Genetic Algorithms for EQ-algebras Automatic Generation

Hashim Habiballa, Vilém Novák, Martin Dyba

Centre of Excellence IT4Innovations - Division University of Ostrava Institute for Research and Applications of Fuzzy Modeling
University of Ostrava
Czech Republic
{hashim.habiballa, vilem.novak, martin.dyba}@osu.cz

21.10.2013
1 Introduction

2 EQ-algebras

3 Specific genetic algorithms for EQ-algebras design

4 Implementation
Motivation

- Finite algebras generation with specific properties
- Task specification
  - \( n \) - number of algebra elements
  - Algebra operations declaration
  - Compulsory properties of operations
  - Optional properties of operations
  - Generate such algebra fulfilling requirements above
- Manual creation with help of properties automated check
- Brute force (combinatorial) approach
- More sophisticated methods?
Why not brute force?

- Example: $n$ elements, $k$ binary operations, $l$ axioms ($m$ elements dependence)
  - $N_c = (n)^{k*n*n}$ possible candidates
  - $l$ axioms check - expression evaluations $N_{ev} = l * (n^m)$ for every candidate
  - total expression evaluations $N_t = N_c * N_{ev}$
  - expression means dozens of simple (CPU level) instructions
  - current common computer about $10^9 - 10^{10}$ instructions per second
    e.g. Intel Atom N270 - 3 GIPS, Intel Core i7 920 (Quad core) - 80 GIPS, SC IT4I (2015) cca $10^{15}$ IPS (FLOPS)...

- Fix $k = 3, l = 10, m = 3$
  - $n = 4, \{0, a, b, 1\}$, $N_c \doteq 7.9 * 10^{28}$, $N_t \doteq 5.1 * 10^{31}$
  - $n = 5, \{0, a, b, c, 1\}$, $N_c \doteq 2.6 * 10^{52}$, $N_t \doteq 3.3 * 10^{55}$
  - $n = 6, \{0, a, b, c, d, 1\}$, $N_c \doteq 1.0 * 10^{84}$, $N_t \doteq 2.4 * 10^{87}$
  - $n = 7, \{0, a, b, c, d, e, 1\}$, $N_c \doteq 1.6 * 10^{124}$, $N_t \doteq 5.8 * 10^{127}$
  - ...
Why not brute force?

- "Hard" computing fails on superexponentiality of the problem (although many optimizations are possible, $o(n^n)$ remains)
Genetic algorithms (GA)

- Genetic algorithms - successful softcomputing method based on evolutionary principles
- User may select such parameters of GA to achieve "optimal" (not necessarily best) results in "reasonable" time in contrast to brute force
- Main characteristics:
  - Population member (candidate solution), its fitness function (evaluates suitability)
  - Population - set of members, starting population (random)
  - New generation created from previous by selection, crossover and mutation
  - Generate new populations until stop condition is fulfilled (fix number of iterations - populations, predefined member fitness being optimal etc.)
**Genetic Algorithm Flowchart**

1. **Initial population**
2. **Fitness evaluation for population (current generation)**
3. **Termination condition?**
   - **Yes**: **Extract solution**
   - **No**: **Generation of new population – selection, crossover**
4. **Mutation**
Introduction

Population and Population Member (GA)

- Candidate solution \( p \) (Population Member / PM) represented by its properties (usually stored in "chromosomes" - bit array, integer array etc.)
- Fitness function of candidate solution \( f \), \( f(x) \in \langle 0, 1 \rangle \), \( x \) is PM - the keystone of time complexity of the task (possible parallelism)
- Population - fix or variable number of PM
  - Population member (candidate solution), its fitness function (evaluates suitability)
  - Population - sets of PMs, best PM, worst PM, median PM
  - Generation - sequence of populations called generations \( G_0, \ldots, G_r \), where \( G_i = \{p_{i,j} | i, j \in N\} \), \( i \) is generation index, \( j \) is PM index in population
  - Starting Generation \( G_0 \) is randomly (partially randomly) generated
Genetic operators (GA)

