Yuan B, Li B. Coverage optimization for defect-tolerance logic mapping on nanoelectronic crossbar architectures. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 27(5): 979–988 Sept. 2012. DOI 10.1007/s11390-012-1278-0

Coverage Optimization for Defect-Tolerance Logic Mapping on Nanoelectronic Crossbar Architectures

Bo Yuan (袁 博), *Student Member*, *IEEE*, and Bin Li* (李 斌), *Member*, *ACM*, *IEEE*

Natural Inspired Computation and Applications Lab, Joint USTC-Birmingham Research Institute in Intelligent Computation and Its Applications, University of Science and Technology of China, Hefei 230026, China

E-mail: yuanbo@mail.ustc.edu.cn; binli@ustc.edu.cn

Received October 14, 2011; revised April 16, 2012.

Abstract Emerging nano-devices with the corresponding nano-architectures are expected to supplement or even replace conventional lithography-based CMOS integrated circuits, while, they are also facing the serious challenge of high defect rates. In this paper, a new weighted coverage is defined as one of the most important evaluation criteria of various defect-tolerance logic mapping algorithms for nanoelectronic crossbar architectures functional design. This new criterion is proved by experiments that it can calculate the number of crossbar modules required by the given logic function more accurately than the previous one presented by Yellambalase *et al.* Based on the new criterion, a new effective mapping algorithm based on genetic algorithm (GA) is proposed. Compared with the state-of-the-art greedy mapping algorithm, the proposed algorithm shows pretty good effectiveness and robustness in experiments on testing problems of various scales and defect rates, and superior performances are observed on problems of large scales and high defect rates.

Keywords coverage optimization, defect-tolerance, nanoelectronic crossbar, logic mapping, evolutionary algorithm

1 Introduction

Conventional lithography-based CMOS techniques are rapidly approaching their realistic limits. It is expected that emerging nanoelectronic techniques[1-2] can achieve very high devices density ($10^{12}/\text{cm}^2$) and operation frequency (excess of 100 GHz) and will supplement or even replace CMOS integrated circuits. Nanoelectronic architectures can be built from nanowires and nanoelectronic devices via bottom-up self-assembly techniques[3]. The regular nature of chemically self-assembly structure makes it well suited for implementing regular crossbar-based arrays similar to programmable logic arrays (PLAs). Recently, the world's first programmable nanoprocessor consisting of programmable, non-volatile nanowire transistor arrays (PNNTAs) has been developed and demonstrated[4].

However, due to the extremely small size of the nanoelectronic devices and the hardness to control the fabricating process from external, both the bottom-up self-assembly techniques and nano-imprint techniques are not able to avoid generating chips of high defect rates. The exact level of defect rate is still unknown, but it is assumed to be reasonable that 1% to 15% of the resources (wires, switches, etc.) on a nano-chip will be defective[5]. The researchers of Harvard and MITRE[4] characterized the threshold voltage values of nodes from the fabricated PNNTA structure in both the active and inactive states. Notably, they found that only 86% nodes in active state and 87% nodes in inactive state met voltage requirement. Discarding the nano-chip containing any defect is not affordable any more as the yield hit will be substantial. Faced with such a high defect rate of nano-architecture, future nano-chip designing and fabricating industry definitely need carefully crafted effective defect-tolerance designing techniques.

Fortunately, the reconfigurability inherent in the nanoelectronic crossbar architecture is well suited for the implementation of defect-tolerance mechanisms. After identifying defective resources in the chip (defect map) via test and diagnosis processes, the defects can be bypassed by a post fabrication configuration process[6-14]: given a defective crossbar(s) with a logic function to be implemented on it, a mapping from the function to the crossbar is to be found with considering the defects,

Regular Paper

This work was partially supported by the National Natural Science Foundation of China under Grant Nos. 61071024, U0835002, the Innovation Fund for Young Researchers of University of Science and Technology of China, and the EU's 7th Framework Programme for Research (FP7) under Grant No. 247619.

*Corresponding Author

which is generally called defect-aware defect-tolerance logic mapping problem.

In this paper, we first present the formal definition of the defect-aware defect-tolerance logic mapping problem mathematically. Then a new weighted coverage is defined as the evaluation criterion for various defect-aware defect-tolerance logic mapping algorithms, and its rationality is testified experimentally. This new criterion is proved by experiments that it can calculate the number of crossbar modules required by the given logic function more accurately than the previous one[9]. Based on this new criterion, a new effective mapping algorithm based on genetic algorithm (GA) is proposed to optimize the coverage. Compared with the state-of-the-art greedy mapping algorithm[9], the proposed algorithm shows pretty good effectiveness and robustness in experiments on testing problems of various scales and defect rates, and superior performances are observed on problems of large scales and high defect rates.

The rest of the paper is organized as follows. Section 2 introduces the preliminary knowledge of the defect-tolerance logic mapping problem and some related work will be reviewed in Section 3. A new weighted coverage is defined and its rationality is testified experimentally in Section 4. Section 5 proposes the GA-based mapping algorithm for maximizing coverage. The experimental results on various scale problems with different defect rates are provided and analyzed in Section 6. Finally, Section 7 concludes this paper.

2 Preliminaries

2.1 Problem Model

A nanoelectronic crossbar consists of two layers of orthogonal nanowires or nanotubes. The region where two wires cross is called junction or crosspoint, which may be configured to implement a nanoelectronic device. The stochastic nature of the assembly or the inaccurate nano-imprint means that the probability of aligning three things will be very low. Therefore, three-terminal devices will be hard to fabricate, while a two-terminal connection can be established merely by overlapping two wires perpendicularly. Two-terminal devices such as nanowire field effect transistors, diodes, and molecular switches will be preferred[5].

For the sake of simplification and without loss of generality, it is assumed that the defect in this paper only indicates the "stuck-at-open" defect which is the most common in nanoelectronic crossbar and the defect probability distribution over the crossbar is uniform as assumed in previous work[6-8,12]. A "stuck-at-open" defect means that there is a nonprogrammable switch at the crosspoint or missing a switch at the crosspoint, thus the two cross wires at this crosspoint are always disconnected. It is notable that the methods presented in this paper can be easily extended to other defect types ("stuck-at-closed" defect, "nanowire open" defect and "nanowire bridging" defect[13]) and other defect distributions (such as clustered distribution[9]) by modifying the following bipartite graph model slightly.

