Identifying CAs with evolutionary algorithms

Witold Bołt[1,2] Jan M. Baetens[1] Bernard De Baets[1]

[1]KERMIT, Department of Mathematical Modelling, Statistics and Bioinformatics, Ghent University, Ghent, Belgium

[2]Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland

AUTOMATA 2013 (17.09.2013)

Agenda

- Notation and definition of the problem
- Evolutionary algorithm layout
- Experimental results
- Future research
- Questions

Agenda

- Notation and definition of the problem
- Evolutionary algorithm layout
- Experimental results
- Future research
- Questions

Agenda

- Notation and definition of the problem
- Evolutionary algorithm layout
- Experimental results
- Future research
- Questions

Agenda

- Notation and definition of the problem
- Evolutionary algorithm layout
- Experimental results
- Future research
- Questions

Agenda

- Notation and definition of the problem
- Evolutionary algorithm layout
- Experimental results
- Future research
- Questions

Basic assumptions

- We start with a **binary image**.
- We consider 1D, deterministic, two–state **CA**s with symmetric neighborhood (of given radius) and periodic boundary conditions.
- We search for a CA that can **reproduce** the image (space–time diagram of this CA needs to match the given image).

Basic assumptions

- We start with a **binary image**.
- We consider 1D, deterministic, two–state **CA**s with symmetric neighborhood (of given radius) and periodic boundary conditions.
- We search for a CA that can **reproduce** the image (space–time diagram of this CA needs to match the given image).

Basic assumptions

- We start with a **binary image**.
- We consider 1D, deterministic, two–state **CA**s with symmetric neighborhood (of given radius) and periodic boundary conditions.
- We search for a CA that can **reproduce** the image (space–time diagram of this CA needs to match the given image).

Notation: images

- **Image:** $I = (I_{t,s})$ for $t \in \mathcal{T}$, $s \in \mathcal{S} = \{0, 1, \ldots, S-1\}$, $I_{t,s} \in \{0, 1\}$.
- The time domain $\mathcal{T}$ is a subset of $\{0, 1, \ldots, T\}$ that includes 0.
- **Time step:** $I_t := (I_{t,s})_{s \in \mathcal{S}}$ – the t-th row of an image.
- **Initial condition:** I_0 represents the initial condition for CA.

Notation: images

- **Image:** $I = (I_{t,s})$ for $t \in \mathcal{T}$, $s \in \mathcal{S} = \{0, 1, \ldots, S-1\}$, $I_{t,s} \in \{0, 1\}$.
- The time domain $\mathcal{T}$ is a subset of $\{0, 1, \ldots, T\}$ that includes 0.
- **Time step:** $I_t := (I_{t,s})_{s \in \mathcal{S}}$ – the t-th row of an image.
- **Initial condition:** I_0 represents the initial condition for CA.

Notation: images

- **Image:** $I = (I_{t,s})$ for $t \in \mathcal{T}$, $s \in \mathcal{S} = \{0, 1, \ldots, S-1\}$, $I_{t,s} \in \{0, 1\}$.
- The time domain $\mathcal{T}$ is a subset of $\{0, 1, \ldots, T\}$ that includes 0.
- **Time step:** $I_t := (I_{t,s})_{s \in \mathcal{S}}$ – the t-th row of an image.
- **Initial condition:** I_0 represents the initial condition for CA.

Notation: images

- **Image:** $I = (I_{t,s})$ for $t \in \mathcal{T}$, $s \in \mathcal{S} = \{0, 1, \ldots, S-1\}$, $I_{t,s} \in \{0, 1\}$.
- The time domain $\mathcal{T}$ is a subset of $\{0, 1, \ldots, T\}$ that includes 0.
- **Time step:** $I_t := (I_{t,s})_{s \in \mathcal{S}}$ – the t-th row of an image.
- **Initial condition:** I_0 represents the initial condition for CA.

