Improving the Survivability of a Simple Evolved Circuit through Co-evolution*

Michael L. Harrison  
University of Idaho  
Computer Science Department  
Moscow, ID 83844-1010  
michaelh@cs.uidaho.edu

Dr. James A. Foster  
University of Idaho  
Computer Science Department  
Moscow, ID 83844-1010  
foster@cs.uidaho.edu

Abstract

For many applications it is not enough for a system to perform correctly under ideal circumstances. Systems benefit from robust and survivable designs that can withstand faults and degrade gracefully as faults occur. System topologies that resist component failures are certainly attractive in many applications. We demonstrate the effectiveness of a co-evolutionary approach for discovering survivable sorting network topologies. Our study shows an improvement in survivability as selective pressure is applied, that it is possible to tune the degree of survivability in a sorting network, and the mechanisms by which this happens are general enough that the results should apply to any evolved hardware with serial components.

1. Introduction

Evolutionary computing is modeled on natural design processes and in nature we typically find survivable designs since the process inherently favors them. We show that selective pressure can be exploited to produce more robust and survivable designs when evolving sorting network topologies. Several related studies focus on fault tolerance in evolved systems and show that evolutionary design techniques produce fault tolerant designs, often without this particular objective[1, 2]. Here we study a specific aspect of fault tolerance, namely survivability, and show that as selective pressure for survivability increases we observe an increase in the survivability of the resulting sorting networks. This preliminary evidence demonstrates the possibility of extending the automated design of circuits with evolutionary approaches to produce survivable solutions.

2. Background

Fault tolerance is the ability of a system to continue a degree of normal operation in the presence of faults[3]. The breakdown of a system due to internal failure can be viewed as a chain of events from fault to error to failure. A fault is an internal defect in a software or hardware system, such as a design flaw or a component failure. Incorrect behavior inside a system, caused by a fault, is termed an error. Finally, a failure occurs when a system is unable to perform its specified task or tasks due to errors[4].

Survivability is one aspect of fault tolerance and measures the expected level of service when a fault occurs within a system[5]. A fundamental concept of survivability is spreading risk, a survivable system shouldn’t depend heavily on any single point of failure.

A fault will not necessarily cause system failure. If a fault is masked by the system, the system maintains its ability to function despite the presence of a fault. A system with active fault masking has abilities to detect and reconfigure and thereby ignore the effect of the faulty component. Systems relying on passive fault masking are characterized by taking no active role in diagnosing or repairing themselves, instead relying on internal structure and redundancy to mask faults. This study is concerned with arranging components into passive fault tolerant structures in which the circuit topology provides the tolerance and therefore survivability. To explore the evolution of fault tolerant structures we utilize sorting networks as a simple test problem and evolve them using co-evolution.

Masner’s studies examined the affect of various algorithmic choices on the fault tolerance of evolved sorting networks. It should be noted that no explicit fault tolerance pressure was placed on the networks during evolution. In his paper, Size Verses Robustness: Is Bigger Better[1], he measured the benefit of simply using more components. He found that a few extra components above the minimum produced the best result with regard to his fault measure and that little or no benefit was realized with significantly larger

---

* This work was supported by the NSF-Idaho EPSCoR Program and by the National Science Foundation under award numbers EPS-0132626 and EPS-00809035.
networks. In his other study, *Representation and Robustness for Evolved Sorting Networks*[2], he found various encodings for sorting networks produced differing levels of robustness.

### 2.1. Sorting Networks

Sorting networks (SN) have long been of interest to computer scientists. Knuth used nearly 350 pages discussing them in his book *The Art of Computer Programming: Sorting and Searching*[6]. SNs provide a rich search space; even for our small 11 input, 45 gate sorting networks there are \( \binom{11}{2}^{45} \approx 2^{200} \) possible networks. Many studies, including the first study using co-evolution by Hillis[7], focus on finding minimally sized sorting networks[8, 9, 10]. Hillis’ work inspired the techniques used in this study, but we emphasize that our goal is to evolve reasonable fault tolerant structures, not minimal sorting networks.