An example of a defective 3×3 nanoelectronic crossbar is shown in Fig.1(a). The crossbar consists of two sets of orthogonal nanowires. The vertical nanowires are the inputs, whereas the horizontal nanowires are the outputs. There is a programmable switch at each crosspoint. The nonprogrammable defective switches at the crosspoints are represented by an "X" for each.

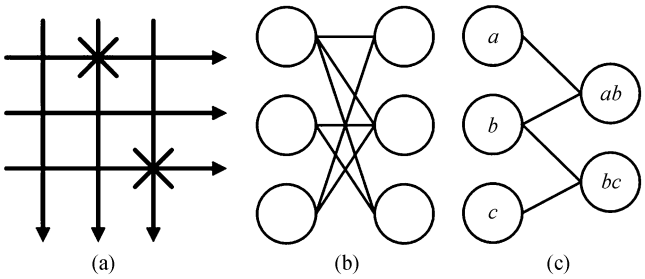

Fig.1. (a) 3×3 nanoelectronic crossbar with two defects. (b) Bipartite graph of crossbar in (a). (c) Bipartite graph of function: $F = ab + bc$.

Given a 2-D crossbar with defect map, it can be represented by a bipartite graph, as shown in Fig.1(b) which is the bipartite graph of the crossbar in Fig.1(a). A bipartite graph of an $n \times n$ crossbar is an undirected bipartite graph $G(U, V, E)$ with partitions U and V, having $|U| = n$ and $|V| = n$. U represents the set of input nanowires, and V represents the output nanowires. E consists of representative edges for all the programmable non-defective crosspoints of the crossbar.

A bipartite graph also can represent a two-level logic function in a sum-of-products form, as shown in Fig.1(c). U represents the set of logic variables, and V represents the product terms. E consists of representative edges for the corresponding product term containing the variable.

When we use a crossbar structure to implement a two-level logic function, the relationships between the product term set and the variable set in the logic function are represented by the connections between horizontal and vertical wires in the crossbar. Such logic-function-to-crossbar mapping problem can be formulated as embedding the logic function bipartite graph into the nanowire crossbar bipartite graph. This bipartite graph embedding problem by its nature is equivalent to the subgraph isomorphism problem which is a well-known NP-complete problem[15].

2.2 Coverage-Based Mapping

We can employ two decision vectors to represent the logic-function-to-crossbar mapping trial M[14]:

Input Mapping Vector (***IMV***): $IMV[v] = i$ if variable v is assigned to vertical nanowire i.

Output Mapping Vector (***OMV***): $OMV[p] = o$ if product term p is mapped to horizontal nanowire o.

For each mapping trial M, its quality can be represented by a "score", namely the coverage in the mapping. The coverage C was firstly defined in [9] as the ratio of the number of product terms been successfully mapped to the total number of product terms of the function, which is regarded as the evaluation criterion of various defect-tolerance logic mapping algorithms[9]. With the coverage C, the number of crossbar modules N required for mapping the given logic function can be estimated to be the upper bound of $1/C$.

However, this estimation is always inaccurate, because the mapping costs (difficulties) of various product terms are different. More variables contained in a product term means more cost required to map it. In this paper, a new weighted coverage is proposed to incorporate the consideration of mapping cost of product terms, in which a weight w_p is assigned to each product term p to represent its mapping cost. Then the coverage C can be reformulated as the ratio of the weighted sum of successfully mapped product terms to the sum of total weights. In this case, the original form of coverage C[9] is a special case with $w_p = 1$ for all the product terms p. In Section 4, we will test the effect of several weighting strategies to the accuracy of estimation.

Now, we will give the generalized definition of the weighted coverage-based defect-tolerance logic mapping problem. The definitions of associated variables are shown in Table 1.

Input:

0-1 function matrix:

$$\boldsymbol{FM} = (f_{v,p})_{V\times P},$$

where $f_{v,p} = 1$ represents input variable v is contained by product term p.

0-1 crossbar matrix:

$$\boldsymbol{CM} = (c_{i,o})_{I\times O},$$

where $c_{i,o} = 1$ represents that there is a stuck-at-open defect between input nanowire i and output nanowire o.

Solution:

Two permutations: ***IMV*** and ***OMV***, $\boldsymbol{IMV} \in S_I$ and $\boldsymbol{OMV} \in S_O$.

$IMV(v) = i$, represents input variable v is mapped on input nanowire i, $1 \leqslant v \leqslant V$ and $1 \leqslant i \leqslant I$.

$OMV(p) = o$, represents product term p is mapped on output nanowire o, $1 \leqslant p \leqslant P$ and $1 \leqslant o \leqslant O$.

Objective Function:

Maximum Formula:

$$Fitness = \sum_{p=1}^{P} w_p \times \Big(sgn\Big(\sum_{v=1}^{V} f_{v,p} \times (f_{v,p} - c_{IMV(v),OMV(p)}) - d_p\Big) + 1\Big)\Big/W. \quad (1)$$

Table 1. Definitions

V	Number of variables
P	Number of product terms
I	Number of input nanowires
O	Number of output nanowires
FR	Density of the function matrix
DR	Defect rate of the crossbar matrix
O_1	Set of all the product terms
O_2	Set of product terms which have been successfully mapped
w_p	Weight w for product term p
W	Sum of all the weights: $W = \sum_{p\in O_1} w_p$
C	Coverage: $C = \sum_{p\in O_2} w_p/W$
$M_{\text{calculated}}$	Number of crossbar modules calculated: $M_{\text{calculated}} = \lceil 1/C \rceil$
M_{actual}	Actual number of crossbar modules required
D	Deviation: $D = \lvert M_{\text{calculated}} - M_{\text{actual}} \rvert / M_{\text{actual}} \times 100\%$
d_p	Degree (number of variables product term p contains) of product term p
$d_{\min}$	Minimum degree of product terms set P
$d_{\max}$	Maximum degree of product terms set P
d_{mean}	Mean degree of product terms set P

Decision vectors ***IMV*** and ***OMV*** represent one possible mapping trial M from logic function to crossbar architecture. One constraint is implicated in the solution: each variable v or product term p must be mapped to a nanowire i or o, and each nanowire can only receive one v or p. Another constraint is implicated in the *sgn* function of the objective function: a "1" of $\boldsymbol{FM}(f_{v,p} = 1)$ mapped to a "1" of $\boldsymbol{CM}(C_{IMV(v),OMV(p)} = 1)$, which means a defect crosspoint is used as a functional switch and the corresponding product term p will never contain the logic variable v, so p is unmapped and weight w_p will not affect *Fitness*; while all the "1"s of a product term p ($f_{v,p} = 1$) are mapped to "0"s of ***CM*** ($C_{IMV(v),OMV(p)} = 0$) means p is mapped successfully and *Fitness* will increase.

