Evolving Fault Tolerant Systems*
CSRP 385

Adrian Thompson

School of Cognitive and Computing Sciences, University of Sussex,
Brighton BN1 9QH, UK. E-mail: adrianth@cogs.susx.ac.uk

Abstract

The conventional mechanism used to gain fault tolerance is *redundancy*. In contrast, this paper suggests that artificial evolution can be used to produce systems that are *inherently* insensitive to faults, with fault tolerance becoming part of the task specification. The possible techniques are investigated, and the study is grounded in a real-world evolved electronic control system for a robot.

1 Introduction

If a defect occurs in the underlying implementation of a fault-tolerant system, it either continues unaffected or undergoes graceful degradation. In a harsh environment or a safety-critical application, a system might be required to retain a certain level of ability even if a computer's memory becomes slightly corrupted, or a few transistors fail. Fault tolerance is especially important in designs for integrated circuits, because it increases the yield of usable chips in the presence of unavoidable silicon defects, permitting larger and cheaper chips. Indeed, wafer-scale integration is not feasible without fault tolerance [1].

This paper investigates the production of fault-tolerant designs by artificial evolution. Firstly, I describe an evolved electronic control system to serve as an example in what follows. Then I shown that in some circumstances, evolution will *automatically* tend to produce designs that are insensitive to some faults. Next, the discussion is broadened to the practicalities of explicitly including fault tolerance as one of the properties required of the evolving system. I then note that defective components can sometimes be *exploited* by evolution as if they were working parts. Finally, we see that the evolution of these designs that are *by their nature* insensitive to faults can go hand-in-hand with more traditional redundancy approaches.

*To appear in the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), Sheffield, September 1995.

2 A Real-World Example

The specimen evolved system used as an example in this paper is a real electronic circuit evolved to control a real robot. The circuit is a simple example of "evolvable hardware" — a reconfigurable electronic architecture that can physically instantiate many possible circuits. By placing the configuration under evolutionary control, it is possible to evolve electronic circuits that are evaluated by their performance as real physical circuits implemented in hardware. This technique has profound implications, but here I will only sketch out the details necessary for the rest of this paper, and the interested reader is referred to [2].

The objective was to cause a two-wheeled robot always to move, but yet to keep away from all four walls of its rectangular enclosure as much as possible. The evolvable hardware architecture for the control system is shown in Figure 1: I call it a "Dynamic State Machine" (DSM). It is based on the well-known "direct-addressed ROM" implementation of a finite-state machine [3], but has been endowed with the potential for much richer dynamical behaviour, by putting many of the temporal constraints under genetic control. As in the finite-state machine, a RAM holds a look-up table of the next state to follow each possi-

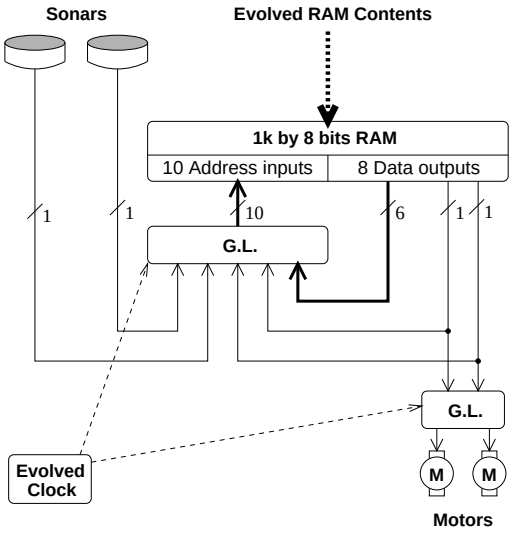

Figure 1: The evolvable DSM.

ble (present-state, input) combination. The clocked state-register that would normally hold the present state has been replaced by a "Genetic Latch" (G.L.), which behaves like the state register except that which of the variables are latched according to the clock, and which are passed straight through asynchronously is under genetic control. Genetic latches also control whether any of the inputs or outputs are clocked. All of the latches run from a common clock, but its frequency is under genetic control, as is the contents of the RAM.

