EVOLVABLE HARDWARE

Julian Miller, chair

Lens System Design and Re-Engineering with Evolutionary Algorithms

Julie Beaulieu[†]**, Christian Gagné**[‡]**,** and **Marc Parizeau**[‡]
Laboratoire de Vision et Systèmes Numériques (LVSN)
[†]Département de Physique, de Génie Physique et d'Optique
[‡]Département de Génie Électrique et de Génie Informatique
Université Laval, Québec (QC), Canada, G1K 7P4
E-mail: jubeauli@phy.ulaval.ca, {cgagne,parizeau}@gel.ulaval.ca

Abstract

This paper presents some lens system design and re-engineering experimentations with genetic algorithms and genetic programming. These Evolutionary Algorithms (EA) were successfully applied to a design problem that was previously presented to expert participants of an international lens design conference. Comparative results demonstrate that the use of EA for lens system design is very much human-competitive.

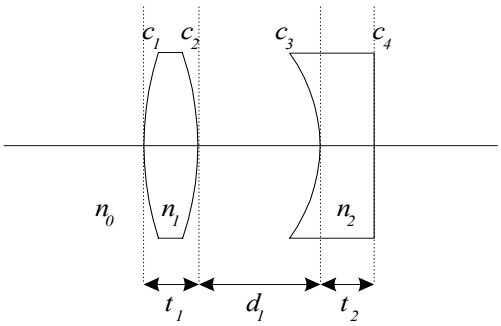

Figure 1: Parameters of a Two Lenses System. The n_i variables denote the refractive indexes of the media, the c_i represent the lens surface curvatures, the t_i are the lens thicknesses, and d_1 is the lens spacing.

1 INTRODUCTION

Designing a lens system is a complex task currently done by experimented optical engineers, using CAD tools that can optimize a roughly shaped design. The work presented in this paper is motivated by a desire to completely automate this design task, using Genetic Algorithms (GA) and Genetic Programming (GP) techniques.

The paper first presents the theory related to lens systems, before addressing a brief survey of modern design methods. Then, experimental results are presented for the automatic design of a benchmark problem for both the design (using GA and GP) and the re-engineering (using GP) of a lens system.

2 THEORY ON LENS SYSTEM DESIGN

A lens system is an arrangement of lenses with different refractive indexes, surface curvatures, and thicknesses. Figure 1 shows an example of a 2 lenses system. Given an object of a certain size, at a certain distance, its function is to produce an image of this object. Although many lens arrangements can generate images of the same size, the problem of lens system design is to seek the one with the least amount of aberration.

Aberrations are the difference between a real image and the corresponding approximated image computed with Gauss optics (O'Shea, 1985). Gauss optics constitute a usable framework to characterize an optical system with various Gaussian constants such as the effective focal length, stop, f-number of the system, and image distance and magnification. Aberrations come from the fact that Gauss optics are used during the design process; real physics of lens systems are too complex to be usable.

To characterize lens systems we need to do what is called ray tracing. Starting at a given point on the object and a given initial angle, a ray trace is the computation of the trajectory of a light ray through the

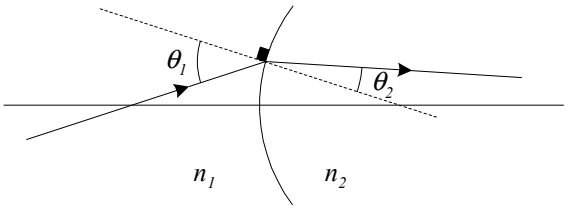

Figure 2: Illustration of Snell-Descartes First Law of Refraction

optical system until it reaches the image plane. The exact (real) ray trace is obtained from the first law of refraction (Snell-Descartes) that governs the behavior of light passing through the interface between two media having different refractive indexes. The path of a ray passing from medium 1 to medium 2 obeys the following equation:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2 \tag{1}$$

where n_1 and n_2 are refractive indexes of media 1 and 2, and θ_1 and θ_2 are incident and refracted angles relative to the normal of the interface between the two media. Figure 2 illustrates this first law of refraction. On the other hand the paraxial approximation consists in assuming that all rays lie close to the optical axis. Using the sine expansion:

$$\sin\phi = \phi - \frac{\phi^3}{3!} + \frac{\phi^5}{5!} - \cdots \tag{2}$$

then $\phi \approx 0 \Longrightarrow \sin\phi \approx \phi$. Equation 1 becomes:

$$n_1\theta_1 \approx n_2\theta_2 \tag{3}$$

This approximation is the basis of Gauss optics or first order optics.

The quantification of the aberrations of an optical system is done by computing the difference between the real image (i.e. the one that stems from Equation 1), and the image that results from the paraxial approximation. In other words, two ray traces emerging from the same point on the object with the same angle, one exact and one approximated[1], will strike the image plane at different positions. These correspondence errors, averaged over a whole set of distinct rays, could provide a convenient basis for building a quality measure.

Finally, in the sine expansion of Equation 2, it is interesting to note that if we also consider the second term, than we obtain what is called third order optics.

[1]The approximative ray trace is virtual and computed with Gauss optics.

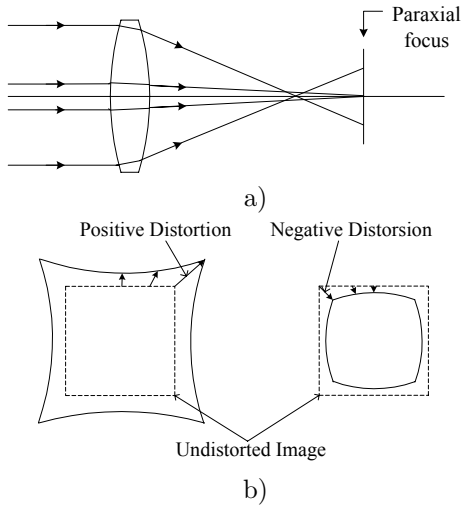

Figure 3: Illustration of: a) Spherical Aberration, and b) Distortion.

The difference between first and third order optics represents the five Seidel aberrations: spherical aberration, coma, astigmatism, field curvature, and distortion (O'Shea, 1985). Figure 3 illustrates two of these. The spherical aberration (Figure 3a) is caused by the fact that, for spherical lenses, rays coming from infinity and parallel to the optical axis do not converge to the same focus point, depending on the ray distance from the optical axis. The result of this type of aberration is a blurred image. Another type of aberration is distortion, that causes pincushion (positive distortion) or barrel (negative distortion) shaped images, as shown in Figure 3b.

3 EXISTING METHODS

Modern design of lens systems is generally done with specialized CAD softwares that help designers to visualize the lens system, evaluate its quality following precise criteria, and locally optimize the system's variables. This optimization is often done by using local search algorithms like the Damped Least Square (DLS) method. But the typical search space of optical system design is a complicated multidimensional space comprising several peaks, non-linearities and strong correlation between parameters (Sturlesi and O'Shea, 1990). Hence, a local search explores only the immediate neighborhood of the initial solution, making the result very dependent on the competence and experience of the designer. But since the beginning of the 1990's, some applications of global search methods have been made in optical design. A few researchers have successfully used simulated annealing for optical

design (Forbes and Jones, 1990; Hearn, 1990). Others have modified local optimization algorithms, like the DLS algorithm, to allow exploration beyond local optima (Isshiki, 1998). These two approaches have been recently integrated in some optical CAD tools.

But not much work has been done using Evolutionary Algorithms (EA). As far as we know, only (Ono et al., 1998) have designed some lens systems using real-coded genetic algorithms. They were able to automatically design lens systems made of more than 10 parts, for some imaging applications. They also experimented with multi-objective optimization of optical systems by using the Pareto optimal selection strategy. Other research on application of EA to optical systems includes (Ben Hamida et al., 1999), which use two evolutionary approaches to design an optical surface, and (Nosato et al., 2001), that use EA for the automatic alignment of optical devices.

4 MONOCHROMATIC QUARTET PROBLEM

To evaluate the capability of our approach for lens system design, we chose a problem stated in the 1990 International Lens Design Conference (ILDC). This conference, held every four years, includes a friendly design competition for its participants. The 1990 problem (O'Shea, 1990) became a benchmark to evaluate the performance of optimization algorithms for lens system design because the 11 best solutions proposed by human experts make up only two different classes of similar solutions, and the organizers concluded that these solutions appear to be global optimums of the solution space.

The problem is named the *Monochromatic Quartet*. Essentially, it consists in finding an optical system made of four spherical lenses. Here is the formal statement of the problem (O'Shea, 1990).

> Design a 4-element, f/3, 100 mm effective focal length lens of BK7 glass, illuminated by helium d wavelength (i.e., $n = 1.51680$). The object is at infinity, the object field covers 30° full field (15° semi-field angle) and the image field is flat.
>
> Constraints on the construction includes: only spherical surfaces, no aspherics, GRIN elements, Fresnel lenses, binary elements, holographic optical elements, etc. The minimum glass thickness is 2 mm, but there is no upper limit on the size of the lens. The distortion must be less than 1% and there should be no vignetting. The last is intended to assure that vignetting could not be used to improve the edge performance on the lens. No requirement is put on the location of the stop of the system.
>
> The merit function consists of the average of the RMS blur spot for three fields : on-axis, 10.5°, and 15°, weighted equally.

The f-number (also written f/#) measures the light-collecting ability of the lens system. An effective focal length for a lens system is similar to the focal length of an equivalent single lens. The focal length itself is the inverse of the lens power, which is the capacity of making rays converge over short distances. The BK7 glass is just an ordinary type of glass frequently used for lens fabrication. The helium d wavelength constraint specifies that the problem is monochromatic, that is the considered wavelength is fixed and thus the refractive indexes are also fixed (otherwise we would have to consider the so-called chromatic aberrations). This system must not have vignetting, i.e. the image must not be truncated. It is also possible to include a stop, that is an aperture in the optical system which limits the amount of light in the system, allowing to reduce aberrations. Its diameter can be linked directly with the effective focal length and the f-number.

The error measure of the problem seeks to separate distortion from other types of aberrations. The problem statement specifies that distortion must not exceed 1% and thus implies that below this level, one should only concentrate on other aberrations. Using exact computations (Equation 1), the RMS blur spot method traces several parallel rays at a given entrance angle. These angles must be set successively at 0°, 10.5°, and 15° as specified by the problem statement. Using paraxial approximation, all the rays with the same entrance angle converge at a single point. But with exact ray traces, they will strike the image plane at different points, generally in the neighbourhood of the approximate point, and form a so-called blur spot, as illustrated in Figure 4. The RMS blur spot is computed from the variances of the position at the image plane of different exact rays with the same entrance angle. A reference ray traced with the paraxial approximation is used to evaluate the distortion, by measuring its distance from the centroid of the exact rays at the image. For more details, the interested reader is referred to (Lambda Research Corporation, 2001).

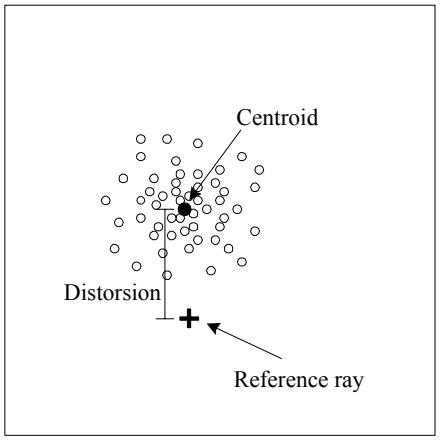

Figure 4: Illustration of the Blur Spot Measure

Table 1: Monochromatic Quartet Lens System Encoding with GA. Units for thickness, distance and stop location are mm. Units of curvatures are mm^{-1}.

Type of parameter	**# of par.**	**Encoding**	**Bounds**
Curvature	7	16 bits	$[-0.04, 0.04]$
Thickness	4	16 bits	$[2, 250]$
Distance	3	16 bits	$[0, 250]$
Stop Location	1	18 bits	$[0, 1000]$

5 LENS SYSTEM DESIGN WITH GA

As a first trial, we tried to design a lens system that meets the monochromatic quartet criteria using classical GA, with bit string representations of fixed length (Holland, 1975). The lens system modeling is straightforward and the problem has a total of 15 parameters to optimize. Table 1 summarizes the encoding and the upper/lower bounds for each parameter encoded in the bit strings. The chosen bounds are large enough to allow the exploration of all physically feasible solutions.

The value of the last curvature and aperture stop are not included in this table because these parameters are not evolved. They are set so that the lens system respects the problem specifications (effective focal length and f-number). The distance between the last surface and the image plane is also calculated in order that approximative rays having the same field angle focus on the image plane. Also, the lens systems are validated during the evolution to ensure that they are physically possible (to prevent lens overlap, etc.). When impossible configurations appear, the problematic lens diameters, distances between lenses or lens thicknesses are corrected until the system become physically feasible.

Table 2: Evolution Parameters with GA

Evolution parameter	**Value**
Population size	5000
Number of generations	1000
Crossover probability	0.03
Mutation probability	0.01
Participants to tournament selection	3

To implement this GA, we used a C++ framework for evolutionary computations named Open BEAGLE (Gagné and Parizeau, 2002). Also a C++ library, named *Library for Lens System Ray Tracing* (Gagné and Beaulieu, 2001), was developed to compute ray tracing through a given lens system. This library facilitates the fitness measure calculation, that needs to evaluate several exact and approximative ray traces.

For the fitness measure, we defined the following equation:

$$Fitness = \begin{cases} \dfrac{1.0}{1.0 + RMS} & \%_{dist} \leq 1.0 \\ \dfrac{1.0}{\%_{dist}} \times \dfrac{1.0}{1.0 + RMS} & \%_{dist} > 1.0 \end{cases}$$

where $\%_{dist}$ is the maximum percentage of distortion observed, and RMS is the average value of the RMS blur spot for the three initial field angles mentioned in the problem statement. This fitness equation provides a measure spectrum, normalized between 0 and 1, that is detailed enough to adequately differentiate individuals. It also ensures that the lens system having more than 1% of distortion are sufficiently penalized during the evolutions. Experimentally, we observed that all the best solutions obtained do not have more than 1% of distorsion.

Preliminary tests were first conducted in order to obtain a good idea of evolution parameters to use for this problem. The used parameters are presented in Table 2. Then, 30 evolution runs were conducted using these parameters, each needing about 2 to 3 hours of CPU time on an Athlon processors running at 1.2 GHz (the runs were conducted in parallel on a Beowulf cluster of 25 nodes).

6 RESULTS USING GA

The best solution obtained with GA produces an averaged RMS blur spot of 0.0019 mm, compared with 0.0024 mm for the best human result reported at the 1990 ILDC monochromatic quartet contest. This RMS

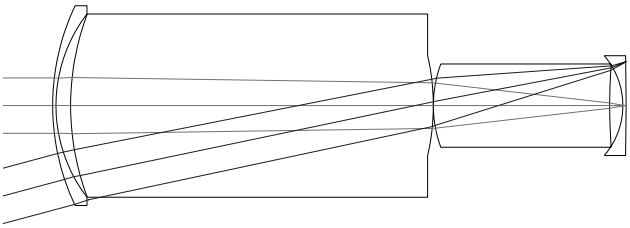

Figure 5: Best Monochromatic Quartet Presented at the 1990 ILDC

Table 3: Parameters of the Best Monochromatic Quartet Presented at the 1990 ILDC

Radius	Thickness	Aperture	Glass
140.0	2.0	60.0	BK7
90.6	8.8	55.0	air
155.86	206.8	55.0	BK7
0.0	**11.2**	**13.757654**	**BK7**
-134.7	0.05	30.0	air
72.6	106.0	25.0	BK7
357.38	7.9	25.0	air
-45.82	2.0	30.0	BK7
-1458.1	0.1	30.0	air

(Bold surface is the aperture stop.)

blur spot is the average of three statistical measures of many exact rays scattering from the three given entrance angle. The RMS value is dependent on both these entrance angles, but also on the choice of the computed rays for each angle. Moreover, when the RMS is very small, as in this case, it is not highly accurate due to rounding errors and image plane placement. Varying the image plane position may lower a little bit the RMS. The RMS measures reported here have been calculated with the well known CAD tool OSLO (Lambda Research Corporation, 2001), in order to eliminate any bias that our own RMS measure might introduce. Using our own RMS measure could have favored the evolved systems relatively to the human designed ones.

