An Improved Chaotic Flower Pollination Algorithm for Solving Large Integer Programming Problems

Ibrahim El-henawy, Mahmoud Ismail

Abstract

Flower pollination Algorithm is a recently-developed method in the field of computational intelligence. In this paper is presented an improved version of Flower pollination Meta-heuristic Algorithm, (FPPSO), for solving integer programming problems. The proposed algorithm combines the standard flower pollination algorithm (FP) with the particle swarm optimization (PSO) algorithm to improve the searching accuracy. Numerical results show that the FPPSO is able to obtain the optimal results in comparison to traditional methods (branch and bound) and other harmony search algorithms. However, the benefits of this proposed algorithm is in its ability to obtain the optimal solution within less computation, which save time in comparison with the branch and bound algorithm (exact solution method).

Keywords: Flower pollination Algorithm, meta-heuristics, optimization, chaos, particle swarm optimization, integer programming

1. Introduction

The real world optimization problems are often very challenging to solve, and many applications have to deal with NP-hard problems [1]. To solve such problems, optimization tools have to be used even though there is no guarantee that the optimal solution can be obtained. In fact, for NP problems, there are no efficient algorithms at all. As a result of this, many problems have to be solved by trial and errors using various optimization techniques [2]. In addition, new algorithms have been developed to see if they can cope with these challenging optimization problems. Among these new algorithms, many algorithms such as particle swarm optimization, cuckoo search and firefly algorithm, have gained popularity due to their high efficiency. In this paper we have used IBACH algorithm for solving integer programming problems. Integer programming is NP-hard problems [3-10]. The name linear integer programming is referred to the class of combinatorial constrained optimization problems with integer variables, where the objective function is a linear function and the constraints are linear inequalities. The Linear Integer Programming (also known as LIP) optimization problem can be stated in the following general form:

\[ \text{Max } cx \]
\[ s.t. \ Ax \leq b, \]
\[ x \in \mathbb{Z}^n \]

where the solution \( x \in \mathbb{Z}^n \) is a vector of \( n \) integer variables: \( x = (x_1, x_2, \ldots, x_n)^T \) and the data are rational and are given by the \( m \times n \) matrix \( A \), the \( 1 \times n \) matrix \( c \), and the \( m \times 1 \) matrix \( b \). This formulation includes also equality constraints, because each equality constraint can be represented by means of two inequality constraints like those included in eq. (2).

Integer programming addresses the problem raised by non-integer solutions in situations where integer values are required. Indeed, some applications do allow a continuous solution. For instance, if the objective is to find the amount of money to be invested or the length of cables to be used, other problems preclude it: the solution must be discrete [3]. Another example, if we are considering the production of jet aircraft and \( x_1 = 8.2 \) jet airliners, rounding off could affect the profit or the cost by millions of dollars. In this case we need to solve the problem so that an optimal integer solution is guaranteed.
The possibility to obtain integer values is offered by integer programming: as a pure integer linear programming, in which all the variables must assume an integer value, or as a mixed-integer linear programming which allows some variables to be continuous, or a 0-1 integer model, all the decision variables have integer values of zero or one [10].

A wide variety of real life problems in logistics, economics, social sciences and politics can be formulated as linear integer optimization problems. The combinatorial problems, like the knapsack-capital budgeting problem, warehouse location problem, travelling salesman problem, decreasing costs and machinery selection problem, network and graph problems, such as maximum flow problems, set covering problems, matching problems, weighted matching problems, spanning trees problems and many scheduling problems can also be solved as linear integer optimization problems [11-14].

Exact integer programming techniques such as cutting plane techniques [15-17]. The branch and the bound both have high computational cost, in large-scale problems [18-19]. The branch and the bound algorithms have many advantages over the algorithms that only use cutting planes. One example of these advantages is that the algorithms can be removed early as long as at least one integral solution has been found and an attainable solution can be returned although it is not necessarily optimal. Moreover, the solutions of the LP relaxations can be used to provide a worst-case estimate of how far from optimality the returned solution is. Finally, the branch method and the bound method can be used to return multiple optimal solutions.