The temporal freedom available in this arrangement means that the evolved DSM robot controller is able to accept directly the echo pulses from a pair of time-of-flight sonars mounted on the robot facing left and right, and to generate the pulse trains to drive the two d.c. motors as its outputs. For the simple wall-avoidance behaviour, only two of the RAM's data outputs and four of its address inputs were enabled, so only 32 bits of RAM and six latches were placed at the evolutionary algorithm's disposal.

A 16-bit binary code for the clock period, a bit for each signal passing through a genetic latch, and the 32 bits of RAM were all encoded directly onto a linear bit-string genotype. A conventional generational genetic algorithm (GA) was used [4], but with elitism (the fittest individual of each generation was always copied once without mutation into the next) and linear rank-based selection. The bitwise mutation probability was set to give an expected rate of one per genotype, the crossover probability was 0.7, and the population size was 30. Fitness was measured according to the performance of the real hardware DSM in controlling the real wheels (which were just spinning in the air), but the sonar echo signals were synthesised in real-time by a simulator. The products of evolution in this "virtual environment" have been verified to perform well in the real world — after about 40 generations, the behaviour is to move reliably to the centre of the arena and wander around there, even if started off facing into a corner.

Armed with this real-world example of a piece of electronics evolved to control a robot, the following sections discuss the evolution of fault tolerance. Note, however, that the concepts are general and are not confined to this particular arrangement.

3 The Evolution of Fault Tolerance

3.1 Via the Genetic Mutation Operator

A single-stuck-at (SSA) fault means that one signal in the system is clamped at an invariant value due to a defect. For a RAM chip, a SSA fault in the memory array causes a particular bit of the RAM always to read the same (either always 0 or always 1) no matter what is written to it, while all of the other bits function correctly. Now recall that for the evolving DSM example, the contents of the RAM chip were directly encoded, bit-for-bit, onto the bit-string genotype. As a consequence of this encoding, an application of the genetic mutation (bit-flip) operator to the section of the genotype coding for the RAM causes one of the RAM's bits to be inverted — the same effect as an adverse SSA fault. This section will show that evolved systems *automatically* tend to have some insensitivity to faults that have the same effect as genetic mutations, as in this example. The phenomenon arises from the way in which the fitness landscape [5] (the "topography" of fitness values assigned over the space of all possible genotypes) influences the distribution of individuals in an evolving population.

In considering the evolution of nucleic

acids, Eigen [6] defines a "quasi-species" as "a mutant 'clan' that is ordered around one or a degenerate set of selected master sequences, containing weighted contributions from all mutants present in the distribution. ...This distribution, and not a single type, is the target of selection." For a converged population, which has arrived at a local optimum, it can be imagined that the population forms a "cloud" on the fitness landscape, unable to converge completely upon the optimal genotype because of the forces of mutation, but held around it by crossover and selection. Consequently, optima that have surrounding regions of high fitness will be favoured over isolated optima standing out alone amongst low fitness genotypes, because it is *around* the optimum that most of the individuals will be found, not exactly *at* the optimal genotype.

Eigen [6] summarised several experiments which demonstrate this effect in a model used for the study of molecular evolution. Here, I adapt one of those experiments to the GA described in the previous section. Consider a population of 5-bit genotypes. Let the Hamming distance of an individual i from the sequence 00000 be $h(i)$, so that $h(i)$ is simply the number of '1's in i's genotype. Then define i's fitness as:

$$\text{fitness}(i) = \begin{cases} 10 & \text{if } h(i) = 0 \\ 9 & \text{if } h(i) = 5 \\ 5 & \text{if } h(i) = 4 \\ 0 & \text{otherwise} \end{cases}$$

This fitness landscape consists of two optima. The first is a global optimum of 10 for the genotype 00000, which is an isolated optimum: all genotypes near it in Hamming space (within three bit-flips) give zero fitness. The second optimum is for the sequence 11111, and has the slightly inferior fitness of 9, but is surrounded by a region of medium fitness, such that all five possible 1-bit mutants of the optimum have fitness 5. All other genotypes confer zero fitness. The GA was as described earlier (expected mutation rate of 1 per genotype, population size 30) except that the elitism mechanism was removed. To initialise the population, *all* of the genotypes were set to the 00000 global optimum, and the GA was then let to run. After 200 generations, the distribution of the population was measured by counting the number of individuals at each of $h(i) = 0, 1, 2, 3, 4, 5$. The measurements

