A Discrete Multi-Objective Optimization Method for Hardware/Software Partitioning Problem Based on Cuckoo Search and Elite Strategy

Wei Xiong[1,2], Bing Guo[1*], Yan Shen[3], Wenli Zhang[1,4]

ABSTRACT

This paper attempts to provide a desirable solution to hardware/software partitioning of the embedded system. For this purpose, the author developed a discrete multi-objective optimization method based on the cuckoo search (CS) algorithm (MODCS) and the elite strategy of stratification and congestion degree comparison. Then, the MODCS was compared with two other typical simulation algorithms. The results show that the MODCS is superior to typical optimization algorithms in terms of many indices, including diversity, stability and generational distance (GD) of optimal solution. The superiority is positively correlated with the number of modules. The findings shed new light on the bionic optimization of hardware/software partitioning.

Key Words: Multi-objective Simulation, Hardware/Software Partitioning, Cuckoo Search, Elite Strategy, Generational Distance (GD), Pareto Diversity

DOI Number: 10.14704/nq.2018.16.5.1304 **NeuroQuantology 2018; 16(5):749-756**

Introduction

The human brain contains over ten billion neurons, each of which is attached with numerous dendrites capable of detecting sensory stimuli like sound, motion, touch, etc. With these dendrites, the neurons are connected into a vast network. If a person is particularly knowledgeable in an aspect, the neural network will be extremely dense in the corresponding brain region. Thus, his/her brain can transmit and process the relevant information in a rapid manner.

Inspired by the biological neural networks in human brain, the perceptron, the perceptron, the first model on artificial neuron, came into being in the 1960s. Since then, remarkable advancement has been made on the ANN, as the artificial neuron network (ANN) evolved from the perceptron, multilayer neural network, and deep neural network to artificial intelligent neural network.

Over the years, the ANN has been applied in many fields, including but not limited to image processing, speech recognition, pattern recognition, and automatic control. In particular, the ANN is often combined with the genetic algorithm (GA) and other bionic algorithms to deal with hardware/software partitioning of the embedded system.

For example, Guo *et al*., (2006) proposed hardware /software partitioning method for system on a chip (SoC) embedded design based simulated annealing algorithm to hardware partitioning. On the upside, this algorithm is simple and immune to the local minimum trap; on the downside, the algorithm consumes too much time, faces difficulty in setting the initial and annealing temperatures, and requires repeated experiments. Zhang *et al*., (2008) adopted artificial immune principles for

Corresponding author: GuoBing
Address: [1]Computer Science College, Sichuan University, Chengdu 610064, China; [2]Leshan Vocational & Technical College, Leshan 614000, China; [3]School of Control Engineering, Chengdu University of Information Technology, Chengdu 610225, China; [4]Chengdu Technological University, Chengdu 611730, China
e-mail ✉ guobing@scu.edu.cn
Relevant conflicts of interest/financial disclosures: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.
Received: 14 March 2018; **Accepted:** 13 April 2018

hardware/software partitioning; their strategy achieves good results but fails to consider the power and area constraints of objective function. Wiang *et al*., (2002) successfully resolved hardware/software partitioning with tabu search and the system of reward and punishment. Through intelligent algorithm optimization, Savage, M. J. W. *et al*., (2004) treated the problem in the extended genetic algorithm etc.

