Designing Minimal Sorting Networks Using a Bio-inspired Technique

Blanca C. López-Ramírez and Nareli Cruz-Cortés

Centro de Investigación en Computación,
Instituto Politécnico Nacional, Mexico City,
Mexico

bcelopez@gmail.com, nareli@cic.ipn.mx

Abstract. Sorting Networks (SN) are efficient tools to sort an input data sequence. They are composed by a set of comparison-exchange operations called *comparators*. The comparators are *a priori* fixed for a determined input size. The comparators are independent of the input configuration. SN with a minimal number of comparators results in an optimal manner to sort data; it is a classical NP-hard problem studied for more than 50 years. In this paper we adapted a biological inspired heuristic called Artificial Immune System to evolve candidate sets of SN. Besides, a local strategy is proposed to consider the information regarding comparators and sequences to be ordered at a determined building stage. New optimal Sorting Networks designs for input sizes from 9 to 15 are presented.

1 Introduction

Sorting Networks (SN) are tools utilized to sort fixed-size input data. An SN is composed by a set of *comparators*. Each comparator executes an action *compare-interchange* between two elements (a, b). The element a must be not greater than b. Otherwise, the values must be interchanged to (b, a). For a given input list of size n, its set of comparators is applied to the list, then the output should be a monotonically non-decreasing ordered list. SN are called *oblivious* meaning that their comparisons are independent of the input data or previous comparisons [13, 14]. Unlike other well known sorting algorithms (bubble sort, quicksort, heapsort, etc.), the sequence and number of comparisons are exactly the same no matter the input configuration (permutations).

Typically, SN are graphically depicted by n horizontal lines (called *buses*) representing the n input data (see Figure 1) and by some vertical lines representing comparisons between the value at its top extreme and the value at its bottom. If the value at the top is greater than the value at the bottom, these values must be swapped. The input data are placed at the left, then they go through the horizontal lines executing the comparisons found. The output is obtained at the right. The data must be ascendant sorted from top to bottom.

Consider, for example, an SN for $n = 4$ inputs illustrated in Figure 1. Each input data is placed on a horizontal line (bus), and the lines are labeled as x_0, x_1, x_2, x_3. The vertical lines are the *comparators* c_0, c_1, c_2, c_3, c_4, each receiving two values. The comparators c_0 and c_1 are executed first, then c_2 and c_3, and finally, c_4 as follows: c_0 evaluates $4 > 2$, thus the values of x_0 and x_1 are swapped; c_1 evaluates $1 < 3$, then the values of x_2 and x_3 remain without change. This process continues until all the five comparators are applied and the final list y_0, y_1, y_2, y_3 is obtained, which satisfies $y_0 \leq y_1 \leq y_2 \leq y_3$.

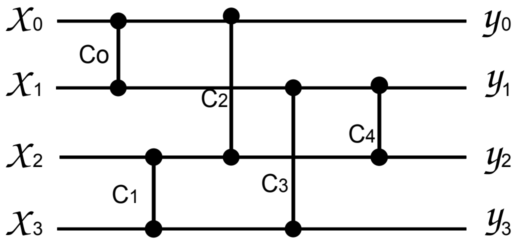

Fig. 1. Example of a Sorting Network with $n = 4$ inputs

To design optimal SN is a hard classical problem extensively studied for decades by researchers. Optimality can be considered according to different criteria such as parallelism or the number of comparisons performed; in this work we will refer to optimal SN as those with minimal number of comparators. In the specialized literature, only SN for small input sizes (less than 16) have been found. Perhaps, the most intensively studied SN are those for input size $n = 16$.

To determine if a certain SN is valid or not, it is necessary to verify its ability to sort any input configuration (permutation), which is an NP-hard process.

The Artificial Immune System is a population-based meta-heuristic utilized to solve complex problems [8, 3, 7, 4]. Some algorithms have been developed based on mechanisms observed in the biological immune system. Specifically, the Clonal Selection Algorithm was proposed to solve numerical and combinatorial optimization problems.

Notice that the problem at hand is highly sensitive to small changes, that is, any change of comparators or their position strongly affects the SN performance by turning it into an invalid one. So, for our goals, the Clonal Selection algorithm seems to be adequate basically because its main variation operator is *mutation*; this results less disruptive than other operators such as *crossover*. Besides, the mutation designed in this work can be controlled to allow small variations in possible solutions.

