: combinatorial optimisation; evolutionary computation; cuckoo search; CS; binary cuckoo search; BCS; knapsack problem; KP; multidimensional knapsack problem; MKP.


Biographical notes: Amira Gherboudj received her Master’s degree from Mentouri University of Constantine, Algeria in 2009 and she is a PhD student from the year 2010 in Mentouri University of Constantine, Algeria. Her current research interests include optimisation methods and their applications to solve several combinatorial optimisation problems.

Abdesslem Layeb is an Associate Professor in the Department of Computer Science at the Mentouri University of Constantine, Algeria. He is a member of MISC Laboratory. He received his PhD in Computer Science from the University Mentouri of Constantine, Algeria. He is interested by combinatorial optimisation methods and their applications to solve several problems from different domains like bioinformatics, imagery, formal methods, etc.

Salim Chikhi received his PhD in Computer Science from the University of Constantine, in 2005. He is currently a Professor at the Mentouri University, Constantine, Algeria. He is the Leader of the SCAL team of MISC Laboratory. His research areas include soft computing and artificial life techniques and their application in several domains.

1 Introduction

The combinatorial optimisation plays a very important role in operational research, discrete mathematics and computer science. The aim of this field is to solve several combinatorial optimisation problems that are difficult to solve. Several methods were developed to solve such problems that can be classified into two classes (Jourdan et al., 2009). Exact methods, like branch and bound, dynamic programming, can give the exact solutions. However, in the worst case, the computation time required for the execution of such methods, increases exponentially with the size of the instance to solve. The second class contains meta-heuristic methods which are able to find sub-optimal solution within a reasonable computation time compared with the exact methods.

Evolutionary computation has been proven to be an effective way to solve complex engineering problems. It presents many interesting features such as adaptation, emergence and learning (Fogel, 2001). Artificial neural networks, genetic algorithms and swarm intelligence are examples of bio-inspired systems used to this end. In recent years, optimising by swarm intelligence has become a research interest to many research scientists of evolutionary computation fields. There are many algorithms-based swarm intelligence like ant colony optimisation (Dorigo and Gambardella, 1997; Hu et al., 2008), eco-systems...
optimisation (Nabti and Meshoul, 2009; Zheng, 2010), etc. The main algorithm for swarm intelligence is particle swarm optimisation (PSO) (Kennedy and Eberhart, 1997; Eberhart et al., 2001), which is inspired by the paradigm of birds grouping. PSO was used successfully in various hard optimisation problems. The simplicity of implementation and use are the main features of PSO compared to other evolutionary computing algorithms. In fact, the updating mechanism in the algorithm relies only on two simple PSO self-updating equations.

One of the most recent variant of PSO algorithm is cuckoo search (CS) algorithm. CS is an optimisation algorithm developed by Yang and Deb (2009, 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). Some bird’s host can involve direct conflicts with the intruding cuckoos. For example, if a bird’s host discovers that the eggs are strange eggs, it will either throw these alien eggs away or simply abandon its nest and build a new nest elsewhere. The cuckoo’s behaviour and the mechanism of Lévy flights (Payne et al., 2005; Pavlyukevich, 2007) have leading to design an efficient inspired algorithm performing optimisation search. The recent applications of CS for optimisation problems have shown its promising effectiveness. For example, a promising discrete CS algorithm is recently proposed to deal with nurse scheduling problem (Tein and Ramli, 2010).

The aim of this paper is two-fold: first, is to propose a discrete binary version of CS algorithm that we have called binary cuckoo search (BCS) to cope with binary optimisation problems. The main difference between the original version of CS algorithm and the proposed discrete binary version is that, in the original CS, the solution is composed of a set of real numbers, while in the proposed discrete binary version; the solution is composed of a set of bits. The main feature of our approach consists in using a sigmoid function and probability model in order to generate binary values. The second aim of this paper is to prove that the CS algorithm is effective in dealing with binary combinatorial optimisation problems. To validate and prove the performance and the effectiveness of our BCS algorithm, we have tested it on some knapsack problem (KP) instances and multidimensional knapsack problem (MKP) instances. Experimental results show the effectiveness of the proposed algorithm and its ability to achieve good quality solutions.

The remainder of this paper is organised as follows. Section 2 presents the KPs formulation. An overview of the CS algorithm is presented in Section 3. In Section 4, the proposed algorithm is described. Experimental results are discussed in Section 5, and a conclusion is provided in Section 6 of this paper.