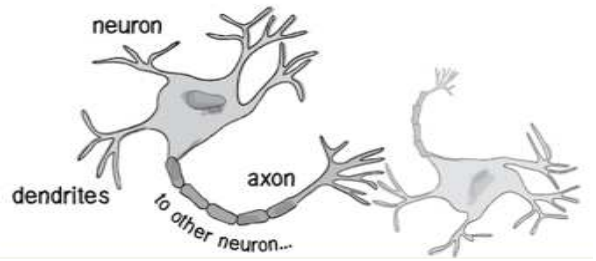

Figure 1. Neuron

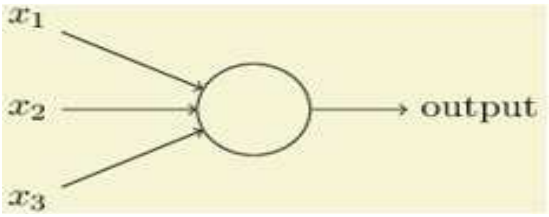

Figure 2. Artificial neuron

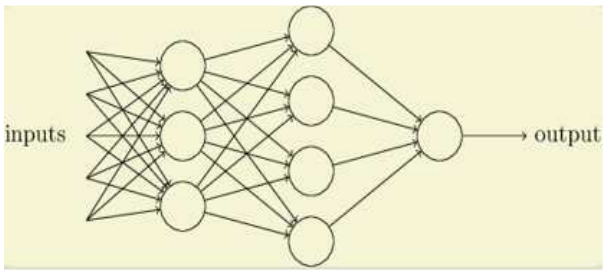

Figure 3. Multilayer neural network

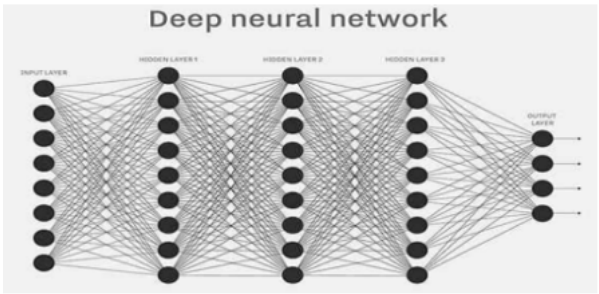

Figure 4. Deep neural network

In light of the above, this paper presents a discrete multi-objective optimization method based on the cuckoo search (CS) algorithm (MODCS) and the elite strategy of stratification and congestion degree comparison, aiming to provide a desirable solution to hardware/software partitioning of the embedded system.

Problem Model

Hardware/software partitioning, an important issue of hardware/software co-design, is an NP-hard problem (Garey and Johnson, 1979). The goal of the problem is to determine whether a system module should be hardware or software, without violating the system requirements on runtime, cost, area, reliability, power consumption, and communication overhead. Both hardware and software modules have their merits and defects. The hardware option features fast operating speed and a high cost, while the software option brings about a low operating cost and a slow speed. Here, the author attempted to achieve the optimal runtime and power consumption under the constraints of cost, embedded area and communication overhead. The target problem is, in essence, a discrete multi-objective optimization problem with constraints.

For a multi-objective optimization problem, it is unlikely to optimize the multiple objectives at the same time. The optimization of one objective may dampen the performance of the other objectives. Taking hardware/software partitioning for example, the minimization of runtime may come at the cost of high power consumption. Therefore, trade-offs are unavoidable to solving multi-objective optimization problems.

Unlike single-objective optimization, no single solution exists that simultaneously optimizes each objective for a multi-objective optimization problem. Instead, there exists a possibly infinite number of Pareto optimal solutions. A solution is called the optimal solution or non-dominated optimal solution.

Definition: For any point y in the search space, it is a non-dominated optimal solution if and only if there exists no i(i=1,2,3...n) in the search space that satisfies $f_i(x) < f_i(y)$. The collection of all such solutions is called the Pareto optimal set of multi-objective optimization problem.

For simplicity, the hardware/software partitioning problem was illustrated with a directed acyclic graph (DAG) (V, E) below.

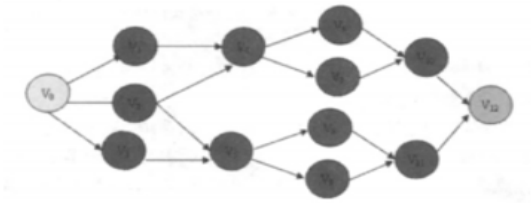

Figure 5. DAG of hardware/software partitioning

In Figure 5, the DAG is formed by a collection of vertices V_i and directed edges, each edge connecting onevertex to another, such t hat there is no way to start at some vertex and foll ow a sequence of edges that eventually loops back to it again. The set of all vertices is denoted as V. It is assumed that the system contains N vertices. For vertex V_i ($1 \le i \le N$), the variable $w_i \in \{0,1\}$ indicates the selection of hardware (w_i=1) or software (w_i=0). For the option of hardware, the runtime, embedded area, cost, power consumption and communication overhead are denoted as th_i, ah_i, ch_i, ph_i, and $comh_i$, respectively; for the option of software, the runtime, power consumption and communication overhead are denoted as ts_i, cs_i, and $coms_i$, respectively.