Sorting networks are simple feed forward circuits used to sort values. Figure 1A demonstrates a 5 input, 9 gate sorting network. Values presented to the input lines (horizontal lines) carry the values from left to right. Along the lines the values encounter compare-exchange (CE) gates (vertical lines). CE-gates are simple two input sorting circuits which exchange their input values if they are out of order. The four possible inputs to the simplest of sorting networks (the 2 input, 1 gate network) is displayed in Figure 1B.

According to the 0-1 principle a sorting network topology is valid if it sorts all possible \( n \) length bit strings, where \( n \) is the number of inputs. Therefore, testing any \( n \) input network requires a series of \( 2^n \) tests. For our 11 input networks this means \( 2^{11} = 2048 \) possible test cases[6].

### 2.2. Co-evolution

A traditional genetic algorithm contains a single population with individuals that compete based on some fitness measure. Co-evolution uses multiple populations to aid in the evolution process. There are two main types of co-evolution, competitive and cooperative.

In this study we are concerned with competitive co-evolution[11, 7]; pitting two populations against one another, competing for fitness. This is a kind of evolutionary arms-race where one population targets the weaknesses of the other for its own benefit and in this way the fitness measure of each population depends on the other. A major benefit of co-evolution is the ability to evolve without a complete fitness evaluation. During evolution it may only be necessary to use a partial fitness evaluation. Hillis showed that this approach worked well for discovering small sorting networks when compared to full fitness evaluations (too slow) and random test cases[7]. This is particularly useful for sorting networks as there are an exponential number of test cases to be considered.

### 3. Methodology

A valid sorting network sorts properly all possible inputs but can be compromised by the presence of faults. A fault is defined as the removal from the network of a CE-gate. If a network depends heavily on a particular CE-gate the faulting of this gate results in many unsorted outputs.

To provide selective pressure toward fault tolerance we evolve two populations simultaneously using co-evolution. One population contains sorting networks, the other is a test population containing the input/fault pairs (IFPs). Each IFP contains two parts; a bit string used as an input to the sorting networks and a fault. The IFPs pressure the sorting networks to evolve correct solutions in the presence of faults. In turn, the sorting networks pressure the IFP population to discover IFPs that cause failures.

We should mention that the evolutionary parameters such as crossover, mutation, elitism, etc. were chosen for convenience and through our initial experiments. We believe that better choices might be made in the future and further improve the results of these experiments.

#### 3.1. Quantifying Survivability in Sorting Networks

We will introduce a fault tolerance calculation for sorting networks so that we can measure the survivability of a
sorting network. Analysis is done after evolution has completed and takes no part in the evolution process. We define a sorting network fault as the removal of a gate from the network. This is typically referred to as a benign fault because the component simply drops out of the system instead of performing the wrong action or malicious behavior. Benign faults were chosen to simplify the fault analysis and because it is difficult to justify a more complicated fault model for such simple digital circuits.

When a gate is removed we expect that the network will incorrectly sort some inputs (unless the fault is completely masked). Removal of certain CE gates will cause more harm than others. To determine the effect of a fault in the network we simply remove the gate we are interested in and evaluate all possible inputs. Every incorrect bit in the output strings is counted and demonstrates the dependence of the network on this particular gate. By evaluating all possible fault (1 fault per gate) and input string combinations (all length 11 bit strings, $2^{11}$ total strings) and we can compare the survivability of sorting networks. The goal is to reduce the affect any fault has on the performance of a sorting network; in other words we don’t want the overall network to depend on any particular gate. So we compare the total number of failures summed over all faults and we can also compare the worst case (the fault that causes the most failures). Using these two measures we can compare sorting networks.

Table 1 shows some of the standard parameters for our genetic algorithm. Initially, both populations are generated using random values for CE-gates in the sorting networks, input strings, and faults. Following evaluation, each population is sorted according to fitness. Selection is performed on both populations using a standard 7 member tournament. Tournaments are conducted within the top 30% of each population and continue until the offspring replace the bottom 70%. This preserves the top 30% in each generation.

3.2. Sorting Network Encoding, Crossover and Mutation

We represent sorting networks as a list of stages, where each stage includes five CE-gates. Each CE-gate is represented by a pair of integers, which indicate the two input lines to that CE-gate. To translate the genotype to its phenotype, a functioning SN, we add CE-gates from left to right in each stage. Each input line is only used once within a stage, so CE-gates that reference previously used lines in the same stage are eliminated from the SN. Our 11 input SNs allow for 9 such stages, resulting in SNs containing between 9 CE-gate (1 in each stage) to 45 CE-gates (5 in each stage). See 2B. The minimum size of a correct 11 input sorting network is 35 with 9 stages so our resulting networks contain the same number of stages but with a few more gates.