The best system evolved with GA is thus 23% better than the best presented at the 1990 ILDC (following the OSLO RMS measure). Figure 5 and Table 3 present the best 1990 ILDC design, while Figure 6 and Table 4 present the best GA design.

7 LENS SYSTEM DESIGN AND RE-ENGINEERING WITH GP

As a second trial, the same monochromatic quartet problem was tackled, using the same EC environment

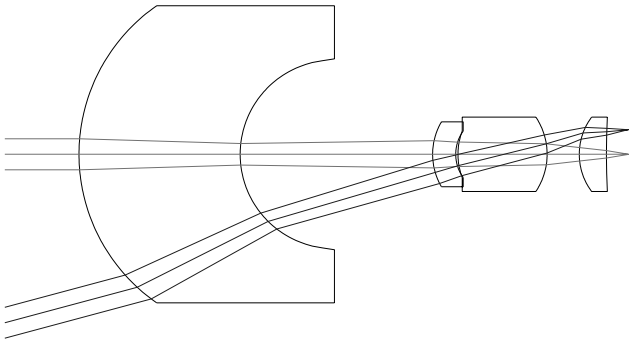

Figure 6: Best Design Found with GA

Table 4: Parameters of the Best Design Found with GA

Radius	Thickness	Aperture	Glass
194.6968	173.638	160.0	BK7
101.6046	207.408	160.0	air
69.2262	24.8946	35.0	BK7
42.8546	2.43763	25.0	air
70.8426	49.7265	25.0	BK7
0.0	**46.3269**	**12.172236**	**BK7**
-72.1277	34.6571	40.0	air
66.8914	29.2579	40.0	BK7
834.9267	23.7954	40.0	air

(Bold surface is the aperture stop.)

(Open BEAGLE), but this time specialized for GP. The evolving genetic programs represent some modifications to apply on a given initial lens system. The evolving programs are made of three types of primitives. The first type includes primitives that can increment/decrement an iterator that points to a lens surface. The second type is composed of primitives that modify a parameter of the current lens surface. The other type includes classic arithmetic operations. The terminals of the genetic trees are ephemeral constants (Koza, 1992), randomly generated between -1 and 1. Two ADFs (Koza, 1994) were also added to favor emergence of building blocks, which can be useful for this type of problem. Table 5 presents the complete set of primitives. Note that we intentionally encouraged the search to be in the initial solution neighborhood by allowing mostly small parameter modifications.

To evaluate each individual, an iterator is affected to the first surface of the initial lens system. Once the system is modified, it is validated to ensure that it respects the problem specifications and that it is physically feasible, as explained in section 5. Thereafter, the

Table 5: GP Primitives Used to Evolve Lens Systems

Primitives	Inputs	Description
`First`, `Last`	1	Set the iterator to the first/last surface.
`Next`, `Prev`	1	Iterate to the next/previous surface.
`Curv+`, `Dist+`	1	Add $0.1 \times \Delta \times$ *input* to the surface curvature/distance value[a].
`Curv*`, `Dist*`	1	Multiply the surface curvature/distance value with $1.0 +$ *input*.
`Stop+`, `Stop*`	1	Add to/multiply the stop location value (as with the curvature/distance).
`+`, `-`, `*`, `/`	2	Add, subtract, multiply, or divide two floating-point numbers.
`Ephemerals`	0	Randomly generated constants in $[-1, 1]$.

[a]The Δ term represents the maximum value for the corresponding variable. For curvature $\Delta = 0.04$, for distance $\Delta = 250$, and for stop location $\Delta = 1000$ (see Table 1).

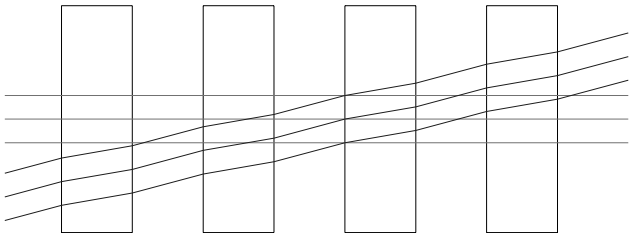

Figure 7: Raw System Used for Lens System Design with GP

Table 6: Parameters of the Raw System Used for Lens System Design with GP

Radius	Thickness	Aperture	Glass
0.0	50.0	80.0	BK7
0.0	50.0	80.0	air
0.0	50.0	80.0	BK7
0.0	50.0	80.0	air
0.0	**50.0**	**80.0**	**BK7**
0.0	50.0	80.0	air
0.0	50.0	80.0	BK7
0.0	50.0	80.0	air

(Bold surface is the aperture stop.)

lens system fitness is calculated. The fitness measure is the same as the one used for GA evolutions.

Two different approaches were applied to solve the monochromatic quartet problem with GP. Firstly, we tried the re-engineering of good solutions by using solutions obtained with GA as initial lens systems for the evolving process. Secondly, we tried the re-engineering using a raw lens system made of four "lenses" of zero curvature (see Figure 7 and Table 6). With this approach, the capacity of the GP to design a lens system from scratch is evaluated.

Using GP this way is interesting because the representation isolates the genotype from the phenotype (Gruau, 1994; Koza et al., 1999). The evolved programs (the genotypes) modify the initial lens system to spawn better ones (the phenotypes).

Again, preliminary tests were made to find adequate evolution parameters (see Table 7). For the raw system as a starting point, 16 differents runs were conducted, while 8 runs were made for the re-engineering of each of the 5 lens system designs selected from the GA results. The chosen GA designs are a representative set of results with GA, i.e. we chose some bad, good and very good lens systems in order to compare the re-engineering in each case. Each evolution needed an average of 36 hours for design and 3 hours for re-engineering on a single Athlon 1.2 GHz processor.

8 RESULTS USING GP

Table 8 presents the best GP re-engineering results for the 5 chosen GA solutions. Note that the best GA design (GA-1) is still the best result after the GP re-engineering, with a RMS blur spot of 0.0016 mm. This overall best result is 34% better than the best 1990 ILDC design. Table 9 presents the parameters of this overall best lens system. For all the best GP re-engineering, the topology remains unchanged from the initial lens system. This indicates that the GP re-engineering is probably doing a local search.

For the GP re-engineering using the raw system as starting point (design from scratch), the best result was an RMS blur spot of 0.0039 mm, which is 60% worse than the best 1990 ILDC design. The corresponding design is presented in Figure 8 and Table 10.

Table 7: Evolution Parameters with GP

Evolution parameter	Value
Population size	5×1000
Migration type	Unidirectional ring
Number of migrants	10
Number of generations	1000 (re-engineering) 5000 (design)
Crossover probability	0.9
Swap mutation probability	0.5
Participants to tournament selection	3
Initial tree height	[4, 7]
Maximum tree height	20

Table 8: Best Results Obtained with GP (Re-Engineering)

Case	Original RMS (mm)	Re-engineered RMS (mm)	RMS Shift
GA-1	0.0019	0.0016	-14%
GA-2	0.0084	0.0071	-16%
GA-3	0.0187	0.0139	-25%
GA-4	0.0308	0.0105	-66%
GA-5	0.0909	0.0843	-7.3%

Table 9: Parameters of the Best Re-Engineering Design Found with GP

Radius	Thickness	Aperture	Glass
194.6968	173.638	160.0	BK7
101.5659	207.496	160.0	air
68.9403	24.829	35.0	BK7
42.8546	2.42726	25.0	air
70.8426	44.7725	25.0	BK7
0.0	**51.2809**	**12.210297**	**BK7**
-72.0648	34.6571	40.0	air
66.4712	31.4814	40.0	BK7
948.4696	21.9564	40.0	air

(Bold surface is the aperture stop.)

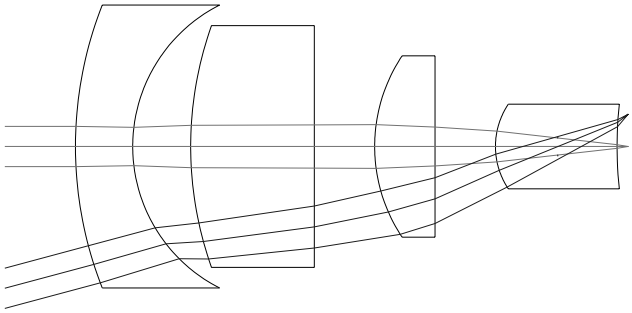

Figure 8: Best Design Found with GP (Using the Raw System as Starting Point)

9 DISCUSSION

Results show that GA and GP are capable of high quality lens system design. Indeed, they have spawn comparable or even better solutions (in terms of RMS blur spot) than the best designs presented by human experts. Thus they meet one of the criteria for human-competitiveness in evolutionary computations (Koza et al., 2000).

The best obtained result, however, has a topology that differs significantly from the two best system classes presented at the 1990 ILDC. Thus, the evolving process has probably discovered a new optimum topology class for the monochromatic quartet problem. The best obtained RMS blur spots with GA design and GP re-engineering are respectively 23% and 34% better than the best previously reported human result (1990 ILDC).

The search space in lens system design is very complex and comprises correlations between the different parameters. This makes the lens system design difficult with GA because good schemes of parameters that are far together on the bit string are likely to be destroyed by the crossover operation. For the lens system design with GP, we intentionally restricted the search in the initial system neighborhood. This has probably disfavored the convergence toward optimums that were situated far from the starting point.

10 FUTURE WORKS

For future works, we plan to experiment with an hybrid GA-GP approach, to benefit from the capacity of GA to optimize numerical parameters, using a GP variable length representation. We also plan to use a multi-objective merit function using different types of Seidel aberrations (see (O'Shea, 1985) for details) and lens costs as evolving criteria. Furthermore, we could use a database of existing commercial lenses, with associated prices to simulate real life design situations.

We will also study more deeply the re-engineering of good solutions for lens system design and for other applicative contexts. We will try to develop methods that make good compromises between local and global optimization, to enable sufficient search space exploration that facilitates the discovery of the best solu-

Table 10: Parameters of the Best Design Found with GP (Using the Raw System as Starting Point)

Radius	Thickness	Aperture	Glass
318.8959	47.4182	117.0	BK7
131.18	47.9761	117.0	air
298.4896	102.302	100.0	BK7
0.0	50.0	100.0	air
135.1501	50.0	75.0	BK7
0.0	50.0	75.0	air
62.973	51.5426	35.0	BK7
0.0	**49.1266**	**7.029558**	**BK7**
334.1364	9.3196	35.0	air

(Bold surface is the aperture stop.)

tions. Finally, we expect to integrate cultural evolving aspects (Spector and Luke, 1996) to define new generic evolutionary way of proceeding for re-engineering.

Acknowledgments

This research was supported by an NSERC-Canada scholarship to C. Gagné and an NSERC-Canada grant to M. Parizeau.

References

Ben Hamida, S., Racine, A., and Schoenauer, M.: 1999, Two evolutionary approaches to design phase plate for tailoring focal-plane irradiance profile, in *Proceedings of Artificial Evolution 1999*, Vol. 2210 of *LNCS*, pp 266–276, Springer Verlag

Forbes, G. W. and Jones, A. E. W.: 1990, Towards global optimization with adaptive simulated annealing, in *Proceedings of International Lens Design Conference 1990*, Vol. 1354 of *SPIE*

Gagné, C. and Beaulieu, J.: 2001, *Library for Lens System Ray Tracing*, http://www.gel.ulaval.ca/~cgagne/llsrt

Gagné, C. and Parizeau, M.: 2002, Open BEAGLE: A new C++ evolutionary computation framework, in *Genetic and Evolutionary Computations COnference (2002)*, New York, NY, USA

Gruau, F.: 1994, *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm, Ph.D. thesis*, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France

Hearn, G. K.: 1990, Pratical use of generalized simulated annealing optimization on microcomputers, in *Proceedings of International Lens Design Conference 1990*, Vol. 1354 of *SPIE*

Holland, J. M.: 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI

Isshiki, M.: 1998, Global optimization with escape function, in *Proceedings of International Optical Design Conference 1998*, Vol. 3482 of *SPIE*

Koza, J. R.: 1992, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA

Koza, J. R.: 1994, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA, USA

Koza, J. R., David Andre, Bennett III, F. H., and Keane, M.: 1999, *Genetic Programming 3: Darwinian Invention and Problem Solving*, Morgan Kaufman

Koza, J. R., Keane, M. A., Yu, J., Bennett III, F. H., and Mydlowec, W.: 2000, *Automatic Creation of Human-Competitive Programs and Controllers by Means of Genetic Programming*, *Genetic Programming and Evolvable Machines* pp 121–164

Lambda Research Corporation: 2001, *OSLO: Optics Software for Layout and Optimization (Reference Manual)*, Lambda Research Corporation

Nosato, H., Kasai, Y., Itatani, T., Murakawa, M., Furuya, T., and Higuchi, T.: 2001, Evolvable optical systems and their applications, in *Evolvable Systems: From Biology to Hardware (Proc. of ICES 2001)*, Vol. 2210 of *LNCS*, pp 327–339, Springer Verlag

Ono, I., Koboyashi, S., and Yoshida, K.: 1998, Global and multi-objective optimization for lens design by real-coded genetic algorithms, in *Proceedings of International Optical Design Conference 1998*, Vol. 3482 of *SPIE*

O'Shea, D. C.: 1985, *Elements of Modern Optical Design*, John Wiley and Sons

O'Shea, D. C.: 1990, The monochromatic quartet: A search for the global optimum, in *Proceedings of International Lens Design Conference 1990*, Vol. 1354 of *SPIE*

Spector, L. and Luke, S.: 1996, Cultural transmission of information in genetic programming, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp 209–214, MIT Press

Sturlesi, D. and O'Shea, D. C.: 1990, Future of global optimization in optical design, in *Proceedings of International Lens Design Conference 1990*, Vol. 1354 of *SPIE*

A Modified Compact Genetic Algorithm for the Intrinsic Evolution of Continuous Time Recurrent Neural Networks

John C. Gallagher
Department of Computer Science and Engineering
Wright State University, Dayton OH 45435-0001
jgallagh@cs.wright.edu

Saranyan Vigraham
Department of Computer Science and Engineering
Wright State University, Dayton OH 45435-0001
svigraha@cs.wright.edu

Abstract

In the past, we have extrinsically evolved continuous time recurrent neural networks (CTRNNs) to control physical processes. Currently, we are seeking to create intrinsic CTRNN devices that combine a hardware genetic algorithm engine on the same chip with reconfigurable analog VLSI neurons. A necessary step in this process is to identify a genetic algorithm that is both amenable to hardware implementation and is sufficiently powerful to effectively search CTRNN spaces. In this paper, we will propose and test several variations of the compact genetic algorithm (CGA) for searching these spaces. We will then benchmark the best variant using the De Jong functions, outline a hardware implementation, and discuss future plans to develop an integrated evolvable hardware device controller.

1. INTRODUCTION

The author has proposed the use of Continuous Time Recurrent Neural Networks (CTRNNs) as an enabling paradigm for evolving analog electrical circuits. In previous work we focused almost exclusively on extrinsically evolved CTRNNs that were created in simulation and only later implemented in hardware. We are currently interested in producing intrinsic CTRNN devices that evolve online as they are controlling physical processes. We feel that a necessary step in achieving this goal is to combine reconfigurable analog CTRNNs and a hardware evolutionary search engine onto a single VLSI device. Half of the problem, the feasibility of implementing CTRNNs in analog VLSI, has been addressed elsewhere [Gallagher and Fiore, 2000]. This paper will address the feasibility of searching CTRNN spaces with hardware based genetic algorithm that can be fabricated on the same chip with reconfigurable CTRNN hardware.