Notation: problem formulation

- Let $\mathscr{A}$ be a CA. By A we will denote its global rule (global map). Let I be a binary image, as defined on previous slide.
- We define an error of reproduction of I by A as:

$$E(A, I) := \sum_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} \left| I_{t,s} - A^t\left(I_0\right)[s] \right|.$$

- The error defined here is simply a count of cells which state in the image is different than the one evolved by the automaton.
- We look for A such that $E(I, A) = 0$ (or as small as possible).

Notation: problem formulation

- Let $\mathscr{A}$ be a CA. By A we will denote its global rule (global map). Let I be a binary image, as defined on previous slide.
- We define an error of reproduction of I by A as:

$$E(A, I) := \sum_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} \left| I_{t,s} - A^t \left(I_0 \right) [s] \right| .$$

- The error defined here is simply a count of cells which state in the image is different than the one evolved by the automaton.
- We look for A such that $E(I, A) = 0$ (or as small as possible).

Notation: problem formulation

- Let $\mathscr{A}$ be a CA. By A we will denote its global rule (global map). Let I be a binary image, as defined on previous slide.
- We define an error of reproduction of I by A as:

$$E(A, I) := \sum_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} \left| I_{t,s} - A^t \left(I_0 \right) [s] \right| .$$

- The error defined here is simply a count of cells which state in the image is different than the one evolved by the automaton.
- We look for A such that $E(I, A) = 0$ (or as small as possible).

Notation: problem formulation

- Let $\mathscr{A}$ be a CA. By A we will denote its global rule (global map). Let I be a binary image, as defined on previous slide.
- We define an error of reproduction of I by A as:

$$E(A, I) := \sum_{t \in \mathcal{T}} \sum_{s \in \mathcal{S}} \left| I_{t,s} - A^t \left(I_0 \right) [s] \right| .$$

- The error defined here is simply a count of cells which state in the image is different than the one evolved by the automaton.
- We look for A such that $E(I, A) = 0$ (or as small as possible).

Evolutionary algorithm: motivation

- Direct identification methods exist and work in the case of “complete” images.
- More flexibility is need, to solve the generalized problem:
  - Incomplete image: missing time steps or individual cell values.
  - Noise in the image: noisy observation or noisy process.
  - Unknown neighborhood structure (partially covered in the direct methods).
- Note that in this presentation we use the evolutionary algorithm to solve a “simple” problem where there is no noise, no missing frames and the neighborhood is known. This is the first step towards solving the general problem.

Evolutionary algorithm: motivation

- Direct identification methods exist and work in the case of "complete" images.
- More flexibility is need, to solve the generalized problem:
  - Incomplete image: missing time steps or individual cell values.
  - Noise in the image: noisy observation or noisy process.
  - Unknown neighborhood structure (partially covered in the direct methods).
- Note that in this presentation we use the evolutionary algorithm to solve a "simple" problem where there is no noise, no missing frames and the neighborhood is known. This is the first step towards solving the general problem.

Evolutionary algorithm: motivation

- Direct identification methods exist and work in the case of "complete" images.
- More flexibility is need, to solve the generalized problem:
  - Incomplete image: missing time steps or individual cell values.
  - Noise in the image: noisy observation or noisy process.
  - Unknown neighborhood structure (partially covered in the direct methods).
- Note that in this presentation we use the evolutionary algorithm to solve a "simple" problem where there is no noise, no missing frames and the neighborhood is known. This is the first step towards solving the general problem.

Evolutionary algorithm: motivation

- Direct identification methods exist and work in the case of "complete" images.
- More flexibility is need, to solve the generalized problem:
  - Incomplete image: missing time steps or individual cell values.
  - Noise in the image: noisy observation or noisy process.
  - Unknown neighborhood structure (partially covered in the direct methods).
- Note that in this presentation we use the evolutionary algorithm to solve a "simple" problem where there is no noise, no missing frames and the neighborhood is known. This is the first step towards solving the general problem.

