IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 17, 2018, accepted September 26, 2018, date of publication October 5, 2018,
date of current version November 8, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2873636

Efficient Hardware/Software Partitioning Based on a Hybrid Algorithm

TAO ZHANG, (Member, IEEE), XIN ZHAO, AND XUAN LI
[1]School of Electrical and Information Engineering, Tianjin University, Tianjin 300072, China
[2]Texas Instruments DSP Joint Laboratory, Tianjin University, Tianjin 300072, China

Corresponding author: Xin Zhao (zhaoxin_16@tju.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61350009.

ABSTRACT Complex embedded systems with multi-processing units are important platforms for running complex tasks. In the development of complex embedded systems, hardware/software partitioning plays an important role. In practical application, there are many dynamic tasks which require the hardware/software partitioning to be done in real time. It is necessary to design efficient algorithms to do this. In this paper, the shuffled frog leaping algorithm (SFLA) and the greedy algorithm (GRA) are used to generate a hybrid algorithm named SFLA-GRA. On the basis of the SFLA, the SFLA-GRA uses the greedy idea to terminate invalid iterations and adjust the search step size. By these greedy strategies, the algorithm can be effectively accelerated. Experimental results show that compared with the other swarm intelligence (SI) algorithms, the efficiency of the proposed algorithm has been improved.

INDEX TERMS Efficient hardware/software partitioning, shuffled frog leaping algorithm, greedy algorithm, hybrid algorithm.

I. INTRODUCTION

With the development of science and technology, the scale and complexity of tasks are increasing. In order to ensure the normal operation of complex tasks, it is necessary to improve the processing performance of the embedded systems. Increasing the number and type of the processing units can effectively improve the performance of the embedded systems. Therefore, complex embedded systems with multi-processing units have become the main platforms for running complex tasks. Hardware/software partitioning is an important part of the development of complex embedded systems. A complex task consists of many subtasks, hardware/software partitioning assigns these subtasks to software units or hardware units. It is obvious that different task assignment schemes would bring different execution results and finding the optimal scheme is the goal of hardware/software partitioning.

For hardware/software partitioning, there are two important parts: the hardware/software partitioning models and the hardware/software partitioning algorithms. To build a model, the architecture of the system and the objectives of the hardware/software partitioning problem should be considered. In aspect of the architecture, some works [1], [2] are based on the architecture consists of a single software unit and a signal hardware unit while some works [3], [4] are based on the architecture consists of different types and quantities of software and hardware units. The system to be partitioned is generally given in the form of a task graph, or a set of task graphs [5], and the parallelism and the communication between two processing units usually should be considered. In aspect of the objectives, a task of hardware/software partitioning may have one [6] or multiple [7] objectives. The common objectives include minimizing the execution time of tasks on the system, minimizing the hardware area of the system, minimizing the power of the system, minimizing the communication overhead of the system. In different codesign environments, the models would be different [5].

Algorithms used to solve hardware/software partitioning problems can be divided into two categories: exact algorithms and heuristic algorithms. Exact algorithms [7]–[9] can accurately find the best solution of the problem. But hardware/software partitioning is an NP-hard problem [10], when its scale is large, using exact algorithms would be difficult and time-consuming. Because heuristic algorithms can obtain the optimal solutions or near-optimal solutions within a reasonable time [11], they have become the main way to solve hardware/software partitioning problems [12]–[15]. As an important class of heuristic algorithms,

SI algorithms [16]–[21] are flexible and highly adaptable [22]. These algorithms are proved to be more suitable for solving complex problems [6].

In recent years, heuristic algorithms are more and more applied in hardware/software partitioning. On the basis of the original heuristic algorithms, a lot of methods are proposed to improve the solution quality of hardware/software partitioning. Such as the two software-oriented and the second hardware-oriented greedy heuristic algorithms [23], the supervised shuffled frog leaping algorithm [24], the position disturbed particle swarm optimization with invasive weed optimization [25]. These algorithms effectively improve the search ability and are able to obtain the better solutions. In practical application, most tasks are dynamic, the hardware/software partitioning usually should be done in real time. Therefore, improving the efficiency of hardware/software partitioning without decreasing their solution quality is another important research content. Based on different ideas, some studies have been done to reduce the running time of hardware/software partitioning. Based on the method of model reduction, work [26] proposes the graph reduction techniques to reduce the design space for hardware/software partitioning. Based on the method of hardware acceleration, work [27] designs the parallel algorithm which can be accelerated by GPUs. In addition to these two methods, the most common way to reduce the running time is improving the performance of the algorithm itself.

This paper would mainly research on hardware/software partitioning algorithms, it would use SI algorithms to solve the problems of hardware/software partitioning. SI algorithms are inexact algorithms, it is difficult to judge whether the solutions obtained by them are the best solutions. They search for optimal solutions through multiple iterations. These iterations are composed of valid iterations that improve the quality of solutions and invalid iterations that reduce the efficiency of algorithms. Increasing the total number of iterations can improve the solution quality, but it would also increase the running time of the algorithms. Therefore, when SI algorithms are used in hardware/software partitioning, they should be improved to reduce the running time. There are two common ways to improve the SI algorithms: generating new algorithms by the fusion of multiple algorithms, designing the adaptive search strategy for the algorithms. Based on the two ways, some improved algorithms have been proposed in recent years [28]–[31]. Compared with the original algorithms, these improved algorithms can effectively improve the quality of the output solutions. But because most improved algorithms mainly focus on improving the solution quality, their computational efficiency sometimes cannot be improved effectively.

Based on the above analysis, this paper focuses on studying efficient algorithms which can be used in hardware/software partitioning. We first select SFLA which has good performance when applied in hardware/software partitioning as the basic algorithm. Then, based on the idea of the fusion of multiple algorithms, SFLA is hybridized with GRA [32] to generate an improved algorithm named SFLA-GRA. Compared with SFLA, the hybrid algorithm can effectively reduce the number of invalid iterations and improve the algorithm efficiency.