Several hardware based evolutionary algorithms (EAs) have been proposed in recent years [Kajitani, T., et.al., 1998] [Scott and Seth, 1995] [Yoshida and Yasuoka., 1999] One in particular, the Compact Genetic Algorithm (CGA) [Harik, Lobo, and Goldberg 1999] is especially well suited for efficient hardware implementation using common VLSI techniques [Aporntewan and Chongstitvatana, 2001]. The Compact Genetic algorithm, however, is a very weak evolutionary algorithm -- only equivalent to a first-order simple GA with uniform crossover and tournament selection [Harik, Lobo, and Goldberg 1999]. Although it is well suited to efficient hardware implementation, the standard CGA is not powerful enough to effectively evolve practical CTRNN device controllers. This paper will propose several simple variations of the CGA designed to increase the effectiveness of the search with respect to CTRNN devices. We will test the performance of the standard CGA and our variants against a benchmark locomotion control problem and show that, with simple modifications that do not significantly complicate hardware implementation, we can effectively evolve effective CTRNN control devices. Further, we will demonstrate that the performance of our modified CGA is, for this problem, superior to the simple genetic algorithms we have employed in the past.

2. THE BENCHMARK PROBLEM

Our benchmark problem is the control of legged locomotion in a simple, single-legged artificial agent. For the agent to walk, the leg must alternate swing and stance phases. A swing begins with the leg in its full backward position (at negative leg angle range limit) and the foot raised in the air. Then the leg rotates clockwise to swing the foot forward. A stance begins by placing the foot of a fully forward leg on the ground. Then the leg rotates counterclockwise to propel the body forward. Figure 1 shows the agent at the beginning of a stance phase. The leg contains three effectors and one sensor that returns the leg's angular position in radians. One effector governs the state of the foot (FT) and the other

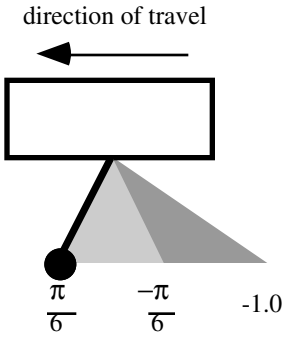

Figure 1: The Single Legged Agent

two generate clockwise (BS) and counter-clockwise torques (FS) about the leg's single joint with the body. The torques about each joint are summed, and depending on the state of the foot will either translate the body forward (foot down) or rotate the leg about its joint (foot up). Each leg has a limited range of angular motion (±π/6 radians - corresponds to the light gray wedge in Figure 1). A supporting leg may stretch outside of this range, but provides no transitional forces when doing so. The leg may not stretch outside an absolute limit (±1 radians - corresponds to the dark gray wedge in Figure 1). This behavior is intended to model the reduced ability of a hyper-extended leg to provide propulsion. When the leg is lifted from the ground, the agent "falls" and its velocity is immediately set to zero.

The agent's behavior is controlled by a fully connected continuous time recurrent neural network (CTRNN) [Beer, 1995] with the following state equation:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_i + \theta_i)$$

where y is the state of each neuron, τ is its time constant, w_{ji} is the strength of the connection from the j^{th} to the i^{th} neuron, θ is a bias term, and $\sigma(x) = 1/(1+e^{-x})$ is the standard logistic activation function. States were initialized to uniform random numbers in the range ±0.1 and circuits were integrated using the forward Euler method with an integration step size of 0.1. In each evolved CTRNN, three units are motor neurons and provide control efforts to the forward swing (FS) and backward swing (BS) and FOOT (also called FT). The remaining neurons in each leg controller have no pre-specified role. For the experiments reported in this paper, the controlling CTRNNs receive no sensory input from the outside world.

3. COMPACT GENETIC ALGORITHM

In a compact genetic algorithm [Harik, Lobo, and Goldberg, 1999], the population is represented by a probability vector that codes the chance that each bit in an individual will be a one or a zero. Each generation is a single tournament between two individuals randomly generated from the global probability vector. The probabilities governing each bit are adjusted according to the result of the tournament. Tournaments are run until the probability vector converges. The basic CGA can be summarized as follows:

1. Initialize probability vector
```
for i:=1 to l do p[i]=0.5
```
2. Generate two individuals from the vector
```
a := generate(p);
b := generate(p);
```
3. Let them compete
```
winner, loser := evaluate(a,b);
```
4. Update the probability vector toward the better one
```
for i := 1 to l do
   if winner[i] <> loser[i] then
     if winner[i] = 1 then p[i] +=1/n
     else p[1] -= 1/n
```
5. Check if the probability vector has converged
```
for i = 1 to l  do
   if p[i] > 0 and p[i] < 1 then
         goto step 2
```
6. P represents the final solution

In the above description, `l` represents the number of bits in the genome and `n` represents the size of the simulated population. In this paper, we will systematically consider two modifications to the basic CGA. These are:

a) Elitism
We will implement elitism (the best individual seen to date stays in the population) by modifying step 2 of the basic CGA so that only the "losing" contestant of each tournament is randomly generated at the beginning of the next tournament. This ensures the best individual seen to date remains in consideration. We may more formally state this modification by rewriting step 2 of the standard CGA as follows:

2) Generate one individual from the vector

```
if fitness(a) > fitness(b) then
      b = generate(p);
else
      a = generate(p);
```

b) Mutation

We will implement mutation by adding a secondary tournament to each cycle that compares the performance of the current elite string with a mutated version of itself. If the mutated version wins, it replaces the old champion as the elite string for the next tournament. The bit positions that changed also have their probabilities returned to 0.5, or some other user selected value. Our implementation of mutation presumes elitism. We can more formally describe this modification by altering step 2 as described above and adding a new step as follows:

4.5) Mutate Champ and Evaluate

```
if fitness(a) > fitness(b)
       { c = mutate(a);
         evaluate(c);
         if fitness(c)>fitness(a) then
              { a = c;
                prob_fix(p);
              }
       }
else   { c = mutate(b);
         evaluate(c);
         if fitness(c) > fitness(b) then
              { b = c;
                prob_fix(p);
              }
       }
```

The `prob_fix()` routine resets `p[x]` to 0.5 for every position `x` that was mutated. Heuristically, this is meant to represent the fact that a mutation in that position seems like a good idea, but that we don't want to commit to it completely. Rather, we want to remain undecided (there's a fifty percent chance of either bit setting occurring) and let a history of evaluations determine which way that bit position should be set.

It should be noted that with the introduction of mutation, CGA convergence can no longer be guaranteed. One must modify the end condition appropriately, perhaps by setting a maximum number of tournaments to be run.

In later sections of this paper, we will refer to the standard CGA simply as "CGA". We will refer to a CGA with the elitism modification as "eCGA". We will refer to a CGA with both the elitism and mutation modifications as the "modified CGA", or the "mCGA".

Search Type	*Yield*	*Avg. Perform*
CGA	0%	0%
ECGA	33.7%	89.0%
MCGA	71.0%	92.9%

Table 1: Relative Performances of CGA Variants
Yield shows the percentage of runs that resulted in a CTRNN controller that was at least 80% of optimal. Avg. Performance shows the mean of the performances of all controllers that achieved better than 80% of optimal. Note that both the yield and relative quality is greater for mCGA than for eCGA. Also note that the standard CGA is totally ineffective for this problem.

4. PRELIMINARY COMPARISON OF CGA VARIATIONS

Our first set of experiments was designed to compare the relative efficacy of the three variations on the CGA described above. For these experiments, the parameter settings of a five neuron, fully connected CTRNN were encoded on a bit string genome using eight bits per parameter. Values were encoded as fixed-point binary numbers and were simply concatenated into a single string. For five neuron CTRNNs, there were 40 parameters resulting in a bit string of length 320. The fitness of a particular CTRNN was the amount of distance it caused the agent to walk in a fixed amount of time. No special scaling was applied to the fitness. One hundred three (103) searches each were run using CGA, eCGA, and mCGA. We found that the CGA failed terribly. Not one of the 103 runs of the CGA produced an agent capable of locomotion. Approximately 34% of the runs of eCGA produced agents capable of walking at a speed of at least 80% of optimal. 71% of the runs of mCGA produced agents capable of walking at a speed of at least 80% optimal. These results are summarized in Table 1. For this benchmark CTRNN search problem, mCGA is clearly superior. Further, mCGA compares very well to the simple genetic algorithm. Previous results using the simple GA for the same problem produced a yield of approximately 75% and an average performance of 91.1% of optimal.

5. mCGA and the CTRNN Benchmark

The preliminary benchmark results of the last section suggest that, of the CGA variants discussed, mCGA is the best suited to searching CTRNN spaces. Further, those benchmarks suggest that mCGA is, for searching CTRNN spaces, at least as effective as the standard simple genetic algorithm. Our second set of experiments was aimed at producing a

Arch Set	*Parameters*	*Bit Length*	*mCGA Yield*	*sGA Yield*	*mCGA Avg*	*sGA Avg*
CPG3	18	144	38.8%	40.4%	88.8%	89.1%
CPG4	28	224	61.2%	66.3%	90.3%	91.9%
CPG5	40	320	71.0%	74.5%	92.9%	93.2%
CPG6	54	432	79.0%	65.4%	92.2%	93.0%
CPG7	70	560	84.5%	63.3%	92.6%	92.0%
CPG8	88	704	87.9%	64.7%	94.4%	89.9%
CPG9	108	864	91.3%	57.0%	93.2%	88.9%
CPG10	130	1040	87.3%	59.8%	93.8%	88.8%

Table 2: mCGA vs. Simple Genetic Algorithm for Varying Search Space Sizes
Yield and avg. performances are as defined in Table 1. mCGA refers to the CGA with mutation and elitism. sGA refers to a standard simple genetic algorithm. Mann-Whitney tests show no significant differences in either yield or average performance up through CPG5. After CPG5, both yields and average performances drop off sharply and significantly for the standard genetic algorithm, while both yield and average performance hold steady for mCGA at least up through CTRNNs of nine neurons.

more detailed picture of the efficacy of mCGA in searching CTRNN spaces.

Approximately 100 GA searches were run over each of eight architecture sets. The base architectures searched were 3, 4, 5, 6, 7, 8, 9, and 10 neuron fully-connected CTRNNs. The purpose of these tests as to examine how well mCGA scaled to more difficult searches. The results of these experiments, as well as the results of previously reported applications of the simple genetic algorithm to the same problems [Gallagher, 1998], are shown in Table 2. mCGA runs were limited to a maximum of 100,000 tournaments to ensure that approximately the same number of candidate evaluations were made in both mCGA and simple genetic algorithm experiments. In terms of either the quality of solutions or yield of successful solutions, there is no significant difference between the simple GA and mCGA for small CTRNNs. However, there is a significant difference in both yield and quality of solution for CTRNNs of six or more neurons. The simple genetic algorithm simply can not cope with longer bit strings, while the mCGA seems quite capable of searching these larger spaces.

The reason for the relatively poor yields of CPG3 networks has been identified and discussed extensively in other works [Beer, Chiel, and Gallagher; 1999][Chiel, Beer, and Gallagher; 1999]. In short, three-neuron CTRNNs lack sufficiently many degrees of freedom to properly solve the locomotion problem. The relative scarcity of three-neuron solutions increases the difficulty of the search. It also puts a cap on the maximum effectiveness of the solutions evolved. Our data shows that mCGA performs no worse than the simple GA in the face of these difficulties. Preliminary experiments with single elimination tournaments in the mCGA, however, show an increase in the yields of CPG3s to 68%. We are currently engaged in further experimentation to better characterize this surprising phenomenon.

6. mCGA and the De Jong Test Functions

mCGA was developed against a specific CTRNN benchmark problem. During that development, we saw that though both the addition of elitism and mutation were useful, it was the addition of mutation that seemed to provide the most benefit. Both as a means of evaluating the effect of differing mutation rates and as means of evaluating the mCGA against standard benchmarks, we tested it against the De Jong test functions [De Jong, 1975]. We ran 100 mCGA searches for each of the five De Jong test functions for bitwise mutation rates of 0.0, 0.5, 0.1, 0.15, and 0.20. Each objective function parameter was coded with same precision and range as in De Jong's thesis. Because mCGA as formulated above doesn't converge for higher mutation rates, we capped the number of generations at 100,000 to be consistent with the CTRNN benchmarks previously discussed. Simulated population size was set to 100 for similar consistency with the CTRNN benchmarks.

mCGA always succeeded in finding the global optimal for De Jong F1. This is perhaps not surprising, as the unimodal and symmetric F1 represents a very easy optimization problem. We did note, however, that for F1, small mutation rates allowed for faster searching. On average, mCGA found the F1 global optimal after about 672 generations with a mutation rate of 0.0. With a mutation rate of 0.05, this was halved to about 332 generations. Higher rates, resulted in delayed appearances of the optimal. At mutation rates of 0.1, 0.15, and 0.2 it took on average 537, 1589, and 13363 generations respectively to find the global optimal.

Figure 2 shows the average scores, based on 100 runs, of the best solutions found for De Jong F2, F3, F4, and F5 for the same bitwise mutation rates listed above. Note that for F2, F3, and F5, increased mutation leads to an increased chance of finding the optimal. Also note, however, that just like with F1, increasing the mutation also increases the number of generations, on average, that one needs to wait for the optimal to appear. F4 behaves differently. A little

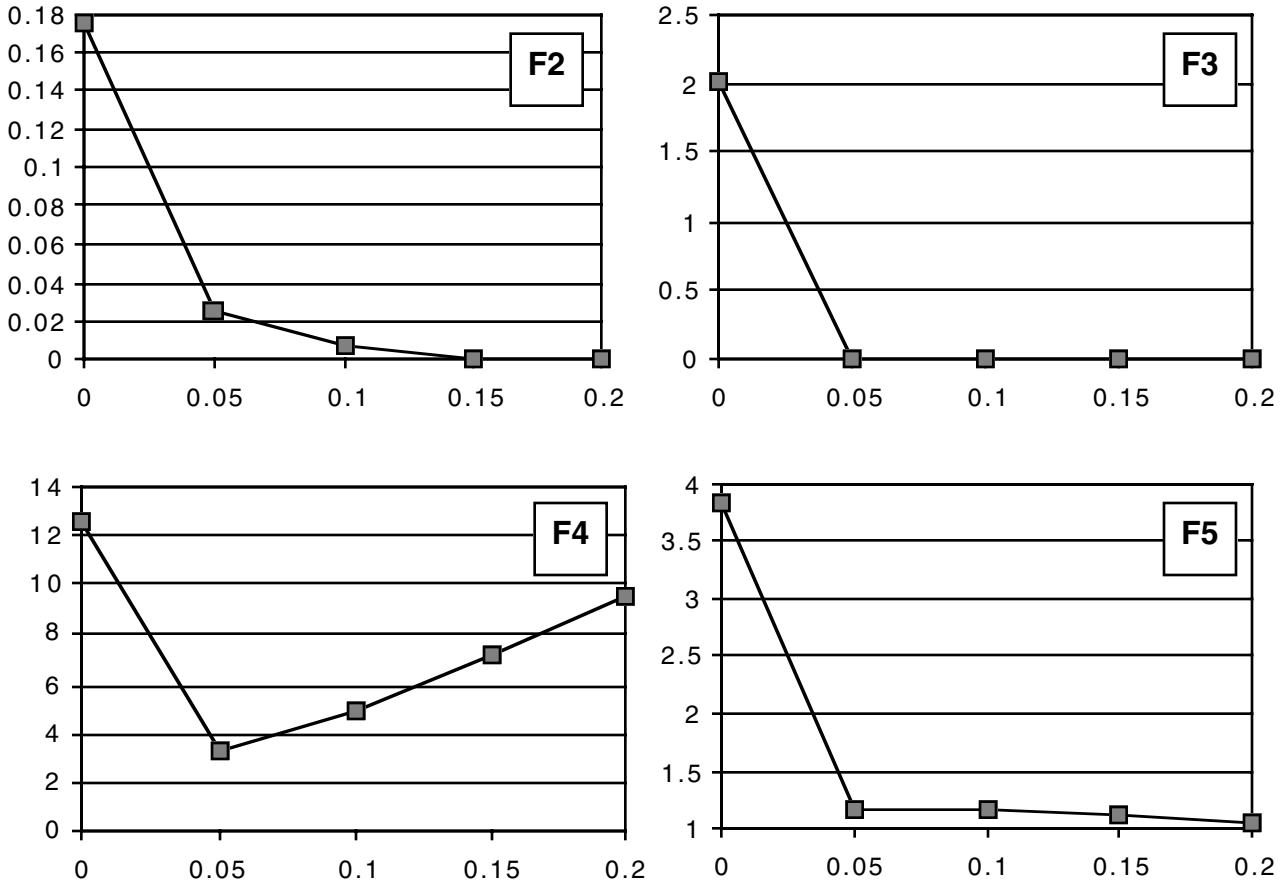

Figure 2: Average Errors Attained on De Jong F2 - F5
For each graph, the y-axis shows the average final error attained based on 100 runs. In all cases, the bottom tick on the y-axis represents the lowest attainable error score. The x-axis shows setting of the mCGA bitwise mutation rate.