Evolutionary algorithm: motivation

- Direct identification methods exist and work in the case of "complete" images.
- More flexibility is need, to solve the generalized problem:
  - Incomplete image: missing time steps or individual cell values.
  - Noise in the image: noisy observation or noisy process.
  - Unknown neighborhood structure (partially covered in the direct methods).
- Note that in this presentation we use the evolutionary algorithm to solve a "simple" problem where there is no noise, no missing frames and the neighborhood is known. This is the first step towards solving the general problem.

Evolutionary algorithm: motivation

- Direct identification methods exist and work in the case of "complete" images.
- More flexibility is need, to solve the generalized problem:
  - Incomplete image: missing time steps or individual cell values.
  - Noise in the image: noisy observation or noisy process.
  - Unknown neighborhood structure (partially covered in the direct methods).
- Note that in this presentation we use the evolutionary algorithm to solve a "simple" problem where there is no noise, no missing frames and the neighborhood is known. This is the first step towards solving the general problem.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (I)

- We use a standard **genetic algorithm** with populations build with CAs encoded by LUTs. Population size is denoted with C.
- Genetic operators:
  - **Selection** and reproduction: random selection with probability weighted with fitness.
  - **Crossover**: random crossover (each LUT entry randomly selected from one of two parents) – mixing probability p_c.
  - **Mutation**: flipping of exactly one, randomly selected bit in the LUT – applied with probability p_m.
- **Elitism** – we consider two algorithm variants:
  - Algorithm E with elite survival, with elite size of C_E individuals,
  - Algorithm NE without elitism.

Evolutionary algorithm: definition (II)

- Fitness function should **be related** to error E defined earlier.
- Using E directly leads to **poor results**!
  - Errors propagate in time
  - We count the same errors multiple times
- **Fitness** used in our algorithms:

$$\mathrm{fit}(A) := 1 - \frac{1}{T\,S} \sum_{t \in \mathcal{T} \setminus \{0\}} \sum_{s \in \mathcal{S}} |I_{t,s} - A(I_{t-1})[s]| \tag{1}$$

- Instead of global error measure (based on E), we use a **pair–wise comparison** of the time frames, *i.e.* we verify if the rule "works" on pairs of time frames.
- Note: for technical simplicity we normalize fitness (the maximum value is one and represents a perfect match) and consider **fitness maximization** problem.

Evolutionary algorithm: definition (II)

- Fitness function should **be related** to error E defined earlier.
- Using E directly leads to **poor results**!
  - Errors propagate in time
  - We count the same errors multiple times
- **Fitness** used in our algorithms:

$$\mathrm{fit}(A) := 1 - \frac{1}{T\,S} \sum_{t \in \mathcal{T} \setminus \{0\}} \sum_{s \in \mathcal{S}} |I_{t,s} - A(I_{t-1})[s]| \qquad (1)$$

- Instead of global error measure (based on E), we use a **pair–wise comparison** of the time frames, *i.e.* we verify if the rule "works" on pairs of time frames.
- Note: for technical simplicity we normalize fitness (the maximum value is one and represents a perfect match) and consider **fitness maximization** problem.

Evolutionary algorithm: definition (II)

- Fitness function should **be related** to error E defined earlier.
- Using E directly leads to **poor results**!
  - Errors propagate in time
  - We count the same errors multiple times
- **Fitness** used in our algorithms:

$$\mathrm{fit}(A) := 1 - \frac{1}{T\,S} \sum_{t \in \mathcal{T} \setminus \{0\}} \sum_{s \in \mathcal{S}} |I_{t,s} - A(I_{t-1})[s]| \tag{1}$$

- Instead of global error measure (based on E), we use a **pair–wise comparison** of the time frames, *i.e.* we verify if the rule "works" on pairs of time frames.
- Note: for technical simplicity we normalize fitness (the maximum value is one and represents a perfect match) and consider **fitness maximization** problem.