Since integer linear programming is NP-complete, for that reason many problems are intractable. So instead of the integer linear programming, the heuristic methods must be used. For example, Swarm intelligence metaheuristics, amongst which an ant colony optimization, artificial bee colony optimization particle swarm optimization [20-24]. Also Evolutionary algorithms, differential evolution and tabu search were successfully applied into solving integer programming problems [25-27]. Heuristics typically have polynomial computational complexity, but they do not guarantee that the optimal solution will be captured. In order to solve integer programming problems, most of the heuristics truncate or round the real valued solutions to the nearest integer values. In this paper, an improved version of flower pollination algorithm is applied to integer programming problems and the performance was compared with other harmony search algorithms.

This paper is organized as follows: after introduction, the original flower pollination algorithm is briefly introduced in section 2. Section 3 introduces the particle swarm optimization algorithm. Section 4 introduces the meaning of chaos. In section 5, the proposed algorithm is described, while the results are discussed in section 6. Finally, conclusions are presented in section 7.

2. The Flower pollination Algorithm

Flower Pollination Algorithm (FPA) was founded by Yang in the year 2012. Inspired by the flow pollination process of flowering plants are the following rules [28]:

**Rule 1**: Biotic and cross-pollination can be considered as a process of global pollination process, and pollen-carrying pollinators move in a way that obeys Le'vy flights.

**Rule 2**: For local pollination, a biotic and self-pollination are used.

**Rule 3**: Pollinators such as insects can develop flower constancy, which is equivalent to a reproduction probability that is proportional to the similarity of two flowers involved.

**Rule 4**: The interaction or switching of local pollination and global pollination can be controlled by a switch probability \( p \in [0, 1] \), with a slight bias toward local pollination.

In order to formulate updating formulas, we have to convert the aforementioned rules into updating equations. For example, in the global pollination step, flower pollen gametes are carried by pollinators such as insects, and pollen can travel over a long distance because insects can often fly and move in a much longer range [56]. Therefore, Rule 1 and flower constancy can be represented mathematically as:

\[
x_i^{t+1} = x_i^t + \gamma L(\lambda)(x_i^t - B)
\]

Where \( x_i^t \) is the pollen i or solution vector \( x_i \) at iteration t, and B is the current best solution found among all solutions at the current generation/iteration. Here \( \gamma \) is a scaling factor to control the step size.
In addition, $L(\lambda)$ is the parameter that corresponds to the strength of the pollination, which essentially is also the step size. Since insects may move over a long distance with various distance steps, we can use a Lévy flight to imitate this characteristic efficiently. That is, we draw $L > 0$ from a Lévy distribution:

$$L \sim \frac{2\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi} \frac{1}{S^{\alpha+1}}, (S \gg S_0 > 0)$$  \hspace{1cm} (5)

where $\Gamma(\lambda)$ is the standard gamma function, and this distribution is valid for large steps $s > 0$. Then, to model the local pollination, both Rule 2 and Rule 3 can be represented as

$$x_i^{t+1} = x_i^t + U(x_j^t - x_k^t)$$  \hspace{1cm} (6)

where $x_i^t$ and $x_k^t$ are pollen from different flowers of the same plant species. This essentially imitates the flower constancy in a limited neighborhood. Mathematically, if $x_i^t$ and $x_k^t$ comes from the same species or selected from the same population, this equivalently becomes a local random walk if we draw $U$ from a uniform distribution in $[0, 1]$. Though flower pollination activities can occur at all scales, both local and global, adjacent flower patches or flowers in the not-so-far-away neighborhood are more likely to be pollinated by local flower pollen than those faraway. In order to imitate this, we can effectively use the switch probability like in Rule 4 or the proximity probability $p$ to switch between common global pollination to intensive local pollination. To begin with, we can use a naive value of $p = 0.5$ as an initially value. A preliminary parametric showed that $p = 0.8$ might work better for most applications [28].

