Hybrid Coevolutionary Population Migration Algorithm for Integer Programming and Its Application in Neural Network

Liangdong QU 1, 2, Dengxu HE1, 2, †, Jinzhao WU1, 2, 3

1 College of Mathematics and Computer Science, Guangxi University for Nationalities, Nanning 530006, China
2 Guangxi Key Laboratory of Hybrid Computation and IC Design Analysis, Nanning 530006, China
3 School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China

Abstract

For integer programming, there exist some difficulties and problems for the direct applications of population migration algorithm (PMA) due to the variables belonging to the set of integers. In this paper, a novel hybrid coevolutionary PMA is proposed for integer programming which evolves on the set of integer space. Several functions and its application problem simulation results show that the proposed algorithm is significantly superior to other algorithms.

Keywords: Integer Programming; Population Migration Algorithm; Hooke-Jeeves Pattern Search Method; Neural Network

1. Introduction

Integer programming [1] is often encountered in many areas. An important area of application concerns the efficient management of a limited number of resources so as to increase productivity and/or profit. Such applications are encountered in Operational Research problems such as goods distribution, production scheduling, and machine sequencing. Capital budgeting, portfolio analysis, network and VLSI circuit design, as well as automated production systems are some other applications in which Integer Programming problems are met [2].

Yet another recent and promising application is the training of neural networks with integer weights, where the activation function and weight values are confined in a narrow band of integers. Such neural networks are better suited for hardware implementations than the real weight ones [3].

The Integer Programming problem can be stated as

\[ \min_{x \in S} f(x) \quad S \subseteq \mathbb{Z}^n \]

where \( \mathbb{Z} \) is the set of integers, and \( S \) is a not necessarily bounded set, which is considered as the feasible region (maximization Integer Programming problems can be easily turned to a minimization problem).

Solving integer programming problems, the traditional methods are branch and bound, cutting plane and implicit enumeration [4]. They apply to the low dimensional variable integer programming, however for

† Corresponding author.

Email addresses: dengxuhe@126.com (Dengxu HE).
high dimensional problems, traditional calculation methods would be very time-consuming. Optimization techniques applied on real search spaces can be applied on such problems and determine the optimum solution by rounding off the real optimum values to the nearest integer. However, in many cases, certain constraints of the problem are violated due to the rounding of the real optimum solutions. Moreover, the rounding might result in a value of the objective function that is far removed from the optimum [3, 5] and their efficiency is not high.

This paper based on the analysis of the basic population migration algorithm (PMA) [6] and Hooke-Jeeves pattern search method [7], we propose a novel hybrid coevolutionary PMA which does evolutionary computation in integer space directly, the method ensures that the population stay in integer space in the evolutionary process, then to avoid searching in non-integral space, to speed up the convergence and improve the calculation accuracy.

The rest of this paper is organized as follows: in Section 2, we give the brief population migration algorithm. In Section 3, we propose Hooke-Jeeves pattern search method and simplified Hooke-Jeeves exploratory move. In Section 4, we propose a hybrid coevolutionary population migration algorithm for integer programming (IPHCPMA). In Section 5, we report numerical results of the methods for some test problems. In Section 6, we give its application. Finally, Section 7 gives conclusions.

2. Population Migration Algorithm

Population Migration Algorithm (PMA) [6] is proposed in 2003, it is a kind of global optimization algorithm that simulates population migration mechanism. As the living groups, population will certainly keep migrate to survive and improve. Population migration can be divided into three basic operators: population flow, population migration and population proliferation, population flow is spontaneously, without an overall plan movement in the local environment; population migration is a selective movement crossing a wide range, the basic rule is following the economic center of gravity; population proliferation is a selective movement that form the beneficial region to the non-preferential one due to the population pressure increased.

As a optimization algorithm, in the PMA, the optimization variable $x$ refers to living places, the objective function $f(x)$ refers to attractive of the residence place, the optimal solution(local optimal solution) refers to the most attractive place (beneficial region), the "up" or "mountain climbing" of algorithm refers to move to beneficial region, escaping from the local optimal refers to move out of beneficial region as a result population pressure; population flow corresponds to random, local searching method; and population migration corresponds to the way to choose the approximate solution like as population struggles upwards; population proliferation combines the overall searching with escaping from the local optimal strategy.

3. Hooke-Jeeves Pattern Search Method

Hooke-Jeeves pattern search method [7] is a direct searching method proposed by Hooke and Jeeves in 1961. It works by creating a set of search directions iteratively. The created search directions should be such that they completely span the search space. In other words, they should be such that starting from any point in the search space any other point in the search space can be reached by traversing along these directions only.
In the Hooke-Jeeves method, a combination of exploratory moves and heuristic pattern moves is made iteratively.