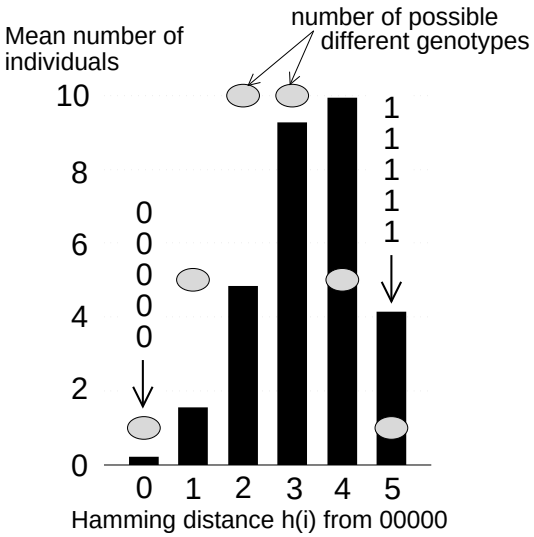

Figure 2: Mean population distribution.

were averaged over 100 runs of the GA.

The results (Figure 2) show that the population nearly always moved *away* from the isolated global optimum, in favour of the slightly suboptimal fitness peak, with its surrounding 1-bit mutant region of medium fitness. In the figure, the bar for $h(i) = 5$ is not the highest, even though the population is converged around this point, because there is only one possible genotype (11111) for $h(i) = 5$, but there are more possibilities for $h(i) = 4, 3, 2, 1$, as indicated. The outcome was similar even when the elitism mechanism was re-introduced, as long as there was more than 10% noise added to the fitness evaluations, or if the two optima were set to be of *equal* fitness.

So far, we have considered a highly contrived fitness landscape having only two optima. To see whether the result applies to more typical situations, the NK model of fitness landscapes was used[5]. N is the number of binary "genes" in the genotype. Each gene's additive contribution to the overall fitness is a real-valued function of the values of K other epistatically linked genes and itself. Each gene's function and epistatic linkages are chosen randomly and then held constant to define a static random landscape. N is the dimensionality of the fitness landscape, and K its ruggedness. K=0 gives a single-optimum landscape, and K=$N-1$ a maximally rugged (uncorrelated) one.

The GA described earlier (but with a pop-

ulation of 1000, a bitwise mutation probability of 0.005, and no elitism) was applied to a random N=20, K=10 landscape.[1] After 100 generations, the fittest individual was taken, and the mean fitness decrease caused by single mutations was calculated (averaged over all possible single mutations). Then, a new random N=20, K=10 landscape was generated, and exhaustive search was used to find that genotype with fitness closest to the one found by the GA on the other landscape. The mean fitness decrease of this non-evolved individual in the presence of single mutations was then compared to that of the evolved one. Repeating the entire process 250 times (until the results were statistically significant) showed that the fitness decrease of the evolved individuals in the presence of a single mutation was 5% less than for non-evolved ones of equal fitness[2], on average.

It can be tentatively concluded that when using a GA, the population will tend to converge upon a high-fitness region of the fitness landscape in which single mutations are less deleterious, on average, than if a similar result had been arrived at through non-evolutionary means.[3] Therefore, if the introduction of some type of fault affects the phenotype in the same way as would a genetic mutation, then the evolved system will automatically tend to tolerate a fault of that type better than a non-evolved (designed) system would. The effect will certainly depend upon the shape of the particular fitness landscape, the mutation rate and the population size, and it is not yet known if it is of any practical importance: the 5% improvement seen on N=20, K=10 landscapes is small.

As mentioned earlier, this phenomenon should cause the evolved DSM wall-avoider to be less sensitive to SSA faults in the RAM memory array than equivalent non-evolved DSMs. Unfortunately, it has not been feasible to produce DSMs as good as the evolved ones by other means (such as design), and it would be too time-consuming to repeat the comparison enough times for it to be

[1] Future work will determine the effect of varying these parameters.

[2] If the exhaustive search could not find an individual with fitness very close to the evolved one, or if the GA produced a freak very-poor result, then that trial was discarded.