Let *Time* and *Pow* be the runtime and power consumption of the system, respectively. Then, the goals of our optimization problem: the minimal runtime and power consumption can be expressed as:

$$Minimize \quad Pow = \sum_{i=1}^{N} w_i * ph_i + (1 - w_i) * ps_i \tag{1}$$

$$Minimize \quad Time = \sum_{i=1}^{N} w_i * th_i + (1 - w_i) * ts_i \tag{2}$$

Suppose the maximum runtime and maximum power consumption of the system are U_{Time} and U_{Pow}, respectively. Then, the runtime and power consumption constraints can be expressed as:

$$Time \le U_{Time} \tag{3}$$

$$Pow \le U_{Pow} \tag{4}$$

Similarly, the cost, embedded area, and communication cost of the system are *Cost*, *Area* and *Com*, respectively, and the maximum cost, embedded area and communication overhead of the system are *UCost*, *UArea* and *UCom*, respectively. Then, the cost, embedded area, and communication overhead constraints can be expressed as:

$$Cost = \sum_{i=1}^{N} w_i * ch_i + (1 - w_i) * cs_i \le U_{Cost} \tag{5}$$

$$Area = \sum_{i=1}^{N} w_i * ah_i \le U_{Area} \tag{6}$$

$$Com = \sum_{i=1}^{N} w_i * comh_i + (1 - w_i) * coms_i \le U_{Com} \tag{7}$$

Our goal is to find the Pareto optimal solutions of runtime and power consumption under the constraints of cost, embedded area and communication overhead.

MODCS

DC search

There are many traditional methods to solve multi-objective optimization problem, such as linear weighting, efficiency coefficient method and stratified sequencing. With the development of Bionic algorithm (Deb, 2001; Konak *et al*., 2006), recent years has seen the emergence of the multi-objective optimization algorithm based on swarm intelligence. Currently, some of the most popular and mature multi-objective optimization algorithms are non-dominated sorting genetic Algorithm (NSGA) (Srinivas and Deb, 1994), NSGA-II (Deb *et al*., 2002) and multi-objective particle swarm optimization (MOPSO) algorithm (Coello and Lechuga, 2002; Zhang *et al*., 2003). For instance, the NSGA-II is mainly implemented in the following steps:

1. Initialize the population P_0 randomly, rank P_0 with the NSGA, and initialize the rank value of every individual.

2. t=0;

3. Select individuals from P_t, and produce a new-generation population Q_t through overlapping and variance operation;

4. Combine P_t and Q_t to produce a composite population $Rt = P_t \cup Q_t$;

5. Rank R_t with the NSGA, and select N individuals to form a new-generation population P_{t+1} through displacement and elite selection; Return to Step 3 and repeat the above steps till the termination condition is fulfilled.

In this research, a discrete multi-objective optimization strategy, denoted as MODCS, was created based on the CS search algorithm.

Inspired by the parasitism and reproduction of cuckoos, the CS search is an emerging heuristic algorithm for optimization problem (Yang and Deb, 2009). In the algorithm, the Le flight (Barthelemy *et al*., 2008) is adopted to satisfy the heavy tailed probability distribution. As a result, the algorithm boasts a more effective random search than the GA and other swarm intelligence algorithms. Besides, it is known for its strong global search ability and limited number of parameters. Over the years, the CS search algorithm has been extensively used in areas like project scheduling, function optimization, structure optimization and integer programming (Yang and Deb, 2010; Yang, 2010). The basic steps of the classic CS search are as follows:

Begin

Objective function f(x),x=$(X_1,...X_d)^T$

1. Generate the initial population of n host nests x_i(i=1,2,...,n);

2. while(t<Maximum number of iterations or termination condition)

3. Get a cuckoo randomly and evaluate its fitness (fi) against Lévy flight;

4. Choose a nest j randomly;

5. If(f_i>f_j)

Replace j by the new solution;

end;