mutation helps, but increasing it too much results in a degradation of the quality of solutions found. F4 is a simple unimodal function with gaussian noise and is meant to test how well an optimization algorithm deals with noisy objective functions. In the simple genetic algorithm, a mutation of an individual that received a deceptively good score by random chance would be likely to drop out of the population eventually. We would expect it would not receive deceptively good scores sufficiently often to allow it to spread through the population. mCGA, however, simulates a mutated individual having spread widely into the population instantaneously by modifying the probability vector that simulates the population. In a sense, the momentum effects provided by having a real population are not present in mCGA, and we could therefore expect it to be quite easily tricked by deceptive evaluations of individuals. This effect is magnified as we allow more mutation events to occur in a search. We will discuss possible fixes to this problem later in this paper.

7. A Proposed Hardware Implementation

A design for a hardware implementation of the compact genetic algorithm is in the literature [Aporntewan and Chongstitvatana, 2001]. A hardware implementation of our mCGA is not much more difficult to achieve. In this section, we will outline one possible hardware implementation of the mCGA. We will present a data path that supports all the operations needed to implement the mCGA as described in section three of this paper. We will also provide a qualitative description of the actions that an on-chip microcontroller needs to take to implement the mCGA.

A proposed data path is provided in Figure 3. The design is similar to that in [Aporntewan and Chongstitvatana, 2001], but contains additional machinery to implement elitism and mutation. The bit probability for a genome position as well as bits for that position for two candidates are held in a number of “bit modules”. In figure 3, two bit modules are

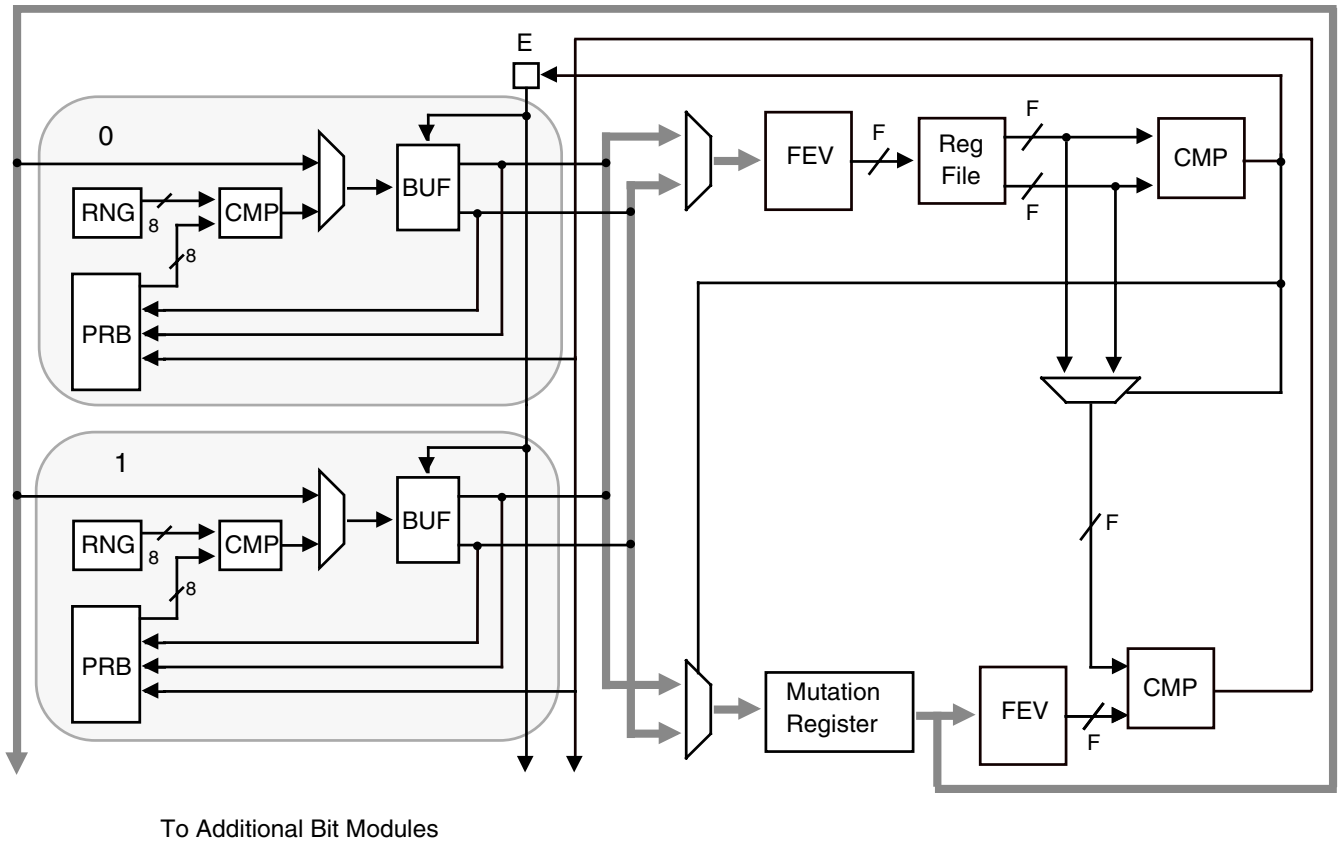

Figure 3: Sample Data Path for the mCGA
A sample data path capable of implementing the mCGA in hardware. Components are defined in the text. Bold gray lines are N bit busses where N is the number of bits in the genome. For large genomes, these wide busses can be replaced with serialized communication channels.

shown enclosed in gray boxes and labeled 0 and 1. Larger genomes can be implemented by adding additional modules as needed. The components in the data path are defined as follows:

RNG : A random number generator. When activated, this device will generate a random eight-bit number

CMP : A standard multibit comparator

PRB : The probability register. This register contains an eight bit value that encodes the probability of that bit position is a 1. The device also has control lines that allow incrementing, decrementing, or setting the probability to 0.5.

BUF : A 2x1 memory device. A select line allows one to load a bit into the "top" or "bottom" bit position. This device is used to hold bit values at a position for the two candidates under consideration.

E : E is a one bit register that encodes which of the two bits stored in each BUF (top or bottom) is part of the current "elite individual.

FEV : The Function Evaluator. This is hardware that provides an error score for the bit pattern supplied to it. This hardware could be on chip or it could obtain the error evaluation with off-chip circuitry. We encode the error score as an F bit number.

MR : Mutation Register. This device acts like a normal register except that it mutates bits according to a user specified mutation rate.

RF: Register File. A simple register file that stores the F bit errors for the "top" and "bottom" candidates stored in the bit modules.

MUX : Standard multibit multiplexers. Shown in the data path as unlabeled trapezoids.

The mCGA would be implemented by augmenting the data path with a microcontroller that would complete the following operations.

1. The Probability Registers (PRBs) are set to hold their initial probabilities of 0.5. All registers and buffers are zeroed.
2. Each RNG generates a random number. This value is compared with the contents of the corresponding PRB. Each CMP produces a 1 if the generated random number is less than the probability and a zero if it is greater. Each generated bit in each of the bit modules is written into the "top" slot of the buffer (BUF). The E bit, which was cleared in step one, designates the "top" slot as holding the current elite individual.
3. Step 2 is repeated, except this time the bits generated by each comparator are stored in the "bottom" slots of each BUF.
4. One at a time, all the bits in the top buffers are routed to the FEV module. Each string is evaluated and the F bit errors are stored in the "top" and "bottom" slots of the register file.
5. The errors sent from the register file to a comparator. The identity of the best score (top encoded as zero, bottom encoded as 1) is sent back to and stored into the E bit. We now know which of the two initially generated individuals is the better.
6. All the bits making up the current elite individual are routed into the mutation register, creating a mutated version of the current best individual.
7. The bits making up the mutated candidate string are routed to a FEV. The error resulting error score is compared with the previously computed error score of the best (taken from the register file). If the mutated individual is better, all the bits from the mutation register are copied into their corresponding bit modules into the currently elite buffer slots. The PRBs inside of bit modules corresponding to bits that changed are also instructed to reset to hold a probability of 0.5.
8. Each RNG generates a random number, which is compared with the corresponding PRB to generate a bit as described in step two. Each new bit is written into the NON ELITE slot in each bit buffer (BUF).
9. If our end condition is not met, go to step 4.

8. Conclusions and Discussion

We desire to combine reconfigurable analog neurons and a hardware-based genetic algorithm into a single device to be used to control and regulate physical processes. For our application, compact size is vital. The CGA is particularly amenable to implementation using standard VLSI techniques and would result in extremely compact circuitry. The CGA, however, is known to be a weak search method. We have shown that the unmodified CGA is not sufficiently powerful to evolve CTRNN controllers. However, by making simple modifications that do not require major increases in the number of gates necessary to implement the CGA in hardware, we can produce a modified CGA that competes well with the simple genetic algorithm. In fact, we have shown that our modified CGA, for this problem, is actually provides superior search performance for approximately the same computational cost. Additionally, we have shown mCGA, for the most part, performs well on the five De Jong test functions. Where it performs less well, specifically on randomized error functions, we have identified the problem and are in a position to fix it. These results alone are significant. At least as significant, however, are the interesting questions raised by these results

First, we expected that unmodified CGA would not be able to search CTRNN spaces effectively. We also expected that with appropriate modification, we would be able to make CGA work without incurring a great penalty in hardware cost. We did not expect that the modified CGA would outperform methods previously employed. We intend to carefully study our modified CGA to determine exactly why we observed this improved performance. We formulated our modifications based on our experience about what works for evolving CTRNNs. Did, however, we stumble on something that searches other spaces well too? We intend to answer this question by applying the mCGA to difficult search problems outside of our target problem domain. If more universally successful, we will more carefully and formally analyze the algorithm.

Second, the introduction of single elimination tournaments, unlike the other modifications we proposed, does somewhat increase the number of gates required for hardware implementation. Tournaments are, therefore, somewhat less attractive for our stated problem domain. However, that we have already observed a significant gain in yield for the difficult CPG3 problem is interesting. We intend to study the tournament modification more carefully in the future. It may be the case that increases in effective yield might be well worth the increased hardware costs.

Third, the single-leg CPG problem was not chosen arbitrarily. It happens to be the best studied, and because the controller may not make use of sensory feedback, one of the more challenging, locomotion control problems we have considered to date. This makes it a reasonable initial benchmark of mCGA efficacy. We need, however, to apply the search method to a wider variety of CTRNN control problems. Initial results are very encouraging. We have successfully used mCGAs to evolve hexapod locomotion controllers and controllers correcting arrhythmia in simulated human hearts. These results also represent important verifications against prior work. However, we desire to be careful and ensure that all our results possess a high degree of statistical significance. Therefore, we need to run more experiments against these other problems before we have enough instances to report more than anecdotally on wider success.

Fourth, we have extensively studied the neural dynamical principles underlying the operation of CTRNN locomotion controllers evolved using the simple genetic algorithm [Beer, Chiel, and Gallagher, 1999][Chiel, Gallagher, and Beer, 1999]. We have yet to rigorously study the principles underlying the operations of the mCGA produced controllers. One would expect them to operate using similar principles -- however, we have three mCGA produced CPG3's that seem to defy explanation using the dynamical systems techniques previously developed. Analysis of these devices is under way. The differences in the resulting products may reveal that the mCGA is searching a portion of CTRNN space difficult for previous methods to reach. The nature of the "difficult to reach space" may provide important clues on why our modifications work so well and perhaps, suggest additional modifications that might help us search CTRNN spaces more reliably.

In the future, we intend to expand and optimize the hardware mCGA. This should be a fairly straightforward task and we expect to have completely validated designs very shortly.

In conclusion, we have demonstrated that the marriage of the mCGA and reconfigurable CTRNN hardware is likely to produce compact, capable EH chips. In addition to providing an important feasibility result, we have also opened several interesting new lines of inquiry that will likely provide equally interesting results as they are pursued. Our mCGA is almost certainly well suited to CTRNN-EH, and may be adaptable to other EH efforts as well.

Acknowledgments

The authors would like to thank Steven Perretta and Wendy Peluso for their comments and suggestions on this manuscript.

References

Aporntewan, C. and Chongstitvatana, Prabhas. (2001). A hardware implementation of the compact genetic algorithm. In *The Proceedings of the 2001 IEEE Congress on Evolutionary Computation.* Seoul, Korea. IEEE Press

Beer, R.D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior* **3**(4):469-509.

Beer, R.D., Chiel, H.J. & Gallagher, J.C. (1999). Evolution and analysis of model CPGs for walking II. General principles and individual variability. *J. Computational Neuroscience* 7(2):119-147. (Kluwer).

Chiel, H.J., Beer, R.D., & Gallagher, J.C. (1999). Evolution and analysis of model CPGs for walking I. Dynamical modules. *J. Computational Neuroscience* 7:(2):99-118.

De Jong, K.A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral dissertation, University of Michigan, Ann Arbor. University Microfilms Num 76-9381.

Gallagher, J.C. (1998). *A dynamical systems analysis of the neural basis of behavior in an artificial autonomous agent.* Ph.D. Doctoral dissertation, Case Western Reserve University. Cleveland, OH. USA.

Gallagher, J.C. and Fiore, J.M. (2000). Continuous time recurrent neural networks: a paradigm for evolvable analog controller circuits. in *The Proceedings of the National Aerospace and Electronics Conference*. IEEE Press.