The Clonal Selection Algorithm is adapted to find efficient SN, that is, SN with low number of comparators. We found new designs for SN with sizes from 9 to 15 with minimal comparators.

The rest of the paper is organized as follows. In Section 2 some concepts concerning Sorting Networks and previous related work are presented. In Section 3 the Clonal Selection Algorithm of the Artificial Immune System is explained. Section 4 presents our proposal. In Section 5 the results and the new SN designs are presented. Finally, in Section 6 some conclusions and future work lines are given.

2 Sorting Networks

To better understand the SN illustrated in Figure 1, it can be interpreted as an algorithm presented below:

```
Require: ${X_0, X_1, X_2, X_3}$
 1: if $(X_0 > X_1)$ then
 2:     swap $(X_0, X_1)$
 3: end if
 4: if $(X_2 > X_3)$ then
 5:     swap $(X_2, X_3)$
 6: end if
 7: if $(X_0 > X_2)$ then
 8:     swap $(X_0, X_2)$;
 9: end if
10: if $(X_1 > X_3)$ then
11:     swap $(X_1, X_3)$;
12: end if
13: if $(X_1 > X_2)$ then
14:     swap $(X_1, X_2)$;
15: end if
16: $Y_0 = X_0$;
17: $Y_1 = X_1$;
18: $Y_2 = X_2$
19: $Y_3 = X_3$;
20: return ${Y_0, Y_1, Y_2, Y_3}$;
```

Algorithm 1: Sorting Network Algorithm for $n = 4$ corresponding to Figure 1

Generally speaking, the SN efficiency can be measured according to two criteria: 1) the number of comparators needed to order n input data, and 2) the parallel execution time spent by the SN. Of course, this last criterion makes sense only if the SN can be executed on a parallel architecture where independent comparators are executed simultaneously. A set of independent comparators is called a *layer*. Therefore, SN with less number of layers are considered to be faster.

If an optimal SN for the input size n can be designed (i. e., with a minimal number of comparators), it means that it is the best manner to sort n data. Designing SN with a minimal number of comparators and/or high parallelism is a classical and interesting open problem in Computer Science. Actually, nowadays only the optimal SN for input sizes $4 \leq n \leq 14$ are known.

In order to verify the SN validity, i. e., if it actually sorts any input configuration, it is necessary to evaluate all the $n!$ possible permutations. However, the so called *Theorem Zero-One* [13] is utilized to verify the SN validity reducing the number of evaluations. This theorem affirms that, if an SN sorts all 2^n sequences of zeros and ones into a non-decreasing order, then it will sort any arbitrary sequence of n numbers.

In our example with $n = 4$, to prove the SN validity by means of the Theorem Zero-One, we have to test if all the 2^4=16 binary sequences with length n are correctly sorted by the SN, i. e., if all the zeros appear at the top followed by all the ones. That is, we have to test each of the following sequences $\{0000, 0001, 0010, ..., 1111\}$. In Figure 2, for example, the sequence $\{1010\}$ is tested by the SN; the output at the right side is correctly sorted. So, if all the 2^4 sequences can be correctly sorted, then the SN is valid.

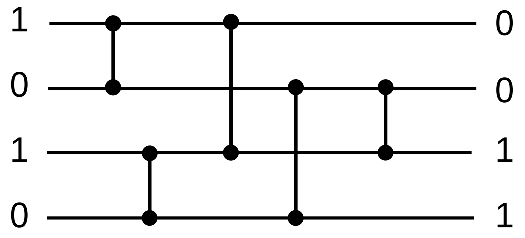

Fig. 2. Example of the SN for $n = 4$ to sort the binary sequence 1010

The first known optimal SN were published in 1945 by P. N. Armstrong, R. J. Nelson, and D. G. O'Connor for input sizes $n = 4, 5, 6, 7, 8$, with 5, 9, 12, 18, and 19 comparators, respectively [14]. In 1964, R. W. Floyd designed a new SN for $n = 9$. Later, in 1968, A. Waksman designed an optimal SN with $n = 10$. G. Shapiro and M. W. Green designed an SN for $n = 12$ with 39 comparators. Since 1960 some researchers have intensively studied the SN with $n = 16$. In 1969, Green found an optimal solution with only 60 comparators.