The rest of this paper is organized as follows. In section 2, the related knowledge of hardware/software partitioning is shown. In section 3, the hybrid algorithm SFLA-GRA is proposed. In section 4, the experimental results are given and in section 5, we conclude our work.

II. HARDWARE/SOFTWARE PARTITIONING PROBLEM

In different codesign environments, the models would be different. Most of the hardware/software partitioning methods can work perfectly within their own codesign environments, but it is impossible to compare them, because of the large differences in their codesign environments and the lack of benchmarks [5]. Compared with the exact algorithms, SI algorithms have stronger adaptability, they can be more easily used to solve different models. In this paper, we mainly focus on studying the efficient algorithms of hardware/software partitioning. In order to facilitate the research of the algorithms, our work is based on a simplified model.

In this model, the architecture is made up by one software unit and multiple hardware units of the same type. In this architecture, tasks assigned to the hardware units can be executed concurrently while tasks assigned to the software unit must be executed sequentially. When there is communication between a software unit and a hardware unit or between two hardware units, the communication time is required.

Directed Acyclic Graph (DAG) is usually used to illustrate the hardware/software partitioning. The graphs before and after partitioning are shown in Figure 1.

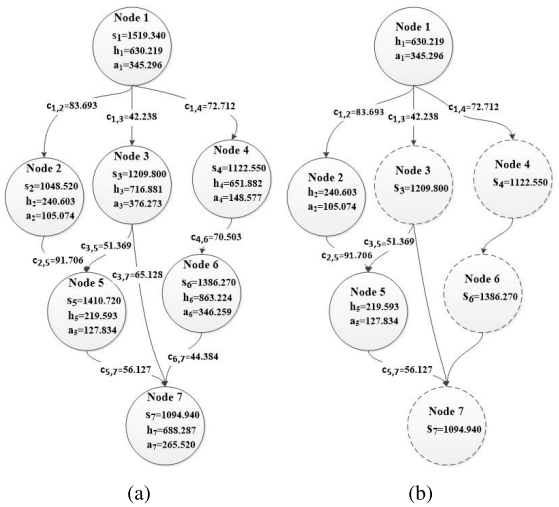


FIGURE 1. DAG of hardware/software partitioning. (a) Before partitioning. (b) After partitioning.

It can be seen in Figure 1(a), node i has three attributes: software execution time s_i, hardware execution time h_i, hardware area a_i. $c_{i,j}$ is the communication time between node i and j. Based on these attributes, tasks would be assigned to

software unit or hardware units. Figure 1(b) shows a result after partitioning, where node 3, 4, 6 and 7 are assigned to software and the other nodes are assigned to hardware.

The partitioning scheme can be encoded to an N-dimensional vector $X = [x_1, x_2, \ldots, x_N]^T$, where N is the number of task nodes, $x_i \in \{0, 1\}$, $x_i = 0$ represents task i is assigned to the software and $x_i = 1$ represents task i is assigned to hardware.

The optimization objective is minimizing the critical path which demonstrates the longest path. The critical path would determine the time required to execute the tasks on the embedded platform. The hardware area is set as a constraint. The optimization problem can be expressed by:

$$\begin{aligned} &\text{min: } T = \max\{TE(k)|0, 1 \ldots M\} \\ &\text{subject to: } \sum_{i=1}^{N} x_i \times a_i \leq A\_limit \end{aligned} \tag{1}$$

Where $TE(k)$ represents the completion time of the kth path, M is the number of paths. A_limit denotes the constraint value of hardware area.

III. SFLA-GRA

A. SFLA AND GRA

1) SFLA

SI algorithms usually have similar characteristics, such as easy to be realized and strong global optimization ability. But for different problems, different SI algorithms usually have different performances. Our work is based on the original SI algorithms. In our previous works [6], [33], we have used different original SI algorithms to solve hardware/software partitioning problems and found that SFLA has the highest efficiency. Therefore, SFLA is selected to generate the effective algorithm. SFLA is inspired by the foraging behavior of frog population. In this algorithm, the position of a frog represents a solution and the searching for optimal solutions is based on multiple iterations. Before the iterations, some solutions would be generated randomly to form the initial population. There are three steps in each iteration: grouping, updating and shuffling. Frogs are divided into several groups in the step of grouping and shuffled together in the shuffling step. Updating is the most important part of SFLA. In this step, the worst solution of each group would move to better solutions to update itself.

In order to further analyze the algorithm process of SFLA, we use SFLA to solve hardware/software partitioning problem with 500 task nodes, the maximum number of iterations is set to 1500 and the fitness curve of solutions is shown in Figure 2.

It can be seen in Figure 2, invalid iterations account for a large proportion of the total iterations, and these invalid iterations obviously reduce the running speed of the algorithm. In addition, when there are a large number of successive invalid iterations, the solutions usually have reached a relatively high quality. Terminating the algorithm when the number of successive invalid iterations arrives at a threshold

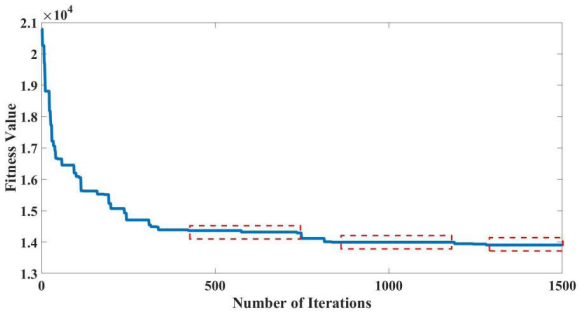


FIGURE 2. The fitness curve of SFLA.

is a common method to reduce the running time. But after the large number of successive invalid iterations, there may also be valid iterations to improve the solution quality. Therefore, this termination condition may make the algorithm miss the higher quality solutions. Based on these phenomena, if the valid iterations can appear after a small number of successive invalid iterations and the suitable algorithm termination condition is set, the running time of the algorithm would be effectively reduced.

2) GRA

GRA is designed by the greedy idea. It is one of the simplest heuristic algorithms. GRA usually starts from an initial solution which is generated randomly and then updates the solution iteratively. At each iteration, some alternative solutions would be generated by the moving of the current solution, and the distances between different alternative solutions and the current solution are the same. The algorithm would choose the optimal alternative solution to replace the current one based on their profits. Because the choice of each iteration is made just based on the current profits, the output solution of GRA is usually a local optimal solution.