Harik, G.R., Lobo, F.G., and Goldberg, D.E. (1999). The compact genetic algorithm. In *IEEE Transactions on Evolutionary Computation* vol. 3 no. 4. pp 287-297

Kajitani, T., Hoshino, T. Nishikawa, D., et.al. (1998). A gate level EHW chip: implementing GA operations and reconfigurable hardware on a single LSI. In *Proceedings of the International Conference on Evolvable Hardware (ICES 1998).*

Scott, S. and Seth, A. (1995). HGA: a hardware-based genetic algorithm, In *Proceedings of the ACM/SIGDA Third Int. Symposium on Field-Programmable Gate Arrays.*

Yoshida, N. and Yasuoka, T. (1999). Multi-gap : parallel and distributed genetic algorithms in VLSI. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*

Evolving Fault Tolerance on an Unreliable Technology Platform

Morten Hartmann[1] **Frode Eskelund**[1] **Pauline C. Haddow**[1] **Julian F. Miller**[2]

[1]Dept. of Computer and Information Science
The Norwegian University of Science and Technology
NO-7491 Trondheim, Norway
{mortehar,eskelund,pauline}@idi.ntnu.no

[2]School of Computer Science
University of Birmingham
Birmingham, B15, UK
j.miller@cs.bham.ac.uk

Abstract

One of the key areas in which evolvable hardware has been shown to excel is in achieving robust analogue and digital electronics. In this paper this domain is investigated further by manipulation of the digital abstraction. Some of the strict requirements of digital gates are relaxed in order to increase the complexity of the functionality available to evolution in order to evolve fault tolerant designs. Results from extrinsic evolution of a 2-by-2 bit multiplier, based on CMOS technology under various noise and fault conditions, illustrate the suitability of the messy gate methodology used herein for evolution of a fault tolerant design.

1 INTRODUCTION

Silicon is not a truly reliable technology. However, by using a digital abstraction we obtain a more robust platform against signal variation and noise whilst sacrificing much intrinsic complexity. That is, above and below the digital thresholds we are not concerned with signal variations.

Each gate and its connections plays a crucial role in the overall behaviour of a digital circuit. Unforeseen events can easily prevent proper operation of a regular digital design. Why is this? The digital abstraction assumes that the technology will always operate within the specifications laid down by the abstraction mechanism and since silicon is not a truly reliable technology, deviations may be expected.

A number of noteworthy efforts have already been conducted within fault detection and repair based on the principles of biological development. In the embryology work conducted at York [OT99] and Ecole Polytechnique Fédérale de Lausanne (EPFL) [MSST00], experiments have been conducted using FPGAs with extended Configurable Logic Blocks (CLBs) to contain a complete genotype of the circuit. Through repeated cell divisions, a circuit develops from a single cell into a full-grown phenotype. An interesting approach was taken in [BOST00] where principles of biological immune systems were adopted to attain fault-tolerance. Work on achieving tolerance to temperature changes includes [TL00] and [SKZ01].

Inherent fault tolerance is present if the design is able to continue its operation undisturbed by fault inducing events without the need for explicit mechanisms for fault detection and recovery. This may be achieved by robust ways of computation or by an underlying fault-detection and repair from within the technology. The work of Haddow and van Remortel [HvR01] considered possibilities for achieving fault detection and repair from within the technology as well as more fault tolerant ways of distributing digital designs onto the technology based on the amorphous computing concept [Aea99]. Hounsell and Arslan [HA01] have developed a fault tolerant hardware platform for the automated design of multiplier-less digital filters. Tyrrell et al [THS01] have used evolutionary strategies to achieve redundancy thus providing inherent fault tolerance in the design.

The approach described herein uses the concept of messy gates [MH01a]. The messy gate concept may be said to be a fault tolerant methodology rather than fault detection and repair methodology. This tolerance is a tolerance to a less reliable technology. It may not only tolerate less than perfect gates but in fact uses evolution to exploit this *messiness* i.e. non perfect digital signals. Other work which exploited features of the technology includes the work of Thompson [Tho96]. Here the focus was not so much on fault tolerance but more towards achieving unique solutions to difficult problems and, as such, illustrating the power of evolution. Messy gates are able to operate in analogue voltage levels that are outside the normal digital scope. In addition, the analogue outputs of the messy gates are allowed to propagate through the circuit.

Circuit designs are evolved using messy gates as their components. As in nature, evolution is not prone to designs where every part is required for satisfactory behaviour, but rather to distributed designs where no single point is crucial. Fault tolerance emerges through the abstraction mechanism as its functionality is exploited by evolution.

Earlier work of Miller and Hartmann [MH01a, MH01b] on the messy gates approach may be said to be a proof of concept. That is, the model did not take into account any particular technology but more investigated the possibility of exploiting non-perfect digital gates to achieve fault tolerance. This work showed promising results and, as such, the model has been extended to a technology specific model. This paper presents the new model, simulator and the results of experimentation.

The model for the messy gates and the simulator that was developed are described in section 2. In section 3, the evolutionary algorithm is explained. Section 4 describes the experiments and section 5 gives a discussion of the results. In section 6 we present some ideas for future work and section 7 presents our conclusions.

2 MESSY GATES AND THE SIMULATOR ENVIRONMENT

Introduced in [MH01a] and elaborated in [MH01b], messy gates is an approach which removes some of the digital abstraction and investigates the impact on evolution of circuits using these gates to achieve fault tolerance. While earlier work is based on an abstraction level not close to a specific hardware technology, this paper introduces a model of messy gates which is very close to one technology, specifically Complementary Metal Oxide Semiconductor (CMOS). The model is parameterised and can be tuned to different semiconductor technologies.

The simulator is based on observations made during analogue simulation of digital gates in Simulation Program for Integrated Circuits Emphasis (SPICE). Each gate was modelled at the transistor level including various error sources. Figure 1 shows the model of a NOR gate which was provided to the SPICE simulator. As shown, the model incorporates three noise generators ($FGEN1$ through $FGEN3$), several capacitances ($C1$ through $C3$), current leaks ($R1$ and $R2$), and output load ($LOAD$). NAND, NOR, NOT, MUX and NMUX (multiplexor with one input inverted) gates were simulated at analogue 5V CMOS transistor level whilst being exposed to different configurations of capacitances, resistances and noise. For the purpose of the experiments herein, only the MUX and NMUX gates were used.

A sigmoid function was used to approximate the fall and rise behaviour of the digital gates. Sigmoid functions for each gate type were individually tuned to approximate behaviour shown in SPICE simulations. For instance, a NOR gate was simulated in SPICE showing a behaviour as shown in Figure 2. Using a sigmoid function this was approximated in our model as shown in Figure 3. Approximations were subject to some limitations due to the fact that a look-up table replaced a full sigmoid function in order to speed up simulations. The size of this table was limited to 64 KB, in order to allow it to fit into the first level cache of most modern processors. It should be noted that the smoothness displayed in Figure 3 is only the core part of the gate model, and more noisy behaviour like the one in Figure 2 is likely to be displayed as noise is added.

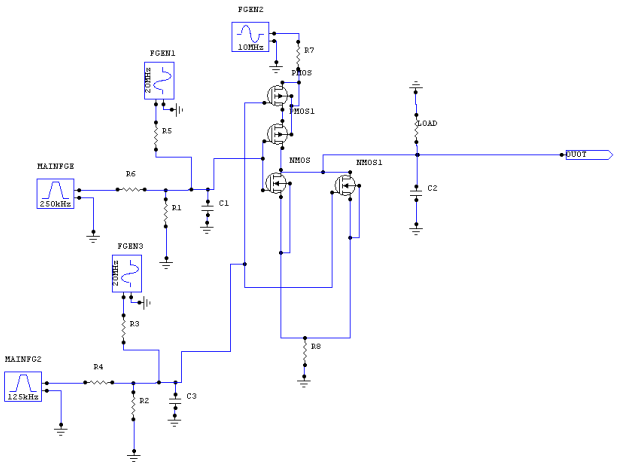

Figure 1: SPICE transistor level layout of NOR gate

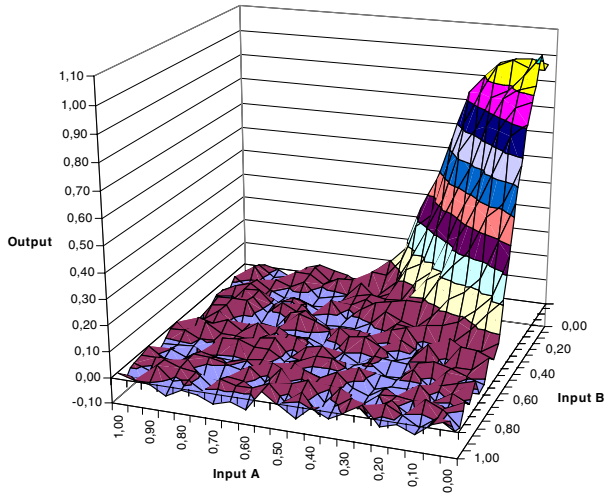

Figure 2: SPICE simulation of NOR gate

The current simulator focuses on internal faults of types stuck-at errors, floating outputs and partly random output,

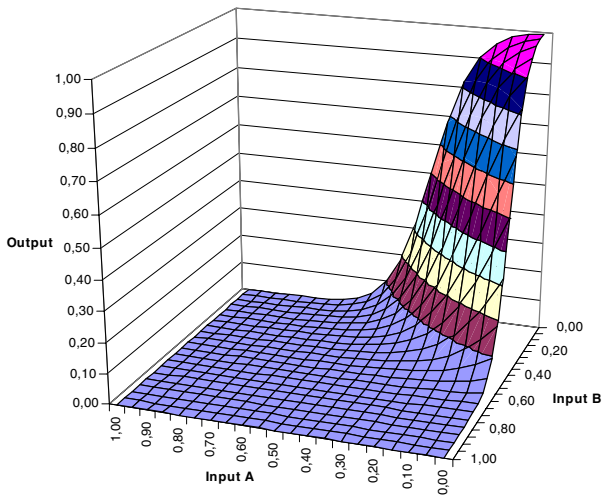

Figure 3: Model approximation of NOR gate

in addition to supporting induced signal noise. Input or output stuck-at errors cover the cases of short-circuit to power or ground and certain cases of inter-signal short-circuits. Floating output errors covers the case where the output is completely random, while partly random output covers the case where the output is correct for one logical value while random for the other logical value e.g. logical 1 is represented as 1 whilst logical 0 is represented as a random number from 0 to 1. The simulator uses a real number internal representation within the range 0 to 1, to represent the CMOS voltage range of 0 to 5 volts.

The transistor layout used in SPICE simulations allows analogue signals to propagate through gates and there is no explicit mechanism for pulling the output signal to either a completely high or low state e.g. a push-pull stage. As shown in Figure 3, the sigmoid behaviour will allow propagation of analogue signals and yet be biased towards the digital endpoints of the analogue scale (the real numbers 0 and 1 in simulation). The model provides evolution with the possibility to exploit this non-digital feature to achieve more robustness in evolved designs.

The resulting gate model used in the simulator is illustrated in Figure 4. E_1, E_2 and E_3 generate one of the supported errors (stuck-at, floating outputs or partly random output) or let the signal propagate through without error. The probability of error is preset as a parameter, whilst the type of error is random with equal probability for each of the four possible faults. F is the sigmoid function approximated to the real behaviour of the corresponding gate in SPICE. Finally, the output noise N is superimposed on the signal to approximate errors that are not explicitly a part of the model e.g. thermal noise, radiation, power supply noise, component variance and cross talk. The errors and noise of this abstracted model is not directly related to the errors of the transistor model, but are present to achieve a behaviour similar to that shown by the transistor model. The simulator currently supports feed-forward networks and realization details such as routing and layout are ignored.

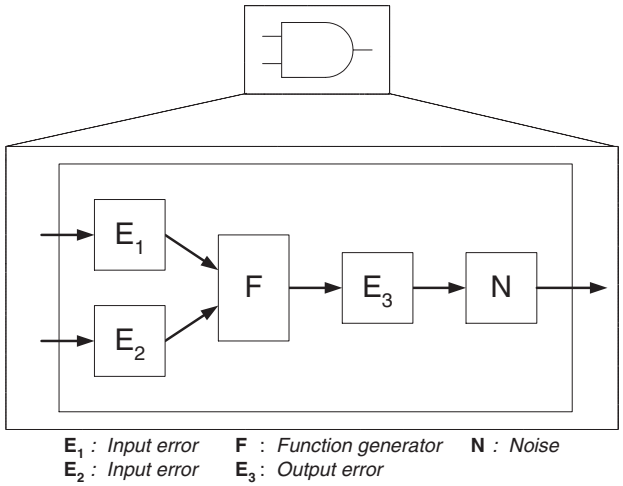

Figure 4: Generic 2-input gate

3 EVOLUTIONARY ALGORITHM

The algorithm used is a (1+λ) evolutionary strategy with neutral genetic drift. That is, a generation consists of the best individual from the former generation and mutations of it. Neutral drift is obtained when in the case of equal fitness amongst the best individuals, one not from the former generation is selected. The work of Vassilev and Miller [VM00] illustrates the advantage of this approach.

The target solution is defined simply via its truth table and the number of inputs and outputs. Under fitness evaluation, every circuit is subject to the complete set of possible inputs and a selection of noise and fault vectors. The outputs generated by the circuit are compared to the target solution truth table and any analogue output values are clamped to their closest digital value for comparability.

The fitness function used is expressed in Equation 1. A circuit C (individual) is tested against the target truth table (T) a number of times (TPI) under different environments. Noise and fault probabilities are used to generate the different environments m for each test.

The average of all tests is computed to yield a penalty for the number of incorrect output vectors. Another term ($G_c * G_p$) penalises the number of gates. Finally, these terms are subtracted from the maximum obtainable fitness Max to yield a fitness score in the range 0 to the total number of

nature	label	select	type	input A	input B
input	0				
input	1				
input	2				
input	3				
gate	4	0	MUX	0	3
gate	5	2	MUX	2	0
gate	6	5	NMUX	0	1
gate	7	2	MUX	2	1
gate (o)	8	7	MUX	7	4
gate (o)	9	8	MUX	5	6
gate (o)	10	4	MUX	7	6
gate (o)	11	3	MUX	4	1

Figure 5: Example genotype of a 4 inputs, 4 outputs circuit

output vectors in the target truth table minus 1 e.g. $2^4 - 1$ in the case of the 2-by-2 bit multiplier. Thus, the range of the fitness of the 2-by-2 bit multipliers described in section 4 is 0 to 15.

$$F = Max - \left(\frac{\sum_{n=1}^{TPI} diff(C_m, T)}{TPI} + G_c * G_p \right) \quad (1)$$

F	Fitness of individual
Max	Maximum obtainable fitness
TPI	Test pr. individual
$diff()$	Number of incorrect output vectors
C_m	Circuit in environment m
T	Target truth table
G_c	Number of gates used
G_p	Penalty pr. gate used

An example of the genotype used to represent a circuit is shown in Figure 5. The connections of this specific genotype can be seen in Figure 7. Connections refer to labels of either the inputs of the circuit (0 to 3 in the example) or to the output of one of the gates in the circuit (4 to 11 in the example). The last gates in the genotype representation are considered to be connected to the external outputs of the circuit (8 to 11 in the example). Only feed-forward connections are allowed. The genotype uses MUX and NMUX (multiplexor with one input inverted) and as such, each gate has three inputs (select, input A and input B).

Mutations are done at gate level. If a gate is mutated, one of its inputs is remapped or its type is changed to a random type within the predefined set of gate types. The gate types available to evolution are predefined. The simulator is not limited to the gates described in section 2, but each new gate requires handcrafted tuning of parameters.

4 PERFORMED EXPERIMENTS

The focus of these experiments is to discover the influence of noise and gate failures on evolution of messy circuits. 2-by-2 bit multipliers were evolved using the algorithm described in section 3 and the model and simulator environment defined in section 2. Only gates of type MUX and multiplexors with one input inverted, NMUX, were made available to evolution to maintain comparability with [MH01a, MH01b].

Five sets of experiments were carried out labeled A to E in Table 1. The noise percentage signifies the amount of noise relative to full signal strength that was superimposed at each gate output. Noise is implemented as a random value within this range. Error probability is the chance of any given gate failing i.e. being subject to one of the errors explained in section 2.

All experiments used a gate mutation probability of 15%, population size of 30 individuals and a maximum gate count of 9. Termination of each run occurred when an individual avoided bit errors at the outputs and its size was less than 10 gates.