Fitness $= 1$ means all the product terms are mapped successfully around the crosspoint defects in nano crossbar.

3 Related Work

Dehon and Naeimi[6] proposed a liner-time greedy bipartite matching algorithm for mapping PLA logic around crosspoint defects in nanoelectronic crossbar.

The author assumed that the PLA inputs have been previously assigned which resulted in very low mapping success rates.

Hogg and Snider[7-8] identified reliability thresholds in the ability of defective crossbars to implement Boolean logic (binary adders in this paper) by exhaustive search algorithms with bounded computing time. These thresholds varied among different implementations of the same logical formula, allowing molecular circuit designers to trade off reliability, circuit area, crossbar geometry and the computational complexity of locating functional components.

Yellambalase and Cho[9] evaluated three different greedy defect-tolerance logic mapping algorithms for circumventing clustered defective crosspoints in nanowire reconfigurable crossbar architectures. Then, a cost-effective repair method was presented and analyzed.

Simsir *et al.*[10] proposed and evaluated a hybrid nanowire-CMOS architecture that addressed defects and faults at the same time. A companion to the architecture was a compiler with heuristics to quickly determine if the logic function can be mapped onto partially defective nanoscale elements. The heuristic algorithm does not attempt to find an optimal pin assignment since that is NP-complete. It utilizes greedy pin assignment and constructs the bipartite matching step by step instead of constructing the complete bipartite graph, which is time-consuming.

Dai *et al.*[11] presented an information-theoretic approach to investigate the intrinsic relationship between defect-tolerance and inherence redundancy in molecular crossbar architectures. By modeling defect-prone molecular crossbars as non-ideal information processing mediums, the information transfer capacity can be interpreted as the bound on reliability that molecular crossbars can achieve. The proposed method allows us to evaluate the effectiveness of redundancy-based defect-tolerance in a quantitative manner.

Rao *et al.*[12] modeled the PLA logic mapping problem as a bipartite graphs embedding problem and proposed a recursive algorithm. Further on, three enhanced heuristic techniques were proposed to improve the algorithm's efficiency by significantly cutting down unnecessary backtracking processes.

Crocker *et al.*[13] examined some fundamental aspects of defect-tolerance for QCA (Quantum-Dot Cellular Automata)-based crossbar architectures, then presented an implementation independent defect model. In this model, techniques were introduced to map Boolean logic functions to defective QCA-based crossbar architectures via graph monomorphism matching.

4 Weighting Strategies

The previous form of coverage C defined in [9] is not reasonable because it regards the mapping cost (difficulty) of each product term p equally which results in inaccurate estimation of the number of crossbar modules N required for mapping the given logic function. In this paper, we assume that the mapping cost of each product term p is determined by the number of variables contained by it, that is, the more variables it contains, the harder to map it.

Based on the above hypothesis, we design the following six forms of weight w_p for each product term p and test the effect of them to the accuracy of estimation:

$$w1_p = 1,$$
$$w2_p = d_p,$$
$$w3_p = d_p - d_{\min} + 1,$$
$$w4_p = \begin{cases} d_p - d_{\text{mean}}, & \text{if } d_p > d_{\text{mean}}, \\ 1, & \text{if } d_p \leqslant d_{\text{mean}}, \end{cases}$$
$$w5_p = \exp(k_1 \times (d_p - d_{\min})/(d_{\max} - d_{\min})),$$
$$w6_p = \exp(k_2 \times (d_p - d_{\text{mean}})/d_{\text{mean}}),$$

where

$$k_1 = k_2 = 2Size \times FR \times DR.$$

Please note that w_1 is the original form of coverage in previous work. We employ the column-matching-first algorithm[9] as defect-tolerance logic mapping algorithm du to its effectiveness. We set $V = P = I = O = Size$ for simplification and if the function size is less than the crossbar size, we can insert all-0 rows and columns to the function matrix to make the matrix have equal size with the crossbar matrix. For each case with the same *Size* and *DR*, we randomly generate ten different test benchmarks F_1 to F_{10}.①

The simulation results over various problem scales and defect rates show that the weight of exponential with degree is the most close to the actual case, especially $w6$. We only show the results of $w1$ and $w6$ to save space in Tables 1∼3 and the simulation results are the statistical mean from 100 random samples, the unit of deviation D is percent.

1) $Size = 16$: For this small-scale problem, we set $DR = 0.25$ and 0.30. Table 2 shows the deviations D of $w1$ and $w6$ $(D1, D6)$ with different DRs. The estimations of both $w1$ and $w6$ are accurate within a certain limit when $DR = 0.25$, because the actual numbers of crossbar modules required are less. The estimations of

① All the problem instances used in the simulation in the paper are available are: http://home.ustc.edu.cn/∼yuanbo/JCST2012.rar.