Evolutionary algorithm: definition (II)

- Fitness function should **be related** to error E defined earlier.
- Using E directly leads to **poor results**!
  - Errors propagate in time
  - We count the same errors multiple times
- **Fitness** used in our algorithms:

$$\mathrm{fit}(A) := 1 - \frac{1}{T\,S} \sum_{t \in \mathcal{T} \setminus \{0\}} \sum_{s \in \mathcal{S}} |I_{t,s} - A(I_{t-1})[s]| \tag{1}$$

- Instead of global error measure (based on E), we use a **pair–wise comparison** of the time frames, *i.e.* we verify if the rule "works" on pairs of time frames.
- Note: for technical simplicity we normalize fitness (the maximum value is one and represents a perfect match) and consider **fitness maximization** problem.

Evolutionary algorithm: definition (II)

- Fitness function should **be related** to error E defined earlier.
- Using E directly leads to **poor results**!
  - Errors propagate in time
  - We count the same errors multiple times
- **Fitness** used in our algorithms:

$$\mathrm{fit}(A) := 1 - \frac{1}{T\,S} \sum_{t \in \mathcal{T} \setminus \{0\}} \sum_{s \in \mathcal{S}} |I_{t,s} - A(I_{t-1})[s]| \tag{1}$$

- Instead of global error measure (based on E), we use a **pair–wise comparison** of the time frames, *i.e.* we verify if the rule "works" on pairs of time frames.
- Note: for technical simplicity we normalize fitness (the maximum value is one and represents a perfect match) and consider **fitness maximization** problem.

Evolutionary algorithm: definition (II)

- Fitness function should **be related** to error E defined earlier.
- Using E directly leads to **poor results**!
  - Errors propagate in time
  - We count the same errors multiple times
- **Fitness** used in our algorithms:

$$\text{fit}(A) := 1 - \frac{1}{T\,S} \sum_{t \in \mathcal{T} \setminus \{0\}} \sum_{s \in \mathcal{S}} |I_{t,s} - A(I_{t-1})[s]| \tag{1}$$

- Instead of global error measure (based on E), we use a **pair–wise comparison** of the time frames, *i.e.* we verify if the rule "works" on pairs of time frames.
- Note: for technical simplicity we normalize fitness (the maximum value is one and represents a perfect match) and consider **fitness maximization** problem.

Evolutionary algorithm: definition (II)

- Fitness function should **be related** to error E defined earlier.
- Using E directly leads to **poor results**!
  - Errors propagate in time
  - We count the same errors multiple times
- **Fitness** used in our algorithms:

$$\mathrm{fit}(A) := 1 - \frac{1}{T\,S} \sum_{t \in \mathcal{T} \setminus \{0\}} \sum_{s \in \mathcal{S}} |I_{t,s} - A(I_{t-1})[s]| \tag{1}$$

- Instead of global error measure (based on E), we use a **pair–wise comparison** of the time frames, *i.e.* we verify if the rule "works" on pairs of time frames.
- Note: for technical simplicity we normalize fitness (the maximum value is one and represents a perfect match) and consider **fitness maximization** problem.

Experiment 1: Rule 154

Algorithm NE with parameters: $p_c = 0.5$, $p_m = 0.02$, $C = 100$, $T = 100$, $S = 100$. We look for radius–2 CA rules that match rule 154.

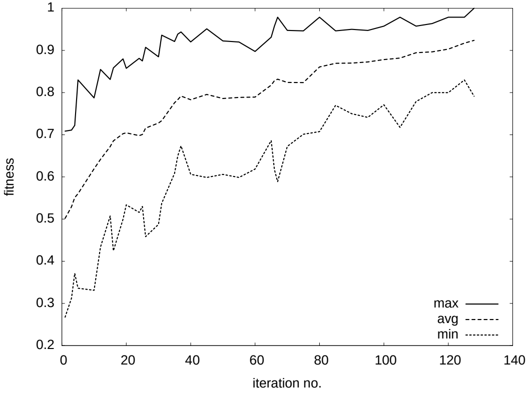


Figure: Evolution of the maximum, average and minimum fitness.