- Selection - simply into next generation or further processing
  - Elitist - usually best $m$ PM from $G_i$ is directly copied into $G_{i+1}$
  - Selection for crossover (SC) - some PMs from $G_i$ are selected for generation of new children for $G_{i+1}$.
  - SC should inhere probability of selection $prob_{SC}(p)$ for PM $p$ non-decreasing with respect to fitness function:
    $$f(p_1) \geq f(p_2) \Rightarrow prob_{SC}(p_1) \geq prob_{SC}(p_2)$$
- Crossover - combination of several PMs to generate new PMs for next generation
  - Simple - two old PMs $p_{old1}, p_{old2}$ generate two children, where first portion of chromosome is from $p_{old1}$ and second from $p_{old2}$ and contrary
  - Exponential - if we can distinguish several portions of chromosome we can generate more children than parents (every possible combination)
Genetic operators (GA)

- **Mutation** - randomly selected PMs from new generation are "altered"
  - Mutation rate - probability of selection PM for mutation
  - Point - single element of chromosome is altered
  - Interval - interval of chromosome elements are altered
  - Overall - whole chromosome is altered

- **Termination** - we have to end iterative application of operators to new generations
  - Best PM (average, median) - best PM in last population has fitness greater or equal to predefined value
  - Step - fixed number of steps (generations) is produced
  - Suitable PMs - predefined number of PMs with required fitness is generated
  - Time - time elapsed restriction to iteration
  - Peak - peak fitness is achieved and $m$ next generations has worse fitness (or more sophisticated dependence on fitness)
Task - EQ-algebras generation

- EQ-algebras as truth value structure for EQ-logics
- Key operation - Fuzzy Equality
- 3 basic binary operations fulfilling several properties
  - Infimum $\land$
  - Multiplication $\otimes$
  - Fuzzy Equality $\sim$
- One possible additional unary operation
  - Delta $\Delta$
- One derivable binary operation
  - Supremum (maximum) $\lor$
- Additional supporting (directly following) connectives
  - Implication $\rightarrow$
  - Negation $\neg$
  - LessOrEqual $\leq$
Task - EQ-algebras definition

EQ-algebra $\mathcal{E}$ - algebra of type $(2, 2, 2, 0)$, $\mathcal{E} = \langle E, \land, \otimes, \sim, 1 \rangle$

(E1) $\langle E, \land, 1 \rangle$ is a commutative idempotent monoid (i.e. $\land$-semilattice with top element $1$). We put $a \leq b$ iff $a \land b = a$, as usual.

(E2) $\langle E, \otimes, 1 \rangle$ is a monoid and $\otimes$ is isotone w.r.t. $\leq$.

(E3) $a \sim a = 1$  \hspace{1cm} (reflexivity axiom)

(E4) $((a \land b) \sim c) \otimes (d \sim a) \leq c \sim (d \land b)$  \hspace{1cm} (substitution axiom)

(E5) $(a \sim b) \otimes (c \sim d) \leq (a \sim c) \sim (b \sim d)$  \hspace{1cm} (congruence axiom)

(E6) $(a \land b \land c) \sim a \leq (a \land b) \sim a$  \hspace{1cm} (monotonicity axiom)

(E7) $a \otimes b \leq a \sim b$  \hspace{1cm} (boundedness axiom)
EQ-algebra - Additional operations

- Implication - $a \rightarrow b = (a \land b) \sim a$
- Negation - If $E$ contains also the bottom element 0 then we put $\neg a = a \sim 0$ and call $\neg a$ a negation of $a \in E$.
- Maximum (supremum) - $\lor$ is derived from $\land$ preserving this condition: $(a \land b = a) \Rightarrow (a \lor b = b)$ (details in algorithms).
- Delta - EQ-algebra $E$ extended by a unary additional operation $\Delta : E \rightarrow E$ fulfilling the following axioms:

  $(E\Delta 1)$ $\Delta 1 = 1$
  $(E\Delta 2)$ $\Delta a \leq \Delta \Delta a$
  $(E\Delta 3)$ $\Delta (a \sim b) \leq \Delta a \sim \Delta b$
  $(E\Delta 4)$ $\Delta (a \land b) = \Delta a \land \Delta b$
  $(E\Delta 5)$ $\Delta a = \Delta a \otimes \Delta a$
Special EQ-algebras

Let $\mathcal{E}$ be an EQ-algebra and $a, b, c, d \in E$. We say that $\mathcal{E}$ is:

1. **separated** if for all $a \in E$, $a \sim b = 1$ implies $a = b$.
2. **good** if $a \sim 1 = a$.
3. **residuated** if for all $a, b, c \in E$, $(a \otimes b) \land c = a \otimes b$ iff $a \land ((b \land c) \sim b) = a$.
4. **involutive** ($\mathcal{I}EQ$-algebra) if for all $a \in E$, $\neg\neg a = a$.
5. **prelinear** if for all $a, b \in E$, $\sup\{a \rightarrow b, b \rightarrow a\} = 1$.
6. **lattice EQ-algebra** ($\mathcal{L}EQ$-algebra) if it is a lattice and $((a \lor b) \sim c) \otimes (d \sim a) \leq (d \lor b) \sim c$.
7. **linear** if for all $a, b \in E$, $((a \land b) = a)$ or $((a \land b) = b)$.
EQ-algebras - former support tool

Manual algebras design with automated axioms check (complicated for larger EQ-algebras)
Basic principles

- Object oriented model of EQ-algebras as GA Population Members
- GA Population (Generation) as list of PMs
- Fitness function based on relative fulfilment of mandatory and optional axioms
- EQ-algebras fulfilling additional criteria called Winners
- Winners are stored during GA process
- Very important is detection of previously generated (identical) candidates (removal)
PM and operations data structures

OperationTriple = array[1..3] of char;
OperationCouple = array[1..2] of char;

PEQAlgebra = ^TEQAlgebra;
TEQAlgebra = class
public
  NEElements : integer;
  Elements : array[1..MaxNEElements] of char;
  NSemilatticeArguments: integer; {number of different tuples in semilattice}
  SemiLattice : array [1..MaxNArguments] of OperationTriple;
  {only specific triples (x, y, x o y) in a triangle without of the operation square are stored}
...

Population data structure

PEQPopulation = ^TEQPopulation;
TEQPopulation = class(TList)
public
    parent, child : PEQPopulation;
    ElementsNo : integer;
...
constructor Create();overload;
procedure GenerateRandom(populationsize, elementsize:integer);
procedure CrossOver(item1, item2:PEQAlgebra; target:PEQPopulation);
procedure Mutate(prob:real);
procedure RecomputeFit(win:PEQPopulation);
procedure RemoveEqual();
...

GA algorithm detailed

- Random (starting) population partially built to fulfil simple properties (e.g. infimum is commutative)
- Fitness evaluation - two phases:
  - Mandatory properties evaluation (e.g. boundedness axiom - \( a \otimes b \leq a \sim b \))
  - Optional properties evaluation (e.g. goodness - \( a \sim 1 = a \))
- String representing a candidate algebra
- Removal of same candidates (based on the string representation)
- Sort of PMs in population through fitness
- Termination condition:
  - Fixed number of steps performed
  - Fixed number of EQ-algebras with required properties
  - Manual (user) termination
EQCreator application

- Algorithms implemented in the form of PC application EQCreator
- GUI based application for MS Windows 32-bit platform
- Former EQAlgebras tool written in Object Pascal language
- Minor usage of code - for backward compactibility (enables to load and save older eqa format)
- Uses abstract types of Visual Component Library (TList)
- Main purpose:
  - Selection of various properties for candidate EQ-algebras
  - Evolution of algebras to attain EQ-algebras even with specific properties
  - Automated check of properties and generation
  - Saving of resulting optimal solutions in suitable form
EQCreator - basic functions

- **Fundamental settings**

  ![EQCreator interface]

  - Algebra elements number - support size (2 - 28)
  - Population limit - max. number of algebras in population
  - Generation steps - max. number of GA steps until one run stops (except stopped manually) (0 - unlimited)
  - Stop after certain number of EQ-algebras found
EQCreator - Genetic algorithms settings

- Children ratio (0 - 100%) - crossover resulting new members relative count (how large portion of new population to be new children, others are old members copied from previous generation)
- Cross ratio (0 - 100%) - portion of BEST members to have possibility to crossover (it is not crossover probability!)
- Mutation ratio (0 - 100%) - probability for new population member to be mutated
- Crossover probability is set arbitrary (fixed) - in descending ordered (by fitness) population of the size $N$ we set probability of member $i$ $p_i = \frac{N - i}{N*(N+1)}$ for $i = 0, ..., N - 1$, where $f(i) \geq f(i + 1)$ (fitness for members)
  e.g. for 5 members: $p_0 = \frac{5}{15}, p_1 = \frac{4}{15}, ..., p_4 = \frac{1}{15}$
EQCreator - Optional settings