2 Knapsack problems

The KP is a NP-hard problem (Pisinger, 2005). Numerous practical application of the KP can be found in many areas involving resource distribution, investment decision making, budget controlling, project selection and so on. The KP can be defined as follows: Assuming that we have a knapsack with maximum capacity C and a set of N objects. Each object i has a profit pi and a weight wi. The problem consists to select a subset of objects that maximise the knapsack profit without exceeding the capacity of the knapsack. The problem can be formulated as:

\[
\text{Maximise } \sum_{i=1}^{N} p_i x_i \quad (1)
\]

Subject to \( \sum_{i=1}^{N} w_i x_i \leq C \quad x_i \in \{0,1\} \quad (2)\)

Many variants of the KP were proposed in the literature including the MKP. MKP is an important issue in the class of KP. It is a NP-hard problem (Chu and Beasley, 1998). In the MKP, each item has a profit pi like in the simple KP. However, instead of having a single knapsack to fill, we have a number M of knapsack of capacity Cj (j = 1, ..., M). Each x_i has a weight w_ij that depends of the knapsack j (example: an object can have a weight 3 in knapsack 1, 5 in knapsack 2, etc.). A selected object must be in all knapsacks. The objective in MKP is to find a subset of objects that maximise the total profit without exceeding the capacity of all dimensions of the knapsack. MKP can be stated as follows (Angelelli et al., 2010):

\[
\text{Maximise } \sum_{i=1}^{N} p_i x_i \quad (3)
\]

Subject to \( \sum_{i=1}^{N} w_{ij} x_i \leq C_j \quad j = 1, ..., M \quad (4)\)

The MKP can be used to formulate many industrial problems such as the capital budgeting problem, allocating processors and databases in a distributed computer system, cutting stock, project selection and cargo loading problems (Vasquez and Vimont, 2005).

Clearly, there are 2^N potential solutions for these problems. It is obviously that KP and its variants are combinatorial optimisation problems. Several techniques have been proposed to deal with KPs (Pisinger, 2005). However, it appears to be impossible to obtain exact solutions in polynomial time. The main reason is that the required computation grows exponentially with the size of the problem. Therefore, it is often desirable to find near optimal solutions to these problems. In this paper, we are interested by applying BCS algorithm to solve these problems.

3 Cuckoo search

In order to solve complex problems, ideas gleaned from natural mechanisms have been exploited to develop heuristics. Nature inspired optimisation algorithms has been extensively investigated during the last decade paving the way for new computing paradigms such as neural networks,
evolutionary computing, swarm optimisation, etc. The ultimate goal is to develop systems that have ability to learn incrementally, to be adaptable to their environment and to be tolerant to noise. One of the recent developed bioinspired algorithms is the CS (Yang and Deb, 2010) which is based on life style of cuckoo bird. Cuckoos use an aggressive strategy of reproduction that involves the female hack nests of other birds to lay their eggs fertilised. Sometimes, the egg of cuckoo in the nest is discovered and the hacked birds discard or abandon the nest and start their own brood elsewhere. The CS proposed by Yang and Deb (2009, 2010) in 2009 is based on the following three idealised rules:

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest.
- The best nests with high quality of eggs (solutions) will carry over to the next generations.
- The number of available host nests is fixed, and a host can discover an alien egg with a probability $p_a \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest so as to build a completely new nest in a new location.

The last assumption can be approximated by a fraction $p_a$ of the $n$ nests being replaced by new nests (with new random solutions at new locations).

The generation of new solutions $x^{t+1}_i$ is done by using Lévy flights [equation (5)]. Lévy flights essentially provide a random walk while their random steps are drawn from a Lévy distribution for large steps which has an infinite variance with an infinite mean [equation (6)]. Here, the consecutive jumps/steps of a cuckoo essentially form a random walk process which obeys a power-law step-length distribution with a heavy tail (Yang and Deb, 2010).

$$x^{t+1}_i = x^t_i + \alpha \odot \text{Lévy}(\lambda)$$  \hspace{1cm} (5)

$$\text{Lévy} \odot u = t^{-\alpha}$$  \hspace{1cm} (6)

where $x^{t+1}_i$ and $x^t_i$ represent the solution $i$ at times $t+1$ and $t$, respectively. $\alpha > 0$ is the step size which should be related to the scales of the problem of interest. Generally, we take $\alpha = O(1)$. The product $\odot$ means entry-wise multiplications. This entry-wise product is similar to those used in PSO, but here the random walk via Lévy flights is more efficient in exploring the search space as its step length is much longer in the long run.