Experiment 2: ECA discoverability

We measure rule discoverability (disc.) which is expressed as the percentage of test runs resulting in finding a perfect solution (30 runs for each rule were used, algorithm is allowed to run for 5000 iterations). Out of 256 ECA rules, 88 rules have zero discoverability and 42 rules have full (100%) discoverability.

rule	avg. iter.	disc.
75	49.57	100%
30	53.10	100%
169	53.63	100%
165	53.80	100%
106	54.77	100%

(a)

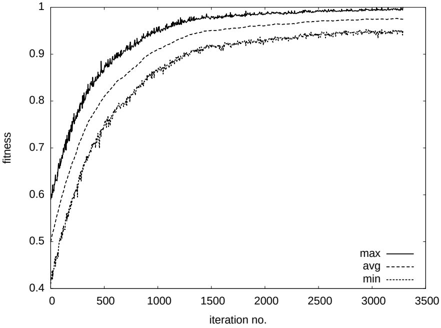
rule	avg. iter.	disc.
245	4997.23	3.33%
98	4993.47	3.33%
188	4987.80	3.33%
6	4982.07	3.33%
43	4981.33	3.33%

(b)

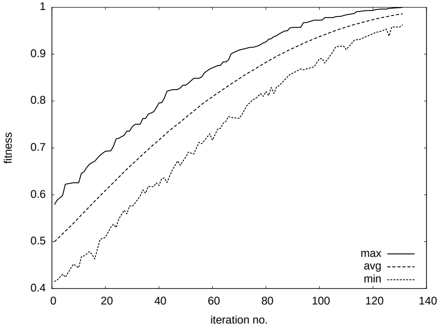
Table: Rule discoverability and average number of iterations for those ECA that give rise to the five lowest (a) and highest (b) numbers of iterations.

Experiment 3: Algorithm E vs. Algorithm NE

Algorithms with parameters: $p_c = 0.5$, $p_m = 0.02$, $C = 5000$, $C_E = 500$, $T = 200$, $S = 200$. We look for a radius–4 CA.



(a) Algorithm NE



(b) Algorithm E

Figure: Evolution of maximum, average and minimum fitness.

Extended fitness function

- To solve the general case with missing time steps, we need to extend the fitness function.
- Let $\mathcal{T} = \{t_i \mid i = 0, 1, \ldots, |\mathcal{T}|\}$ and assume that $t_{max} > 0$ is the length of maximum time gap (for any i we assume $t_{i+1} - t_i \leq t_{max}$).
- For $t_i \in \{0, \ldots, |\mathcal{T}| - 1\}, j \in \{1, 2, \ldots, t_{max}\}$ we define the time gap error as:

$$E_{i,j}(A, I) := \sum_{s \in \mathcal{S}} \left| I_{t_{i+1},s} - A^j(I_{t_i})[s] \right|,$$

  and with that we define a generalized pair–wise error:

$$E_i(A, I) := \min_j E_{i,j}(A, I).$$

- The generalized fitness is defined as:

$$\widehat{\mathrm{fit}}(A, I) := 1 - \frac{1}{(|\mathcal{T}| - 1)\, S} \sum_i E_i(A, I).$$

Extended fitness function

- To solve the general case with missing time steps, we need to extend the fitness function.
- Let $\mathcal{T} = \{t_i \mid i = 0, 1, \ldots, |\mathcal{T}|\}$ and assume that $t_{max} > 0$ is the length of maximum time gap (for any i we assume $t_{i+1} - t_i \leq t_{max}$).
- For $t_i \in \{0, \ldots, |\mathcal{T}| - 1\}, j \in \{1, 2, \ldots, t_{max}\}$ we define the time gap error as:

$$E_{i,j}(A, I) := \sum_{s \in \mathcal{S}} \left| I_{t_{i+1},s} - A^j(I_{t_i})[s] \right|,$$

and with that we define a generalized pair–wise error:

$$E_i(A, I) := \min_j E_{i,j}(A, I).$$

- The generalized fitness is defined as:

$$\widehat{\text{fit}}(A, I) := 1 - \frac{1}{(|\mathcal{T}| - 1)\, S} \sum_i E_i(A, I).$$

