Assoc. Prof. Rustem Popa, PhD

http://www.etc.ugal.ro/rpopa/index.htm
The development of new algorithms for testing and automatic reconfiguration of digital systems

I have studied some evolutionary algorithms for synthesis, testing and reconfiguration of programmable digital structures in case of single fault, for the purpose of self repair and adaptation to the changes of the environment. I have developed standard genetic algorithms and algorithms obtained by hybridizing genetic algorithms with other search methods, such as search by induction or simulated annealing.
Teaching experience

**Undergraduate:**

- Digital Electronics - since 1992
- Medical Electronics - since 1994

**Graduate:**

- Audio, Speech and Image Processing - (2006 – 2009)
- Audio and Speech Processing - since 2009
- Evolutionary Techniques in Signal Processing - since 2009
Research interests

- Evolutionary Computation
- Evolvable Hardware (EHW)
- Digital Electronics, including FPGA technology
- Soft Computing
- Digital Signal Processing
- Medical Electronics
- Artificial Consciousness
About Genetic Algorithms

Evolutionary computation is a subfield of artificial intelligence (AI) (or more particularly computational intelligence (CI)), involving combinatorial optimization problems. It may be recognised by the following criteria:

- iterative progress, growth or development
- population based
- guided random search
- parallel processing
- often biologically inspired
About Genetic Algorithms

A typical structure of an Evolutionary Algorithm (EA):

- Initial population of chromosomes
- Offspring
- Population
- Calculate fitness value
- Solution found?
- Evolutionary operations
- Stop

No

Yes
About Genetic Algorithms

- GA begins with a set of solutions (represented by chromosomes) called population. Solutions from one population are selected according to their fitness, and form new solutions (offspring) by using genetic operators (crossover, mutation). This is motivated by a hope, that the new population will be better than the old one.

- This is repeated until some condition (for example number of generations or improvement of the best solution) is satisfied.
About Genetic Algorithms

The structure of a Genetic Algorithm:

begin
  generate randomly the initial population of chromosomes;
repeat
  calculate the fitness of chromosomes in population;
  repeat
    select 2 chromosomes as parents;
    apply crossover to the selected parents;
    calculate the fitness of new child chromosomes;
  until end of the number of new chromosomes
  apply mutation to the new chromosomes;
  update the population;
until end of the number of generations
end
About Genetic Algorithms

Encoding of a chromosome in a GA:

Chromosome 1  1101100100110110
Chromosome 2  1101111000011110

• A chromosome should in some way contain information about solution that it represents. The most used way of encoding is a binary string.

• The encoding depends mainly on the solved problem. For example, one can encode directly integer or real numbers, sometimes it is useful to encode some permutations and so on.
About Genetic Algorithms

Crossover:

<table>
<thead>
<tr>
<th>Chromosome 1</th>
<th>11011</th>
<th>00100110110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome 2</td>
<td>11011</td>
<td>11000011110</td>
</tr>
<tr>
<td>Offspring 1</td>
<td>11011</td>
<td>11000011110</td>
</tr>
<tr>
<td>Offspring 2</td>
<td>11011</td>
<td>00100110110</td>
</tr>
</tbody>
</table>

This example shows a crossover in a single point. There are other ways how to make crossover, for example we can choose more crossover points. Crossover can be quite complicated and depends mainly on the encoding of chromosomes.
About Genetic Algorithms

Mutation:

<table>
<thead>
<tr>
<th>Original offspring 1</th>
<th>1101111000011110</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original offspring 2</td>
<td>1101100100110110</td>
</tr>
<tr>
<td>Mutated offspring 1</td>
<td>1100111000011110</td>
</tr>
<tr>
<td>Mutated offspring 2</td>
<td>1101101100110100</td>
</tr>
</tbody>
</table>

Mutation operation randomly changes the offspring resulted from crossover. In case of binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. The probability of mutation is low and it prevents the falling of all solutions in the population into a local optimum.
About Genetic Algorithms