In early 90's, D. Hillis [10] used an evolutionary strategy based on a co-evolutive schema, finding an optimal SN with $n = 16$ by using a subset of comparators from Green's SN as an initialization process. In 1995, Hugues Jullie [11] proposed a method called Evolving Non-Determinism to find some solutions for SN with input sizes $9 \leq n \leq 16$.

Some recent works can be found in specialized literature mainly focused on reducing the execution time and algorithms' complexity. However, there have not been found better solutions than the previous ones in terms of the number of comparators that conform SN. For example, in 1997, J. Koza applied Genetic Programming able to find optimal solutions only for an SN with the size $n = 7$. In 2005, Choi and Moon [2] applied a Genetic Algorithm with local search capable to find some SN with $n = 16$ using 20 comparators based on Greens's optimal SN.

This paper proposes a scheme to design minimal SN without using predefined comparators. This approach is based on the Clonal Selection Algorithm of the Artificial Immune System. Some modifications are proposed to make it suitable for the SN optimization problem. The results show the capability of our technique to generate SN different from those obtained by previous works.

3 The Clonal Selection Algorithm

From the information processing point of view, the biological immune system has a number of interesting features such as intruder detection, memory, fault tolerance, pattern recognition, among others. The immune system functioning is very complex and not completely understood by the scientific community; however, there exist some models and theories trying to explain some specific process, for example, the Clonal Selection Theory proposed by Burnet in 1959 [1]. In most general terms, this theory establishes that only antibodies with high affinity regarding the external antigens will proliferate by cloning themselves. Then, these clones undergo some random changes named *somatic hypermutation* to improve their affinity and increase their capability to attach to and eliminate the antigens.

The biological immune system has inspired a set of meta-heuristics to solve difficult problems in Computer Science and Engineering. Specifically, the Clonal Selection Algorithm was proposed in [8] and used to solve mainly Computer Security and Optimization problems [6, 9, 5, 12]. A general idea of this algorithm is that a set of antibodies includes potential solutions, and the antigen is the objective optimization problem at hand. A numerical value (called*affinity*) is assigned to each antibody, which represents how well the antibody solves the problem. The antibody's cloning probability is assigned proportional to its affinity value. That is,

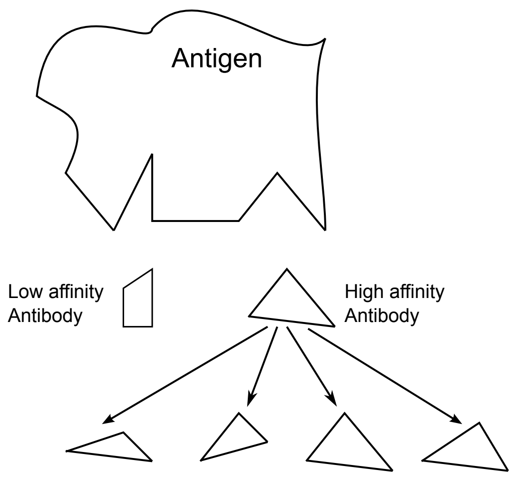

Fig. 3. According to the Clonal Selection Theory, only antibodies with high affinity are allowed to be cloned; then some mutations are applied to them

for antibodies with high affinity values, high cloning probabilities are assigned, and vice versa. In other words, best antibodies receive a higher quantity of clones, meanwhile worst ones create only a small quantity of clones.

Besides, the antibodies' hypermutation probabilities are inversely proportional to their affinity values. So, antibodies with high affinity undergo small changes through hypermutation, and vice versa. After these cloning and hypermutation processes are performed, a large quantity of cells appear, but only the best cells are allowed to survive in order to initiate a new iteration of the algorithm. This idea is illustrated in Figure 3.

4 Proposal

In general, our idea is to use an Artificial Immune System by means of the Clonal Selection Algorithm as a global strategy to minimize the number of comparators of the SN (which is defined as the *affinity* value). Besides, a local strategy is designed to select the comparators that will conform the SN. It is a process that assigns a fitness value to candidate comparators to be selected at a determined building stage.