6. Abandon a part of inferior solutions and establish new solution;

7. Keep the optimal solutions;

8. Rank the solutions and find out the current optimal solution;

9. end while;

10. Display the result

Position update

In the MODCS, the position update of the search for host nest is realized by the DC search algorithm. The procedure is detailed as follows:

Let $W=(w_1,w_2,\ldots,w_N)$ be the set of individual cuckoos, that is, the positions of host nests. Then, x_i^t is used to indicate the position of host nest *i*, and the corresponding $x_{ij}^t \in \{0,1\}$ to indicate component *j*. Thus, the position can be updated in the following manner.

1) When $r=\text{Random}[0,1] \le pr$:

$$x_{ij}^{t+1}=\begin{cases}1, & Random[0,1]\le\dfrac{1}{1+\exp(-s)}\\ 0, & other\end{cases} \quad (8)$$

2) When r > pr

If $s \le 0$

$$x_{ij}^{t+1}=\begin{cases}0, & Random[0,1]\le 1-\dfrac{2}{1+\exp(-s)}\\ x_{ij}^{t}, & other\end{cases} \quad (9)$$

If $s > 0$

$$x_{ij}^{t+1}=\begin{cases}1, & Random[0,1]\le\dfrac{2}{1+\exp(-s)}-1\\ x_{ij}^{t}, & other\end{cases} \quad (10)$$

Where s is the step size of Lévy flight: $s=\alpha*\mu/|v|^{1/\beta}$, with $\mu\sim N(0,\sigma_\mu^2)$ and $v\sim N(0,1)$. Specifically, the σ_μ^2 can be expressed as:

$$\sigma_\mu^2=\left\{\frac{\Gamma(1+\beta)\sin(\pi\beta/2)}{\Gamma[(1+\beta)/2]2^{(\beta-1)/2}\beta}\right\}^{1/\beta} \quad (11)$$

In general,$\alpha=0.01$ and $\beta=1.5$.

The procedure of the MODCS is detailed below:

Step 1. Initialize the probability that the egg laid by a foreign bird can be discovered by the host bird *pa*, binary coding control parameter *pr*, population size *n*, maximum number of iterations *G*, and the parameter of the *t-th* iteration $t=0$. Generate *n* host nests as the initial population $X^0=\{x_1^0,x_2^0,\ldots,x_n^0\}$, and obtain the fitness of every individual x_i^0 $fit_i^0=(Pow(x_i^0),Time(x_i^0))$. Initialize the optimal position of host nest $\text{bestIndex}_i=x_i^0$ of every individual x_i^0 in the population and find the optimal fitness of every individual $\text{bestfit}_i=fit_i^0$.

Step 2. Terminate the algorithm if t>G.

Step 3. The cuckoo chooses a new host nest by Lévy flight. The position of the new nest x_i^{t+1} depends on (8)~(10). Under the given constraints, compare the fitness of the current x_i^{t+1} with the optimal fitness bestfit_i. If fit_i^{t+1} is better than bestfit_i, then $\text{bestfit}_i=fit_i^{t+1}$, $\text{bestIndex}_i=x_i^{t+1}$.

Step 4. Compare a random number with the probability that the egg laid by a foreign bird can be discovered by the host bird *pa*. If *r*>*pa*, perform crossover operation to search r= Random[0,1] new host nest, update the value of x_i^{t+1}, and compare it with the optimal fitness bestfit_i.

Step 5. Perform Steps 2~5 repeatedly and produce the population P_0.

Step 6. Initialize n=0.

Step 7. Perform crossover and mutation operations for the population P_0 generated from Steps 2~5.

Step 8. Produce a new parent generate by applying the elite strategy to offspring generation Q_n and parent generation P_n.

Step 9. Update the current iterations: $t = t + 1$, and return to Step2; Otherwise, go to Step 10;

Step 10. Rank P_{t+1} by the NSGA, and treat the top solution as the Pareto optimal solution.