In order to further analyze the algorithm process of GRA, we use GRA to solve the same hardware/software partitioning problem which is solved by SFLA in the previous part, the algorithm would be terminated when there is no better alternative solution can be chosen to replace the current one. The fitness curve of solutions is shown in Figure 3.

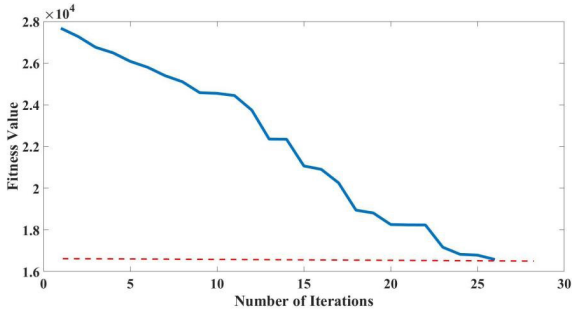


FIGURE 3. The fitness curve of GRA.

It can be seen from Figure 3, GRA is terminated after 26 iterations and the quality of the solution obtained now has been obviously improved compared with the initial solution.

TABLE 1. Pseudo code of function GRA().

Function: GRA()

begin
1 $X_{temp} = X_{BEST}$; /* Use the current optimal solution in the population as the initial solution */
2 $GRA\_Iter\_Num = 0$; /* Initialize the current number of iterations of GRA*/
3 **While** $GRA\_Iter\_Num < Max\_GRA\_Iter$ /*When the maximum number of iterations is not arrived */
3.1 $Fit_{temp} = Cal\_Fit(X_{temp})$ /*Get the fitness of X_{temp}*/
3.2 $i = 0$;
3.3 **do**
$i = i + 1$;
$X_{temp,i}(i) = !X_{temp,i}(i)$; /*Change the value of the ith dimension of $X_{temp,i}$ */
$Fit_{temp,i} = Cal\_Fit(X_{temp,i})$;
While $Fit_{temp,i} \geq Fit_{temp} \&\& i < Node\_Num$ /*When $X_{temp,i}$ is not better than X_{temp} and i is less than the number of task nodes */
3.4 **If** $Fit_{temp,i} \geq Fit_{temp}$ **then**
break; /* Terminate GRA() if a better solution is not found */
Else
$X_{temp} = X_{temp,i}$
End if
3.5 $GRA\_Iter\_Num = GRA\_Iter\_Num + 1$;
End while
4 $X_{output} = X_{temp}$ /*Output the best solution*/
End.

GRA can get better solutions in each iteration and keep a fast descending speed. These show that GRA has the advantage of high efficiency. But when the algorithm is terminated, the quality of the output solution is much worse than that of SFLA, which proves that it is easy for GRA to fall into local optimum.

B. THE HYBRID ALGORITHM

To ensure the quality of the solutions, SFLA with strong global optimization ability should be used in hardware/software partitioning problems. Although SFLA has higher efficiency compared with some SI algorithms, its efficiency still should be further improved. To achieve this goal, there are three methods:

1) Reducing the invalid iterations. Invalid iterations account for a large proportion of the total iterations, so reducing the invalid iterations can obviously reduce the number of iterations.
2) Terminating the algorithms with the effective termination conditions. If the algorithm is terminated too early, the quality of the solutions would be poor, while if the algorithm is terminated too late, the running time of the algorithm would be long. So it is important to find the effective termination conditions.
3) Accelerating the search efficiency of each iteration. When the search efficiency of each iteration is improved, the total running time would be reduced.

Based on the above analysis, the greedy idea of GRA is introduced into SFLA to generate a new algorithm SFLA-GRA. SFLA has strong global optimization ability and GRA has high efficiency. Therefore, the fusion of the two algorithms can obtain their respective advantages. The three most important parts of the hybrid algorithm are shown as follows:

1) TERMINATING INVALID ITERATIONS OF SFLA WITH GRA.

When the algorithm starts, SLFA-GRA would be run in accordance with the steps of SFLA. During this process, the number of successive invalid iterations would be calculated and if it is higher than the threshold *Inv_Limit*, GRA function would be run. Because GRA has the ability to get better solutions in one iteration, it would help the algorithm to terminate the invalid iterations. When the GRA can no longer find a better solution or reaches its maximum number of iterations, it would be stopped and the SFLA would be continued. The pseudo code of the process of GRA function is shown in Table 1.

2) TERMINATING THE ALGORITHM WITH A NEW TERMINATION CONDITION.

For SFLA, when a large number of successive invalid iterations appears, there is a big possibility that the obtained solution is near to the best solution. For GRA, when a better solution can't be obtained, the current obtained solution is at least a local optimal solution. Therefore, if neither SFLA (with a certain number of iterations) nor GRA can find a better solution, there would be a larger probability that the current solution is equal to or close to the best solution in the solution space. Therefore, after a certain number of successive invalid iterations, GRA function would be run, if a better solution still can't be found by GRA function, the algorithm would be terminated.That is the termination condition of SFLA-GRA.

3) ACCELERATING THE SEARCH WITH GREEDY STEP SIZE.

In SFLA, a bad solution moves toward a better one to find a new solution. The generation process of a new solution is

TABLE 2. The pseudo code of SFLA-GRA.