The algorithm handles a set S of valid SN. Each element s, with $s \in S$, is an SN represented as a list of comparators. The set S is evolved by the Clonal Selection Algorithm to minimize the length of each s which is equivalent to minimizing the number of comparators that conform the SN.

The Clonal Selection Algorithm for a given input size n is presented as Algorithm 2. It generates a set S of valid SN (Line 1). The quantity of comparators of an element s is assigned as its affinity value. Next, the elements s are cloned (Line 3), and the resulting clones are mutated (Line 4). The best solutions are selected to update S (Line 6). This process is repeated a predetermined number of times.

More details about the representation, initial population, cloning, and mutation are explained in what follows.

```
Require: $n$ = input size of the SN.
1: Generate a random initial set $\{S\}$
   of valid SN (of size $|S|$);
2: Assign affinity value to each element $s$ with
   $s \in S$;
3: Clone each $s$ proportionally to their affinity
   value
   to conform the set of clones $\{K\}$;
4: Apply mutation to $k$, with $k \in K$, with
   probability inversely
   proportional to its affinity value;
5: Compute the affinity value
   for each modified $k$;
6: Select the best $|S|$ solutions from $\{S \cup K\}$
   to update $S$;
7: Repeat from Step 3 a determined
   number of times;
8: return The element of $\{S\}$ with the best
   affinity value.
```

Algorithm 2: Proposed Clonal Selection Algorithm to generate minimal SN

4.1 Representation

A potential SN is represented as a list of valid comparators. Each comparator is written as a pair (x, y) where x and y are indexes to the top and bottom buses, respectively, with $0 \leq x, y \leq n-1$ and $x < y$. This representation allows solutions with variable length which depends on the number of comparators conforming the SN.

For example, the SN shown in Figure 1 would be represented as

(0,1)(2,3)(0,2)(1,3)(1,2).

4.2 Initial Set S of SN

The initial set $\{S\}$ of SN (Algorithm 2, Step 1) is built randomly, but it is restricted to contain only *valid* SN. That is, each $s \in S$ should sort[1] any binary representation of the numbers from 0 to $2^n - 1$ (see Theorem Zero-One in Section 2). For that sake the Algorithm 3 is proposed. The inputs of this algorithm are the following sets:

— The set $\{B\}$ conformed by the integer numbers from 0 to $2^n - 1$ represented as binary chains of length n, and

— The set $\{C\}$ of all possible comparators. Each comparator $c \in C$ is written as a pair (x, y) where x and y are the comparator indexes to the top and bottom buses, with $0 \leq x, y \leq n - 1$, and $x < y$. The complete list is

$$C = \{(0,1), (0,2), ..., (0, n-1), (1,2), (1,3), ..., (1, n-1), ..., (n-2, n-1)\}. \tag{1}$$

This algorithm returns the set $\{S\}$ of size $|S|$ with a random initial population. Notice that the elements of S can be of different length.

4.3 Affinity Function

Affinity is defined as the number of comparators that conform s ($s \in S$). We try to minimize this function.

[1]Recall that a sorted binary chain in B means that all the zeros are at the top and the ones at the bottom.

```
**Require:** $n$, $\{B\}$, $\{C\}$
1: for i=0 to $|S|$ do
2:     $j = 0$;
3:     Copy $\{B\}$ to $\{B'\}$
4:     Randomly select an element $c$ with $c \in C$;
5:     Add $c$ into $S_{i,j}$ ;
6:     Apply $c$ to each element in $B'$ ;
7:     Remove the sorted binary chains of $B'$;
8:     j++;
9:     Repeat from Step 4 until $B'$ is empty;
10: end for
11: return $\{S\}$
```

Algorithm 3: Generation of the random initial set $\{S\}$ (from Step 1, Algorithm 2)

4.4 Cloning

Cloning (Line 3, Algorithm 2) consists in producing copies of current solutions s. Solutions with better affinity values will produce more copies of themselves, and vice versa. The number of clones is computed as follows:

— Sort the elements s ($s \in S$) according to their affinity value from the best to the worst,

— The number of clones $\#$ for the i-th element s_i is computed as

$$\#s_i = \sum_{i=1}^{|S|} \frac{|S|}{i}, \tag{2}$$

where $|S|$ is the number of elements of the set S.