A noteworthy fact is that each circuit is evaluated ten times under different noise and error conditions. Each of these evaluations gives rise to a fitness value. Fitness for this circuit (individual) is an average of these fitness values. This means that an individual needs to be tolerant to more than one specific configuration of faults and noise and exhibit an overall tolerance to the environmental settings in order for it to survive the selection process.

EXPERIMENT	NOISE	ERROR PROB.
A	10%	0%
B	30%	0%
C	0%	10%
D	0%	30%
E	35%	10%

Table 1: The experimental set of noise and error probabilities

5 RESULTS AND EVALUATIONS

The evolutionary system successfully evolved circuits able to perform a 2-by-2 bit multiplication in all the specified environments. Figure 6 illustrates how the average of the most fit individual for experiments B and E grow towards 100% fitness (15). As expected, the rougher environment in experiment E made it harder to obtain better fitness when compared to experiment B.

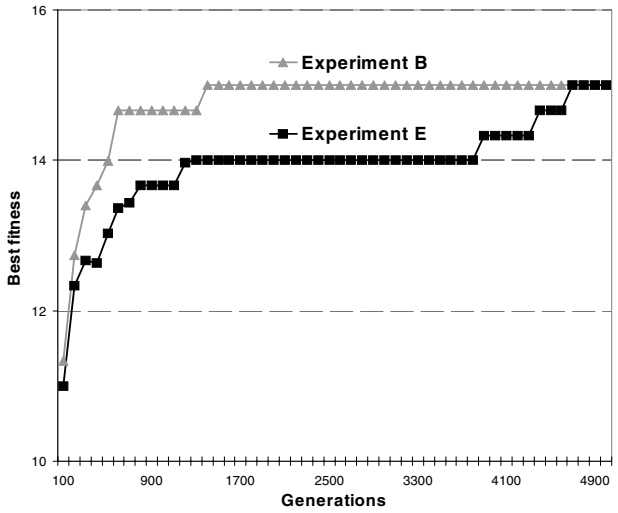

Figure 6: Best fitness individuals averaged over several runs

An example of an evolved multiplier from experiment A is illustrated in Figure 7. The genotype of this specific multiplier is the one shown earlier in Figure 5.

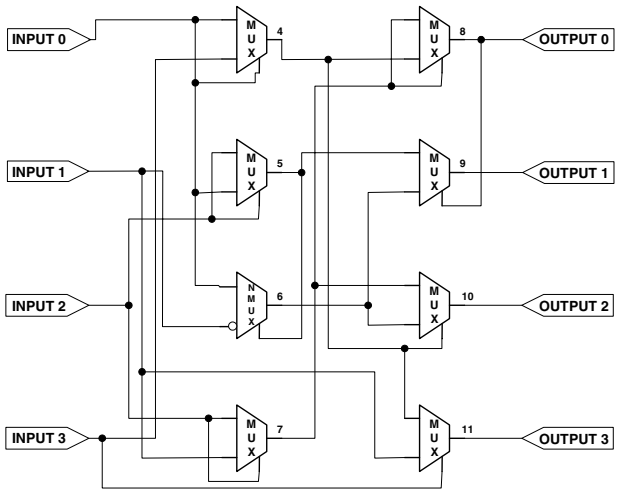

Figure 7: Example of evolved multiplier

As seen in Figure 6 and 8 the number of generations required before a correct individual was evolved increased as noise and error probability got more severe. Experiments A, B and C took on average below 2000 generations. Experiment E required slightly more computational labour, around 3000 generations. This may be said to be due to the combination of severe noise and substantial error probability. Experiment D clearly separates from the other experiments. The gate error probability presented a difficult design task. However, the fact that evolution was able to create such a fault tolerant architecture is quite an achievement. The computational time required is not really that severe, on average the performance is about 100 generations per second.

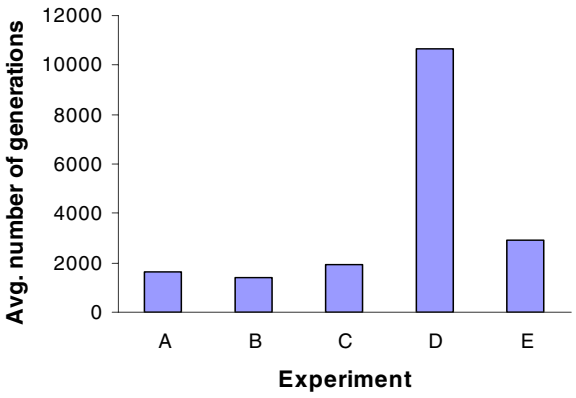

Figure 8: Generations used before termination

In the experiments performed here evolution was not given much room to play with in terms of gates. The maximum number of gates was set to nine. Due to this maximum only a slight variation in the number of gates was seen, as shown in Figure 9. Also, during the experiments, G_p in equation 1 was so low (0.001) that the output correctness always had highest priority. In experiment D the average number of gates used is below the general average. The reason for this is that evolution avoids extra gates, as each gate increases the chance of gate failure. In experiment E, evolution may be assumed to perform similarly where the rough environment in terms of both noise and gate errors forces a bias towards small circuits.

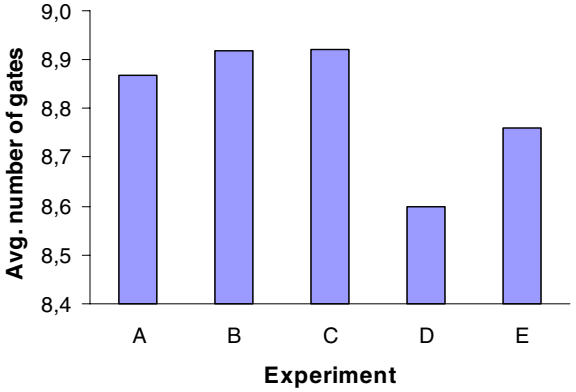

Figure 9: Gates used in final solutions

6 FUTURE WORK

Several extensions of the work described are planned, primarily exploiting the speed of the simulator in order to run extensive numbers of experiments. Such experiments would include allowing evolution to use more gates to increase the genetic variation, evolving with more common type of logic gates like NAND and NOR, evolving circuits with different functionality and exploring the suitability of different evolutionary algorithms. Extensive investigation of the evolved circuits true fault tolerance compared to other approaches would also be advantageous. In addition, verification of the circuits on both digital and analogue levels is important.

A feature of the developed simulator is that its states and signals are fully observable. Combining this with the increased freedom of the semi digital environment may yield results that exploit complex intrinsic silicon functionality not available in a traditional evolutionary digital design environment. Such exploitation of intrinsic features have shown stunning results e.g. in [Tho95, Tho96], but the features of such circuits have been hard or impossible to observe and understand [Lay98]. A goal of our project is to move into hardware implementation with new knowledge on how to exploit the intrinsic properties of the underlying technology.

7 CONCLUSIONS

In this work a simulator has been developed that takes the messy gates approach much closer to real electronic hardware. In this simulated environment evolution is able to perform circuit design tasks that present a serious challenge. Within certain limits, noise influence and fault probability does not even seem to affect evolution. In fact, external influences that human engineers view as destructive and problematic may be exploited by evolution e.g. increasing the noise may in fact decrease the number of generations needed to evolve a complete circuit. When faults and noise influence are increased to severe levels, evolution seems to find it harder to find solutions. Still, it does in fact manage to do so and very quickly when compared to a human designer. Just imagine the design task of creating a 2-by-2 bit multiplier using eight or nine multiplexors in an environment where almost one third of the gates fail on average.

References

[Aea99] H. Abelson et al. Amorphous Computing. Technical report, Massachusetts Institute of Technology, 1999.

[BOST00] Daryl Bradley, Cesar Ortega-Sanchez, and Andy M. Tyrrell. Embryonics + immunotronics: A bio-inspired approach to fault-tolerance. In J. Lohn, A. Stoica, D. Keymeulen, and S. Colombano, editors, *Proc. The Second NASA/DoD Workshop on Evolvable Hardware, EH 2000*, pp 215–224. IEEE Computer Society, 2000.

[HA01] Ben I. Hounsell and Tughrul Arslan. Evolutionary Design and Adaptation of Digital Filters Within an Embedded Fault Tolerant Hardware Platform. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pp 127–135. IEEE Computer Society, 2001.

[HvR01] Pauline C. Haddow and Piet van Remortel. From here to there: Future robust EHW technologies for large digital designs. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pp 232–239. IEEE Computer Society, 2001.

[Lay98] P. Layzell. A New Research Tool for Intrinsic Hardware Evolution. In *2nd International Conference on Evolvable Systems (ICES98)*, Lecture Notes in Computer Science, pp 47–56. Springer, 1998.

[MH01a] J. Miller and M. Hartmann. Evolving Messy Gates for Fault-Tolerance: Some preliminary Findings. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pp 116–123. IEEE Computer Society, 2001.

[MH01b] J. Miller and M. Hartmann. Untidy Evolution: Evolving Messy Gates for Fault-Tolerance. In Y. Liu et al, editor, *Evolvable Systems: From Biology to Hardware. Fourth Int. Conf., ICES 2001*, volume 2210 of *Lecture Notes in Computer Science*, pp 14–25. Springer, 2001.

[MSST00] Daniel Mange, Moshe Sipper, André Stauffer, and Gianluca Tempesti. Toward Self-repairing and self-replicating hardware : the embryonics approach. In *The 2nd NASA/DoD Workshop on Evolvable Hardware*, pp 205–214, 2000.

[OT99] Cesar Ortega and Andy Tyrrell. Reliability Analysis in Self-Repairing Embryonic Systems. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proc. The First NASA/DoD Workshop*

on Evolvable Hardware, EH 1999, pp 120–128. IEEE Computer Society, 1999.

[SKZ01] A. Stoica, D. Keymeulen, and R. Zebulum. Evolvable Hardware Solutions for Extreme Temperature Electronics. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pp 93–97. IEEE Computer Society, 2001.

[Tho95] A. Thompson. Evolving Electronic Robot Controllers that Exploit Hardware Resources. In *The 3rd European Conference on Artificial life (ECAL95)*, 1995.

[Tho96] A. Thompson. An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics. In *1st International Conference on Evolvable Systems, ICES*, Lecture Notes in Computer Science, pp 390–405. Springer, 1996.

[THS01] A. M. Tyrrell, G. Hollingworth, and S. L. Smith. Evolutionary strategies and intrinsic fault tolerance. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pp 98–106. IEEE Computer Society, 2001.

[TL00] Adrian Thompson and Paul Layzell. Evolution of Robustness in an Electronics Design. In Julian F. Miller, Adrian Thompson, Peter Thomson, and Terence C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware. Third Int. Conf., ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, pp 218–228. Springer, 2000.

[VM00] V. K. Vassilev and J. F. Miller. The advantages of landscape neutrality in digital circuit evolution. In Julian F. Miller, Adrian Thompson, Peter Thomson, and Terence C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware. Third Int. Conf., ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, pp 252–263. Springer, 2000.

An Evolvable Micro-controller or what's new about mutations?

Uwe Tangen
FhG – Fraunhofer Society
Biomolecular Information Processing
Schloss Birlinghoven
53754 St. Augustin, Germany

Abstract

Un-supervised learning of complex software projects is still an issue. Additionally, the objective to create open-ended evolutionary systems is lacking a confident realization. Long-term evolutionary behavior results in noisy regimes or stable fixed points including limit cycles. The evolving organizations are more or less simple or directly reflecting the programmer's point of view of a complex organization. One possible way to overcome these limitations is to drastically increase population sizes and look at longer timescales. The work reported serves two purposes: the introduction of an evolving micro-controller inclusive evolving assembler code on the new massively parallel configurable hardware MereGen^{TM}and it provides new insights on how mutation probabilities may influence the evolutionary outcome of evolving populations.

With 648 MBytes of fast SRAM and about a Gigabyte of SDRAM, millions of programs and thousands of micro-controllers can be evolved concurrently in hardware. Typical speedups are about five orders of magnitude compared with current day high-end PCs. Thus, longterm evolution of large populations is no longer the bottleneck and limitations in our understanding and the observability of evolving systems immediately grabs the focus of our attention in evolutionary experiments.

1 Introduction

The study of evolving hardware aims at creating a new power level of man-made machines. Evolving hardware is expected to be able to adapt to unknown environments and to find solutions hardly found by humans. Evolution, at least in the biological context, is intimately bound to information processing with strings typically considered as information carriers. Machines working on their own description are especially well suited to study the scope of evolutionary concepts.

The idea of information processing machines operating on themselves was first published in 1842 [1] by Menabrea in a translation from Ada Augusta, Countess of Lovelace:

> ...
> Considered under the most general point of view, the essential object of the machine being to calculate, according to the laws dictated to it, the values of numerical coefficients which it is then to distribute appropriately on the columns which represent the variables, it follows that the interpretation of formula and of results is beyond its province, unless indeed this very interpretation be itself susceptible of expression by means of the symbols which the machine employs. Thus, although it is not itself the being that reflects, it may yet be considered as the being which executes the conceptions of intelligence. ...

Though they didn't really attempt to build a machine capable of understanding itself – the idea of such an endeavor was not completely out of the question. They had at that time established the principles of a programmable computer (e.g. conditional execution and loops have been explicitly mentioned). Today we have the hardware to really delve into this domain where electronic hardware is able to self-organize and to evolve.

Despite the famous book of von Neumann [2] and his

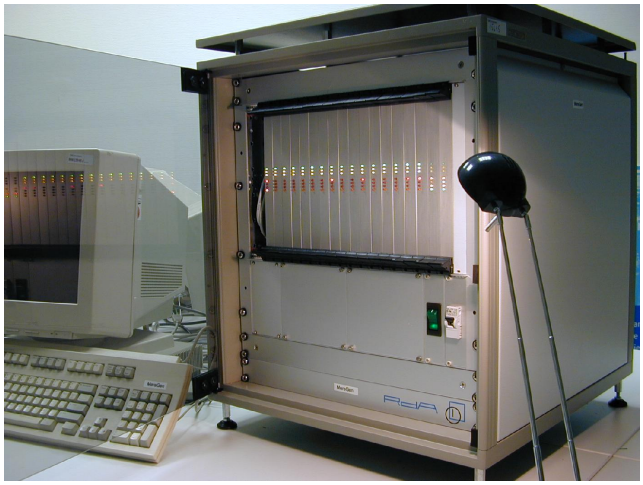

Figure 1: MereGen^{TM}
The 19 inch rack of MereGen^{TM} is shown with 18 boards.

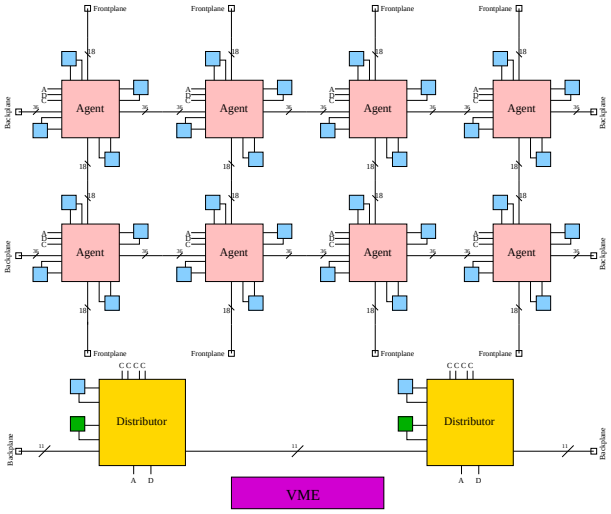

Figure 2: Sketch of the board layout
A board in MereGen^{TM} is organized hierarchically. At the bottom the VME-bus interface is drawn (Xilinx xc4013xl). The two distributors (Xilinx xcv300e) each control a block of four agents (Xilinx xcv600e). The evolving microcontrollers (μCs) are configured into the agents. A further μC, configured into each of the distributors, is dedicated to the sequential part of controlling the experiments. All agents are working continuously and simultaneously with a 64 MHz clock. Attached to the agents four fast 7.5ns 9 MBit SRAMs are used to store the evolving program data.

self-reproducing automata – it was not until 1984 that the extension of the original idea of Charles Babagge and Ada Augusta was revived – probably not known to the author then – by Dewdney in his article in Scientific American [3] about Core Wars. Dewdney had been inspired by the first worms and viruses being observed in the computer labs of Xerox. In the aftermath, several people tried to evolve programs [4, 5, 6] and seriously attempted to realize self-organized software systems. There, non evolvable register machines determine the artificial physics evolving programs are subjected to. A special minimalistic evolving system has been investigated in [7] with only two instructions available for evolution – replicate the genome or do nothing.