Algorithm SFLA-GRA

begin
1 $Population\_size = P, Group\_num = G, Max\_repeat\_num = MR, Inv\_LIM = IL$;
2 Init_frogs(); /*Initialize frog population*/
3 **For** $repeat\_num = 1$ to MR **do**
  3.1 Cal_Fit(all_solutions); /*Calculate the fitness of all solutions*/
  3.2 Group();/*Group solutions*/
  3.3 **For** $i = 1$ to G **do**
    $X_{i,temp}$=$X_{i,worst} + Rand * Greedy\_step * (X_{i,best} - X_{i,worst})$
    $Fit_{i,temp}$ = Cal_Fit($X_{i,temp}$);
    If $Is\_valid(X_{i,temp})\&\&Fit_{i,temp} < Fit_{i,worst}$ **then**
      $X_{i,worst} = X_{i,temp}$;
    Else
      $X_{i,temp}$=$X_{i,worst} + Rand * Greedy\_step * (X_{BEST} - X_{i,worst})$;
      $Fit_{i,temp}$ = Cal_Fit($X_{i,temp}$);
      If $Is\_valid(X_{i,temp})\&\&Fit_{i,temp} < Fit_{i,worst}$ **then**
        $X_{i,worst} = X_{i,temp}$;
      Else
        do
          $X_{i,temp}$=$L\_bound + Rand * (U\_bound - L\_bond)$;
        While !Is_valid($X_{i,temp}$)
        $X_{i,worst} = X_{i,temp}$;
      End if
    End if
  End for
3.4 Shuffle() /*Shuffle all frogs*/
3.5 $X_{BEST} = Get\_best(all\_solutions)$; /*Get the best solution in the population*/
  $X_{WORST} = Get\_worst(all\_solutions)$; /*Get the worst solution in the population*/
3.6 $Inv = Cac\_inv()$; /* Calculate the number of successive invalid iterations*/
  If $Inv > IL$ **then**
    GRA(); /* Running the GRA function*/
    $X_{WORST} = X_{output}$; /* Replace the worst solution with the output solution*/
  End if
3.7 **If** Cal_Fit(X_{output}) $\geq$ Cal_Fit(X_{BEST}) **then**
    break;/* Terminate the algorithm*/
  End if
4 Output solution;
End.

shown as follows:

$$\underbrace{\begin{bmatrix} x^1_{new} \\ x^2_{new} \\ \vdots \\ x^N_{new} \end{bmatrix}}_{(X_{new})} = \underbrace{\begin{bmatrix} x^1_{bad} \\ x^2_{bad} \\ \vdots \\ x^N_{bad} \end{bmatrix}}_{(X_{bad})} + \underbrace{\begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix}}_{(Rand)} \times Step \times \Big(\underbrace{\begin{bmatrix} x^1_{better} \\ x^2_{better} \\ \vdots \\ x^N_{better} \end{bmatrix}}_{(X_{better})} - \underbrace{\begin{bmatrix} x^1_{bad} \\ x^2_{bad} \\ \vdots \\ x^N_{bad} \end{bmatrix}}_{(X_{bad})} \Big) \tag{2}$$

Where X_{new}, X_{bad} and X_{better} represent the new, the bad and the better solutions respectively. N is the number of task nodes. *Rand* is an N-dimensional vector composed of random values between 0 and 1. *Step* is the search step size of SFLA.

To accelerate the algorithm, greedy step size is proposed in SFLA-GRA. When x^i_{bad} is 0 and x^i_{better} is 1, the greedy step size is denoted by $g^i_{0,1}$, when x^i_{bad} is 1 and x^i_{better} is 0, the greedy step size is denoted by $g^i_{1,0}$, $g^i_{0,1}$ and $g^i_{1,0}$ can be calculated by:

$$\begin{aligned} C^i_{0,1} &= (s_i - h_i)/a_i \\ C^{\max}_{0,1} &= \max\{C^i_{0,1} | i = 1, 2 \cdots N\} \\ C^{\min}_{0,1} &= \min\{C^i_{0,1} | i = 1, 2 \cdots N\} \\ g^i_{0,1} &= mis + (mas - mis) \times (C^i_{0,1} - C^{\min}_{0,1})/(C^{\max}_{0,1} - C^{\min}_{0,1}) \\ g^i_{1,0} &= mis + mas - g^i_{0,1} \end{aligned} \tag{3}$$

Where $C^i_{0,1}$ is the profit when x^i_{bad} is changed from 0 to 1. s_i, h_i and a_i are the software execution time,the hardware execution time and the hardware area of task node i. *mas* and *mis* are the maximum search step size and the minimum search step size. The generation of a new solution based on the greedy step size is shown as:

$$\begin{aligned} Greedy\_step &= \begin{bmatrix} g^1_{0,1} \times x^1_{better} + g^1_{1,0} \times (!x^1_{better}) \\ g^2_{0,1} \times x^2_{better} + g^2_{1,0} \times (!x^2_{better}) \\ \vdots \\ g^N_{0,1} \times x^N_{better} + g^N_{1,0} \times (!x^N_{better}) \end{bmatrix} \\ X_{new} &= X_{bad} + Rand \times Greedy\_step \times (X_{better} - X_{bad}) \end{aligned} \tag{4}$$

The pseudo code of SFLA-GRA is shown in Table 2. Where $X_{i,worst}$, $X_{i,best}$, X_{BEST} are the worst solution in group i, the best solution in group i and the best solution in the population, respectively. U_bond and L_bond are the upper and lower bounds of the solution space. In order to further