Table 2. Deviations of $w1$ and $w6$ for Problem Scale $Size = 16$ with Different DRs

Size	$DR = 0.25$			$DR = 0.30$		
$= 16$	M_{actual}	$D1$	$D6$	M_{actual}	$D1$	$D6$
$F1$	2.1	5.2	**4.3**	2.5	20.2	**8.7**
$F2$	2.3	13.0	**1.3**	3.4	41.2	**15.6**
$F3$	2.1	5.7	**2.8**	2.6	21.0	**8.2**
$F4$	2.2	8.7	**3.7**	2.8	29.3	**3.9**
$F5$	2.1	6.1	**3.8**	2.7	22.6	**17.0**
$F6$	2.4	17.7	**8.6**	3.4	37.4	**19.3**
$F7$	2.3	14.2	**9.0**	3.4	37.1	**18.8**
$F8$	2.4	18.0	**1.6**	3.4	40.5	**23.1**
$F9$	2.1	6.5	**5.6**	2.8	28.6	**17.5**
$F10$	2.4	16.0	**8.0**	3.2	36.9	**1.9**

Table 3. Deviations of $w1$ and $w6$ for Problem Scale $Size = 24$ with Different DRs

Size	$DR = 0.25$			$DR = 0.30$		
$= 24$	M_{actual}	$D1$	$D6$	M_{actual}	$D1$	$D6$
$F1$	3.7	44.8	**20.2**	5.9	59.5	**17.8**
$F2$	3.2	31.5	**22.4**	4.7	38.8	**12.1**
$F3$	3.0	32.4	**14.4**	4.7	48.2	**5.8**
$F4$	3.6	39.9	**23.3**	5.8	52.9	**14.6**
$F5$	3.0	32.3	**21.2**	4.5	46.5	**2.0**
$F6$	3.6	44.2	**6.0**	6.9	62.9	**14.0**
$F7$	3.9	37.3	**23.1**	5.8	42.4	**7.4**
$F8$	4.3	47.3	**4.0**	7.5	61.4	**26.6**
$F9$	3.3	33.9	**24.8**	5.4	48.3	**32.2**
$F10$	3.4	37.3	**29.0**	5.5	48.8	**23.0**

$w1$ are not accurate any more when $DR = 0.30$, and the maximal deviation is 41.2%; while the estimations of $w6$ can keep accurate and the deviations range from 1.9% to 23.1%.

2) $Size = 24$: For this medium-size problem, we set $DR = 0.25$ and 0.30. Table 3 shows the deviations D of $w1$ and $w6$ with different DRs. The estimations of $w1$ are inaccurate and the deviations range from 31.5% to 47.3% and 38.8 to 62.9% with $DR = 0.25$ and 0.30, respectively; while the deviations of $w6$ can keep in a reasonable range below 30% in most cases, except for F_9 with $DR = 0.30$.

3) $Size = 32$: For this large-size problem, we set $DR = 0.20$ and 0.25. Table 4 shows the deviations D of $w1$ and $w6$ with different DRs. The estimations of $w1$ are inaccurate and the deviations range from 18.3% to 40.5% and 42.3 to 59% with $DR = 0.20$ and 0.25, respectively; while the deviations of $w6$ can keep in a reasonable range below 30% in all cases.

The simulation results above show that the weight of exponential with degree $w6$ is most close to the actual case, especially for large-scale problems with high defect rates. Consequently, this weight can estimate N, the number of crossbar modules required accurately and can be taken as a new evaluation criterion of the performance for various defect-tolerance mapping algorithms. Therefore we can use it in (1) which can be employed as the fitness function for evolutionary algorithms. Next, we will propose an effective mapping algorithm to maximize the coverage based on GA with specific design for the mapping problem.

Table 4. Deviations of $w1$ and $w6$ for Problem Scale $Size = 32$ with Different DRs

Size	$DR = 0.25$			$DR = 0.30$		
$= 32$	M_{actual}	$D1$	$D6$	M_{actual}	$D1$	$D6$
$F1$	3.2	35.5	**8.8**	6.39	55.9	**16.3**
$F2$	2.5	18.3	**16.3**	4.42	42.3	**28.7**
$F3$	3.3	39.5	**22.0**	6.43	57.7	**13.2**
$F4$	3.7	39.1	**13.6**	6.63	49.8	**23.1**
$F5$	3.1	36.0	**20.7**	5.70	51.2	**6.8**
$F6$	3.5	38.8	**14.4**	7.09	55.6	**6.8**
$F7$	3.4	40.5	**22.4**	6.24	53.8	**13.5**
$F8$	2.8	27.3	**23.6**	4.65	52.7	**27.1**
$F9$	3.1	35.0	**12.4**	6.05	59.0	**13.4**
$F10$	3.2	36.7	**14.7**	6.41	57.6	**12.2**

5 GA-Based Mapping Algorithm

5.1 Framework of GA-Based Algorithm

Evolution algorithms simulate the behavior of nature and social group, such as genetic algorithm[16], particle swarm optimization[17], and differential evolution[18]. They make a remarkable success in the combinatorial optimization field, for example, traveling salesman, quadratic assignment, flow shop scheduling, protein structure prediction. The defect-tolerance logic mapping problem is also a typical combinatorial optimization problem similar to the quadratic 3-D assignment problem (Q3AP)[19] which is known as NP-hard in the mathematical formulation, where the whole solution contains two (in)complete permutations, ***IMV*** and ***OMV*** for this problem. Therefore, we try to solve this mapping problem by GA-based method for coverage optimization which is expected to have a very good effect.

The flow graph of the GA-based method for coverage optimization is shown in Fig.2 which mostly follows a standard GA. We initialize the population randomly and use roulette wheel selection. In order to guarantee the GA-based method converge to optimum solution or suboptimum solution, we employ simple elite reservation strategy that selects one best group as the new parent population in the combination of parent and offspring population. The fitness of the individual is the coverage computed after decoding according to (1). When the coverage arrives to 1 or the given maximum fitness evaluation times is arrived, the algorithm will stop.

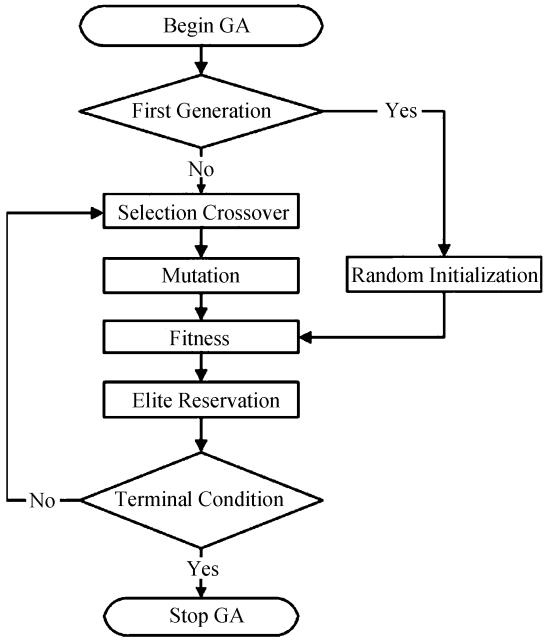

Fig.2. GA-based mapping method for coverage optimization.