The main characteristic of CS algorithm is its simplicity. In fact, comparing with other population or agent-based meta-heuristic algorithms such as PSO and harmony search, there are few parameters to set. The applications of CS into engineering optimisation problems have shown its encouraging efficiency. For example, a promising discrete CS 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, 2011). The basic steps of CS algorithm are presented in a pseudo code shown in Figure 1.

**Figure 1** Cuckoo search

- **Objective function** $f(x) = (x_1, \ldots, x_d)^T$.
- **Initial a population** of $N$ host nests $x_i (i = 1, 2, \ldots, N)$;
- **while** ($t < \text{MaxGeneration}$) or (stop criterion);
- Get a cuckoo (say $i$) randomly by Lévy flights;
- Evaluate its quality/fitness $F_i$;
- Choose a nest among $n$ (say $j$) randomly;
- if ($F_i > F_j$),
  - Replace $j$ by the new solution;
- end
- Abandon a fraction ($p_a$) of worse nests
- build new ones at new locations via Lévy flights;
- Keep the best solutions (or nests with quality solutions);
- Rank the solutions and find the current best;
- **end while**

*Source: Yang and Deb (2010)*

## 4 The proposed algorithm (BCS)

Optimisation problems can be classed into two main classes: continuous optimisation problems and discrete optimisation problems. In continuous optimisation problems, the solution is represented by a set of real numbers. However, in discrete optimisation problems, the solution is represented by a set of integer numbers. Discrete binary optimisation problems are a sub-class of the discrete optimisation problems class in which a solution is represented by a set of bits. Many optimisation problems can be modelled as a discrete binary search space such as, flowshop scheduling problem (Liao et al., 2007), job-shop scheduling problem (Pongchairerks, 2009), routing problems (Zhan and Zhang, 2009), KP (Gherboudj and Chikhi, 2011) and its variants such as the MKP (Kong and Tian, 2006), the quadratic KP (Julstrom, 2005), the quadratic multiple KP (Singh and Baghel, 2007) and so one.

The original CS algorithm is based on Lévy flights, and it operates in continuous search space. Consequently, CS algorithm gives a set of real numbers as a solution of the handled problem. However, a binary optimisation problem needs a binary solution and the real solutions are not acceptable, because they are considered as illegal solutions. Therefore, the solutions must be converted from real values to binary values. In the aim to extend the CS algorithm to discrete binary areas, we propose in this paper a discrete binary version of CS that we called BCS. In BCS algorithm, the problem solution must be represented by a set of bits.

The BCS architecture contains two essential modules. The first module contains the main binary cuckoo dynamics. This module is composed of two main operations: Lévy flights and binary solution representation (BSR) operations. These two operations combine the CS algorithm and the
sigmoid function to obtain a BCS. In the first operation, Lévy flight is used to get a new cuckoo. In the second operation, the sigmoid function is used to calculate the flipping chances of each cuckoo. Then, the binary value of each cuckoo is computed using their flipping chances. The second module contains the objective function and the selection operator. The selection operator is similar to the elitism strategy used in genetic algorithms. Figure 2 shows the flowchart of the proposed architecture. In the following, we will explain in more detail the BCS architecture.

Figure 2 Flowchart of the BCS architecture

4.1 Binary solution representation

The main objective of the BCS algorithm is to deal with the binary optimisation problems. Therefore, the main feature of BCS algorithm is to transform a solution $x_i$ from real area to binary area, and consequently obtain a BSR $x'_i$. To meet this need, we propose to constrain the solution $x_i$ in the interval $[0, 1]$ using the Sigmoid function as follows:

$$S(x_i) = \frac{1}{1 + e^{-x_i}}$$

where $S(x_i)$ is the flipping chance of bit $x'_i$. It represents the probability of bit $x'_i$ takes the value 1.

To obtain the binary solution $x'_i$, we have to generate a random number from the interval $[0, 1]$ for each dimension $i$ of the solution $x$ and compare it with the flipping chance $S(x_i)$ as mentioned below in equation (8). If the random number is lower than the flipping chance of bit $x'_i$, than $x'_i$ takes the value 1. Otherwise, $x'_i$ takes the value 0.

$$x'_i = \begin{cases} 1 & \text{if } \gamma < S(x_i), \ r \in [0,1] \\ 0 & \text{otherwise} \end{cases}$$

Consequently, having a solution $x_i$ encoded as a set of real numbers, the sigmoid function is used to transform the solution $x_i$ into a set of probabilities that represents the chance for bit $i$ to be flipping. The flipping chance is then used to compute the binary solution $x'_i$.