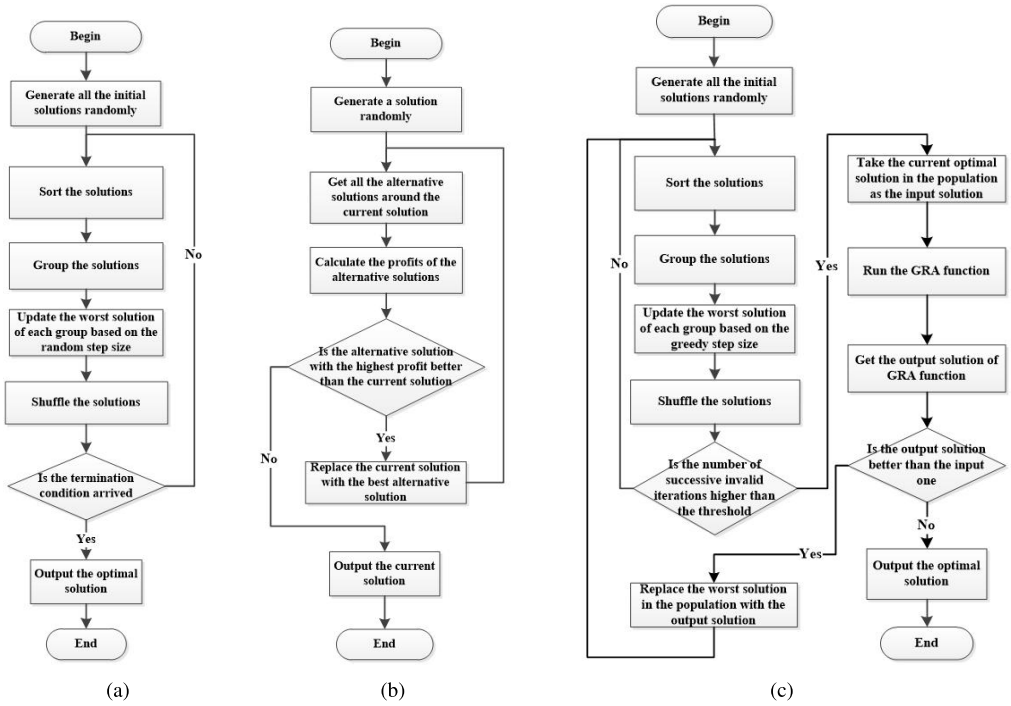


FIGURE 4. Flow charts of SFLA, GRA and SFLA-GRA. (a) SFLA. (b) GRA. (c) SFLA-GRA.

TABLE 3. Comparison results under the first termination condition.

Number of nodes	200		300		500		700		1000	
	Q(%)	R(%)	Q(%)	R(%)	Q(%)	R(%)	Q(%)	R(%)	Q(%)	R(%)
SFLA-GRA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SFLA	2.1	53.0	1.1	8.8	6.54	-39.0	9.2	-34.9	1.0	-30.9
GRA	35.9	-76.3	40.2	-68.5	53.5	-45.1	46.6	-42.1	38.2	-33.4
ABC	14.6	446.1	22.4	299.9	35.0	128.9	33.4	-1.86	36.1	115.9
PSO	8.5	59.5	11.6	20.6	15.4	-26.9	15.3	-35.6	9.4	-20.1
ASFA	0.9	1310.1	0.3	1407.1	1.4	709.4	1.7	827.8	0.5	603.4
GA	9.9	204.1	9.9	272.7	16.8	47.5	14.9	65.6	5.5	191.8
APABC	1.5	198.5	2.5	276.7	8.2	24.1	10.7	4.9	2.6	28.7
IPSO	4.2	1093.3	8.5	764.8	8.9	680.0	11.4	531.3	10.4	556.0
MSFLA	4.1	92.7	5.5	14.7	8.0	30.7	7.2	4.8	2.2	27.1
IGA	0.0	554.7	0.0	912.2	0.6	1205.2	4.1	436.2	0.5	486.3

compare the three algorithms, the flow charts of SFLA, GRA and SFLA-GRA are show in Figure 4.

IV. SIMULATION RESULTS

The proposed algorithm SFLA-GRA is simulated in C++ on an Intel Core i5-6400, 2.70GHz CPU, 8.00GB of RAM, running Microsoft Windows 10 operating system. Random instances are generated by TGFF tool to test the performance of the algorithms.These instances include five task sets with different scale (200 task nodes, 300 task nodes, 500 task nodes, 700 task nodes and 1000 task nodes). The constraint value is set to 1/2 of the maximum area (the total hardware area when all tasks are assigned to hardware units). To test the performance of SFLA-GRA, it is compared with the two original algorithms: SFLA and GRA, four original SI algorithms: Artificial Bee Colony Algorithm (ABC), Artificial Fish Swarm Algorithm (ASFA), Genetic Algorithm (GA) and Particle Swarm Optimization Algorithm (PSO), and four improved SI algorithms proposed in recent years: Mnemonic Shuffled Frog Leaping Algorithm(MSFLA) [34], Improved Particle Swarm Optimization(IPSO) [35], novel Artificial Bee Colony Algorithm(called APABC) [36] and Improved Genetic Algorithm(IGA) [37]. SFLA-GRA is terminated based on the termination condition introduced in section III. GRA is terminated when no better alternative solution can be found. Other algorithms are terminated by two common termination conditions. The first condition is that there are 150 successive invalid iterations. The second condition is that the number of iterations reaches 1500. The simulation results are averaged by 10 runs.

A. COMPARISON BASED ON THE SOLUTION QUALITY AND RUNNING TIME

Table 3 and Table 4 show the comparison results of the 11 algorithms in terms of solution quality and running time. In Table 3, SFLA and other SI algorithms used for

TABLE 4. Comparison results under the second termination condition.

Number of nodes	200		300		500		700		1000	
	Q(%)	R(%)	Q(%)	R(%)	Q(%)	R(%)	Q(%)	R(%)	Q(%)	R(%)
SFLA-GRA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SFLA	0.5	215.9	0.7	128.1	1.4	30.4	2.3	16.1	0.0	48.0
GRA	35.9	-76.3	40.2	-68.5	53.5	-45.1	46.6	-42.1	38.2	-33.4
ABC	12.1	1730.3	15.2	1636.7	31.6	672.5	29.1	469.2	24.2	2060.6
PSO	8.5	930.3	11.6	616.5	15.4	327.1	15.3	210.2	9.4	1537.2
ASFA	0.0	4213.9	0.0	3809.1	0.0	1736.1	0.0	1282.5	0.0	2527.6
GA	3.9	1056.1	6.0	742.1	11.4	378.3	12.0	249.1	2.7	446.6
APABC	0.2	538.0	0.6	850.5	4.9	230.6	7.5	142.4	0.4	229.0
IPSO	3.8	2449.1	4.6	1847.4	7.3	1194.3	8.0	901.6	3.1	1452.9
MSFLA	1.3	368.4	1.4	218.7	3.9	147.8	4.8	82.2	0.4	179.7
IGA	0.0	3322.4	0.0	2586.3	0.0	1757.8	0.3	1327.3	0.0	2025.8

comparison are terminated by the first termination condition(150 successive invalid iterations). In Table 4, SFLA and other SI algorithms used for comparison are terminated by the second termination condition(1500 iterations). Q and R are calculated by:

$$Q = \frac{fit_{comp} - fit_{sfla-gra}}{fit_{sfla-gra}} \times 100\%$$
$$R = \frac{run_{comp} - run_{sfla-gra}}{run_{sfla-gra}} \times 100\% \quad (5)$$

Where fit_{comp} and $fit_{sfla-gra}$ are the fitness values of the solutions obtained by the comparison algorithm and the SFLA-GRA, respectively. run_{comp} and $run_{sfla=gra}$ are the running time of the comparison algorithm and the SFLA-GRA, respectively. It can be seen from formula 5, when Q and R are positive values, the solution quality of SFLA-GRA is better than the comparison algorithm and the running time of SFLA-GRA is shorter than the comparison algorithm.