Elite strategy

The elite strategy was introduced to Step 8 to enhance the accuracy of sampling space and optimization result. The strategy is usually implemented in four steps. First, combine the new population and parent generation into U_n, whose population size is 2N; Second, divide U_n into L layers of non-dominate solution set $Z_i(1 \leqslant i \leqslant L)$, and input Z_1, Z_2,... into the new parent population P_{n+1} ; Third, if k solutions Z_k are inputted and $\sum_{j=1}^{k}|Z_j| = N$, then terminate the operation; Fourth, if $\sum_{j=1}^{k}|Z_j| < N < \sum_{j=1}^{k+1}|Z_j|$, then compare the congestion degrees of the individuals in Z_{k+1}, and input the initial $\sum_{j=1}^{k+1}|Z_j| - N$ individuals into the new parent population P_{n+1}.

Stratification and congestion degree comparison are two main techniques of the elite strategy. Below is a description of each of the two techniques.

Stratification

The population is divided into a Pareto non-dominated set and a dominated set, such that the individuals of the non-dominated set are not dominated by any individual of the current or subsequent non-dominated set. For this purpose, the non-dominated individuals dominated by other individuals are deleted from the population in each iteration; this process is repeated until the entire population is replaced. The pseudo-code is as follows:

```
def fast_nondominated_sort(P):
F = []
for p in P:
  Sp = []
  np = 0
  for q in P:
  if p > q:  Sp.append(q)
  else if p < q: np += 1
  if np == 0:
  p_rank = 1
F1. append (p)
F. append(F1)
i = 0
while F[i]:
  Q = []
  for p in F[i]:
  for q in Sp:
  nq -= 1
  if nq == 0
  q_rank = i+2
  Q. append(q)
  F. append(Q)
  i += 1
```

Congestion degree comparison

The first step of congestion degree comparison is to calculate the degree of every individual. Let $W = (w_1, w_2, \ldots, w_N)$ be the collection of individuals in the population. For a given population, the congestion degree can be calculated in four steps. First, initialize the congestion degree of every individual $d_i=0$, and normalize the value of each objective function; then, rank the individuals for each objective O_j, and assume that the congestion degree of two individuals at the boundary of O_j is infinite, that is, $d_{Low}^{O_j} = d_{Upper}^{O_j} = \infty$; next, compute the congestion degree of individuals for O_j by $d_i^{O_j} = \left|f_{i-1}^{O_j} - f_{i+1}^{O_j}\right|$, with $f_{i-1}^{O_j}$ and $f_{i+1}^{O_j}$ being the values of the objective function of the individual before and after individual i, respectively; finally, calculate the congestion degree of individual i for all objectives: $d_i^{O_j} = \sum_{j=1}^{m}\left|f_{i-1}^{O_j} - f_{i+1}^{O_j}\right|$.

During the comparison of congestion degrees, individual i is considered better than individual j under any of the two conditions:

1) If the position of non-dominated layer individual i is better than that of individual j, i.e. $level_i < level_j$.

2) If individual i and individual j are located at the same non-dominated layer, and the congestion degree of individual i is higher than that of individual j, i.e. $level_i = level_j$ and $d_i > d_j$.

Comparison Experiments

Comparison indices

To compare performance of the MODCS with NSGA-II and MOPSO, generational distance (GD) and Pareto diversity (PD) were introduced to this research.

The GD measures the proximity between the Pareto optimal solution to be solved and the optimal Pareto solution obtained by brute-force search (hereinafter referred to as the authentic Pareto solution). The value of the GD is positively correlated with the physical proximity between *Pa* and *Pa**. This index is defined as:

$$GD = \frac{1}{|Pa|}\left(\sum_{i=1}^{|Pa|} dist_i^m\right)^{1/m} \quad (12)$$

Where $dist_i$ is the Euclidean distance between individual *i* and the corresponding authentic Pareto frontier; *m* is the number of objective functions.

The PD reflects the diversity of *Pa*. The value of PD is negatively correlated with the variety of Pareto optimal solutions. This index is defined as:

$$PD = \frac{1}{\sum_{i=1}^{|m|} dist_i^e + |pa|}\left[\sum_{i=1}^{|m|} dist_i^e + \sum_{i=1}^{|pa|-1} |dist_i - \overline{dist}|\right] \quad (13)$$

where $\overline{dist}$ is the mean value of all *dist*; $dist_i^e$ is the boundary value distance between *Pa* and *Pa** for objective *i*.