### 3.1. Exploratory Move

**Step 1.** Calculate \( f = f(x) \), \( f^+ = f(x + \Delta) \) and \( f^- = f(x - \Delta) \).

**Step 2.** \( f_{\text{min}} = \min\{f, f^+, f^-\} \). Set \( x \) corresponds to \( f_{\text{min}} \).

**Step 3.** Is \( i = N \)? if no, set \( i = i + 1 \) and go to Step 1. Else \( x \) is the result and go to Step 4.

**Step 4.** If \( x \neq x^e \), success; Else failure.

### 3.2. Pattern Move

A new point is found by jumping from the current best point \( x^e \) along a direction connecting the previous best point \( x^{(k-1)} \) and the current base point \( x^{(k)} \) as follows:

\[
x_p^{(k+1)} = x^{(k)} + \left(x^{(k)} - x^{(k-1)}\right)
\]

The Hooke-Jeeves method comprises of an iterative application of an exploratory move in the locality of the current point and a subsequent jump using the pattern move. If the pattern move does not take the solution to a better region, the pattern move is not accepted and the extent of the exploratory search is reduced. Therefore, the Hooke-Jeeves method is also called step acceleration or pattern search method.

### 3.3. Simplified Hooke-Jeeves Method (SHJ)

In this paper, we simplify the Hooke-Jeeves method. Where we restrict the exploratory move in the set of integer space and discard pattern move.

The specific steps of SHJ are as follows: Input the iteration \( l \), the dimension of search space \( N \), the step of the exploratory move \( n_0 \).

1. Initialize the initial point \( x^{(0)} \);
2. \( k = j = 1 \);
3. \( y^{(1)} = x^{(0)} \);
4. while \( k \leq l \)
5. for every \( j \)
6. if \( f(y^{(1)} + n_0 e_j) < f(y^{(1)}) \)
   
   \[
y^{(1)} = y^{(1)} + n_0 e_j
   \]
else if \( f(y^{(l)} - n_\alpha e_j) < f(y^{(l)}) \)

\[ y^{(l)} = y^{(l)} - n_\alpha e_j; \]

else

\text{break;}

end if

7. end for

8. \( k = k + 1; \)

9. end while

4. Hybrid Coevolutionary Population Migration Algorithm for Integer Programming

4.1. The Principle of Hybrid Coevolutionary Population Migration Algorithm for Integer Programming (IPHCPMA)

The basic PMA is used for integer programming, because three operators: population flow, population migration and population proliferation are in real space. The next state may be a non-integral number, even if the current state is an integer, so its evolution is in real space. If directly solving integer programming using the basic PMA, its efficiency will be lower. For integer programming, it should be in integer space only, so the three operators: population flow, population migration and population proliferation are redesigned in integer space. Literature [8] proposed a population migration algorithm for integer programming, however, its population flow is blind and completely random. So we add SHJ method to PMA. The new algorithm denote IPHCPMA. For Hooke-Jeeves method, we reserve exploratory move and discard pattern move, population migration and population proliferation instead of pattern move. For PMA, the simplified exploratory move instead of population flow.

4.2. The Process of Hybrid Coevolutionary Population Migration Algorithm for Integer Programming (IPHCPMA)

The procedure of the hybrid coevolutionary algorithm of PMA and Simplified Hooke-Jeeves Method for Integer Programming is summarized as follows:

**Step 1.** Initialize the population size \( NP \), the maximum number of iterations \( \text{max\_num} \), the contraction coefficient \( \Delta \), population pressure parameter \( \alpha \), the largest number of flow \( l \). Set the initial iteration number \( num = 0 \).

**Step 2.** Generate the initial populations \( NP \) in the feasible integer region, form \( NP \) flow integer regions.

**Step 3.** Each Population executes simplified Hooke-Jeeves exploratory move method in search integer region, record and update of the optimal value. If the flow number is more than pre-given number, then do population migration, contract the beneficial region and change the points uniformly, also record and update of the optimal value. When population pressure is greater than a certain population parameters, then do population proliferation, record the optimal value.

**Step 4.** Determine whether the number of iterations achieve the given maximum number, if so, output the result, otherwise, to Step 3.
5. The Mathematical Function Experiments

Four typical test functions [9, 10] are used to estimate the performances of IPHCPMA. Searching theoretical optimum is as the stopping condition of the algorithm, if having not searched theoretical optimum yet when reaching the \( \text{max\_num} \), then the algorithm also stops. To decrease the impact of randomicity, 100 independent runs were performed for each test case. Recording satisfied stopping condition required the average iteration, the success rate of searching theoretical optimum to compare with the basic PMA (BPMA), IPPMA in Literature [8] and PSO in Literature [9, 10]. The parameters are set in Table 1, the step of the exploratory move \( n_e = 1. NP \) is the same to Literature [8-10].