Parameters of GA:

• Population size
• Encoding (binary, permutation, value, tree)
• Selection (roulette wheel, elitism, tournament)
• Crossover type
• Crossover probability
• Mutation type
• Mutation probability
Some results

Evolvable Hardware (EHW):


Some results (EHW)

Intrinsic evolution of a modulo 5 counter on a motherboard with CMOS integrated circuits (2004)

---

**Fig. 9.** The building-block for intrinsic digital EHW

**Fig. 10.** An example of an evolved counter with 5 states
Some results (EHW)

Two examples of extrinsic evolution of a state machine with 8 states (2006)

Fig. 5. Evolved optimal circuit solution of the sequence detector (first solution).

Fig. 6. Evolved optimal circuit solution of the sequence detector (second solution).
Some results (EHW)

Two examples of extrinsic evolution of a boolean function with 3 variables in a network with 4 gates, using 3 GAs: (2010)

- CGA (Canonical Genetic Algorithm)
- SCQGA (Single Chromosome Quantum Genetic Algorithm)
- QIGA (Quantum Inspired Genetic Algorithm)
Fig. 4. The evolutions of CGA, SCQGA and QIGA in attempt to synthesize the function $f$ given in (12). It was represented the fitness evaluation of the best chromosome in population after 10 successful runs of each algorithm on 300 generations.
## Some results (EHW)

### Table II
A Comparison Between CGA, SCQGA and QIGA

<table>
<thead>
<tr>
<th>Parameter</th>
<th>CGA</th>
<th>SCQGA</th>
<th>QIGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global time</td>
<td>73.990 s</td>
<td>38.599 s</td>
<td>19.263 s</td>
</tr>
<tr>
<td>Self time</td>
<td>2.447 s</td>
<td>1.417 s</td>
<td>1.390 s</td>
</tr>
<tr>
<td>Evaluation time</td>
<td>59.561 s</td>
<td>31.536 s</td>
<td>11.750 s</td>
</tr>
<tr>
<td>Calls of eval. func.</td>
<td>25200</td>
<td>19200</td>
<td>4836</td>
</tr>
<tr>
<td>Ratio between eval. and global time</td>
<td>80.5 %</td>
<td>81.7 %</td>
<td>60.9 %</td>
</tr>
<tr>
<td>Nr. of generations</td>
<td>300</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Successful runs in 10 attempts (with fitness of 100%)</td>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
Some results

The Hybridisation of GAs:


Some results

The Hybridisation of the Selfish Gene Algorithm:

Figure 5. Runs of the two algorithms based on the Selfish Gene Theory on a fault coverage problem of 50 possible faults

Figure 6. A comparison between two hybridated algorithms on a fault coverage problem of 50 possible faults
Some results

Multiple Hybridisation in GAs:

Figure 5. The fault coverage as a function of the number of input test vectors

Figure 7. Runs of the three hybridated algorithms on a fault coverage problem of 50 possible faults

Figure 6. Runs of the two genetic algorithms on a fault coverage problem of 50 possible faults

Figure 8. Runs of the three hybridated algorithms on a fault coverage problem of 200 possible faults
Some results

Image Processing with GAs:


Some results

Two Different Images of the Same Melanocytic Lesion, an Affine Transformation and Two Different Lesions:

\[
\begin{bmatrix}
  x_1' & y_1' & 1 \\
  x_2' & y_2' & 1 \\
  \vdots & \vdots & \vdots \\
  x_N' & y_N' & 1 \\
\end{bmatrix}
\begin{bmatrix}
  a_1 & b_1 \\
  a_2 & b_2 \\
  a_3 & b_3 \\
  \vdots & \vdots \\
\end{bmatrix}
= \begin{bmatrix}
  x_1 & y_1 \\
  x_2 & y_2 \\
  \vdots & \vdots \\
  x_N & y_N \\
\end{bmatrix},
\]