Experimental setup

The experimental parameters were divided into the parameter set of test cases and parameter set of algorithms. A total of 6 different test cases were adopted, respectively containing 10, 12, 14, 16, 18 and 20 vertices. These test cases are denoted as TEST1, TEST2, TEST3, TEST4, TEST5 and TEST6. Then, MODCS, NSGA-II and MOPSO were contrasted with the authentic Pareto solution.

Table 1. Parameter set of test cases

Test case	TEST					
	1	2	3	4	5	6
Number of node	10	12	14	16	18	20
(U_{Time}) Time limit	60	70	80	100	120	140
(U_{Area}) Area limit	40	60	60	70	70	80
(U_{Pow}) Power consumption limit	120	130	140	180	200	220
(U_{Cost}) Cost limit	50	70	80	100	110	120
(U_{Com}) Communication cost limit	60	80	90	110	130	140
(*Pa**) Authentic Pareto Optimum solution (*Pa**)	(57,109) (58,107) (59,106) (60,104)	(59,120) (61,117)	(69,140) (71,137)	(70,147) (74,146)	(79,168) (81,167) (82,165) (86,164)	(92,190) (93,188) (96,186) (90,191) (94,187)

Table 2. Parameter set of MODCS

alpha (α)	0.01
beta (β)	1.5
Elimination probability	0.25
Encode control parameter	0.3
Population scale	100
Iterations	120
Crossover probability	0.7
Mutation probability	0.3

Table 3. Parameter set of NSGA-II

Crossover probability	0.7
Variation probability	0.3
Population scale	50
Iterations	100

Table 4. Parameter set of MOPSO

Dynamic constant *w*	0.4
Local search control parameters r_1	0.25
Global search control parameters r_2	0.25
EA maximum capacity (*C*)	10
Grid number (*M*)	3
Population size (*n*)	50
Number of iterations (*G*)	100

The experiments were performed in the environment of Intel Core Duo i5-3337U 1.8 GHz CPU, 4GB RAM, and Matlab R2012b. To control the data randomness, the MODCS, NSGA-II and MOPSO were run ten times in each of the six test cases, respectively, and then the mean $\overline{GD}$, $\overline{PD}$ and variance σ_{GD}^2, σ_{PD}^2 of GD and PD were computed one by one.

Results and Discussion

According to the above figures and tables, the mean PDs of MODCS, NSGA-II and MOPSO algorithms were 0.176, 0.350018 and 0.957023, respectively. Thus, the three algorithms were ranked as MODCS<NSGA-II<MOPSO by mean PD, indicating that MODOC has a more diverse Pareto solution set than the contrast algorithms. The mean GDs of MODCS, NSGA-II and MOPSO algorithms are 0.036283, 0.286683 and 0.313597, respectively.

Table 5. Results of $\overline{PD}$ and σ_{PD}^2

Algorithm	TEST1			TEST2			TEST3		
	$\overline{PD}$	σ_{PD}^2	time	$\overline{PD}$	σ_{PD}^2	time	$\overline{PD}$	σ_{PD}^2	time
MODCS	0	0	1.8081	0	0	2.4258	0.401	0.373	1.2433
NSGA-II	0	0	0.99685	0.10811	0.09886	1.8533	0.627	0.235	1.546
MOPSO	1	0	0.52572	0.98153	0.00341	0.53352	0.96306	0.0061	0.44616
Algorithm	TEST4			TEST5			TEST6		
	$\overline{PD}$	σ_{PD}^2	time	$\overline{PD}$	σ_{PD}^2	time	$\overline{PD}$	σ_{PD}^2	time
MODCS	0.3	0.233	2.9079	0.1	0.1	3.683	0.255	0.30	2.7534
NSGA-II	0.327	0.219	2.145	0.4	0.271	1.8783	0.638	0.303	1.8252
MOPSO	0.966	0.012	0.54444	0.9	0.013	0.5772	0.93155	0.00842	0.52572