4.5 Mutation

The hypermutation (or just mutation) is a process applied to clones to induce random changes to their configuration. This process is applied by choosing a point m of s, wiping out all the comparators after it, then the missing part is built again. The mutation rate should be high for clones with worse affinity values, and vice versa. A big change is introduced if the point m is closer to the beginning of s (at the left), and small changes, if m is closer to the end. The mutation process is performed by

Algorithm 4. Two different fitness functions are proposed to select a comparator at each stage. The first function (named $F1$) considers the quantity of binary chains in B that remain unsorted after the comparator is applied. The second function (named $F2$) considers the total quantity of bits in B that are wrong after the comparator is applied. Each of these fitness functions is randomly selected with a probability $pF1$ (Lines 7 to 22). The inputs of this algorithm are the following items:

— The clone k to be mutated;

— L which is the length of the original clone k, that is, the number of comparators that form k;

— The probability $pF1$ to choose the fitness function F1;

— The set $\{B\}$ conformed by the integer numbers from 0 to $2^n - 1$ represented as binary chains of length n;

— The set $\{C\}$ of all the possible comparators.

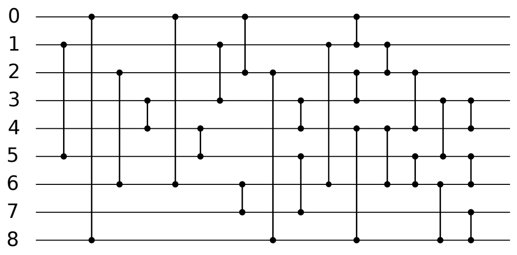

Fig. 4. New SN design for $n = 9$ with 25 comparators and 9 layers

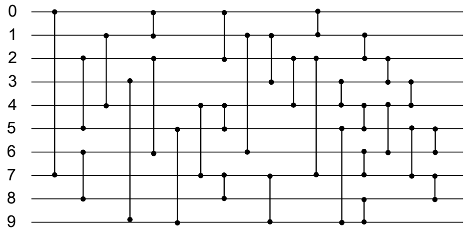

Fig. 5. New SN design for $n = 10$ with 30 comparators and 8 layers

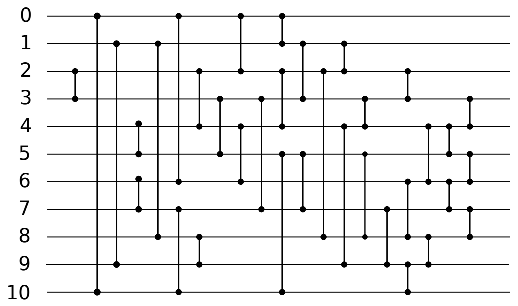

Fig. 6. New SN design for $n = 11$ with 35 comparators and 11 layers

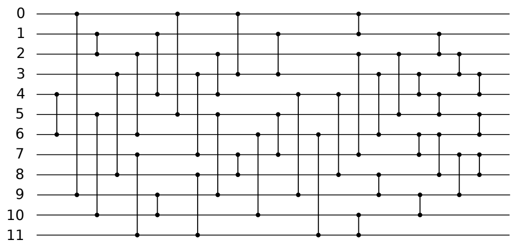

Fig. 7. New SN design for $n = 12$ with 39 comparators and 9 layers

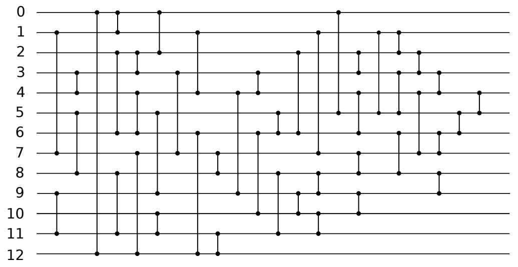

Fig. 8. New SN design for $n = 13$ with 45 comparators and 11 layers

5 Experimental Results

The proposed algorithm was used to design efficient SN for input sizes from $n = 9$ to $n = 15$. Table 1 shows comparison against the algorithms presented in Valsalam [15], and Choi and Moon [2]. The columns labeled with L show the number of comparators in the SN. The symbol P denotes the number of layers which represent the SN parallelism. Besides, the columns with Ini indicate the number of initial comparators that the algorithm takes from Green's SN [13].

```
Require: $k$ = A clone to be mutated
    $L$ = Length of $k$
    $pF1$= Probability to select F1 as fitness function $\{C\}$, $\{B\}$
 1: Copy $\{B\}$ to $\{B'\}$
 2: Select a mutation point $m$ with $0 \leq m \leq L$
 3: $j \leftarrow m$
 4: Wipe out all the elements in $k$ from $m$ to $L$
 5: Apply the comparators of $k$ to $B'$
 6: Remove the sorted elements from $B'$
 7: if pF1 then
 8:     for i=0 to $|C|$ do