5.2 Coding Scheme

Effective coding schemes are very important to GA. According to the nature of defect-tolerance mapping problem, this paper makes use of the features of decision vector $\boldsymbol{IMV}$ and $\boldsymbol{OMV}$ and proposes the integer coding scheme as shown in Fig.3.

Chromosome i

$\boldsymbol{IMV}_i$	2	7	6	4	9	5
	1	2	3	4	5	6
$\boldsymbol{OMV}_i$	3	8	5	6	4	1
	1	2	3	4	5	6

Fig.3. Coding scheme.

Each chromosome i contains two parts: $\boldsymbol{IMV}_i$ and $\boldsymbol{OMV}_i$, where $\boldsymbol{IMV}_i$ represents the assignment from variables to input nanowires, $\boldsymbol{OMV}_i$ represents the assignment from product terms to output nanowires. The index of the genes represents the index of variables or product terms in the function. The values of the genes represent the index of input nanowires or output nanowires in the crossbar architecture.

5.3 Genetic Operators

Genetic operators play the key role in GA. In the crossover stage of our method, we cross the parent individuals both in $\boldsymbol{IMV}$ and $\boldsymbol{OMV}$. In order to guarantee the rationality, we use order crossover method similar to that used in TSP (traveling salesman problem) as shown in Fig.4.

First, select two random crosspoints k_1 ($1 \leqslant k_1 < n-1$) and k_2 ($k_1 < k_2 < n$) for $\boldsymbol{IMV}_1$ and $\boldsymbol{IMV}_2$. Second, the genes from k_1 to k_2 in offspring $\boldsymbol{IMV}_1$' succeed the genes (1, 2, 8) from k_1 to k_2 in parent $\boldsymbol{IMV}_2$. Third, delete the genes (2) in $\boldsymbol{IMV}_1$ whose values have already existed in $\boldsymbol{IMV}_1$'. Forth, add the remaining genes (7, 6, 4, 9, 5) in $\boldsymbol{IMV}_1$ to $\boldsymbol{IMV}_1$' (7, 6, 4) in sequence and ignore the excess part (9, 5). The same strategy is used for $\boldsymbol{IMV}_2$' that succeed the genes from

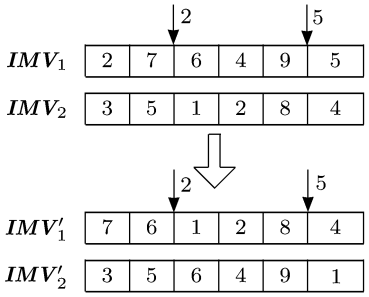

Fig.4. Crossover operator.

k_1 to k_2 in parent $\boldsymbol{IMV}_1$ firstly and then succeed the genes which have never existed in parent $\boldsymbol{IMV}_2$. We use similar crossover method for $\boldsymbol{OMV}$ as $\boldsymbol{IMV}$.

The mutation method is simple in this paper. First, test if there are genes whose values have never appeared in $\boldsymbol{IMV}_i$ or $\boldsymbol{OMV}_i$. If any, we will select a random gene whose value has never appeared to replace the gene to be mutated (corresponding to the case that the function size is less than the crossbar size). Conversely, select a random gene in $\boldsymbol{IMV}_i$ or $\boldsymbol{OMV}_i$ and interchange it with the gene to be mutated (corresponding to the case that the function size is equal to the crossbar size).

6 Simulation Results

In our simulation, we assume that $V = P = I = O = Size$ for simplification. Note that if the function size is less than the crossbar size, we can insert all-0 rows and columns to the function matrix to make the matrix have equal size with the crossbar matrix.

The experimental parameters of the GA-based coverage optimization algorithm are set as follows:

Population size: 40.

Crossover probability: 100%.

Mutation Probability: we will discuss it in next subsection.

Maximum fitness evaluation times: 20 000, 40 000, 60 000 for different problem scales with $Size = 16$, 24, 32 respectively.

For each case with same problem scale and same function density CR and defect rate DR, we generate 10 different benchmark functions from F_1 to F_{10} with random distribution model for function matrix and crossbar matrix as used in Section 4. The simulation results

of the GA-based coverage optimization algorithm are statistic from 30 independent runs for each instance.

6.1 Mutation Probability

In our simulation, we find that the performance of the GA-based mapping algorithm is very sensitive to the mutation probability of chromosomes. In general, an appropriate larger mutation probability means a better performance. It is notable that in this paper, the mutation probability is for each chromosome but not for each gene.

Under different mutation probabilities 0, 0.2, 0.5, 0.8, 1, we record the results (statistical mean from 30 independent runs) for 10 $Size = 24$ benchmark functions from F_1 to F_{10} with $CR = 40\%$ and $DR = 30\%$ in Table 5. Obviously, a more appropriate mutation probability between 0.5 and 1 can result in better results. The same obviations can be found in other problems with different scales, *CR*s and *DR*s in our simulation. Hence, the mutation probability is set at 0.8 for all benchmark functions in the following simulation.

Table 5. Coverage Obtained by the GA-Based Algorithm with Different Mutation Probability

Size	Mutation Probability				
$= 24$	0	0.20	0.50	0.80	1.00
F_1	0.58	0.77	0.83	**0.87**	0.80
F_2	0.27	0.53	0.59	**0.61**	0.52
F_3	0.55	0.75	**0.82**	**0.82**	0.72
F_4	0.44	0.62	0.70	**0.75**	0.67
F_5	0.46	0.69	0.72	**0.75**	0.70
F_6	0.69	0.82	0.85	**0.86**	0.82
F_7	0.35	0.52	0.62	**0.66**	0.59
F_8	0.41	0.54	0.59	**0.67**	0.64
F_9	0.43	0.67	0.72	**0.74**	0.66
F_{10}	0.41	0.64	0.69	**0.72**	0.62

6.2 Effectiveness and Robustness

The average convergence curves from 30 independent runs of the GA-based algorithm are shown in Figs. 5, 6 and 7 corresponding to different problem scales $Size = 16$, 24 and 32 with the density of the function 40% and various defect rates.