The basic steps of FP can be summarized as the pseudo-code shown in Figure 1.

### Flower pollination algorithm

<table>
<thead>
<tr>
<th>Define Objective function $f(x)$, $x = (x_1, x_2, ..., x_d)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize a population of $n$ flowers/pollen gametes with random solutions</td>
</tr>
<tr>
<td>Find the best solution $B$ in the initial population</td>
</tr>
<tr>
<td>Define a switch probability $p \in [0, 1]$</td>
</tr>
<tr>
<td>Define a stopping criterion (either a fixed number of generations/iterations or accuracy)</td>
</tr>
<tr>
<td>while $t &lt; $MaxGeneration</td>
</tr>
<tr>
<td>for $i = 1 : n$ (all $n$ flowers in the population)</td>
</tr>
<tr>
<td>if rand $&lt;$ $p$,</td>
</tr>
<tr>
<td>Draw a ($d$-dimensional) step vector $L$ which obeys a Lévy distribution</td>
</tr>
<tr>
<td>Global pollination via $x_i^{t+1} = x_i^t + L(B - x_i^t)$</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>Draw $U$ from a uniform distribution in $[0,1]$</td>
</tr>
<tr>
<td>Do local pollination via $x_i^{t+1} = x_i^t + U(x_j^t - x_k^t)$</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>Evaluate new solutions</td>
</tr>
<tr>
<td>If new solutions are better, update them in the population</td>
</tr>
<tr>
<td>end for</td>
</tr>
<tr>
<td>Find the current best solution $B$</td>
</tr>
<tr>
<td>end while</td>
</tr>
<tr>
<td>Output the best solution found</td>
</tr>
</tbody>
</table>

Figure 1. Pseudo code of the Flower pollination algorithm

3. Particle Swarm Optimization

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart in 1995 based on the swarm behavior such as fish and bird schooling in nature [29]. Since then, PSO has generated much
wider interests and forms an exciting, ever expanding research subject called swarm intelligence. This algorithm searches the space of an objective function by adjusting the trajectories of individual agents, called particles, as the piecewise paths formed by positional vectors in a quasi-stochastic manner. The movement of a swarming particle consists of two major components: a stochastic component and a deterministic component. Each particle is attracted toward the position of the current global best \( g \) and its own best location \( x_i^* \) in history, while at the same time it has a tendency to move randomly. Let \( x_i \) and \( v_i \) be the position vector and velocity of particle \( i \), respectively. The new velocity vector is determined by the following formula:

\[
v_i^{t+1} = v_i^t + c_1 r_1 (g - x_i^t) + c_2 r_2 (x_i^* - x_i^t)
\]

Where \( r_1 \) and \( r_2 \) are two random vectors and each entry takes the values between 0 and 1. The parameters \( c_1 \) and \( c_2 \) are the learning parameters or acceleration constants, which can typically be taken as, say, \( c_1 = c_2 = 2 \). The initial locations of all particles should be distributed relatively uniformly so that they can sample over most regions, which is especially important for multimodal problems. The initial velocity of a particle can be taken as zero, i.e. \( v_i^{t=0} = 0 \). The new positions can then be updated by:

\[
x_i^{t+1} = x_i^t + v_i^{t+1}
\]

Although \( v_i \) can be any value, it is usually bounded in some range \([0, v_{\text{max}}]\).

4. Chaos

Generating random sequences with longer periods and good consistency is very important for easily simulating complex phenomena, sampling, numerical analysis, decision making and especially in heuristic optimization [30]. Its quality determines the reduction of storage and computation time to achieve a desired accuracy [31]. Chaos is a deterministic, random-like process found in a nonlinear, dynamical system, which is non-period, non-converging and non-bounded. Moreover, it depends on its initial condition and parameters [32-34]. Applications of chaos has several disciplines including operations research, physics, engineering, economics, biology, philosophy and computer science [35-37].