Figure 2. (A) Integer/fault pair encoding, input string followed by index of gate to fault. (B) Sorting network encoding showing 9 stages and demonstration of single stage with some CE-Gates not expressed. Gates [4:9] and [1:7] conflict with the previous gates [4:7] and [2:1] so are left unexpressed.
Crossover is standard 2-point crossover with randomly selected crossover points selected along the list of CE-gates. Crossover doesn’t split up integers that comprise a CE-gate. Two parent networks produce two child networks. A child may be mutated with some probability (denoted in Table 1) by randomly choosing a CE-gate in the array and randomly modifying one of its input lines.

### 3.3. Input/Fault Pair (IFP) Encoding, Crossover and Mutation

A input/fault pair is encoded as a bit string and fault pair as shown in Figure 2A. The binary string on the left is the 11 bit input to the sorting network and the integer on the right is a value from 1 to 45 indicating the index of the CE-gate to be faulted. It is possible for a sorting network to have fewer than 45 CE-gates so these faults may refer to nonexistent CE-gates.

Crossover on input strings is performed using standard 1-point crossover with a random crossover point resulting in 2 children. The faults are then assigned randomly to the two children. After crossover each child may be selected for mutation with the probability noted in Table 1. If selected one of either the fault or the bit string is selected for mutation with even probability. The bit string is mutated by randomly selecting a bit and flipping it. The fault is mutated by randomly assigning its value to another CE gate index.

### 3.4. Evaluating Sorting Network and Input/Fault Pair Fitness

As described previously, a fault is the removal of a CE-gate from a network. When an IFP is evaluated by a SN the faulty CE-gate associated with that IFP isn’t allowed to participate. This missing CE-gate or fault may cause the sorting network to sort its inputs incorrectly which affects the fitness of both the SN(negatively) and the TS(positively).

![Figure 3. Evaluation of a sorting network using an input/fault pair. The fault from the IFP is applied to the sorting network and then the input is applied. In this case the number of errors is 2.](image)

The effect of a fault is topology specific, so a input/fault pair may not work well on testing a wide variety of sorting networks. However, we believe the sorting network population will periodically converge on a similar set of solutions which the test strings/faults will exploit.

Sorting networks are evaluated based on their ability to correctly evaluate all the individuals in the IFP population. Every IFP in the test population is applied to every individual in the sorting network population. Given a SN and an IFP we calculate their relative scores in the following manner:

1. Apply the fault from the IFP by removing the CE-gate indicated in the IFP.
2. Sort the input bit string using the sorting network.
3. Determine the bitwise error of the output by calculating the hamming distance between the sorting net-
Table 2. This table demonstrates the fitness evaluation for a input/fault pair(IFP) population and a sorting network (SN) population. For clarity we use smaller, 4 bit strings. Individual are evaluated by applying each 4 bit string to all members of the sorting network population and summing the bitwise errors (hamming distance from correct output) in the output. Similarly, sorting networks are evaluated by summing the bitwise errors (hamming distance from correct output) over all test strings in the test string population.

<table>
<thead>
<tr>
<th>IFP</th>
<th>Input</th>
<th>Correct</th>
<th>Output</th>
<th>Score</th>
<th>Output</th>
<th>Score</th>
<th>Output</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFP1</td>
<td>1010</td>
<td>0011</td>
<td>0011</td>
<td>0</td>
<td>0011</td>
<td>0</td>
<td>1001</td>
<td>2</td>
</tr>
<tr>
<td>IFP2</td>
<td>0110</td>
<td>0011</td>
<td>0101</td>
<td>2</td>
<td>0011</td>
<td>0</td>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>IFP3</td>
<td>1000</td>
<td>0001</td>
<td>0010</td>
<td>2</td>
<td>0001</td>
<td>0</td>
<td>0001</td>
<td>0</td>
</tr>
</tbody>
</table>