For example, assuming that we have a problem with $N = 6$ objects i.e., problem size is 6. $x_i = [2.314, -3.4510, 1.9412, 0.3498, 1.9412, 3.8461]$ is the obtained solution with original CS algorithm; $S(x_i) = [0.9100, 0.0307, 0.8745, 0.5866, 0.1732, 0.9791]$ is the set of flipping chances (probabilities) of each bit $x'_i$ calculated by the Sigmoid function; Then the chance for each bit to be flipping is: chance $i = [91\% , 3.07\% , 87.45\% , 58.66\% , 17.32\% , 97.91\% ]$; In order to obtain a binary solution, we must generate six random numbers $r$ from the interval $[0, 1]$, for example, $r = [0.8147, 0.1270, 0.9134, 0.6324, 0.0975, 0.5469]$.

Following the defined instructions in equation (8), the first, fifth and sixth bits of the binary solution take the value 1 because there flipping chances (0.9100, 0.1732, 0.9791, respectively) are higher than the generated random numbers (0.8147, 0.0975, 0.5469, respectively). However, the second, third and fourth bits take the value 0, because there flipping chances (0.0307, 0.8745, 0.5866, respectively) are lower than the generated random numbers (0.1270, 0.9134, 0.6324, respectively). Thus, $x'_i = [1, 0, 0, 1, 1]$ is the BSR. Which mean that the selected objects are 1, 5 and 6. A pseudo code of the BSR algorithm is shown in Figure 3.

Figure 3 BSR algorithm

Input: Real solution representation $x_i$

Output: s $x'_i$

For (i = 1 to (problem size)) {
    $S(x_i) = \frac{1}{1 + e^{-x_i}}$
    If (random number $\gamma < S(x_i)$)
        $x'_i = 1$
    Otherwise
        $x'_i = 0$
}

4.2 Outlines of the proposed algorithm

Now, we describe how the CS algorithm scheme including the sigmoid function has been embedded within a new population-based meta-heuristics, and how it can find a binary solution to a binary optimisation problem. The powerful of BCS algorithm is lies essentially in three
components: selection of the best solution, local exploitation by random walk, and global exploration by randomisation via Lévy flights. A pseudo code of BCS algorithm is shown in Figure 4.

Figure 4 Binary cuckoo search

Input: $N$ and $p_a$

Output: the last best solution

Objective function $f(x)$, $x = (x_1, \ldots, x_d)^T$;

Initial a population of $N$ host nests $x_i$ ($i = 1, 2, \ldots, N$);

while ($t <$ MaxGeneration) or (stop criterion)

• Get a cuckoo (say $i$) randomly by Lévy flights;

• Get its binary representation by BSR algorithm;

• Evaluate its quality/fitness $F_i$;

• Choose a nest among $N$ (say $j$) randomly;

• Get its binary representation by BSR algorithm;

• Evaluate its quality/fitness $F_j$;

• if ($F_i > F_j$){
    Replace $j$ by the new solution;
}

• Get the binary representation by BSR algorithm of all the nests and evaluate their qualities;

• Abandon a fraction ($p_a$) of worse nests;

• Build new ones at new locations via Lévy flights;

• Get there binary representation by BSR algorithm and evaluate their qualities;

• Keep the best solutions (or nests with quality solutions);

• Rank the solutions and find the current best;

}

Like any other population-based meta-heuristics, the first step in the BCS algorithm involves setting the parameters for the algorithm. The main advantage of the BCS algorithm is that there are fewer parameters to be set in this algorithm than in PSO. In BCS, there are essentially two main parameters to initialise, population size $N$ and a fraction $p_a$ of the worst nests to be rejected and replaced. In the second step, a swarm of $N$ host nests is created at random positions to represent some possible solutions. However, it should be noted that in order to reduce the convergence time, it is recommended to start with a diverse population containing both good and bad solutions. For this purpose, we can use some heuristics to build good initial solutions. The algorithm progresses through a number of generations according to the BCS dynamics. During each iteration, the following main tasks are performed. A new cuckoo is built using the Lévy flights operator. The Lévy flight provides a random walk that consists of taking successive random steps in which the step lengths are distributed according to a heavy tailed probability distribution. The next step is to evaluate the current cuckoo. For that, we apply the BSR algorithm (Figure 3) to get a binary solution which represents a potential solution for the binary optimisation problems. After this step, we replace some worst nests by the current cuckoo if it is better, or by new random nests generated by Lévy flights. The selection phase in BCS of the best nests or solutions is comparable to some form of elitism selection used in genetic algorithms, which ensures that best solution is kept always in the next iteration. Finally, the global best solution is then updated if a better one is found and the whole process is repeated until reaching a stopping criterion.