Recently chaos has been extended to various optimization areas because it can more easily escape from local minima and improve global convergence in comparison with other stochastic optimization algorithms [34-38]. Using chaotic sequences in flower pollination Algorithm can be helpful to improve the reliability of the global optimality, and they also enhance the quality of the results.

4.1. Chaotic maps

At random-based optimization algorithms, the methods using chaotic variables instead of random variables are called chaotic optimization algorithms (COA) [34]. In these algorithms, due to the non-repetition and ergodicity of chaos, it can carry out overall searches at higher speeds than stochastic searches that depend on probabilities [43-48]. To resolve this issue, herein one-dimensional and non-invertible maps are utilized to generate chaotic sets. We will illustrate some of well-known one-dimensional maps as:

4.1.1. Logistic map

The Logistic map is defined by:

\[
Y_{n+1} = \mu Y_n (1 - Y_n) \quad \text{for} \quad 0 < \mu \leq 4
\]

4.1.2. The Sine map

The Sine map is written as the following equation:

\[
Y_{n+1} = \frac{\mu}{4} \sin(\pi Y_n) \quad \text{for} \quad 0 < \mu \leq 4
\]

4.1.3. Iterative chaotic map
The iterative chaotic map with infinite collapses is described as:

$$Y_{n+1} = \sin \left( \frac{\mu Y_n}{Y_n} \right) \mu \in (0, 1)$$  \hspace{1cm} (11)

4.1.4. Circle map

The Circle map is expressed as:

$$Y_{n+1} = Y_n + \alpha - \left( \frac{\beta}{2\pi} \right) \sin(2\pi Y_n) \mod 1$$  \hspace{1cm} (12)

4.1.5. Chebyshev map

The family of Chebyshev map is written as the following equation:

$$Y_{n+1} = \cos(k\arcsin(Y_n)) Y \in (-1, 1)$$  \hspace{1cm} (13)

4.1.6. Sinusoidal map

This map can be represented by

$$Y_{n+1} = \mu Y_n ^2 \sin(\pi Y_n)$$  \hspace{1cm} (14)

4.1.7. Gauss map

The Gauss map is represented by:

$$Y_{n+1} = \begin{cases} 0 & Y_n = 0 \\ \frac{\mu}{Y_n} \mod 1 & Y_n \neq 0 \end{cases}$$  \hspace{1cm} (15)

4.1.8. Sinus map

Sinus map is formulated as follows:

$$Y_{n+1} = 2.3(Y_n)^{2\mu} (\pi Y_n)$$  \hspace{1cm} (16)

4.1.9. Dyadic map

Also known as the dyadic map bit shift map, 2x mod 1 map, Bernoulli map, doubling map or saw tooth map. Dyadic map can be formulated by a mod function:

$$Y_{n+1} = 2Y_n \mod 1$$  \hspace{1cm} (17)

4.1.10. Singer map

Singer map can be written as:

$$Y_{n+1} = \mu(7.86Y_n - 23.31Y_n^2 + 28.75Y_n^3 - 13.3Y_n^4)$$  \hspace{1cm} (18)

\(\mu\) between 0.9 and 1.08
4.1.11. Tent map

This map can be defined by the following equation:

\[
Y_{n+1} = \begin{cases} 
\mu Y_n < 0.5 \\
(\mu(1 - Y_n))Y_n \geq 0.5 
\end{cases}
\] (19)

5. The Proposed Algorithm (FPPSO) for Solving Integer Programming Problems

In the proposed algorithm, we used chaotic maps to tune the flower pollination algorithm parameters and improve the performance [33]. The steps of the proposed Algorithm for solving integer programming problems are as follows:

Step 1 Initialize the swarm by randomly assigning each particle to an arbitrarily initial velocity and a position in each dimension of the solution space.

Step 2 Evaluate the desired fitness function to be optimized for each particle’s position.

Step 3 for each individual particle; update its historically best position so far, \textit{BestPi}, if its current position is better than its historically best one.