 9:         Copy $B'$ to $Temp$
10:         Apply the comparator $c_i$ to the chains in $Temp$
11:         Remove the sorted chains from $Temp$
12:         Assign the fitness of $c_i$ as the number of chains in $Temp$
13:     end for
14: else
15:     for i=0 to $|C|$ do
16:         Copy $B'$ to $Temp$
17:         Apply the comparator $c_i$ to the chains in $Temp$
18:         Remove the sorted chains from $Temp$
19:         Compute the Hamming distance between the elements of $Temp$ and their corresponding
            already sorted sequences
20:         Assign the fitness of $c_i$ as that Hamming distance
21:     end for
22: end if
23: Select the comparator $c*$ with the minimal fitness value
24: Aggregate the comparator $c*$ in $k_j$
25: j++;
26: Apply the comparator $c*$ to all the elements in $B'$
27: Remove the sorted binary chains of $B'$
28: Repeat from Line 7 until $B'$ is empty
29: $k' \leftarrow k$
30: return $\{k'\}$
```

Algorithm 4: Hypermutation process applied to a clone k (from Line 4, Algorithm 2)

All the SN found by the proposal are new designs. They are presented in Figures 4, 5, 6, 7, 8, 9, and 10, for $n = 9$ to $n = 15$, respectively.

For the case of $n = 10$ in D. Knuth's book [13], two different SN are presented: the first with 29 comparators and 9 layers, and the second with 31 comparators and 7 layers. Our algorithm found an SN with 30 comparators and 8 layers which is a compromise solution between these two.

6 Conclusions

The proposed Clonal Selection algorithm works as a global strategy to look for SN with minimal number of comparators. However, it was necessary to incorporate a local strategy in order to improve the results. Such strategy was designed by means of the mutation operator. It considers a fitness function related to a specific comparator at a determined building stage. Local information is related to the quantity of sequences that a determined

Table 1. Comparison of Choi and Moon [2], Valsalam [15], and the proposed technique for SN with input sizes from $n = 9$ to $n = 15$

n	Best Known		Choi Moon			Valsalam			SIA		
	L	P	L	Ini	P	L	Ini	P	L	Ini	P
9	**25**	**7**	-	-	-	25	-	9	**25**	-	9
10	**29**	**9**	29	-	-	29	-	10	**30**	-	**8**
11	**35**	**8**	-	-	-	35	-	11	**35**	-	11
12	**39**	**9**	39	18	-	39	-	11	**39**	-	**9**
13	**45**	**10**	-	-	-	45	-	16	**45**	22	11
14	**51**	**9**	-	-	-	51	-	16	**51**	24	13
15	**56**	**9**	-	-	-	56	-	15	**56**	29	15

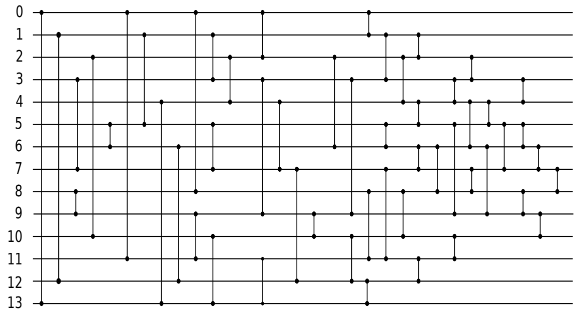

Fig. 9. New SN design for $n = 14$ with 51 comparators and 13 layers

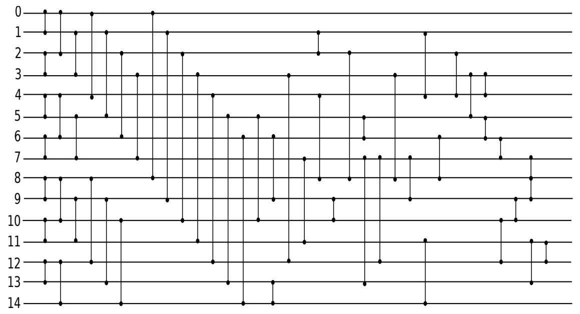

Fig. 10. New SN design for $n = 15$ with comparators and layers

comparator can sort at that moment and how disordered these sequences are. The experimental results showed that this strategy is able to find new optimal SN in terms of the number of comparators and parallelism.