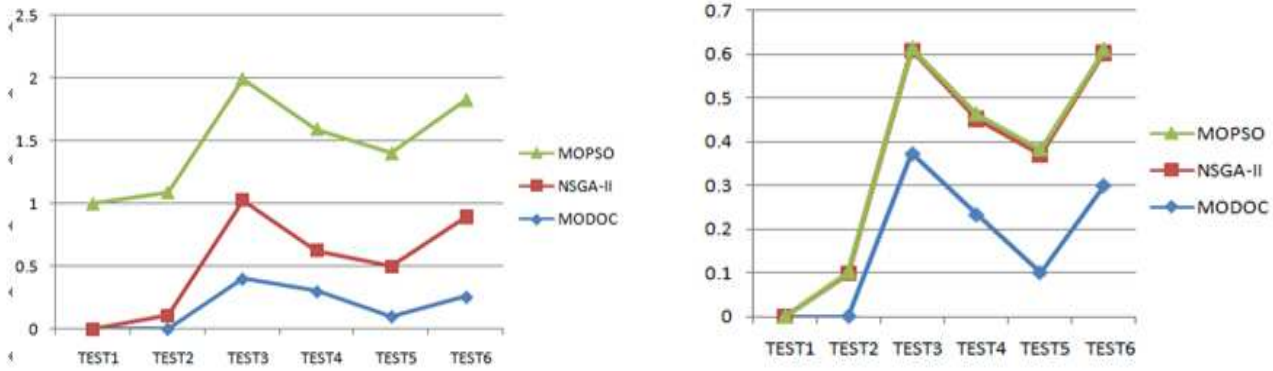

Figure 6. Figure 6 Results of $\overline{PDe}$ and σ_{PD}^2

Table 6. Results of $\overline{GD}$ and σ_{GD}^2

Algorithm	TEST1			TEST2			TEST3		
	$\overline{GD}$	σ_{GD}^2	time	$\overline{GD}$	σ_{GD}^2	time	$\overline{GD}$	σ_{GD}^2	time
MODCS	0	0	1.8081	0	0	2.4258	0.213	0.173	1.2433
NSGA-II	0	0	0.99685	0.014	0.002	1.8533	0.614	0.249	1.546
MOPSO	0	0	0.52572	0.1	0.1	0.53352	0.6	0.26667	0.44616
Algorithm	TEST4			TEST5			TEST6		
	$\overline{GD}$	σ_{GD}^2	time	$\overline{GD}$	σ_{GD}^2	time	$\overline{GD}$	σ_{GD}^2	time
MODCS	0	0	2.9079	0	0	3.683	0.0047	0.0002	2.7534
NSGA-II	0.93	0.048	2.145	0.082	0.068	1.8783	0.0801	0.0185	1.8252
MOPSO	0.2	0.4	0.54444	0.66945	0.58538	0.5772	0.31213	0.16953	0.52572

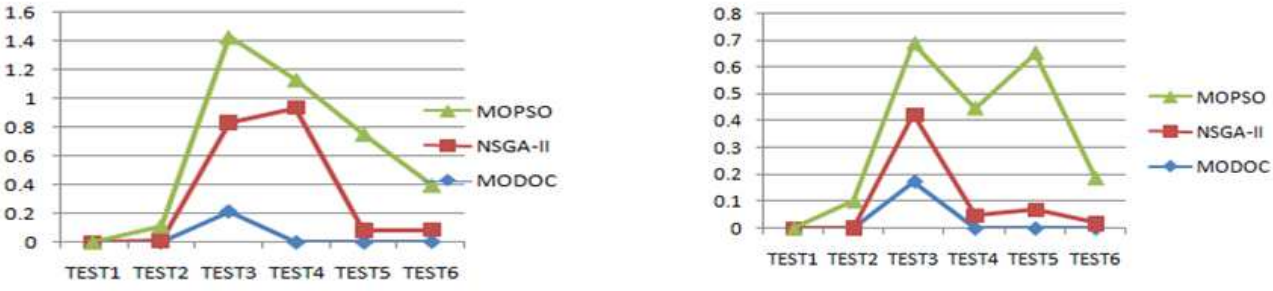

Figure 7. Value of $\overline{GD}$ and σ_{GD}

Thus, the three algorithms were ranked as MODCS<NSGA-II<MOPSO by mean GD. This means the Pareto solution set of MODOC is closer to the authentic solution set than those of NSGA-II and MOPSO. To sum up, the experiments show that MODCS performs much better than NSGA-II, MOPSO in terms of GD, PD and stability.