Step 4 Identify/Update the swarm’s globally best particle that has the swarm’s best fitness value, and set/reset its index as \textit{g} and its position at \textit{gbestP}.

Step 5 Update the velocities of all the particles using equation (7).

Step 6 Move each particle to its new position using equation (8).

Step 7 Repeat steps 2–6 until convergence or a stopping criterion is met.

Step 8 The best solution found by PSO is regarded as initial points for FP algorithm. \textit{B}

Step 9 Calculate \textit{p} by the selected chaotic maps.

Step 10 If (rand < \textit{p}) then global pollination via \(x_i^{t+1} = x_i^t + (f \gamma) L(\lambda)(x_i^t - \textit{B}) \) (\textit{f \gamma}) chaotic \textit{Le}vy flights

else do local pollination via \(x_i^{t+1} = x_i^t + U(x_i^t - x_i^t)\).

Step 11 Evaluate new solutions if better, update them in the population.

Step 12 Find the current best solution \textit{B}.

Step 13 Output the best solution found.

6. Numerical Results

Several examples have been done to verify the weight of the proposed algorithm. The initial parameters setting of the algorithms is as follows: HMS=50 and \textit{itermax}=1000, HMCR = 0.9; \textit{PARmax} = 1; \textit{PARmin} =0.1; \textit{bwmax} = 1; \textit{bwmin} = 0.01. The results of FPPSO algorithm are conducted from 50 independent runs for each problem and measured according to the best values in these runs. The selected chaotic map for all examples is the Sinusoidal map, whose equation is shown below:

\[
Y_{n+1} = \mu Y_n \sin(\pi Y_n) 
\] (20)

Where \(n\) is the iteration number, all the experiments were performed on a Windows 7 Ultimate 64-bit operating system; processor Intel Core i5 760 running at 2.81 GHz; 4 GB of RAM and codes were implemented in C#.

6.1 Test Problem 1

\[
\begin{align*}
\text{Max } z &= 7x_1 + 9x_2 \\
\text{s.t.} & \quad -x_1 + 3x_2 \leq 6 \\
& \quad 7x_1 + x_2 \leq 35 \\
& \quad x_1, x_2 \geq 0 \text{ and integer.}
\end{align*}
\]
6.2 Test Problem 2

Max $w = 4x_1 + 6x_2 + 2x_3$

s.t.
$4x_1 - 4x_2 \leq 5$
$-x_1 + 6x_2 \leq 5$
$-x_1 + x_2 + x_3 \leq 5$
$x_1, x_2, x_3 \geq 0$ and integer.

6.3 Test Problem 3

Min $z = -5x_1 + 7x_2 + 10x_3 - 3x_4 + x_5$

s.t.
$x_2 - 2x_3 - x_4 - x_5 \leq -2$
$x_1 + 3x_2 - 5x_3 + x_4 + 4x_5 \geq 0$
$2x_1 + 6x_2 - 3x_4 + 2x_5 + 2x_6 \geq 4$
$x_i = 0$ or $1, i = 1, 2, ..., 5.$

6.4 Test Problem 4

Max $z = 3x_1 - 2x_2 + 4x_3 + 5x_4 + x_5 + x_6 + 2x_7 - 6x_8 + x_9 + x_{10}$

s.t.
$x_1 + 5x_2 + 3x_3 \geq 8$
$x_5 + 5x_6 + 2x_9 + x_{10} = 7$
$4x_1 + x_2 + 3x_3 + x_4 + x_7 \leq 4$
$6x_3 + x_4 + x_6 + 3x_9 - 2x_5 \leq 5$
$4x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 \geq 2$
$-x_1 + x_2 + 2x_3 - x_4 + x_5 + 7x_7 \leq 3$
$-3x_1 - x_2 + 4x_3 - x_4 + 2x_5 + 3x_6 + 9x_8 + x_{10} = 10$
$x_i \geq 0, i = 1, 2, ..., 10$ and integer