In a more general context, these register machines might be approximated by simpler cellular automata, see e.g. Codd [8] or Langton [9]. The comprehensive field of genetic programming introduced by Koza also uses evolving programs but not in a self-organizing context – they solve optimization problems though some work has been undertaken to realize these optimizations problems in hardware [10].

The work reported in this paper solely focuses on the self-organizing context without any optimization problems in mind. These are to be tackled in a subsequent step when self-organizing systems are better understood. The rational behind this work is not only to try to understand the transition from the non-living to the living world but more importantly to lay the foundation for sparsely controlled nano-systems. Massively parallel dynamical systems in the nanometer range can no longer be controlled in a detailed manner. They cannot even be observed in a detailed manner. In order to cope with these types of systems we must allow them to organize themselves – otherwise we will not be able to use dynamical nano-systems.

Independent of the description level used, programmable machines being physically realized have to obey certain constraints. Most often conservation laws are mentioned. Taking into account dissipative structures, conservation of energy can be abstracted relatively easily. The results of this work show the great importance of the following aspects: 1) physical properties like limited space – physical entities require space to be situated, 2) timing delays – information traversing from point A to point B needs a certain amount of time and 3) routing – why does point X know that point C is in need of exactly this data and of nothing else? Of course these physical properties might be simulated easily on current day PCs or workstations – paying the price of slower simulations. What

is not easily simulated, is the inherent explicit parallelism of chemical or biological systems. The overhead for these to simulate is considerable. Fine grained massively parallel machines have a great advantage in this respect. The more the simulations become hardware dependent, the less effort has to be made to model physical properties – they are inherent. Of course, the price is now a restriction in the number of possible physical model systems.

The work described here has been done on MereGen^{TM} [11], fig. 1, a massively parallel configurable hardware using up to 144 high performance field programmable gate arrays from Xilinx [12] dedicated solely to user designs. With MereGen speedups of up to 200.000 (five orders of magnitude, two to three orders due to parallelism and another two to three orders due to exploitation of hardware instantiated physics) have been realized.

MereGen^{TM} uses field-programmable-gate-array (FPGA) and VME64x-bus technology and was especially designed for the simulation of biological model systems and for doing research on evolving hardware. With an approximated four tera instructions per second and over 600 MByte of high speed static memory, MereGen^{TM} allows research which has not been possible with computer hardware in the quarter million dollar regime. The eighteen boards provide more than 8.5 million gates equivalents for user designs. In fig. 2, the organization of a board is shown. A Linux operated 400 MHz PowerPC connects via the VME-bus standard to the bus-interface at each board and controls the chip configuration as well as the readout and experiment scheduling.

2 The hardware model

Having studied evolving Boolean hardware [13, 14] the apparent mismatch between the needed length of description and the computational power of the resulting Boolean apparatus made us use micro-controller (μC) type machines. It was hoped that the brittleness [15] observed in the evolving Boolean hardware could be managed more easily in a micro-controlling context.

The idea of the evolvable μC is derived from [16] a minimalistic form of a RISC machine with only one instruction (MOV) left. Manipulating data is realized here through side effects when moving data from the input port to the output port or a register. Several thousand μCs (the maximum achieved were 3456 μCs in MereGen^{TM}) are chained up in such a way that each μC has access to the upper element of the stack of the neighboring μCs and to one of their registers. This

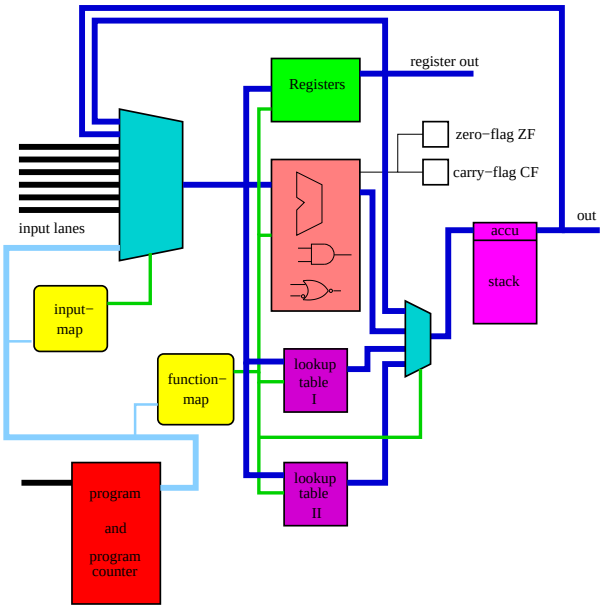

Figure 3: The evolving micro-controller (μC)
Only one instruction (MOV) is available for programs using this 8-bit μC. With just one instruction left, only the parameters are coded in one byte of the program code. The upper four bits of the instruction stream determine the location from where data is moved towards the location determined by the lower four bits of an instruction. Whether flags are tested or functions are executed depend only on the particular location addressed. With an additional mapping of the possible 16 input (input map) and 16 output (function map) locations to different input ports and different functions, extremely dense code can be realized. This mapping is defined at runtime and can therefore be subjected to evolution. In addition, two arbitrary function generators (lookup-tables I and II) which are able to realize any unary function with 8 bit wide I/O can be used to be changed by the evolving system at runtime. Thus a plastic μC is available for the hardware evolution experiments. The program size can be varied between 128 memory partitions at 8 instructions each and two partitions at 512 instructions each. Two special side-effects SWI_SRC and SWI_DEST are available to change the active partitions.

chain actually is a closed loop like in the work of Rasmussen [4]. In the experiments reported, 16 μCs per chip were used. These 16 μCs share the four SRAMs available at each agent. Each μC gets access – in a round robin fashion – to the attached SRAMs for a period of 1024 clock cycles and writes its programs or stack values in the SRAMs which are operated as shift-registers. The SRAM data on the other hand is available to all μCs on the chip at the same time.

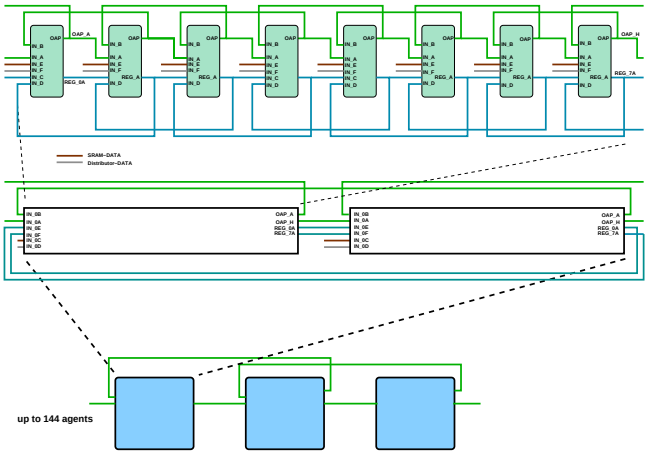

Figure 4: Chain of up to 3456 μCs

Up to 24 μCs configured into a chip give a chain of up to 3456 μCs working in parallel and at a clock rate of 64 MHz with a minimum number of one instruction per clock cycles. The design for 16 μCs is shown which has been used in most experiments. The four SRAMs attached to each agent are operated as 64 Bit memory and each word containing 8 instructions which are serially fed into the μCs at a special input port.

This means that a program in principle could be easily replicated by a factor of 16 if and only if the μCs on the chip are listening.

As already mentioned, routing of information is a central issue for evolving spatially extended information processing systems. To test the capabilities of the evolving μCs and to find a measure for evolutionary abilities [17] the topology of the system is made scalable in the range of more than two orders of magnitude. This means that by pure parameterization of the system at runtime the chain of μCs can be partitioned in 144, 36 and 18 pieces or used with full length. When designs are used with only eight μCs per chip even smaller evolving chain partitions can be investigated. According to [17] a shadow model is defined as an evolutionary model with interactions between individuals in the evolving population disabled. Here, isolating μCs serves the same purpose. Comparing the different topologies with each other, the experiment with the larger number of partitions can be seen as a shadow model. Experiments with special disabled functions allow to grab the semantics in the system as it is a typical procedure in (gene-) knock-out experiments in micro-biology.

3 Observing the system – complexity measures

Observing a self-organized evolution system is not at all trivial. In contrast to Ray [5] and others, the system is not seeded with a known self-replicator. The reason for this is that functional cells are NOT thought to exist *a priori* like the evolving systems of the Tierran type. Thus, it is not known what to expect in the system. The next problem connected with observation is the sheer amount of data being processed by the machine – several tera-bytes per second! Both problems ask for an independent area of research.

> A preliminary attempt to observe the system follows the idea that what is important will be existent in several copies in the evolving system!

An algorithm has been designed to extract all patterns in the genomic data which occur at least $k = 14$ times [18] and $min\_len = 30$ instructions. The patterns themselves are arbitrary. This algorithm is part of a compression algorithm and approximates Kolmogorov entropy [19] to a certain extent. The maximum length of the pattern searched for is restricted to 270 instructions – the reason being a limited amount of main-memory in the host computer.

All complexity measures used in this work are taking this pattern analysis data as their base and not the original genomic data. Of course, tera-bytes per second cannot even be written in the main-memory of the computer. Only a small fraction of programs is read out of the agent chips via the distributors in a regular fashion. These programs are then picked up by the host and are used for the pattern analysis.

The complexity measures used are as follows:

- EXPENSE: The pattern searching algorithm searches for repeating pairs of instructions in all picked up programs. From these pairs of instructions repeating quadruple instructions are searched until no further repeats with a minimum occurrence are found. In a last step it is attempted to elongate the longest patterns found so far. Every attempted combination of new patterns is recorded as a unit effort and counted. The total number of unit efforts divided by the total number of instructions gives the complexity measure EXPENSE.
- DIVERSITY: This is the most simple complexity measure used here. The frequency of all instruction pairs is counted. There are 2^{16} different

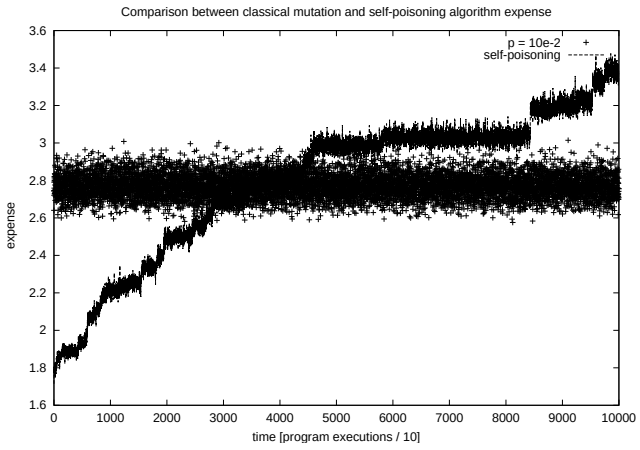

Figure 5: Two different variation systems
Two different mutation processes are shown. The classical mutation procedure with a fixed mutation probability, in this case $p = 10^{-2}$ is used, is shown with the curve '+'. With the second mutation procedure the frequency of each instruction in the genomic data is determined. In the next program iteration the most frequent instruction is replaced by a random value. Not only can different longterm evolutionary behavior be observed but also a higher value of the complexity measure EXPENSE is achieved. This means that the pattern search algorithm has increasingly more difficulties to determine repeats in the genomic data.

instructions pairs possible. The diversity is calculated as:

$$diversity = \frac{(max - min) * nr\_pair}{nr\_inst},$$

with min the count of the instruction pair with minimum frequency, max the count of the instruction pair with maximum frequency and nr_pair the number of all different instruction pairs found in the genomic data of the system.

- PATCOMP: The number of pattern levels (pair, quadruple ...) in the sequence data times the number of patterns of $min\_len = 30$ instructions found in the first iteration of the pattern search algorithm.

4 Mutations – what's new?

So far, in almost all evolutionary models published, e.g. [20] and followers, mutation is thought to be a random event, like a cosmic ray changing the information content of a sequence of letters in a uniformly distributed manner. Of course, in molecular biology it is well known that mutational events are only in rare cases evenly distributed over the genomic sequences which is thought to be a consequence of the selection pressure subjected to the system investigated. Mutation in this work always means exchange of an instruction by a random instruction if a mutational event occurs. It is futile to try and carry out bit-mutations because the fitness consequences per bit-change cannot be expected to be smooth!

Realizing a first C-simulation to test different μC-variants, it soon turned out that the evolving system more or less immediately got trapped in a stationary behavior, fig. 5 (curve at $p = 10^{-2}$). Considering the Gaya-hypothesis and the role oxygen might have played at that time, the idea of mutation as a **self-poisoning** process emerged. The idea is simple: the most successful sequence or species produces so much waste (waste is useless or even toxic per definition) that it destroys its own base of existence.

In the μC-model this could easily be established, e.g. by counting the number of instructions used in all programs. After determining the instruction with the highest frequency this instruction is destined to be completely randomized in the following round of program execution. Whenever this instruction is encountered in the μC, a uniformly distributed random number is taken instead. Other mutational events have been completely abandoned. The resulting behavior of the system changed dramatically, fig. 5 (increasing curve), non-trivial longterm evolution could immediately be observed. The reaction of the evolving system is to develop increasingly complex strategies to avoid dominance of only one program part in the system because this program would almost certainly be destroyed.

