Genetic Algorithms and their Use in the Design of Evolvable Hardware.

Abhishek Joglekar, Manas Tungare
abhij21@hotmail.com, manas@manastungare.com

3 April, 2000.

Abstract

Genetic Algorithms are an important area of Evolutionary Computing, which is a rapidly growing area of Artificial Intelligence. They are a class of algorithms which mimic the natural process of Evolution and Darwin's principle of Survival of the Fittest – in this case, it refers to the acceptance of the best solution, generated from previous solutions by the use of genetic operators such as crossover and mutation. The next section takes a more detailed look at the background of GAs and outlines the basic concepts in its computer model. Genetic Algorithm as in the case of Darwinian model of evolution relies heavily on random experiments of reproduction. From where does this apparently simple model of problem-solving derive its power? This has been a topic of intense research work, covered in the next section. Section 3 of this paper discusses design of evolvable hardware (EHW), which is a promising approach towards autonomous and on-line reconfigurable machines capable of adapting to real-world problems.

1 Introduction

1.1 Genetic Algorithms

The buzzword doing the beats at all hierarchical levels of the industry today is optimization. Calculus had been the reigning emperor of optimization techniques, until recently. One such optimization technique which mimics the natural process of evolution is Evolutionary Computing. Impressed by Charles Darwin, Prof. John Holland of the University of Michigan viewed the process of Biological Evolution as a process of optimization, where nature selects the best genetic settings to survive in the next generation of offspring. These offspring are then optimized further to give successively better offspring. A Genetic Algorithm similarly selects the most optimal solutions from a set, and uses the genetic operators of Crossover and Mutation to generate further solutions. Each such solution is 'more' optimal than its predecessors. An important feature of biological evolution is robustness – which is what genetic algorithms strive to achieve.

1.2 Evolvable Hardware

Now, moving to the domain of the application of Genetic Algorithms, Configurable Hardware is an approach for realizing optimal performance by tailoring its architecture to the characteristics of the given problem. When the characteristics of a problem are known in advance, and they never change with respect to time, it is relatively easy to build configurable hardware using programmable devices like FPGAs (Field Programmable Gate Arrays) because the designer knows how the hardware should be configured. However for problems where designers cannot know in advance how the hardware should be configured, it is required for configurable hardware to have a capability of autonomous and on-line adaptation to a given problem.

Evolvable Hardware (EHW) is a promising approach towards autonomous and on-line reconfigurable intelligent machines. The basic idea of EHW is to use Genetic Algorithms (GAs) to find the best hardware configuration autonomously (without human intervention), on-line (without need to power down the system, reconfigure and then restart).

2 Development of a Genetic Algorithm

In the words of John Koza, *"What we want to use [in GAs] is nature's method to evolve optimal solutions without the hindrance of preconceived knowledge."* A GA Search proceeds as follows: Two chromosomes chosen randomly from the population are mated and they go through genetic operations like Crossover to yield better chromosomes for the next generation. This is repeated until about a half of the population are replaced with new chromosomes. Because the population size is fixed, chromosomes with lower fitness values tend to be eliminated from the population, therefore after several generations of GA search, relatively high fitness chromosomes remain in the population and some of them are chosen as solutions to the problem.

The process of developing a GA for a particular application consists of the following chief phases:

```
SimpleGeneticAlgorithm()
{
    Initialize the Population;
    Calculate Fitness Function;
    While(Fitness Value != Optimal Value)
    {
        Selection;
        Crossover;
        Mutation;
        Calculate Fitness Function;
    }
}
```

2.1 Initialization

The Algorithm is started with a set of solutions (represented by chromosomes) called population. Each chromosome represents a possible solution to the problem. The most-used way (though not the only way) of encoding chromosomes is a binary string.

Chromosome 1	110110|0110
Chromosome 2	011010|0100

2.2 Calculation of Fitness Function

An evaluation function, called fitness function needs to be defined for a problem to be solved in order to evaluate chromosomes. A chromosome with a high fitness value is likely to be a good solution to the problem.

2.3 Selection of the Best Individuals

There are many methods for selecting the best chromosome – such as: Roulette Wheel Selection, Boltzmann Selection, Tournament Selection, Rank Selection, Steady-State Selection and others.

2.4 Crossover

Crossover selects random genes from parent chromosomes and creates a new offspring. Chromosomes of the two parents are split into two (equal or unequal) halves each. Both the chromosomes are cut similarly. The halves are interchanged and combined to form the child chromosome.