As shown in Table 3 and Table 4. GRA has the shortest running time among all the algorithms, but its solution quality is poor, which proves GRA is easy to fall into local optimum. Under the first termination condition, when the numbers of task nodes are 500, 700 and 1000, the running time of SFLA is less than SFLA-GRA, but its solution quality is also worse than SFLA-GRA. That because SFLA is terminated too early to get the high quality solution. Under the second condition, both the solution quality and the running time of SFLA are worse than SFLA-GRA, which proves SFLA-GRA effectively improves the efficiency while guaranteeing the quality of solutions.

Compared with four other original SI algorithms and four improved algorithms, SFLA-GRA can get higher quality solutions within a shorter running time in most cases. Under the first termination condition, the running time of ABC and PSO is sometimes shorter than SFLA-GRA, but their short running time leads to the poor solution quality. In some cases, ASFA and IGA can obtain the solutions whose quality is the same as SFLA-GRA , but ASFA and IGA are the two most time-consuming algorithms among the 8 comparison algorithms and their efficiency is much lower than SFLA-GRA.

Compared with the fist termination condition, when SI algorithms are terminated under the second termination condition, their running time would be longer but their quality would be higher. That is because the 150 successive invalid iterations are easy to appear before the total number of iterations reaches 1500, and the quality of the solutions obtained at this time usually has room for further improvement. This result also shows that it is important to reduce the invalid iterations and set the suitable termination conditions.

B. COMPARISON BASED ON THE FITNESS CURVES

The fitness curves of the 11 algorithms when they are used to solve the instances of 300 nodes and 700 nodes are shown in Figure 5.

It can be seen from Figure 5(a), GRA keeps the fastest descending speed and is terminated after a small number of iterations, but the solution quality of GRA is the worst. This further proves that GRA has high efficiency but easily falls into local optimum. In the early stage, the descending speeds of SFLA-GRA and SFLA are similar. But as the number of iterations increases, the large number of invalid iterations would reduce the descending speed of SFLA while SFLA-GRA would still keep a fast speed. That proves introducing GRA into SFLA can effectively terminate the invalid iterations.

It can be seen from Figure 5(b) and Figure 5(c) that with the number of iterations increases, the descending speeds of most algorithm curves are getting slower and slower. But the fitness curve of SFLA-GRA can keep a fast descending speed until the algorithm is terminated. In the early stage, some algorithms may have faster speeds than SFLA-GRA, but as the increase of their invalid iterations, the descending speeds of these algorithms would be exceeded by SFLA-GRA. It should also be noted that the time required for these algorithms(especially ASFA and IGA) to complete one iteration is much longer than that of the SFLA-GRA. It also can be seen from these curves that SLFA-GRA would usually be terminated in less than 700 iterations but the quality of its output solution is highest among all the algorithms. This proves the validity of the termination condition of SFLA-GRA.