Further on, in Tables 6∼8 we compare the coverage obtained by our GA-based mapping algorithm C_{GA} (statistical mean and standard deviation from 30 independent runs) with the state-of-the-art greedy mapping algorithm specialized for maximizing coverage: column-matching-first algorithm[9] C_{Greedy} which is a deterministic algorithm.

Please note that coverage C obtained above 0.5 means the number of crossbar modules N required for mapping the given logic function is less than two. In this case, we think a good result has been achieved by the algorithm.

1) $Size = 16$: For this small-scale problems, we set $DR = 0.25$ and 0.30. Fig.5 shows the coverage convergence curves of the GA-based algorithm and Table 6 shows the coverage obtained by different algorithms. The GA-based algorithm can achieve very high coverage varying from 0.79 to 1 for different instances with $DR = 0.25$, 0.30. The results of the greedy algorithm

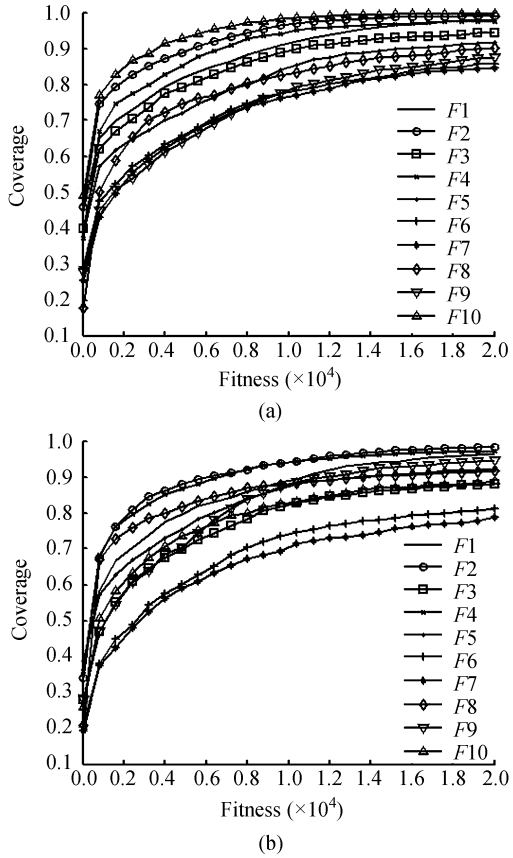

Fig.5. Coverage convergence curves of the GA-based algorithm for $Size = 16$, $DR = 0.25$ (a) and 0.30 (b).

Table 6. Coverage Obtained by the GA-Based Algorithm (C_{GA}) and Greedy Algorithm (C_{Greedy}) for Problem Scale $Size = 16$

$Size = 16$	$DR = 0.25$		$DR = 0.30$	
	C_{GA}	C_{Greedy}	C_{GA}	C_{Greedy}
F_1	$\mathbf{0.98} \pm 0.01$	0.81	$\mathbf{0.96} \pm 0.03$	0.69
F_2	$\mathbf{0.99} \pm 0.01$	0.81	$\mathbf{0.98} \pm 0.01$	0.75
F_3	$\mathbf{0.94} \pm 0.02$	0.81	$\mathbf{0.88} \pm 0.03$	0.63
F_4	$\mathbf{0.98} \pm 0.02$	0.88	$\mathbf{0.97} \pm 0.01$	0.56
F_5	$\mathbf{0.92} \pm 0.03$	0.81	$\mathbf{0.92} \pm 0.03$	0.81
F_6	$\mathbf{0.86} \pm 0.04$	0.63	$\mathbf{0.81} \pm 0.05$	0.69
F_7	$\mathbf{0.85} \pm 0.04$	0.56	$\mathbf{0.79} \pm 0.06$	0.44
F_8	$\mathbf{0.90} \pm 0.04$	0.69	$\mathbf{0.92} \pm 0.04$	0.63
F_9	$\mathbf{0.87} \pm 0.04$	0.63	$\mathbf{0.95} \pm 0.03$	0.56
F_{10}	$\mathbf{1.00} \pm 0.01$	0.94	$\mathbf{0.89} \pm 0.06$	0.63

are also acceptable. However, its performance is low at some complex problem, such as F_7 benchmark function with $DR = 0.30$, the coverage is 0.44.

2) $Size = 24$: For this medium-size problem, we set $DR = 0.25$ and 0.30. Fig.6 shows the coverage convergence curves of the GA-based algorithm and Table 7 shows the coverage obtained by different algorithms. We can see that the worst coverage obtained by the greedy algorithm is reduced to about 0.46 and 0.25 at $DR = 0.25$ and 0.30 respectively, while the GA-based algorithm also can reach high coverage varying from 0.64 to 0.92 in either case.

3) $Size = 32$: For this large-size problem, we set $DR = 0.20$ and 0.25. Fig.7 shows the coverage convergence curves of the GA-based algorithm and Table 8 shows the coverage obtained by different algorithms. Obviously, the coverage obtained by the greedy algorithm is low for this large and complex problem,

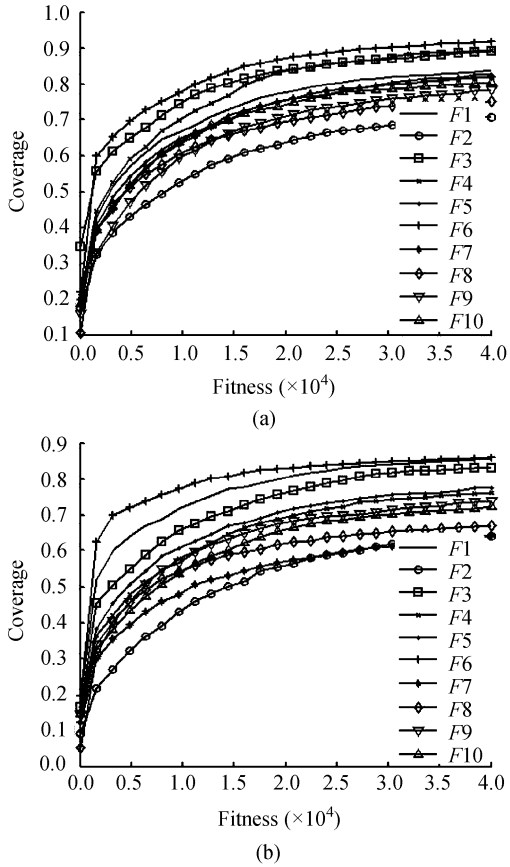

Fig.6. Coverage convergence curves of the GA-based algorithm for $Size = 24$, $DR = 0.25$ (a) and 0.30 (b).