- weight of optional properties - relative weight of special EQ-algebras requirements (e.g. linear EQA, involutive EQA) - should be significantly less than for compulsory axioms (experimental best - 15%)

- notion of colourfulness - required number of distinct elements in variable positions for operator function values (some combinations are determined e.g. $a \land 0 = 0$ in every EQA)

- **colourfulness** assures non-trivial EQ-algebras to be generated e.g. for fuzzy equality when 3 of 5 required - at least 3 different elements occur as functional values in non-determined cases

- colourfulness experimentally needed for Product ($\otimes$) and Fuzzy Equality ($\sim$) - higher means computationally harder!
EQCreator - Optional settings

- Extension of EQ-algebras - Max and Delta operators - some **additional axioms** must hold for these operators!
- Special EQ-algebras holding (or not holding) additional axioms as optional selection:
  
  Good       Commutative      Involutive      Residuated  
  Prelinear   Lattice         Semiseparated  Linear       
  Separated   etc.             

- Important setting - **removal of equal** EQ-algebras from population!
EQCreator - Population members or Winners (EQA)

Browsing

![Diagram of EQ-algebras]

- Infimum
- Delta
- Product
- Maximum
- Equality
- Implication

Hashim Habiballa, Vilém Novák, Martin DGenetic Algorithms for Finite EQ-algebras
Open old EQA format (user can use formerly created algebras)

Save both old EQA format of EQP - EQP is EQ-algebras (or general algebras) Population List

EQP is in contrast to EQA readable text file with operator tables

Number of GA steps could be limited
### EQCreator - EQP output

<table>
<thead>
<tr>
<th>0000: 1</th>
<th>^ 0 a b c 1</th>
<th>* 0 a b c 1</th>
<th>~ 0 a b c 1</th>
<th>m 0 a b c 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0 1 b b b b</td>
<td>0 0 a b c 1</td>
</tr>
<tr>
<td>a</td>
<td>a 0 a b a a</td>
<td>a 0 a b a a</td>
<td>a b 1 a 1 1</td>
<td>a a a a c 1</td>
</tr>
<tr>
<td>b</td>
<td>b 0 b b b b</td>
<td>b 0 b 0 b b</td>
<td>b b a 1 a a</td>
<td>b b a b c 1</td>
</tr>
<tr>
<td>c</td>
<td>c 0 a b c c</td>
<td>c 0 a b c c</td>
<td>c b 1 a 1 1</td>
<td>c c c c c 1</td>
</tr>
<tr>
<td>1</td>
<td>1 0 a b c 1</td>
<td>1 0 a b c 1</td>
<td>1 b 1 a 1 1</td>
<td>1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i 0 a b c 1</th>
<th>&lt; 0 a b c 1</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1 1 1</td>
<td>0 1 1 1 1 1</td>
<td>0 1</td>
</tr>
<tr>
<td>a b 1 a 1 1</td>
<td>a 0 1 0 1 1</td>
<td>a b</td>
</tr>
<tr>
<td>b b 1 1 1 1</td>
<td>b 0 1 1 1 1</td>
<td>b b</td>
</tr>
<tr>
<td>c b 1 a 1 1</td>
<td>c 0 0 0 1 1</td>
<td>c b</td>
</tr>
<tr>
<td>1 b 1 a 1 1</td>
<td>1 0 0 0 0 1</td>
<td>1 b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0001: 1</th>
<th>^ 0 a b c 1</th>
<th>* 0 a b c 1</th>
<th>~ 0 a b c 1</th>
<th>m 0 a b c 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0 1 b b b b</td>
<td>0 0 a b c 1</td>
</tr>
<tr>
<td>a</td>
<td>a 0 a b a a</td>
<td>a 0 a b a a</td>
<td>a b 1 a 1 1</td>
<td>a a a a c 1</td>
</tr>
<tr>
<td>b</td>
<td>b 0 b b b b</td>
<td>b 0 b 0 b b</td>
<td>b b a 1 a a</td>
<td>b b a b c 1</td>
</tr>
<tr>
<td>c</td>
<td>c 0 a b c c</td>
<td>c 0 a b c c</td>
<td>c b 1 a 1 1</td>
<td>c c c c c 1</td>
</tr>
<tr>
<td>1</td>
<td>1 0 a b c 1</td>
<td>1 0 a b c 1</td>
<td>1 b 1 a 1 1</td>
<td>1 1 1 1 1 1</td>
</tr>
</tbody>
</table>
Implementation