5 Experimental results and discussion

The proposed BCS algorithm was implemented in MATLAB 7.3. Several experiments were performed to assess the efficiency and performance of our algorithm (BCS). In the first experiment, we have tested and compared our BCS algorithm with a harmony search algorithm (NGHS) on some small KP instances taken from Zou et al. (2011). In the second experiment, we have used some KP instances used in Gherboudj and Chikhi (2011) to test and compare the BCS algorithm with the standard binary particle swarm optimisation (BPSO) algorithm (Kennedy and Eberhart, 1997) which has a common point with the proposed BCS algorithm. In fact, the two algorithms (BCS and the standard BPSO) used the sigmoid function to generate the binary solution. The used instances are seven different instances with different problem sizes, in which the weights and profits are selected randomly. The different problem sizes $N$ are 120, 200, 500, 700, 900, 1,000 and 2,000. In these instances, the knapsack capacity is calculated by using the following formula (Gherboudj and Chikhi, 2011). The factor 3/4 indicates that about 75% of items are in the optimal solution.

$$C = \frac{3}{4} \sum_{i=0}^{N} w_i$$

In the third experiment, we have evaluated the performance of our BCS algorithm on some MKP benchmarks taken from OR-Library. First, we have tested the BCS algorithm on some small size MKP instances taken from seven benchmarks named mknaps1. Moreover, we have tested the BCS algorithm on some big size MKP instances taken from benchmarks named mknapsb1 and mknapsb2. We have used five tests of the benchmarks mknapsb1 (5,100) which have five constraints and 100 items, and we have used five tests of the benchmarks mknapsb4 (10,100) which have ten constraints and 100 items. The obtained results are compared with the exact solution (best known), the obtained solution by the standard binary PSO with penalty function technique (denoted as PSO-P) (Kong and Tian, 2006) and the quantum inspired cuckoo search (denoted as QICSA) (Bayeb, 2011). In all experiments, we have set the parameters of BCS as follows: $p_a = 25\%$, the number of cuckoos is 40. Finally, statistical tests of Friedman were carried out to test the significance of the difference in the accuracy of each method in this experiment.

Table 1 shows the experimental results of our BCS algorithm and the harmony search algorithm (NGHS) on ten
KP tests with different sizes. The first column, indicates the instance name, the second column indicates the problem size, i.e., number of objects. The third column indicates the obtained results by the BCS algorithm and the last column indicates the obtained results by the NGHS algorithm. Observation of the presented results in Table 1 indicates that the proposed discrete BCS algorithm performs well than the NGHS algorithm in two tests (f6 and f8) and they have the same results in the other instances.

Table 1

<table>
<thead>
<tr>
<th>Test</th>
<th>Size</th>
<th>BCS</th>
<th>NGHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>10</td>
<td>295</td>
<td>295</td>
</tr>
<tr>
<td>f2</td>
<td>20</td>
<td>1,024</td>
<td>1,024</td>
</tr>
<tr>
<td>f3</td>
<td>4</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>f4</td>
<td>4</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>f5</td>
<td>15</td>
<td>481.0694</td>
<td>481.0694</td>
</tr>
<tr>
<td>f6</td>
<td>10</td>
<td>52</td>
<td>50</td>
</tr>
<tr>
<td>f7</td>
<td>7</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>f8</td>
<td>23</td>
<td>9,767</td>
<td>9,761</td>
</tr>
<tr>
<td>f9</td>
<td>5</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>f10</td>
<td>20</td>
<td>1,025</td>
<td>1,025</td>
</tr>
</tbody>
</table>

Table 2 shows the experimental results of BCS and BPSO algorithms on KP random instances. The first column represents the problem size (i.e., instance). The second and third columns represent the obtained results by the BCS and the BPSO algorithms respectively. The purpose of this experiment was to compare the performance of our algorithm and that of BPSO algorithm. The presented results show that the performance of the BCS algorithm outperforms the BPSO algorithm performance, especially with big instances. Indeed, the statistical Freidman test (Figure 5) indicates that the BCS algorithm is better than the BPSO algorithm in this experiment.