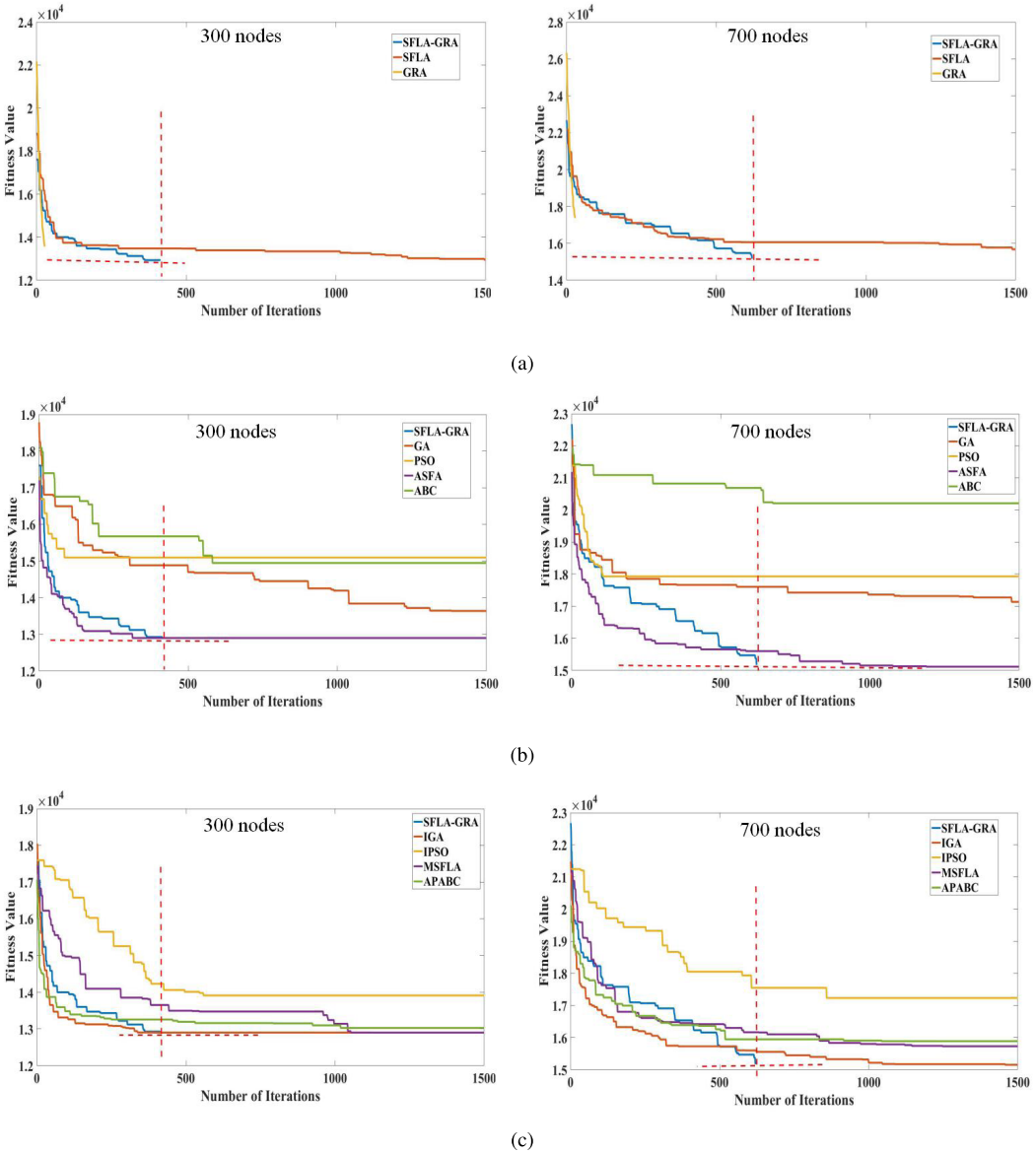


FIGURE 5. Fitness curves of 11 algorithms. (a) Fitness curves of SFLA-GRA, GRA and SFLA. (b) Fitness curves of SFLA-GRA and four original SI algorithms. (c) Fitness curves of SFLA-GRA and four improved SI algorithms.

V. CONCLUSION

In this paper, we first analyze the importance of improving the efficiency of the hardware/software partitioning. Then, based on the idea of the fusion of multiple algorithms, SFLA and GRA are hybridized to generate a hybrid algorithm SFLA-GRA. On the basic of SFLA, the new algorithm uses GRA function to terminate the invalid iterations and sets greedy search step size to further accelerate the algorithm. Experimental results show that the proposed algorithm SFLA-GRA outperforms all comparison algorithms, especially in terms of the algorithm efficiency.

There are some future research suggestions: 1) It is found during our research that the profit function is an important factor which affects the performance of GRA function. So it should be further studied. 2)Our proposed algorithm is based on the original SFLA, but there are many improved heuristic algorithms. Hybridizing GRA with these improved heuristic algorithms may further improve the efficiency. 3) The methods of model simplification and hardware acceleration can also be studied to accelerate the proposed algorithm.

REFERENCES

[1] J. Henkel and R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 2, pp. 273–289, Apr. 2001.

[2] M. López-Vallejo and J. C. López, "On the hardware-software partitioning problem: System modeling and partitioning techniques," *ACM Trans. Design Autom. Electron. Syst.*, vol. 8, no. 3, pp. 269–297, 2003.

[3] J. Wu, Q. Sun, and T. Srikanthan, "Algorithmic aspects for multiple-choice hardware/software partitioning," *Comput. Oper. Res.*, vol. 39, no. 12, pp. 3281–3292, 2012.

[4] S. B. Saoud, S. B. Saoud, and S. B. Saoud, "An efficient technique for hardware/software partitioning process in codesign," *Sci. Program.*, vol. 2016, May 2016, Art. no. 6382765.