Chrom. 1 110110|0110
Chrom. 2 011010|0100

After Crossover:
Child 1: 110110|0100
Child 2: 011010|0110

2.5 Mutation and Elitism

Child 1: 110110|0100
Child 2: 011010|0110

After Mutation:
Child 1: 110010|0100
Child 2: 011011|0110

After a crossover is performed, the resulting solution might fall into a local optimum – hence some genes of the child chromosome are randomly changed. (In Figure, bits are randomly toggled in case of binary strings.)

However, when creating a new population by crossover and mutation, the best chromosome might be lost. Hence, Elitism is a method which first copies the best chromosome(s) to the new population. Elitism rapidly increases the performance of the GA, by preventing loss of the best-found solution.

3 Basic Idea of Evolvable Hardware

The basic idea of Evolvable Hardware (EHW) is to regard the configuration bits of a software-reconfigurable device as the chromosome of GA. The search-space of configurable bits is very huge, but GAs are very effective without prior knowledge of the search-space.

As a fitness function, we choose the performance of the hardware circuit. For example, in Data Compression with EHW, we use a *predictive function* implemented with hardware. As a fitness function we choose the data compression rate. When a good chromosome is obtained, it is immediately downloaded into the reconfigurable device.

In EHW, it is not required to specify the detailed hardware design. Instead we define a fitness function. A fitness function is the *instinct* of the circuit to evolve itself. If the fitness value of a hardware circuit is degraded due to partial malfunction or some changes in the environment, then the GA-process of EHW is invoked, and the search for a better hardware configuration is initialized. Hence, EHW continues to reconfigure itself in order to get better performance.

The chromosome of EHW specifies two things. One is the *function type* of the evolution unit. In figure, the evolution units correspond to gates like AND-gate and OR-gate. The other is the interconnection among the evolution units. EHW can be classified into two classes according to the grain-size of an evolution unit; gate-level and function-level. The figure is an example of gate-level evolution. In function-level evolution, each evolution unit is higher hardware function than gate-level evolution.