6.4 Test Problem 5

Min $z = 3x_1 - 2x_2 + 5x_3 + x_4 - x_5 + 2x_6 + 7x_7 + 3x_8 + 11x_9 + 22x_{10} - 15x_{11} + 9x_{12} + 7x_{13} - 13x_{14} + x_{15} + 8x_{16} + 19x_{17} + 4x_{18} - 9x_{19} + 10x_{20}$

s.t.
$x_1 + 3x_{18} + 9x_{19} + x_{20} \geq 11$
$3x_3 + 7x_{17} + 8x_{19} + x_{20} \geq 18$
$x_7 + x_{13} + x_{15} + 3x_{16} + 3x_{18} \leq 25$
$x_1 + x_2 + 12x_3 + 5x_8 + 6x_{15} - 3x_8 \leq 6$
$-3x_3 - 2x_4 + 7x_5 - 9x_6 + x_{13} + 5x_{17} \geq 21$
$x_6 - 7x_{10} + 3x_{17} + x_{12} + 8x_{13} - 9x_{14} \leq 4$
$-3x_1 + x_2 + 8x_3 + x_{13} + 22x_{19} - x_{20} \leq 83$
$5x_1 + 8x_2 + 7x_3 + x_6 + 2x_7 + 9x_{12} + 7x_{13} \leq 9$
$10x_1 + 2x_6 + x_{10} + 6x_{11} + 11x_{19} + 8x_{15} - 9x_{19} = 40$
$2x_1 + 3x_2 - 7x_4 + 4x_5 + 18x_9 + x_{11} + 9x_{13} + 4x_{16} \geq 10$
$3x_1 - 7x_2 + x_3 + 3x_4 + 8x_8 - 4x_{10} + 11x_{12} + 5x_{14} + 7x_{19} \leq 32$
$22x_5 + 8x_3 + 8x_5 + 13x_2 + 18x_3 + 4x_9 - 3x_{10} + 19x_{11} - 8x_{17} - x_{18} \geq 17$
$x_1 + 3x_2 - x_3 + 7x_4 + 11x_5 + 18x_9 + 5x_{14} + 11x_{17} + x_{18} + 4x_{20} \geq 22$
$-8x_1 + 5x_3 + x_5 + x_6 + x_{10} + 7x_{11} + x_{13} + x_{15} + 8x_{17} + 2x_{20} \geq 9$
$4x_1 + 5x_2 + x_8 + 14x_{10} + 11x_{12} + x_{13} + 5x_{14} + x_{15} + 11x_{16} + 3x_{18} + 3x_{20} \geq 16$
$x_i \geq 0, i = 1, 2, ..., 20$ and integer.
6.4 Test Problem 6

\[
\max \quad Z = x_1 + 2x_2 + 7x_3 + 2x_4 + x_5 + 4x_6 + x_7 -5x_8 + x_9 + 2x_{10} + 4x_{11} + 7x_{12} + 5x_{13} + 3x_{14} + 9x_{15} + 5x_{16} + x_{18} + 8x_{19} + 6x_{20} + x_{21} -3x_{22} + 7x_{23} - x_{24} -3x_{25} + 2x_{26} + x_{27} + 9x_{28} + 7x_{29} + 4x_{30}
\]

