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.

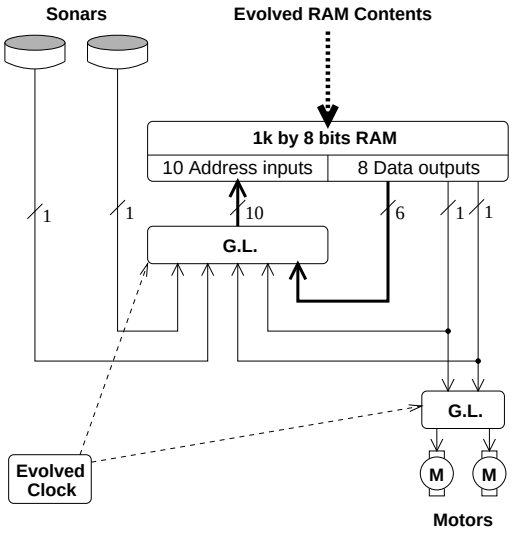

Figure 1: The evolvable DSM.

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 The clocked state-register that would normally hold the present state has been replaced by a "Genetic Latch" (G.L.), which behaves like the state register except that which of the variables are latched according to the clock, and which are passed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

con44he roo

acids, Eigen [6] defines a “quasi-species” as

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

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