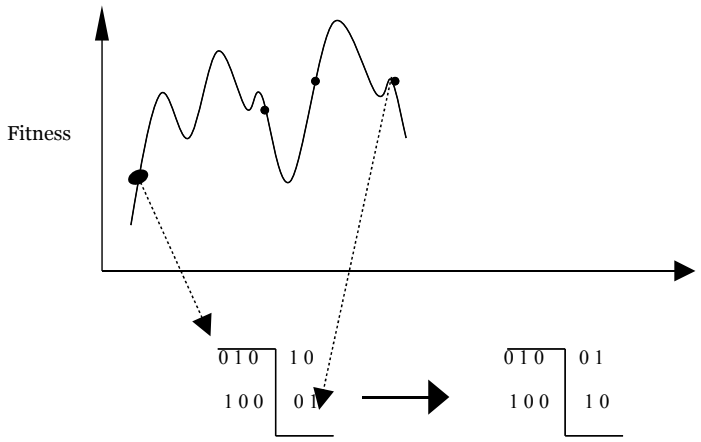

Figure 1: Fitness function of a population

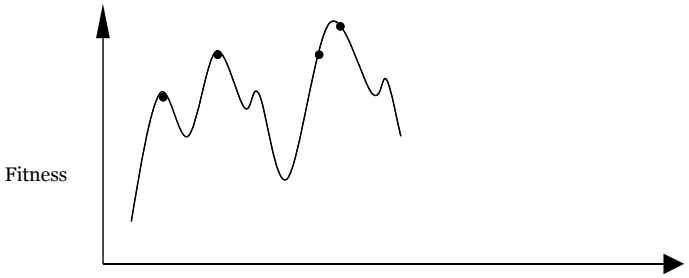

Figure 2: Fitness function of a population after several generations

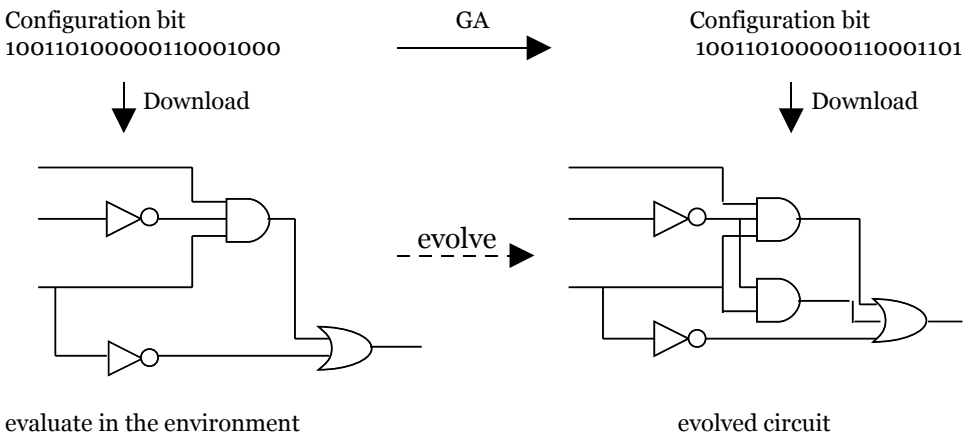

Figure 3: Evolvable Hardware

4 Application of Evolvable Hardware in Neural Networks

Artificial Neural Networks are computational paradigms developed in part because of inspiration by consideration of processes in biological brains. Artificial Neural Networks can perform many different functionalities such as unsupervised learning, supervised learning and optimization.

Most of the industrial Neural Networks (NN) applications are limited to Neural Networks with offline learning, where the learning phase and the execution phase (recognition) are separate. Such a Neural Network never changes during its execution and therefore lacks the flexibility needed. In order to use Neural Networks in a broader range of practical applications, they have to be capable of on-line learning. On-line learning allows neural networks to adapt dynamically to changing problems.

4.1 Ontogenic Neural Networks

The advantage of a neural network is the ability to adapt to the problems by changing interconnection weights on-line. However it is very difficult for the designer to determine the topology of neural network (the number of hidden layers and units per layer) in advance. GAs free the designer of this drudgery of trial-and-error which is inevitable in conventional designing.

Dynamically reconfigurable NN is the key to optimal performance of NNs such the logical NN structure matches the physical NN hardware structure.

5 Scope and Limitations of Genetic Algorithms in Hardware Design

GAs are no doubt a very promising technique for solving optimization problems with a tradeoff between speed and perfection. However, not all problems lend themselves very well to a solution with GAs.

In case of problems involving design of electronic circuits consisting of more than 20 components including resistors, capacitors, inductors, the barrier to applications of the GA technique is that the possible combinations' are too vast to search using a conventional genetic algorithm. In the standard approach, a population of random candidates evolves toward incrementally better solutions. But there are too many beginning analog circuits from which to choose.

The concept of embryonic circuits can be used to bring the analog-circuit-design problem into the realm of the solvable. Embryonic circuits are very minimal circuits designed with the specific problem in perspective. A two-component LC circuit, for example, could serve as an embryonic circuit. Using the embryonic circuit as a starting point, the genetic algorithm operates on it with functions that either add a circuit component or modify a connection among the existing circuit components.

6 Conclusion

We discussed in this paper, the application of Genetic Algorithms to the problem of developing Evolvable Hardware. Evolvable Hardware can deal with practical industrial applications, especially those that require the ability to cope with time-varying problems and real-time constraints.

This paper also explores, though to a limited extent, the fundamentals of genetic algorithms – operators such as crossover, mutation and elitism, and the process of developing a generic genetic algorithm; specifically, the formulation of a genetic algorithm for reconfigurable machines. Further, Neural Networks is an arena where reconfigurable hardware can be used to a great extent; such application of GAs was presented.

To conclude, Genetic Algorithms is currently a fertile ground for research and application development. While a rich set of techniques and models are available, covering a range of domains, there are many areas remaining to be understood and exploited. It must however be noted, that this technique (as every other technique!) poses some limitations over the application areas it can be used in, and hence scope for further development and research in this area is vast.

References

1. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine-Learning,* Addison-Wesley, 1989.
2. M. Murakawa, S. Yoshizawa, I. Kajitani, X. Yao, N. Kajihara, M. Iwata, T. Higuchi, *The GRD Chip: Genetic Reconfiguration of DSPs for Neural Network Processing,* IEEE Transactions on Computers.
3. D. W. Pearson, N. C. Steele, R. F. Albrecht*, Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag Wien.
4. R. Colin Johnson, *Genetic program auto-designs analog circuits, EETimes, June 03*, 1996, Issue: 904.
   http://www.techweb.com/se/directlink.cgi?EET19960603S0059
5. Integrated Systems Group Website: *Evolvable Hardware and Genetic Algorithms*:
   www.ee.ed.ac.uk/~neural/research/eh.html
6. Marek Obitko, *Genetic Algorithms*,
   http://cs.felk.cvut.cz/~xobitko/ga/main.html