s.t.
\[
2x_1 + 2x_3 + x_{12} + 9x_{15} -3x_{20} + 3x_{29} \leq 70 \\
2x_1 + 5x_5 + 2x_{13} + 2x_{16} + 2x_{28} \leq 19 \\
x_2 + x_3 + x_7 + 2x_8 + x_{11} + 8x_{14} + 2x_{25} \leq 20 \\
x_9 - x_{11} -3x_{12} -3x_{13} + 2x_{19} + x_{20} + 3x_{22} + x_{23} \leq 11 \\
-3x_{11} + x_{14} + 3x_{15} + 2x_{16} + x_{18} + 2x_{22} + 2x_{26} \leq 95 \\
x_{12} + x_{14} + 2x_{15} + 2x_{18} + 2x_{24} + 2x_{25} + 3x_{27} \leq 40 \\
x_2 + x_3 + x_4 + 2x_8 + 2x_{10} + x_{17} + x_{18} + x_{20} + x_{21} + x_{22} \leq 35 \\
4x_1 + 2x_9 + x_{10} + 3x_{13} + 3x_{16} -9x_{17} + x_{18} + x_{24} + 5x_{27} + 4x_{29} + x_{30} \leq 40 \\
-3x_5 + x_9 + 2x_{12} + 5x_{13} + 4x_{16} + x_{17} + x_{19} + x_{21} + 4x_{25} + 3x_{27} + 2x_{30} \leq 100 \\
5x_1 + x_3 + 3x_6 + 2x_9 + x_4 + 2x_9 -5x_{10} + 5x_{12} + x_{14} + 3x_{16} -9x_{17} + x_{18} \leq 7 \\
x_1 + 2x_6 + 2x_7 + 2x_{14} + 11x_{15} + x_{16} -3x_{21} + 10x_{21} + 2x_{25} + 8x_{26} + 3x_{28} + 11x_{29} \leq 62 \\
x_2 + x_3 + x_7 + x_8 + 2x_{11} -9x_{13} + 2x_{17} + 5x_{18} + x_{20} + x_{21} + 4x_{24} + 3x_{26} + 5x_{27} + 4x_{30} \leq 51 \\
x_1 + x_3 + 2x_4 + 2x_8 + 3x_7 + 2x_9 -2x_{10} + 2x_{11} -5x_{13} + 2x_{19} + 3x_{20} + 4x_{21} + 3x_{23} + 4x_{28} \leq 22 \\
x_2 + x_4 + 5x_3 + 4x_6 + 2x_7 + 3x_{13} + 8x_{17} + 2x_{19} + x_{21} + 2x_{22} + 2x_{23} + 2x_{24} + 10x_{25} -3x_{26} + 2x_{27} + 3x_{28} + 2x_{29} + x_{30} \leq 60 \\
x_i \geq 0, \ i=1,2,...,30 \ and \ integer.
\]

<table>
<thead>
<tr>
<th>No.of Variables</th>
<th>Optimal sol.</th>
<th>Optimal values</th>
<th>The Best Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>55</td>
<td>X_i=(4,3)</td>
<td>HS[50]</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>X_i=(2,1,6)</td>
<td>IHS[50]</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>X_i=(1,1,0,0,0)</td>
<td>FPPSO</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>X_i=(0,2,0,2,3,1,0,0,2,3)</td>
<td>FPPSO</td>
</tr>
<tr>
<td>20</td>
<td>16</td>
<td>X_i=(0,0,0,0,0,0,0,0,0,1,4,0,4,3,0,2,4,0,3,0)</td>
<td>FPPSO</td>
</tr>
<tr>
<td>30</td>
<td>446</td>
<td>X_i=(0,0,0,0,0,0,0,0,16,20,4,4,0,3,0,0,0,24,3,0,0,0,0,4,0,1,0,8)</td>
<td>FPPSO</td>
</tr>
</tbody>
</table>

Table 1. Optimal solution of different integer programming problems

Table 1 shows the results of FPPSO algorithm are privileged compared with the results of Standard harmony search algorithm (HS) [51], improved harmony search algorithm (IHS) [52] and by [50]. In comparison with exact values we find that the results of FPPSO algorithm are very close to the exact values of selected problems under the study.

7. Conclusions

This paper has introduced an improved lower pollination Algorithm by blending with particle swarm optimization algorithm for solving integer programming problems. Several examples have been used to prove the effectiveness of the proposed methods. The proposed algorithm managed to solve a large scale of problems that traditional method could not solve due to exponential growth in time and space complexities. The solution procedure will not face the same time waste in going through non-converging iterations as traditional methods do. FPPSO algorithm is superior to both HS and IHS in terms of both efficiency and success rate. This implies that FPPSO is potentially more powerful in solving NP-hard problems.

8. References