As future work we plan to experiment for SN with larger input sizes, and to treat the problem as a biobjective one, in which parallelism is also considered in an objective function.

References

1. **Burnet, F.** (**1976**). A modification of jerne's theory of antibody production using the concept of clonal selection. *A Cancer Journal for Clinicians*, Vol. 26, pp. 119–121.
2. **Choi, S.-S. & Moon, B. R.** (**2005**). A graph-based lamarckian-baldwinian hybrid for the sorting network problem. *IEEE Trans. Evolutionary Computation*, Vol. 9, No. 1, pp. 105–114.
3. **Coelho, G. P. & Von Zuben, F. J.** (**2006**). Omni-ainet: an immune-inspired approach for omni optimization. *Proceedings of the 5th international conference on Artificial Immune Systems*, ICARIS'06, Springer-Verlag, Berlin, Heidelberg, pp. 294–308.
4. **Cruz Cortés, N. & Coello Coello, C. A.** (**2003**). Multiobjective optimization using ideas from the clonal selection principle. *GECCO*, pp. 158–170.
5. **Cutello, V. & Nicosia, G.** (**2002**). An immunological approach to combinatorial optimization problems. *Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence*, IBERAMIA 2002, Springer-Verlag, pp. 361–370.
6. **Dasgupta, D.** (**1999**). *Artificial Immune Systems and Their Applications*. Springer, Verlag, Berlin.
7. **de Castro, L. N. & Timmis, J.** (**2002**). *Artificial immune systems - a new computational intelligence paradigm*. Springer.

8. **de Castro, L. N. & Zuben, F. J. V.** (**2002**). Learning and optimization using the clonal selection principle. *IEEE Trans. on Evolutionary Computation*, Vol. 6, No. 3, pp. 239–251.

9. **Harmer, P. K., Williams, P. D., Gunsch, G. H., & Lamont, G. B.** (**2002**). An artificial immune system architecture for computer security applications. *IEEE Transactions on Evolutionary Computation*, Vol. 6, pp. 252–280.

10. **Hillis, W. D.** (**1990**). Co-evolving parasites improve simulated evolution as an optimization procedure. *Phys. D*, Vol. 42, No. 1-3, pp. 228–234.

11. **Juillé, H.** (**1995**). Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces. *Third European Conference on Artificial Life*, pp. 27–32.

12. **Kim, J., Bentley, P. J., Aickelin, U., Greensmith, J., Tedesco, G., & Twycross, J.** (**2008**). Immune system approaches to intrusion detection - a review. *CoRR*, Vol. abs/0804.1266.

13. **Knuth, D. E.** (**1998**). *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

14. **O'Connor, D. G. & Nelson, R. J.** (**1962**). Sorting system with n-line sorting switch. *United States Patent number 3,029,413*, Vol. 6.

15. **Valsalam, V. K. & Miikkulainen, R.** (**2011**). Utilizing symmetry and evolutionary search to minimize sorting networks. Technical Report AITR-11-09, Department of Computer Sciences, The University of Texas at Austin, Austin, TX.

Blanca C. López-Ramírez obtained her PhD in August, 2014. She is professor at the Departament of Systems and Computing of Instituto Tecnologico de Roque, Celaya, Gto., Mexico. Her research interests are bio-inspired algorithms and their applications.

Nareli Cruz-Cortés obtained her PhD in 2004 from Cinvestav-IPN, Mexico. She is research professor at Centro de Investigación en Computación, Insituto Politécnico Nacional, Mexico City, Mexico. She is National Researher of Mexico (SNI) level 1. Her research interests are cybersecurity algorithms, bio-inspired algorithms and their applications.

Article received on 12/04/2014; accepted on 03/06/2014.