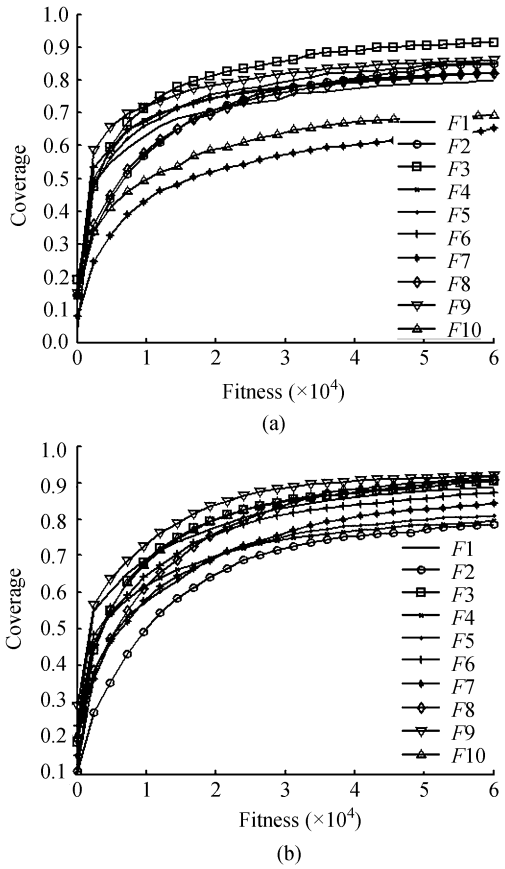

Fig.7. Coverage convergence curves of the GA-based algorithm for $Size = 32$, $DR = 0.20$ (a) and 0.25 (b).

Table 7. Coverage Obtained by the GA-Based Algorithm (C_{GA}) and Greedy Algorithm (C_{Greedy}) for Problem Scale $Size = 24$

$Size = 24$	$DR = 0.25$		$DR = 0.30$	
	C_{GA}	C_{Greedy}	C_{GA}	C_{Greedy}
F_1	**0.84** ± 0.05	0.54	**0.86** ± 0.04	0.54
F_2	**0.71** ± 0.04	0.50	**0.64** ± 0.06	0.38
F_3	**0.89** ± 0.03	0.63	**0.83** ± 0.04	0.58
F_4	**0.89** ± 0.04	0.58	**0.76** ± 0.09	0.38
F_5	**0.82** ± 0.05	0.63	**0.78** ± 0.07	0.33
F_6	**0.92** ± 0.03	0.67	**0.86** ± 0.02	0.46
F_7	**0.82** ± 0.04	0.54	**0.64** ± 0.06	0.25
F_8	**0.75** ± 0.04	0.46	**0.67** ± 0.10	0.46
F_9	**0.78** ± 0.05	0.54	**0.74** ± 0.05	0.54
F_{10}	**0.80** ± 0.03	0.50	**0.72** ± 0.06	0.46

Table 8. Coverage Obtained by the GA-Based Algorithm (C_{GA}) and Greedy Algorithm (C_{Greedy}) for Problem Scale $Size = 32$

$Size = 32$	$DR = 0.20$		$DR = 0.25$	
	C_{GA}	C_{Greedy}	C_{GA}	C_{Greedy}
F_1	**0.89** ± 0.03	0.56	**0.80** ± 0.04	0.41
F_2	**0.79** ± 0.06	0.59	**0.85** ± 0.05	0.50
F_3	**0.90** ± 0.05	0.69	**0.91** ± 0.04	0.63
F_4	**0.80** ± 0.04	0.50	**0.82** ± 0.07	0.47
F_5	**0.81** ± 0.06	0.56	**0.85** ± 0.06	0.50
F_6	**0.87** ± 0.04	0.59	**0.82** ± 0.03	0.47
F_7	**0.84** ± 0.05	0.63	**0.65** ± 0.08	0.41
F_8	**0.91** ± 0.03	0.72	**0.82** ± 0.05	0.66
F_9	**0.92** ± 0.02	0.56	**0.86** ± 0.04	0.50
F_{10}	**0.92** ± 0.03	0.69	**0.69** ± 0.07	0.34

especially when $DR = 0.25$. The GA-based algorithm can still find a good result (from 0.65 to 0.92) in any instance.

The simulation results above show the effectiveness and robustness of the GA-based mapping algorithm that the coverage C obtained increases obviously as the fitness evaluation and good results (0.5 to 1) are obtained in the given fitness evaluation times for various instances randomly generated.

6.3 Discussion

Obviously in our experiments, although the runtime of GA-based mapping algorithm is limited and acceptable, it is much more than the runtime of the greedy mapping algorithm which has linear time complexity. However, the coverage convergence curves of the GA-based mapping algorithm (Figs. 5∼7) show that the coverage obtained by GA can surpass the result of the greedy algorithm in the early stage. Next we will discuss how to improve the efficiency of the GA-based mapping algorithm.

In fact the defect-tolerance mapping problem contain two assignment sub-problems, one is variables assignment (***IMV***) and the other is product terms assignment (***OMV***). Decision vectors ***IMV*** and ***OMV*** represent one possible mapping trial M and all combinations of ***IMV***s with ***OMV***s are the whole solution space. In this paper, we try to search the whole solution space which is very large ($Size! \times Size!$). If we fix either one of them (***IMV*** or ***OMV***), the search space will reduced to $Size!$. Moreover, the remaining assignment sub-problem will be very similar to some existing assignment problems, such as quadratic assignment problem[20], terminal assignment problem[21] and task assignment problem[22]. Take the task assignment problem on heterogeneous computing systems[22] as an example, if we set the amount of resources $w_i = 1$ for all the tasks T_i and maximum resources available $r_i = 1$ for all the processors P_i, the EAs adapted in [22] can also solve either sub-problem in this paper, and of course the fitness function will be specialized for the defect-tolerance mapping problem.