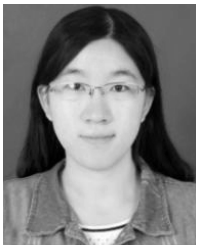
[5] J. Wu, T. Srikanthan, and G. Chen, "Algorithmic aspects of hardware/software partitioning: 1D search algorithms," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 532–544, Apr. 2010.
[6] T. Zhang, X. Zhao, X. An, H. Quan, and Z. Lei, "Using blind optimization algorithm for hardware/software partitioning," *IEEE Access*, vol. 5, pp. 1353–1362, 2017.
[7] W. Shi, J. Wu, S.-K. Lam, and T. Srikanthan, "Algorithmic aspects for bi-objective multiple-choice hardware/software partitioning," in *Proc. 6th Int. Symp. Parallel Archit., Algorithms Program. (PAAP)*, Jul. 2014, pp. 7–12, doi: 10.1109/PAAP.2014.42.
[8] W. Jigang, B. Chang, and T. Srikanthan, "A hybrid branch-and-bound strategy for hardware/software partitioning," in *Proc. 8th IEEE/ACIS Int. Conf. Comput. Inf. Sci.*, Jun. 2009, pp. 641–644.
[9] E. Demirel, N. Demirel, and H. Gökçen, "A mixed integer linear programming model to optimize reverse logistics activities of end-of-life vehicles in turkey," *J. Cleaner Prod.*, vol. 112, pp. 2101–2113, Jan. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959652614011226
[10] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 10, no. 1, pp. 136–156, 2005.
[11] Z. Li, Y. Liu, and G. Yang, "A new probability model for insuring critical path problem with heuristic algorithm," *Neurocomputing*, vol. 148, pp. 129–135, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231214009230
[12] P.-A. Mudry, G. Zufferey, and G. Tempesti, "A hybrid genetic algorithm for constrained hardware-software partitioning," in *Proc. IEEE Design Diagnostics Electron. Circuits Syst.*, Apr. 2006, pp. 1–6.
[13] L. Li, J. Sun, W. Li, Z. Lv, and F. Guan, "Hardware/software partitioning based on hybrid genetic and tabu search in the dynamically reconfigurable system," *Int. J. Control Autom.*, vol. 8, no. 1, pp. 29–36, 2015.
[14] N. Hou, F. He, Y. Zhou, and H. Ai, "A GPU-based tabu search for very large hardware/software partitioning with limited resource usage," *J. Adv. Mech. Des., Syst., Manuf.*, vol. 11, no. 5, p. JAMDSM0060, 2017.
[15] M. Jemai, S. Dimassi, B. Ouni, and M. Abdellatif, "A metaheuristic based on tabu search for hardware-software partitioning," *Turkish J. Elect. Eng. Comput. Sci.*, vol. 25, pp. 901–912, Mar. 2016.
[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4, Nov. 1995, pp. 1942–1948.
[17] M. M. Eusuff and K. E. Lansey, "Optimization of water distribution network design using the shuffled frog leaping algorithm," *J. Water Sources Planning Manage.*, vol. 129, no. 3, pp. 210–225, 2003.
[18] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, Apr. 2007.
[19] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, vol. 13. Reading, MA, USA: Addison-Wesley, no. 7, 1989, pp. 2104–2116.
[20] X. Li, Z. Shao, and J. Qian, "An optimizing method based on autonomous animats: Fish-swarm algorithm," *Syst. Eng.-Theory Pract.*, vol. 22, no. 11, pp. 32–38, 2002.
[21] Y. Tan and Y. Zhu, "Fireworks algorithm for optimization," in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, and K. C. Tan, Eds. Berlin, Germany: Springer, 2010, pp. 355–364.
[22] X.-S. Yang, "Recent advances in swarm intelligence and evolutionary computation," in *Studies in Computational Intelligence*. Cham, Switzerland: Springer, 2015.
[23] J. W. Tang, Y. W. Hau, and M. N. Marsono, "Hardware/software partitioning of embedded system-on-chip applications," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Oct. 2015, pp. 331–336.
[24] X. Zhao, T. Zhang, X. An, and L. Fan, "An improved blind optimization algorithm for hardware/software partitioning and scheduling," in *Advances in Swarm Intelligence*, Y. Tan, Y. Shi, and Q. Tang, Eds. Cham, Switzerland: Springer, 2018, pp. 225–234.
[25] X.-H. Yan, F.-Z. He, and Y.-L. Chen, "A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization," *J. Comput. Sci. Technol.*, vol. 32, no. 2, pp. 340–355, 2017.
[26] G. Jiang, J. Wu, S.-K. Lam, T. Srikanthan, and J. Sun, "Algorithmic aspects of graph reduction for hardware/software partitioning," *J. Supercomput.*, vol. 71, no. 6, pp. 2251–2274, 2015.
[27] Y. Zhou, F. He, and Y. Qiu, "Dynamic strategy based parallel ant colony optimization on GPUs for TSPs," *Sci. China Inf. Sci.*, vol. 60, p. 068102, Jun. 2017.
[28] J. Luo, X. Li, M.-R. Chen, and H. Liu, "A novel hybrid shuffled frog leaping algorithm for vehicle routing problem with time windows," *Inf. Sci.*, vol. 316, pp. 266–292, Sep. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515002571
[29] W.-H. Ho, J.-T. Tsai, J.-H. Chou, and J.-B. Yue, "Intelligent hybrid taguchi-genetic algorithm for multi-criteria optimization of shaft alignment in marine vessels," *IEEE Access*, vol. 4, pp. 2304–2313, 2016.
[30] K. K. Bhattacharjee and S. P. Sarmah, *Modified Swarm Intelligence Based Techniques for the Knapsack Problem*. Norwell, MA, USA: Kluwer, 2017.
[31] R.-I. Chang, H.-M. Hsu, S.-Y. Lin, C.-C. Chang, and J.-M. Ho, "Query-based learning for dynamic particle swarm optimization," *IEEE Access*, vol. 5, pp. 7648–7658, 2017.
[32] C. Cerrone, R. Cerulli, and B. Golden, "Carousel greedy: A generalized greedy algorithm with applications in optimization," *Comput. Oper. Res.*, vol. 85, pp. 97–112, Sep. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054817300801
[33] T. Zhang, X. Zhao, Y.-K. Yu, and X. Cai, "Reserch on hardware/software partitioning method of improved shuffled frog leaping algorithm," *J. Signal Process.*, vol. 9, pp. 1055–1061, Sep. 2015.
[34] H.-B. Wang, K.-P. Zhang, and X.-Y. Tu, "A mnemonic shuffled frog leaping algorithm with cooperation and mutation," *Appl. Intell.*, vol. 43, no. 1, pp. 32–48, Jul. 2015, doi: 10.1007/s10489-014-0642-x.
[35] B. Yao, B. Yu, P. Hu, J. Gao, and M. Zhang, "An improved particle swarm optimization for carton heterogeneous vehicle routing problem with a collection depot," *Ann. Oper. Res.*, vol. 242, no. 2, pp. 303–320, Jul. 2016, doi: 10.1007/s10479-015-1792-x.
[36] L. Cui *et al.*, "A novel artificial bee colony algorithm with an adaptive population size for numerical function optimization," *Inf. Sci.*, vol. 414, pp. 53–67, Nov. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025517307545
[37] R. Zhang and J. Tao, "A nonlinear fuzzy neural network modeling approach using an improved genetic algorithm," *IEEE Trans. Ind. Electron.*, vol. 65, no. 7, pp. 5882–5892, Jul. 2018.

TAO ZHANG received the M.S. degree from the School of Electronic Information Engineering from Tianjin University, Tianjin, China, in 2001, and the Ph.D. degree from Tianjin University in 2004.

He is currently an Associate Professor with the School of Electrical and Information Engineering and the Tianjin University and Texas Instruments DSP Joint Laboratory, Tianjin University, China. His current interests include image, video, and acoustic signal processing, auditory model, speech enhancement, and hardware/software partitioning.

XIN ZHAO received the B.S. degree from the College of Electronic Information Engineering, North China Institute of Aerospace Engineering, Hebei, China, in 2014. He is currently pursuing the Ph.D. degree with the School of Electrical and Information Engineering, Tianjin University.

His research interests are evolutionary computation, optimization, and image processing.



XUAN LI received the B.S. degree from the School of Physics and Electronic-Electrical Engineering, Ningxia University, in 2016. She is currently pursuing the Ph.D. degree with the School of Electrical and Information Engineering, Tianjin University.

Her research interests are hardware/software partitioning and image processing.