Figure 5

Friedman test compares BCS and BPSO on knapsack tests

Table 2

<table>
<thead>
<tr>
<th>Instance</th>
<th>BCS</th>
<th>BPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>4,210</td>
<td>4,296</td>
</tr>
<tr>
<td>200</td>
<td>6,989</td>
<td>7,456</td>
</tr>
<tr>
<td>500</td>
<td>16,364</td>
<td>13,116</td>
</tr>
<tr>
<td>700</td>
<td>22,734</td>
<td>18,276</td>
</tr>
<tr>
<td>900</td>
<td>29,322</td>
<td>22,857</td>
</tr>
<tr>
<td>1,000</td>
<td>31,679</td>
<td>24,933</td>
</tr>
<tr>
<td>2,000</td>
<td>61,497</td>
<td>47,674</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Instance</th>
<th>No.</th>
<th>n</th>
<th>m</th>
<th>Best known</th>
<th>BCS solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>10</td>
<td>3,800</td>
<td>3,800</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>8,706.1</td>
<td>8,706.1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>10</td>
<td>4,015</td>
<td>4,015</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>10</td>
<td>6,120</td>
<td>6,120</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>10</td>
<td>12,400</td>
<td>12,400</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>39</td>
<td>5</td>
<td>10,618</td>
<td>10,618</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>50</td>
<td>5</td>
<td>16,537</td>
<td>16,537</td>
<td></td>
</tr>
</tbody>
</table>

Table 4

<table>
<thead>
<tr>
<th>Benchmark name</th>
<th>Problem size</th>
<th>Best known</th>
<th>BCS solution</th>
<th>PSO-P solution</th>
<th>QICSA solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>mknaps1</td>
<td>5.100.00</td>
<td>24,381</td>
<td>23,510</td>
<td>22,525</td>
<td>23,416</td>
</tr>
<tr>
<td>mknaps1</td>
<td>5.100.01</td>
<td>24,274</td>
<td>22,938</td>
<td>22,244</td>
<td>22,880</td>
</tr>
<tr>
<td>mknaps1</td>
<td>5.100.02</td>
<td>23,551</td>
<td>22,518</td>
<td>21,822</td>
<td>22,525</td>
</tr>
<tr>
<td>mknaps1</td>
<td>5.100.03</td>
<td>23,534</td>
<td>22,677</td>
<td>22,057</td>
<td>22,727</td>
</tr>
<tr>
<td>mknaps1</td>
<td>5.100.04</td>
<td>23,991</td>
<td>23,232</td>
<td>22,167</td>
<td>22,854</td>
</tr>
<tr>
<td>mknaps1</td>
<td>10.100.00</td>
<td>23,064</td>
<td>21,841</td>
<td>20,895</td>
<td>21,796</td>
</tr>
<tr>
<td>mknaps1</td>
<td>10.100.01</td>
<td>22,801</td>
<td>21,708</td>
<td>20,663</td>
<td>21,348</td>
</tr>
<tr>
<td>mknaps1</td>
<td>10.100.02</td>
<td>22,131</td>
<td>20,945</td>
<td>20,058</td>
<td>20,961</td>
</tr>
<tr>
<td>mknaps1</td>
<td>10.100.03</td>
<td>22,772</td>
<td>21,395</td>
<td>20,908</td>
<td>21,377</td>
</tr>
<tr>
<td>mknaps1</td>
<td>10.100.04</td>
<td>22,751</td>
<td>21,453</td>
<td>20,488</td>
<td>21,251</td>
</tr>
</tbody>
</table>

Table 3 and Table 4 show the performance of BCS on MKP instances. Table 3 shows experimental results of our BCS algorithm over the 7 easy instances of mknaps1 benchmarks. The first column indicates the instance index. The second and third columns indicate the number of object and knapsack dimension respectively and the fourth and fifth columns indicate the best known and the BCS solution, respectively. As we can see, our algorithm is able to find the best solution of all the mknaps1 instances. Finally, Table 4 shows the experimental results of the BCS, PSO-P and QICSA algorithms on some hard instances of mknaps1 and mknaps4. Column 1 shows the benchmark name. Column 2 indicates the problem size. Column 3 indicates the best known solutions from OR-Library. Columns 4, 5 and 6 indicate the BCS, the PSO-P and the QICSA solutions.
respectively. The obtained results show that for all cases the BCS algorithm gives good and promising results compared with PSO-P algorithm which is based on a penalty function technique to deal with the constrained problems (Kong and Tian, 2006). Moreover, the BCS algorithm outperforms the QICSA algorithm in several cases (i.e., 5.100.00, 5.100.01, 5.100.04, 10.100.00, 10.100.01, 10.100.03 and 10.100.04).