[3] This may also apply, to a decreasing extent, to greater numbers of simultaneous mutations.

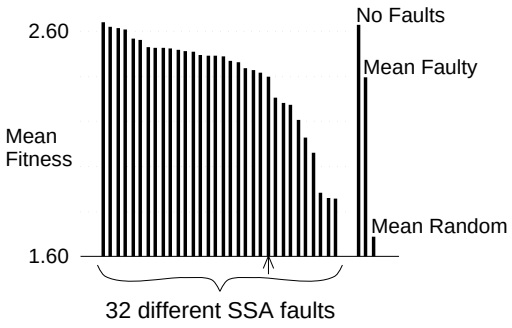

Figure 3: Sensitivity to adverse SSA faults.

statistically significant. However, Figure 3 shows that the evolved wall-avoider DSM is quite robust to adverse SSA faults — observation of the robot's qualitative behaviour bears this out — but it is not known how much is due to the effect described above, and how much is simply a property of the DSM architecture. The 32 possible adverse SSA faults were each emulated in turn by writing the opposite value to that specified by the genotype to the RAM bit in question. For each fault, the DSM was then used to control the robot (in the virtual environment) for sixteen 90-second runs from the same starting position, and the average fitness was measured to give the data in the figure.

This section has shown that when the introduction of some type of fault has the same effect on the phenotype as would a genetic mutation, then evolved systems will tend to be less sensitive to those faults than equivalent systems produced by non-evolutionary means. The observation is not confined to SSA faults: if, for example, the connectivity matrix of a neural network is directly encoded onto the genotype, then evolved networks should tend to be less sensitive to spurious breaking and creation of connections than non-evolved ones. The magnitude of the effect in a practical application will depend upon the particular fitness landscape; it is not yet known if it is great enough to be useful.

3.2 With an Environment of Faults

The argument of the previous section applies only when the genetic encoding is such that the faults of interest have the same phe-

notypic effect as a genetic mutation. What about faults and encoding schemes where that is not so? What if greater tolerance to faults is required than can be obtained in that way? Then the evolving system needs to be deliberately subjected to the faults of interest during its fitness evaluations, so that tolerance to them is an explicit part of the task to be performed[4]: the phenotype must operate in an environment of faults. The exposure to faults can most easily be done in a software simulation, but some fault emulation is also possible in an evolvable hardware architecture — the ability to introduce SSA faults into the DSM's RAM is an example (see previous section).

A problem with the "environment of faults" method arises when only a small proportion of the possible faults of interest have a serious effect on the system, but it is not known beforehand which those will be: it depends on how the system happens to evolve. If, when assessing the fitness of an individual, it is not subjected to *all* of the faults during its evaluation, but rather to a random selection of them, then it will often be those individuals which are lucky enough not encounter any crucial faults that score best, instead of those which are actually *better.* Such very noisy fitness evaluations reduce the efficacy of the evolutionary process. For all but the smallest systems, it is prohibitively time-consuming to test each individual in the presence of every possible fault, so some way of adaptively choosing those faults likely to disrupt the evolving system is required.

Hillis [8] faced a directly analogous problem in generating test cases for the evaluation of evolved sorting networks. They quickly evolved to sort all but a few test cases correctly, but it could not be determined *a priori* which would be the "problem" tests. Hillis' solution was to co-evolve test cases along with the sorting networks: the networks were scored according to how well they sorted the test cases, and the test cases by how well they found flaws in the sorters. The continuous and automatic adaptation of test cases by co-evolution was found to be superior to simply varying the test cases over time or over the two-dimensional grid upon which the population was spatially distributed.

[4] There is also the possibility that the Baldwin effect could occur, aiding the evolutionary process[7].

Hillis' result strongly suggests that the use of a *co-evolving population of faults* may be a way to subject individuals to faults during their evaluations, but without wasting time on faults to which they are already robust. It may then be possible to evolve tolerance to *all* of a large set of faults of interest, because the co-evolving faults would soon adapt to thwart a group of individuals that could be seriously affected by any subset of them. There is a danger that the co-evolving populations will become trapped in a cycle, without making useful progress: more empirical investigation into the applicability of this approach is needed.