Therefore, our future work will employ some efficient greedy or heuristic algorithms, for example the greedy pin-assignment[10], to solve one assignment sub-problem in defect-tolerance logic mapping and use some excellent published EAs[21-22] to optimize the other sub-problem which is expected to be a very good balance between effectiveness and efficiency.

7 Conclusions

This paper redesigns the form of coverage which can calculate the number of crossbar modules required more accurately than the original form[9]. Based on this new criterion, an effective GA-based mapping algorithm for defect-tolerance logic design of nano crossbar architecture is proposed. The experiment results over various benchmark functions show the effectiveness and robustness of the proposed algorithm.

It is expected that hybrid domain-specific knowledge with evolutionary algorithms will improve the performance of the mapping algorithm, which is our future research direction.

References

[1] Bourianoff G, Brewer J E, Cavin R, Hutchby J A, Zhirnov V. Boolean logic and alternative information-processing devices. *Computer*, 2008, 41(5): 38-46.

[2] Cavin R, Hutchby J A, Zhirnov V, Brewer J E, Bourianoff G. Emerging research architectures. *Computer*, 2008, 41(5): 33-37.

[3] Lu W, Lieber C M. Nanoelectronics from the bottom up. *Nat. Mater.*, 2007, 6: 841-850.

[4] Yan H, Choe H S, Nam S W, Hu Y, Das S, Klemic J F, Ellenbogen J C, Lieber C M. Programmable nanowire circuits for nanoprocessors. *Nature*, 2011, 470: 240-244.

[5] Haselman M, Hauck S. The future of integrated circuits: A survey of nanoelectronics. *Proc. IEEE*, 2010, 98(1): 11-38

[6] DeHon A, Naeimi H. Seven strategies for tolerating highly defective fabrication. *IEEE Des. Test Comput.*, 2005, 22(4): 306-315.

[7] Hogg T, Snider G. Defect-tolerant adder circuits with nanoscale crossbars. *IEEE Trans. Nanotechnol.*, 2006, 5(2): 97-100.

[8] Hogg T, Snider G. Defect-tolerant logic with nanoscale crossbar circuits. *J. Electr. Testing: Theor. Appls,* 2007, 23(2-3): 117-129.

[9] Yellambalase Y, Choi M. Cost-driven repair optimization of reconfigurable nanowire crossbar systems with clustered defects. *J. Syst. Archit.*, 2008, 54(8): 729-741.

[10] Simsir M O, Cadamibi S, Ivancic F, Roetteler M, Jha N K. A hybrid nano-CMOS architecture for defect and fault tolerance. *ACM J. Emerg. Technol. Comput. Syst.*, 2009, 5(3): Article No. 14.

[11] Dai J, Wang L, Jain F. Analysis of defect tolerance in molecular crossbar electronics. *IEEE Trans. VLSI Syst.*, 2009, 17(4): 529-540.

[12] Rao W, Orailoglu A, Karri R. Logic mapping in crossbar-based nanoarchitectures. *IEEE Des. Test Comput.*, 2009, 26(1): 68-76.

[13] Crocker M, Hu X S, Niemier M. Defects and faults in QCA-based PLAs. *ACM J. Emerging Technol. Comput. Syst.*, 2009, 5(2): Article No. 8.

[14] Tunc C, Tahoori M B. Variation tolerant logic mapping for crossbar array nano architectures. In *Proc. the 15th Asia and South Pacific Design Automation Conference*, January 2010, pp.858-860.

[15] Garey M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco, USA, W.H. Freeman, 1979.

[16] Mitchell M. An Introduction to Genetic Algorithms (Complex Adaptive Systems). Cambridge, MA, USA: MIT Press, 1998.

[17] Kennedy J, Eberhart R C, Shi Y. Swarm Intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2001.

[18] Price K, Storn R, Lampinen J. Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series). Berlin, Germany: Springer-Verlag, 2005.

[19] Hahn P M, Kim B-J, Stutzle T, Kanthak S, Hightower W L, Samra H, Ding Z, Guignard M. The quadratic three-dimensional assignment problem: Exact and approximate solution methods. *Eur. J. Oper. Res.*, 2008, 184(2): 416-428.

[20] Loiola E M, de Abreu N M M, Bpavemtira-Netto P O, Hahn P, Querido T. A survey for the quadratic assignment problem. *Eur. J. Oper. Res.*, 2007, 176(2): 657-690.

[21] Salcedo-Sanz S, Yao X. A hybrid Hopfield network-genetic algorithm approach for the terminal assignment problem. *IEEE Trans. Syst., Man, Cybern. — Part B: Cybern.*, 2004, 34(6): 2343-2353.

[22] Salcedo-Sanz S, Xu Y, Yao X. Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems. *Comput. Oper. Res.*, 2006, 33(3): 820-835.

Bo Yuan received the B.S. degree in electronic information science and technology from University of Science and Technology of China (USTC), China, in 2009. He is currently pursuing the Ph.D. degree in electronic science and technology from USTC. His research interests include particle swarm optimization, memetic algorithm, defect tolerance for nanoscale systems, hardware/software codesign, maximum balanced biclique and subgraph isomorphism.

Bin Li received the B.S. degree in electrical engineering from Hefei University of Technology, China, in 1992, the M.Sc. degree in automatic control from Institute of Plasma Physics, China Academy of Science, Hefei, China, in 1995, and the Ph.D. degree in signal and information processing from University of Science and Technology of China, in 2001. He is currently an associate professor in the Department of Electronic Science and Technology, School of Information Science and Technology, University of Science and Technology of China. He has published 10 international journal papers, and more than 20 international conference papers. His major research interests include computational intelligence and its applications, electronic design automation. He has undertaken more than 10 projects sponsored by National Natural Science Foundation of China, Natural Science Foundation of Anhui Province, Ministry of Education of the People's Republic of China, the Department of Science and Technology of Anhui Province, Microsoft Research Asia, and Nokia Research China. He is a member of ACM and IEEE, He served as Technical Program Committee member of more than 20 international conferences, and as reviewer of several international and domestic journals.