EQCreator - EQP output with axiom fulfilment info

0000: 1

\[
\begin{array}{c}
\wedge 0 \text{ abc1} \\
\wedge 0 \text{ abc1} \\
\sim 0 \text{ abc1} \\
\sim 0 \text{ abc1}
\end{array}
\]

\[
\begin{array}{c}
\wedge 0 \text{ abc1} \\
\sim 0 \text{ abc1} \\
\sim 0 \text{ abc1} \\
\sim 0 \text{ abc1}
\end{array}
\]

| 0 0 0 0 0 | 0 0 0 0 0 | 0 1 b b b b | 0 0 a b c 1 |
| a 0 a b a a | a 0 a b a a | a b 1 a 1 1 | a a a a a c 1 |
| b 0 b b b b | b 0 b 0 b b | b b a 1 a a | b b a b c 1 |
| c 0 a b c c | c 0 a b c c | c b 1 a 1 1 | c c c c c 1 |
| i 0 a b c 1 | i 0 a b c 1 | i 0 a b c 1 | i 0 b 1 a 1 1 |

| 0 1 1 1 1 1 | 0 1 1 1 1 1 | 0 1 |
| a 1 b 1 a 1 1 | a 0 1 0 1 1 | a b |
| b b 1 1 1 1 | b 0 1 1 1 1 | b b |
| c b 1 1 1 1 | c 0 0 1 1 1 | c b |
| 1 b 1 a 1 1 | 1 0 0 0 0 1 | 1 b |

Associative Infimum: 125/125 OK
Commutative Infimum: 25/25 OK
Neutral Infimum: 10/10 OK
Idempotent Infimum: 5/5 OK
Associative Product: 125/125 OK
Neutral Product: 10/10 OK
Isotone Product: 125/125 OK
Reflexive FEQuality: 5/5 OK
Congruence: 625/625 OK
Substitution: 625/625 OK
Monotone Implication: 125/125 OK
Boundedness: 25/25 OK
Colourfulness: 3/3 OK
Non-Goodness: 1/5, Errors: 4
Non-Involutive: 1/5, Errors: 4
Non-Semiseparated: 3/5, Errors: 2
Non-Separated: 19/25, Errors: 6
Non-Residuated: 114/125, Errors: 11
Commutative: 25/25 OK
Linear: 25/25 OK
Lattice: 625/625 OK
Prelinear: 25/25 OK
EQCreator - Compulsory and Optional Axioms real-time view

- Associative Infimum: 125/125 OK
- Commutative Infimum: 25/25 OK
- Neutral Infimum: 10/10 OK
- Idempotent Infimum: 5/5 OK
- Isotone product: 125/125 OK
- Associative product: 125/125 OK
- Neutral product: 10/10 OK
- Reflexive FEquality: 5/5 OK
- Substitution: 625/625 OK
- Congruence: 625/625 OK
- Monotone Implication: 125/125 OK
- Boundedness: 25/25 OK

- Colorfulness: 3/3 OK
- Colorfulness(*): 4/3 OK
- Non-Goodness: 1/5, Errors: 4
- Non-Involutive: 1/5, Errors: 4
- Non-Semiseparated: 3/5, Errors: 2
- Non-Separated: 19/25, Errors: 6
- Non-Residuated: 114/125, Errors: 11
- Commutative: 25/25 OK
- Linear: 25/25 OK
- Lattice: 625/625 OK
- Prelinear: 25/25 OK
- Sup-product distributivity: 125/125 OK
- Non-Equality over ProdEquality: 6/125, Errors: 119

Fitness: 100%
EQCreator - time efficiency

Tested on Pentium 4 - 2.8 GHz. In contrast to state space searching significant difference (no superexponentiality)
Conclusions

- Genetic algorithms made the task solvable in sensible time
- Specific GA properties are required:
  - Elitism must be used at least of minimal level (5% was acceptable - of course higher usage leads to worse convergence)
  - High mutation ratio must be set in contrast with traditional use of GA (best results with 20 - 30%)
  - Optional axioms and requirements need to have significantly less weight (experimentally 15% has best results)
  - Optional properties negatively affect convergence
  - Colourfulness was defined to prevent trivial solutions (evolution tends to most simple way of achieving results)
- EQ Creator - software for EQ-algebras only, but we suppose to bring fully general generator for algebras
Thank you for attention.