3.3 By Exploiting Resources

If a particular defect persists for an extended period of time while the system is evolving, then the behaviour of the faulty part becomes just another component to be used: the evolutionary algorithm does not "know" that the part is supposed to do something else. For example, one of the SSA faults (the one marked with an arrow in Figure 3) was introduced as a permanent feature in the DSM, and the evolved controller was allowed to evolve some more. At first, the fitness of the population was dramatically lowered, with none of the individuals performing as well as the best of the population used to, but after 10 generations the mean and best fitnesses of the population had recovered to their previous values. In this case, the faulty part was *tolerated* rather than *used*, but in general this need not be so. This mode of fault tolerance may prove useful when transferring an evolved system between pieces of hardware having different defects, or to cope with slowly changing faults in the same hardware.

3.4 By Redundancy

This paper has concentrated on how the nature of the evolutionary process may be used to produce designs that are inherently fault-tolerant. However, the work reported in [9, 10, 11] shows that the more traditional fault tolerance technique of *redundancy* (the use of spares when faults are identified) may be integrated into an evolutionary framework. A special architecture for a field-programmable gate array integrated-circuit is presented, which sup-

ports the "embryological" development of a circuit specified by a genome (which could be evolved). During this development, and even during run-time, if some of the self-testing cells of the array are found to be faulty, the chip can automatically redistribute the expression of the genome so as to avoid those cells. This promising approach implies that the use of artificial evolution may be able to *augment* the highly effective fault-tolerance techniques already developed for hand-designed systems.

4 Conclusion

Traditionally, humans design fault-tolerant systems by providing spare parts. In contrast, artificial evolution can produce systems that are inherently tolerant to faults by the nature of their construction, without explicit redundancy. Viewing artificial evolution as an automatic design process, fault-tolerance can be integrated with the behavioural requirements and respected in all aspects of the design. Some insensitivity to faults that have the same influence on the system as a genetic mutation will tend to arise "for free." Tolerance to an arbitrary and large set of faults can possibly be achieved efficiently through the use of a co-evolving population of faults, which adaptively targets weak-spots. Implementation defects that are permanent or slowly changing may even have whatever properties they happen to exhibit put to use. Finally, the evolutionary approach can be used *as well as* more traditional redundancy methods. This early study suggests that artificial evolution may be well suited to the difficult, yet rewarding, challenge of fault-tolerant design, but much more empirical investigation is needed.

5 Acknowledgements

This research is funded by a D.Phil. scholarship from the School of Cognitive and Computing Sciences, for which I am very grateful. Special thanks to Phil Husbands, Dave Cliff and Inman Harvey.

References

[1] Moritoshi Yasunaga et al. Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed of 576 digital neurons. In *Int Joint Conf on Neural Networks (IJCNN'91)*, volume II, pages 527–535. IEEE, New York, 1991.

[2] Adrian Thompson. Evolving electronic robot controllers that exploit hardware resources. In F. Morán et al., editors, *Advances in Artificial Life: Proceedings of the 3rd European Conference on Artificial Life (ECAL95)*, pages 640–656. Springer-Verlag, 1995.

[3] David J. Comer. *Digital Logic & State Machine Design.* Holt, Rinehart and Winston, 1984.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimisation & Machine Learning.* Addison-Wesley, 1989.

[5] Stuart A. Kauffman. *The Origins of Order.* Oxford University Press, 1993.

[6] M. Eigen. New concepts for dealing with the evolution of nucleic acids. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume LII, 1987.

[7] G.E. Hinton and S.J. Nowlan. How learning guides evolution. *Complex Systems*, 1:495–502, 1987.

[8] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C. Langton et al., editors, *Artificial Life II*, pages 313–324. Addison-Wesley, 1992.

[9] P. Marchal et al. Embryological development on silicon. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 365–366. MIT Press, 1994.

[10] P. Marchal and A. Stauffer. Binary decision diagram oriented FPGAs. In *ACM International Workshop on Field-Programmable Gate Arrays (FPGA'94)*, Berkeley, February 1994.

[11] S. Durand and C. Piguet. FPGA with self-repair capabilities. In *ACM Int Workshop on Field-programmable gate arrays (FPGA'94)*, Berkeley, February 1994.