Extended fitness function

- To solve the general case with missing time steps, we need to extend the fitness function.
- Let $\mathcal{T} = \{t_i \mid i = 0, 1, \ldots, |\mathcal{T}|\}$ and assume that $t_{max} > 0$ is the length of maximum time gap (for any i we assume $t_{i+1} - t_i \leq t_{max}$).
- For $t_i \in \{0, \ldots, |\mathcal{T}| - 1\}, j \in \{1, 2, \ldots, t_{max}\}$ we define the time gap error as:

$$E_{i,j}(A, I) := \sum_{s \in \mathcal{S}} \left| I_{t_{i+1},s} - A^j(I_{t_i})[s] \right|,$$

  and with that we define a generalized pair–wise error:

$$E_i(A, I) := \min_j E_{i,j}(A, I).$$

- The generalized fitness is defined as:

$$\widehat{\text{fit}}(A, I) := 1 - \frac{1}{(|\mathcal{T}| - 1)\, S} \sum_i E_i(A, I).$$

Extended fitness function

- To solve the general case with missing time steps, we need to extend the fitness function.
- Let $\mathcal{T} = \{t_i \mid i = 0, 1, \ldots, |\mathcal{T}|\}$ and assume that $t_{max} > 0$ is the length of maximum time gap (for any i we assume $t_{i+1} - t_i \leq t_{max}$).
- For $t_i \in \{0, \ldots, |\mathcal{T}| - 1\}, j \in \{1, 2, \ldots, t_{max}\}$ we define the time gap error as:

$$E_{i,j}(A, I) := \sum_{s \in \mathcal{S}} \left| I_{t_{i+1},s} - A^j(I_{t_i})[s] \right|,$$

  and with that we define a generalized pair–wise error:

$$E_i(A, I) := \min_j E_{i,j}(A, I).$$

- The generalized fitness is defined as:

$$\widehat{\text{fit}}(A, I) := 1 - \frac{1}{(|\mathcal{T}| - 1)\, S} \sum_i E_i(A, I).$$

Future research

- Using $\widehat{\text{fit}}$ in GA described earlier is possible, but algorithm performance (and convergence) differs greatly depending on the CA rule and time gaps.
- Current research concentrates on more flexible rule representation, rule decomposition and evolutionary algorithm improvements, so that $\widehat{\text{fit}}$ could be fully utilized.
- Relationship between rule "discoverability" (measure of algorithm performance) and dynamic complexity is being evaluated.

Future research

- Using $\widehat{\text{fit}}$ in GA described earlier is possible, but algorithm performance (and convergence) differs greatly depending on the CA rule and time gaps.
- Current research concentrates on more flexible rule representation, rule decomposition and evolutionary algorithm improvements, so that $\widehat{\text{fit}}$ could be fully utilized.
- Relationship between rule "discoverability" (measure of algorithm performance) and dynamic complexity is being evaluated.

Future research

- Using $\widehat{\text{fit}}$ in GA described earlier is possible, but algorithm performance (and convergence) differs greatly depending on the CA rule and time gaps.
- Current research concentrates on more flexible rule representation, rule decomposition and evolutionary algorithm improvements, so that $\widehat{\text{fit}}$ could be fully utilized.
- Relationship between rule "discoverability" (measure of algorithm performance) and dynamic complexity is being evaluated.

Summary

- We shown that the method works in simple cases and that performance is reasonable.
- The method is flexible (can be extended to more general cases).
- We summarized the goals for future research in the project.

Summary

- We shown that the method works in simple cases and that performance is reasonable.
- The method is flexible (can be extended to more general cases).
- We summarized the goals for future research in the project.

Summary

- We shown that the method works in simple cases and that performance is reasonable.
- The method is flexible (can be extended to more general cases).
- We summarized the goals for future research in the project.

Questions

Questions?

Questions

Questions?

Witold Bołt
e-mail: witold.bolt@hope.art.pl

Thank you!