5 Experimental setup and results

A typical experiment lasts about seven minutes. At the beginning all the chips (distributors and agents) are configured and the parameters are written to local registers of the designs. Then the agents are activated and for 400 seconds a 64MHz clock continuously drives all chips in parallel. Micro-controllers in the distributors read out the programs evolving in the agents and when necessary count the frequency of instructions in the programs. Instructions, according to the parameters chosen, with the highest frequency are then written back into the agents in a certain register. If the system is working in a self-poisoning mode these instructions are exchanged by random instructions. The total design is constructed such that the evolving agents are not directly coupled with the ob-

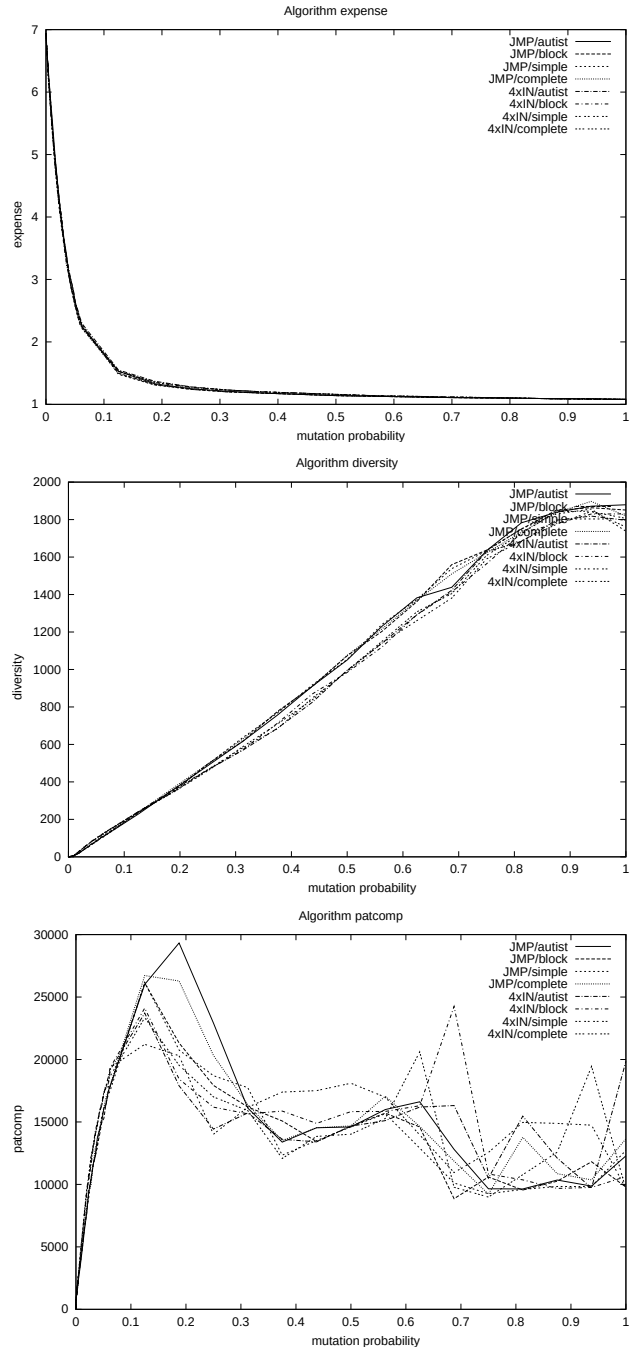

Figure 6: Different mutation probabilities

Each plot comprises eight series. Each two series have different connection partitions (autist = 128 partitions, block = 36 partitions, simple = 18 partitions and complete = 1 partition). JMP curves have a JMP-operator in the mapped function space and '4xIN' curves have four input lanes for reading the SRAM data. The top-down sequence is EXPENSE, DIVERSITY and PATCOMP at the bottom.

serving host. Because of the continuous behavior and the lack of naturally defined generations (what are the species and how long do they live?) in the system the time-base chosen is real-time.

In fig. 6, a bunch of experiments on MereGen^{TM} is shown. Each plot needed about one and a half day of operation time on MereGen^{TM}. The values plotted are average values of the last twenty complexity measure evaluations per run. A variation of the mutation rate has been investigated. As expected, it has turned out that the system depends on the mutation rate which is changed from rare mutations $p = 0, \frac{1}{2^{16}}, ..$ towards high mutation rates $p = 1$. Taking e.g. the upper plot in fig. 6, the effort needed to find all repeats is highest at low mutation rates. This is quite natural because if all instructions are identical, many different levels of repeats exist. Below two evolved program are shown. The first one is taken with a mutation rate of $p \sim 10^{-5}$ the second with a mutation rate of $p \sim 10^{-1}$ when PATCOMP has its maximum value. The listing only reflects one brief glimpse on an evolved program. Though it could be observed that programs might dominate the system for several seconds, trivial stationary behavior is not apparent.

```
# p = 1.5e-5 pattern length = 270 inst. (maximum)
   input section (4 bits)    function section (4 bits)
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _IN_B_ -----         -> _SWI_SRC_A_
 #    _COM_ -----          -> _WRI_REGA_A_
 #    _COM_ -----          -> _WRI_REGA_A_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _OAP[7:0]_ -----     -> _IF_CF_PUSH_
 #    _IN_E_ -----         -> _SWI_SRC_A_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _IN_B_ -----         -> _SWI_SRC_A_
e.g. take value of register _COM_ and store this value
                    in register _REGA_
 #    _COM_ -----          -> _WRI_REGA_A_
                 .
                 .
```

In the second listing two commands are sometimes depicted on the input section. This is a consequence of the possibility to execute more than one instruction per clock cycle. The notation _WRI_REGA_A_ means that the value taken from the input section is written into register A with command mapping A. In addition, most of the commands can be executed via the command mapping B which is activated if the command _EXE_COM_ is encountered. This allows, at the price of lower execution probabilities, the increase of the possible command set.

```
# p = 1.25e-1 pattern length = 210 instructions
                 .
                 .
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _COM_ -----          -> _IF_CF_ZF_PUSH_
 #    _IN_B_ -----         -> _WRI_COM_A_
 #    _OAP[7:0]_ _IF_ZF_POP_     -> _WRI_COM_A_
```

```
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _OAP[7:0]_ _IF_ZF_POP_     -> _WRI_REGC_A_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _REGA_ -----       -> _WRI_REGB_A_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
#    _OAP[7:0]_ _POP_   -> _ADD_VAL_A_
#    _REGC_ -----       -> _IF_CF_PUSH_
#    _COM_ -----        -> _PUSH_
#    _COM_ -----        -> _IF_CF_ZF_PUSH_
                .
                .
```

6 Discussion

In the research on self-organized dynamical systems there are two basic questions: firstly the question on the transition from the non-living to the living world and secondly the need to abandon detailed control of complex nano-systems. Though the first question might sound academic, the opposite is the case. If we know how artificial life can be created we probably will know much more about creativity as such and other higher mental processes. The second question springs to mind when e.g. in DNA-computing [21] the I/O of data becomes an extraordinary issue.

The use of dedicated hardware in the research of biological model systems has both benefits and drawbacks. The benefits are obvious – the systems to be investigated are closer to real biological systems – in scale and because physics are inherently present. On the other hand, the large amounts of data processed by the hardware and its speed forbid the detailed analysis of all sequences in the machine. Now, as it is the case with molecular biology, filtering the essential data, realizing experimental controls and being content with much less powerful tools is the prize which, inevitably, has to be paid. The work reported truly lies on the verge between computer science, electronics and molecular biology.

A short discussion of the results:

- It could be shown that it is indeed possible to realize pure computer science models of self-organizing systems in hardware. It can be argued what self-replication really means. Looking at the sequences sometimes gives the impression that these are selected because of their bit-properties and not of their functional behavior. Nevertheless, these repeats occur in high numbers and are not random at all. There are a lot of intermediate pattern forming processes between the repeating structures found in crystals and e.g. bacterial cells.

- A highly parameterizable and evolvable μC has been presented with a minimum number of one instruction per clock-tic and thus a power of 64 MIPS. Optimizing the design should allow for μCs in the 80MHz range within MereGen^{TM}. With e.g. 3456 processors, a total power of 221 GIPS of customized μCs have been realized.

- The view on mutational events has broadened in the sense that self-poisoning promises to open up a new class of evolutionary power. Certainly, it is different from fitness sharing [22] or resource constrained evolutionary systems, see the appendix below.

- The task to build artificial living systems still remains. Longterm experiments done so far did not exhibit an escape from stationary behavior. The scaling experiments, fig. 6, did not show any dependency on the connection topology. All different partitions resulted in more or less exactly the same complexity measures. It seems that the selection pressure towards robustness of the evolving programs is so strong that inputs from other processors are deferred as much as possible. Looking at the sequences of the evolved programs (data not shown), no essential differences between the partition sizes can be observed. Another reason might be that no substantial organization developed and the resulting inhomogeneous sequences are a pure consequence of the physical constraints being felt by the system.

  Though this is bad news, the good news are that if one finds the means to let the evolving system display a particular scaling behavior then something important has been learned. Up to now, a statement often made that only the system size has to be sufficiently large to realize the emergent behavior desired cannot be validated.

- With this hardware realization, even non-hardware designers are able to investigate complex hardware models because the flexible μCs are suited to many different applications. With the additional broadband interconnect available on each board, even external apparati might be attached.

7 Conclusions

With MereGen^{TM}, several absolute measures become feasible: Firstly, simulation time is no longer the bottleneck – the speedup due to the hardware is sufficient to analyze the systems in the asymptotic limit. Evolutionary time scales – comparable to the evolution on earth – can be established. Secondly, due to the hardware resources scaling behavior can be measured. If a system does not scale appropriately it can not be expected to show something different if the system size is doubled (the combinatorics involved in these systems are far beyond any hope to achieve exhaustive search), thus a strong limit on possible biological models is established. Thirdly, physical properties like routing delays and routing of information as such can now be studied *in extenso* and and might help – besides biological applications – in optimizing network traffic or other systems where routing is the limiting constraint.

We succeeded in the construction of an information processing machine with the "interpretation be itself susceptible of expression by means of the symbols which the machine employs. Thus, although it is not itself the being that reflects," [1] we now might hope to build and understand self-reflecting machines.

Appendix: Fitness sharing[1]

Fitness sharing [22] is probably the concept most similar to self-poisoning proposed in this article. Though a detailed comparison is out of scope of this work, some distinguishing properties should be mentioned:

- Fitness sharing is a phenotypic concept. Whether an individual in the population shares the available fitness for its sequence with other individuals of the same or similar type or is only in an indirect manner (via e.g. mutation or crossover) determined by the genotype of this individual.

- With self-poisoning the frequency of instructions or genes throughout the population is counted. The parameter σ_{sh}, see tab. 1, should thus be maximized. This would disrupt the ability of the genetic algorithm to adapt to multi-modal fitness landscapes.

- With self-poisoning only dominant genes are affected. Individuals not using these dominant genes or using them as a genetic pool do not suffer from the consequences of randomization or even better, are perhaps able to utilize the newly generated genetic information. Thus, self-poisoning is a genotypic concept.

[1] Thanks to the two referees who made me aware of this research

Fitness sharing	Self-poisoning
$f_i \equiv f(i)$	$f_i \equiv f(i)$
$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{sh}}\right)^{\alpha_{sh}} & \text{if } d_{ij} < \sigma_{sh} \\ 0 & \text{otherwise} \end{cases}$	n_{Ij} = number of instructions or genes of type j N = number of instruction or gene types
$m_i = \sum_{j=1}^{N} sh(d_{ij})$	if $(n_{Ii} = \max(n_{Ij}), 0 < j < N)$ then {
$f_{sh,i} = \frac{f_i}{m_i}$	$f_i \simeq 0$ in most cases }

Table 1: Fitness sharing [22] versus self-poisoning
The formulas on fitness sharing are taken from [23]. With f_i the original fitness of an individual, d_{ij} the Hamming-distance between two genomes and $f_{sh,i}$ the effective shared fitness of an individual. The two parameters σ_{sh} and α_{sh} determine the behavior of the genetic algorithm applied to a usually multi-modal problem. With $0 < \sigma_{sh} < 1$ and $\alpha_{sh} \neq 0$ only identical sequences are reduced linearly in fitness. To compare with self-poisoning σ_{sh} should be chosen as $\sigma_{sh} = \max(d_{ij})$ with the consequence that only unimodal fitness landscapes can be adapted properly. The main difference between both concepts is the target of fitness reduction: in fitness sharing the total fitness receivable by a sequence (phenotype) is constrained, but with self-poisoning the genotype is affected through randomization of the most abundant instructions or genes in the population of sequences.

Whether the problem of using explicit fitness as in genetic algorithms or implicit fitness used here as in many other artificial life models is of importance has to be evaluated further.

Acknowledgments

The help and support of J. S. M^{c}Caskill, T. Maeke, E. Rambow and the German Ministry of Science (BMBF) is greatly acknowledged. Also many thanks to the referees for their thorough study of this paper.

References

[1] Menabrea L. F. The Analytical Engine: Invented by Charles Babbage *Bibliothèque Universelle de Genève*, No. 82, 1842.

[2] von Neumann J. Theory of Self-Reproducing Automata Burks, A. W. University of Illinois Press, Urbana, 1966.

[3] Dewdney A. K. Computer-Kurzweil *Spektrum der Wissenschaft*, 5:8–12, 1985.

[4] Rasmussen S., Knudsen C., Feldberg R., Hinsholm M. The Coreworld: Emergence and Evolution of Cooperative Structures in a Computational Chemistry *Physica D*, 42:111–134, 1990.

[5] Ray T. S. An Approach to the Synthesis of Life In Langton C. G., Taylor C., Farmer J. D., Rasmussen S., editors, *Artificial Life II*, pages 371–408 Addison-Wesley, New York, 1991.

[6] Adami C., Brown C. T. Evolutionary Learning in the 2D Artificial Life System "Avida" In Brooks R., Maes P., editors, *Artificial Life IV*, pages 377–381 MIT Press, Cambridge, MA, 1994.

[7] Tangen U. The Extension of the Quasi-Species to Functional Evolution PhD Thesis, Jena, 1994.

[8] Codd E. F. Cellular Automata "Academic Press", New York, 1968.

[9] Langton C. G. Artificial Life Addison-Wesley, New York, 1989.

[10] Koza J. R., Benett III F. H., Hutchings J. L., Bade S. L., Keane M. A. "Evolving Computer Programs Using Rapidly Reconfigurable Field-Programmable Gate Arrays and Genetic Programming" In *"Proc. of the 1998 ACM/SIGDA seventh international symposium on Field Programmable Gate Arrays"*, pages 209–219 "ACM", 1998.

[11] Tangen U., Maeke T., M^{c}Caskill J. S. Advanced Simulation in the Configurable Massively Parallel Hardware MereGen In *Caesarium 2000, LNCS* Springer, 2001 In press.

[12] Xilinx Virtex^{TM}-E 1.8V Field Programmable Gate Arrays, DS022 Xilinx Corporation, San Jose, USA, 2000.

[13] Tangen U., Schulte L., M^{c}Caskill J. S. A Parallel Hardware Evolvable Computer POLYP: Extended Abstract *Proceedings of the FCCM'97 in IEEE Symposium*, 1:238–239, 1997.

[14] Tangen U. Self-Organisation in Micro-Configurable Hardware In Bedau M., M^{c}Caskill J. S., Packard N. H., Rasmussen S., editors, *Artificial Life VII*, pages 31–38 The MIT Press, Cambridge, Massachusetts, 2000.

[15] Holland J. H. Escaping Brittleness: the Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems In Michalski R. S., Carbonell J. G., Mitchell T. M., editors, *Machine Learning II*, pages 593–623 Morgan Kaufman, Los Altos, CA, 1986.

[16] Donlin A. Self Modifying Circuitry - A Platform for Tractable Virtual Circuitry *Lect. Not. Comp. Sci.*, 1482:199–208, 1998.

[17] Bedau M. A., Snyder E., Packard N. H. A Classification of Long-Term Evolutionary Dynamics In Adami C., Belew R. K., Kitano H., E. Taylor C., editors, *Artificial Life VI*, pages 228–237 MIT Press, Cambridge, Massachusetts, 1998.

[18] Karlin S., Morris M., Ghandour G., Leung M.-Y. Efficient Algorithms for Molecular Sequence Analysis *Proc. Natl. Acad. Sci. USA*, 85:841–845, 1988.

[19] Kolmogorov A. N. Three Approaches to the Quantitative Definition of Information *Int. J. Comp. Math.*, 2:157–168, 1968.

[20] Eigen M. Selforganization of Matter and the Evolution of Biological Macromolecules *Z. Naturwissenschaften*, 58:465–523, 1971.

[21] McCaskill J. S., Wagler P. From Reconfigurability to Evolution in Construction Systems: Spanning the Electronic, Microfluidic and Biomolecular Domains. In Hartenstein R. W., Grünbacher H., editors, *"FPL 2000, LNCS"*, Vol. 1896, pages 286–299 "Springer, Berlin Heidelberg", 2000.

[22] Goldberg D. E., Richardson J. "Genetic Algorithms with Sharing for Multimodal Function Optimization" In Grefenstette J., editor, *"Proc. of the Second Intern. Conf. on Genetic Algo."*, pages 41–49 "Lawrence Erlbaum Associates, Hillsdale NJ", 1987.

[23] Horn J. The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Cooperative Populations "Diss., Cornell Uni.", Urbana, Illinois, 1997.