Conclusions

Targeting at the hardware/software partitioning of the embedded system, this paper proposes the MODCS, a bionic algorithm based on discrete DC, and includes the elite strategy into the algorithm. Thanks to the stratification and congestion degree comparison of the strategy, the MODCS enjoys a strong global searching ability, a large and diverse sampling space of population, and thus highly accurate optimization results. Through comparison experiments, it is proved that the MODCS is superior to typical optimization algorithms in terms of many indices, including diversity, stability and GD of optimal solution. The superiority is positively correlated with the number of modules. Of course, MODOC also has a shortcoming, the long runtime incurred by Lévy flight, which can be solved through hardware upgrading and parallel computing.

References

Savage, M. J. W., Salcic, Z., Coghill, G., & Covic, G. (2004). Extended genetic algorithm for codesign optimization of DSP systems in FPGAs. IEEE International Conference on Field-Programmable Technology, 2004. Proceedings (pp.291-294).

Barthelemy P, Bertolotti J, Wiersma DS. A Lévy flight for light. Nature 2008; 453(7194):495.

Coello CAC, Lechuga MS. MOPSO: a proposal for multiple objective particle swarm optimization, Evolutionary Computation, 2002. CEC '02, Proceedings of the 2002 Congress on, IEEE 2002; 1051-56.

Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 2002; 6(2): 182-97.

Deb K. Multi-objective optimization using evolutionary algorithms, New York: John Wiley & Sons, 2001.

Ernst R, Henkel J, Benner T. Hardware-Software Cosynthesis for Microcontrollers. Design & Test of Computers IEEE, 1993; 10(4): 64-75.

Garey MR, Johnson DS. Computers and intractability: A guide to the theory of NP-completeness, W H Freeman Company, 1979.

Guo B, Wang D, Shen Y, Liu Z. Hardware–software partitioning of real-time operating systems using Hopfield neural networks. Neurocomputing 2006; 69(16-18): 2379-84.

Konak A, Coit DW, Smith AE. Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety 2006; 91(9): 992-1007.

Ma T, Wang X, Li Z. Neural Network Optimization for Hardware-Software Partitioning, International Conference on Innovative Computing, Information and Control. IEEE Computer Society 2006; 423-26.

Pan Z. Hardware-software partitioning for the design of system on chip by neural network optimization method, International Symposium on Precision Engineering Measurements and Instrumentation, International Society for Optics and Photonics, 2011; 8321: 83211T-6.

Srinivas N, Deb K. Multiobjective Function Optimization Using Nondominated Sorting Genetic Algorithms, IEEE Transactions on Evolutionary Computation 1994; 2(3): 1301-08.

Wiangtong T, Cheung PYK, Luk W. Tabu Search with Intensification Strategy for Functional Partitioning in Hardware-Software Codesign, Field -Programmable Custom Computing Machines 2002; (4): 297-98.

Yang XS, Deb S. Cuckoo Search via Lévy flights, World Congress on Nature & Biologically Inspired Computing, IEEE 2009: 210-14.

Yang XS, Deb S. Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling and Numerical Optimisation 2010; 1(4): 330-43.

Yang XS. Engineering optimisation: An introduction with metaheuristic applications, John Wiley and Sons, 2010.

Zhang LB, Zhou CG, Liu XH, Ma ZQ, Liang YC. Solving multiobjective optimization problems using particle swarm optimization, Proceedings of the 2003 congress on evolutionary computation CEC2003, Canberra Australia: IEEE Press 2003; 2400-05.

Zhang Y, Luo W, Zhang Z, Li B, Wang X. A hardware/software partitioning algorithm based on artificial immune principles. Applied Soft Computing 2008; 8(1): 383-91.