SN Scores: 0+2+2=4 0+0+0=0 2+2+0=4

Figure 4 shows the analysis of the sorting networks with the lowest total number of failures across all faults from each of the three test runs. This is a valid measure for comparison as the expected number of failures for a sorting network given a randomly chosen CE-gate is simply the average failures per gate and so a lower expected failure rate indicates a more survivable sorting network. For each SN a list of the number of faults on a gate by gate basis was created and then sorted for comparison. The y-axis represents the number of failures and the CE-gates are along the x-axis. These have been sorted from highest failure rate to lowest for comparison. The graph indicates that as selective pressure is increased the dependence on any single CE-gate is reduced. The reduction of faults in Figure 4 is typical across all runs as shown in Table 3. It is worthwhile to note that since the number of CE-gates affects the failure rate we would expect a lower number of failures as the number of gates increase. The three best sorting networks were of similar size: 39 for co-evolved faults, 40 for random faults, and 39 for no faults.

We should also consider the worst case in each of the best sorting networks. The worst CE-gate in the 'no faults' run caused 310 failures, the 'random faults' run caused 239, and the 'co-evolved faults' caused 125. This is a reduction of 23% from no faults to random faults, and a reduction of 55% from no faults to co-evolved faults. This reduction indicates that in the worst case the sorting network co-evolved with faults will fail 60% less than the sorting network evolved without this benefit, a significant decrease.

Table 3 shows summary data from the 36 runs about the worst case single faults and total faults overall. The results show a reduction as the fault pressure is increased. One interesting result is the narrow range of sized for the sorting networks. The representation in the GA allows networks to have between 35 and 45 CE-gates but all resulting networks are tightly concentrated around 40.

We believe that this experiment shows that a co-
<table>
<thead>
<tr>
<th>Fault Pressure</th>
<th>Worst Single Failures</th>
<th>Total Failures</th>
<th>Gates</th>
<th>Mean</th>
<th>Stdev</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Stdev</th>
<th>Min</th>
<th>Max</th>
<th>Generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>367</td>
<td>310</td>
<td>645</td>
<td>2467</td>
<td>555</td>
<td>1542</td>
<td>4011</td>
<td>40</td>
<td>41</td>
<td>37</td>
<td>1384</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1384</td>
</tr>
<tr>
<td>Random</td>
<td>307</td>
<td>239</td>
<td>566</td>
<td>1988</td>
<td>304</td>
<td>1418</td>
<td>2292</td>
<td>41</td>
<td>43</td>
<td>39</td>
<td>2137</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2137</td>
</tr>
<tr>
<td>Co-evolve</td>
<td>240</td>
<td>125</td>
<td>425</td>
<td>1397</td>
<td>178</td>
<td>985</td>
<td>2017</td>
<td>40</td>
<td>43</td>
<td>39</td>
<td>2855</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2855</td>
</tr>
</tbody>
</table>

Table 3. Data collected over 36 runs each based on the fault analysis of the resulting sorting networks.

Figure 4. Each line represents an analysis of the best network from each trial. Each gate in the three SNs was analyzed to determined the number of faults that result from its removal and then the list was sorted for presentation (worst on left, best on right). The number of faults per gate is on the y-axis and the sorted gates are on the x-axis.

evolution approach can be exploited to increase survivability in circuits and that the data presented is good evidence that robust system topologies can be evolved. We observed a reduction in the expected number of failures as well as a reduction in the worst case single point of failure as selective pressure was increased.

One weakness of this approach is the longer run times when introducing faults into the algorithm. The random and co-evolved faults tests averaged 1.5 times the number of generations as the no faults tests. We have not compared the co-evolution approach to using a complete evaluation (fitness based on all inputs and all faults). However, we believe that the complete evaluation approach would be too time consuming given its exponential nature.

In future experiments we will to apply this technique to real world applications. In addition to this we need to study the scalability of this approach and compare it with established survivability design methods. Initial experiments have been conducted on larger 16 bit sorting networks and the results are comparable to the experiments demonstrated here. It should be possible to extend these concepts to other beneficial attributes in circuits. We will also apply the technique to more complicated fault models. Also, aspects of the co-evolutionary process should be studied to determine the interaction of the two population and how evolutionary operators can be tuned to benefit them.

References