The statistical Friedman test in Figure 6 represents a comparison of the best known, the BCS, the PSO-P and the QICSA results. Our BCS algorithm ranks second in the Friedman test (Figure 6). This statistical test shows that there is a significant difference between best known, the PSO-P and the QICSA results, whereas the difference between best known and BCS results is not statistically significant. Consequently, the obtained results confirm that BCS algorithm outperforms the PSO-P and QICSA algorithms and prove that the proposed algorithm gives good and promising results compared to the best known result (Figure 6). However, the performances of BCS can be considerably increased by the introduction of some specified knapsack heuristic operators utilising problem specific knowledge like the repair operator used in Gupta and Garg (2009).

The CS algorithm is a new swarm optimisation algorithm. Unfortunately, there are few applications in optimisation problems based on CS algorithm. Furthermore, there is no discrete version of CS for the resolution of binary optimisation problems. The main purpose of this paper is to validate that the CS technique is also effective for binary combinatorial optimisation problems. That is why we have introduced the sigmoid function in the core of the CS algorithm, which led to an efficient BCS algorithm able to deal with binary optimisation problems. In order to assess the performance of the proposed BCS algorithm, we have used it for solving two NP-hard combinatorial optimisation problems: KP and MKP. These problems are very important for modelling many industrial problems. Moreover, KP and MKP have received the attention of many researchers due to their importance and their NP-hardness. The results presented in this paper are quite encouraging, showing the potential of BCS in dealing with KPs. We can see that BCS is much more efficient in finding the global optima compared to PSO, harmony search and quantum CS algorithms. Although, the proposed algorithm and the two BPSO models (BPSO and PSO-P) used in comparison use the sigmoid function and the probability model to generate a binary solution, the experimental results show that the BCS algorithm outperforms obviously the BPSO and PSO-P algorithms. The effectiveness of our approach is explained by the good balance between the exploitation of the local random walk and the exploration of Lévy flights, which leads the algorithm to effectively explore the search space and locate potential solutions. It is notable that the convergence speed of the algorithm is insensitive to its parameters such as \( p_a \). The diversity of the BCS algorithm is assured by the use of the elitism selection which guarantees that a fraction of the best solutions are kept in each generation of BCS, while the detection of alien eggs with some probability makes sure that some of the worse solutions are replaced by new ones. Subsequently, the proposed BCS is more generic and can be implemented easily for other binary optimisation problems, and it needs few parameters to set (problem size and \( p_a \)) compared to BPSO algorithm. Finally, our algorithm BCS can be improved by the introduction of the information exchange among the best solutions in an attempt to speed up convergence to the optimum (Walton et al., 2011).

6 Conclusions

In this work, we have presented a discrete binary version of CS algorithm called BCS. Our contribution has a two-fold aim: the first aim is to propose a binary version of CS algorithm to deal with binary NP-hard optimisation problems. The second aim is to prove the effectiveness of the CS algorithm in solving NP-hard combinatorial optimisation problems. The main feature of our approach consists in using of sigmoid function and probability model in order to generate binary solutions. To evaluate and prove the performance and the effectiveness of the proposed algorithm BCS, we have applied and tested it on some KP and MKP instances. On the other hand, we have compared BCS with the harmony search algorithm (NGHS), the standard BPSO algorithm the standard binary PSO with penalty function technique (PSO-P) and the QICSA. The experimental studies prove the feasibility and the effectiveness of our approach. Indeed, in the most cases BCS is able to give comparable or close results to the optimal. However, there are several issues to improve our algorithm. Firstly, in order to improve the performance of our algorithm, it is better to integrate a local search method like tabu search in the core of the algorithm. In addition, integration of other operations inspired from other popular algorithms such as PSO or GA will also be potentially fruitful. Finally, to raise the speed of our program, it is better to use parallel machines because it was verified effectively that population-based algorithms can work better on parallel machines.
References