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>max_num</th>
<th>I</th>
<th>( \Delta )</th>
<th>( \alpha )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPMA</td>
<td>100</td>
<td>100</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>IPPMA</td>
<td>100</td>
<td>100</td>
<td>0.01</td>
<td>1</td>
</tr>
<tr>
<td>IPHCPMA</td>
<td>100</td>
<td>100</td>
<td>0.01</td>
<td>1</td>
</tr>
</tbody>
</table>

The experiment conditions are: Intel(R) Core (TM) 2 Duo 2.20GHz CPU, 1G Memory and Windows XP operation system. The programs are realized in Matlab7.

**Example 1.** \( \min f_1(x) = xx^T = \sum_{i=1}^{5} x_i^2, \quad -100 \leq x_i \leq 100, x_i \in Z. \)

As Literature [9], let \( NP = 20 \), run 100 times experiment using IPHCPMA, IPPMA and BPMA, respectively. Their success rate are all 100%, however, their average iteration was 2.01, 4.31 and 5.08. For PSO in Literature [10], the average iteration was 48, and it was 440.72 in Literature [9].

**Example 2.** \( \min f_2(x)=100(x_i + x_j)^2 + 5(x_i - x_j)^2 + (x_i - 2x_j)^2 + 10(x_i - x_j)^4, -100 \leq x_i \leq 100, x_i \in Z. \)

As Literature [9], let \( NP = 40 \), run 100 times experiment using IPHCPMA, IPPMA and BPMA, respectively. Their success rate are all 100%, however, their average iteration was 2, 2.44 and 2.88. For PSO in Literature [9], the success rate was 92%, the average iteration was 460.92.

As Literature [10], let \( NP = 30 \), run 100 times experiment using IPHCPMA, IPPMA and BPMA, respectively. Their success rate are all 100%, however, their average iteration was 2, 2.72 and 3.55. For PSO in Literature [10], the success rate was 98%, the average iteration was 432.

**Example 3.** \( \min f_3(X) = -(15, 27, 36, 18, 12)X' + X' \)

\[
\begin{bmatrix}
35 & -20 & -10 & 32 & -10 \\
-20 & 40 & -6 & -31 & 32
\end{bmatrix}
\]

\( -50 \leq x \leq 50, x_i \in Z, \) with best know solutions \( X^* = (0, 11, 25, 16, 6) \) and \( X^* = (0, 12, 23, 17, 6). \)
As Literature [10], let $NP = 30$, run 100 times experiment using IPHCPMA, IPPMA and BPMA, respectively. The success rate of IPHCPMA is 100% and its average iteration was 7.73. IPPMA and BPMA both are failure, i.e., the success rates of IPPMA and BPMA are 0%. For PSO in Literature [10], the average times was 412, the success rate was 99%.

In Table 2, we gave the average run time of 100 times experiment for each function. We can find the efficiency of IPHCPMA is more than IPPMA and BPMA.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPMA</td>
<td>2.1461s</td>
<td>1.8031s</td>
<td>1.6940s</td>
<td>81.9865s</td>
</tr>
<tr>
<td></td>
<td>($NP=30$)</td>
<td>($NP=30$)</td>
<td>($NP=40$)</td>
<td>($NP=40$)</td>
</tr>
<tr>
<td>IPPMA</td>
<td>0.8356s</td>
<td>0.5337s</td>
<td>0.5384s</td>
<td>30.8532s</td>
</tr>
<tr>
<td></td>
<td>($NP=30$)</td>
<td>($NP=30$)</td>
<td>($NP=40$)</td>
<td>($NP=40$)</td>
</tr>
<tr>
<td>IPHCPMA</td>
<td>0.2924s</td>
<td>0.3450s</td>
<td>0.4200s</td>
<td>3.4224s</td>
</tr>
<tr>
<td></td>
<td>($NP=30$)</td>
<td>($NP=30$)</td>
<td>($NP=40$)</td>
<td>($NP=40$)</td>
</tr>
</tbody>
</table>

6. The Application in Neural Network

An Artificial Neural Network (ANN) [11] is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information, in other words, is an emulation of biological neural system. An ANN is configured for a specific application, such as pattern recognition or data classification, statistics and cognitive psychology, through a learning process [12].

FNNs can be simulated in software, but in order to be utilized in real life applications, where fast speed of execution is required, hardware implementation is needed. The problem is that the conventional multilayer FNNs, which have continuous weights is expensive to implement in digital hardware. Another major implementation obstacle is the weight storage. FNNs having integer weights and biases are easier and less expensive to implement in electronics as well as in optics and the storage of the integer weights is much easier to achieve.

A typical FNN consisting of $L$ layers, where the first layer denotes the input, the last one, $L$ is the output, and the intermediate layers are the hidden layers. It is assumed that the ($l$-1) layer has $N_l$ neurons. Neurons operate according to the following equations

$$ net_l^i = \sum_{j=1}^{N_{l-1}} w_{lj} x_j + \theta_l^i, \quad y_l^i = \sigma_l^i (net_l^i) $$

where $w_{lj}$ is the connection weight from the $i$ th neuron at the ($l$-1) layer to the $j$ th neuron at the $l$ th layer, $y_l^i$ is the output of the $i$ th neuron belonging to the $l$ th layer, $\theta_l^i$ denotes the bias of the $j$ th neuron at the $l$ th layer, and $\sigma$ is a nonlinear activation function.

From the optimization point of view, supervised training of an FNN is equivalent to minimizing a global error function which is a multivariate function that depends on the weights in the network. The square error over the set of input–desired output patterns with respect to every weight, is usually taken as the function to be minimized. Specifically, the error function for an input pattern $t$ is defined as follows:
$e_j(t) = y_j^d(t) - d_j(t), \ j = 1, 2, \cdots, N_L$

where $d_j(t)$ is the desired response of an output neuron at the input pattern $t$. For a fixed, finite set of input-desired output patterns, the square error over the training set which contains $T$ representative pairs is:

$$E(w) = \sum_{i=1}^{T} E_j(w) = \sum_{i=1}^{T} \sum_{j=1}^{N_L} e_j^2(t),$$

where $E_j(w)$ is the sum of the squares of errors associated with pattern $t$. Minimization of $E$ is attempted by using a training algorithm to update the weights. The updated weight vector describes a new direction in which the weight vector will move in order to reduce the network training error. Efficient training algorithms have been proposed for trial and error based training, but it is difficult to use them when training with discrete weights [13].

A test problem-XOR [14] has been used for testing the functionality, which is a difficult classification problem and thus a good benchmark, and computer simulations have been developed to study the performance of the IPCHPMA. If $E \leq 0.01$, then the reported parameters for simulations that have reached solution are: succ. simulations succeeded out of 100 within the generation limit $\text{max\_num}$. The success rate was used to study the performance of the algorithm.

For all the simulations we used bipolar input and output vectors and hyperbolic tangent activation functions in both the hidden and output layer neurons. The weight population was initialized with random integers from the interval $[-W, W]$.

Now we consider the exclusive-OR (XOR) Boolean function. The XOR function maps two binary inputs to a single binary output. A 2-2-1 FNN (six weights and three biases, Fig. 3) was used for these simulations and the training was stopped when the value of the error function $E$, was $E \leq 0.01$ within $\text{max\_num} = 100$. The population size was $NP = 18$. The results of the simulation are shown in Table 3. Their average iteration was 4.06 iterations and the average run time was 0.448 second for 100 runs. In table 3, DE1~6 methods appeared in Literature [3].
Table 3 The Results of Simulations for the XOR Problem

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>DE1</th>
<th>DE2</th>
<th>DE3</th>
<th>DE4</th>
<th>DE5</th>
<th>DE6</th>
<th>IPHCPMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>succ.</td>
<td>91%</td>
<td>80%</td>
<td>95%</td>
<td>82%</td>
<td>81%</td>
<td>93%</td>
<td>100%</td>
</tr>
</tbody>
</table>

A typical weight vector after the end of the training process is \( w = (–10, –5, –10, –3, –4, 8, 7, 8, –4) \) and the corresponding value of the error function was \( E = 6.4902 \times 10^{-6} \). The six first components of the above vector are the weights and the remaining three are the biases. For this problem, IPHCPMA have shown excellent performance. The success rates are better than any other well-known continuous weight training algorithm, as far as we know.

7. Conclusions

In this paper, we designed three operators: simplified Hooke-Jeeves exploratory move method, population migration and population proliferation are in integer region. Then we proposed a novel hybrid coevolutionary PMA for integer programming (IPHCPMA). This algorithm avoids and overcomes the defect that the convergence of traditional methods has much relationship with initial values, whose parallelism satisfies the question of parallel solving integer programming in engineering technique.

References