P Systems inference via Grammatical Evolution

Giorgia Nadizar and Gloria Pietropolli

Department of Mathematics and Geosciences, University of Trieste, Italy

CMC 2022, Trieste (Italy)

Motivation

- *P systems* can be exploited to simulate theoretical and application-focused investigations
- But their design is non-trivial, time-consuming, and requires precision and expertise
- Would it be easier to specify their behavior and automatically infer the rules from that? Certainly!

Contribution

We propose a Grammatical Evolution based approach to infer the rules of a P system from observations of its steps.

Motivation

- *P systems* can be exploited to simulate theoretical and application-focused investigations
- But their design is non-trivial, time-consuming, and requires precision and expertise
- Would it be easier to specify their behavior and automatically infer the rules from that? Certainly!

Contribution

We propose a Grammatical Evolution based approach to infer the rules of a P system from observations of its steps.

Motivation

- *P systems* can be exploited to simulate theoretical and application-focused investigations
- But their design is non-trivial, time-consuming, and requires precision and expertise
- Would it be easier to specify their behavior and automatically infer the rules from that? Certainly!

Contribution

We propose a Grammatical Evolution based approach to infer the rules of a P system from observations of its steps.

Motivation

- *P systems* can be exploited to simulate theoretical and application-focused investigations
- But their design is non-trivial, time-consuming, and requires precision and expertise
- Would it be easier to specify their behavior and automatically infer the rules from that? Certainly!

Contribution

We propose a Grammatical Evolution based approach to infer the rules of a P system from observations of its steps.

Motivation

- *P systems* can be exploited to simulate theoretical and application-focused investigations
- But their design is non-trivial, time-consuming, and requires precision and expertise
- Would it be easier to specify their behavior and automatically infer the rules from that? Certainly!

Contribution

We propose a Grammatical Evolution based approach to infer the rules of a P system from observations of its steps.

Outline

Outline

P Systems: Definition

Definition

A P system with active membranes and cooperative rules, of initial degree $d \geq 1$, is a tuple

$$\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$$

where

- Γ is an alphabet
- Λ is a finite set of labels
- μ is a membrane structure (represented as a rooted unordered tree) consisting of d membranes labeled by elements of Λ
- $w_{h_1}, \ldots, w_{h_d}$, with $h_1, \ldots, h_d \in \Lambda$, are multisets describing the initial contents of each of the d regions of μ
- R is a finite set of rules.

P Systems: Definition

Definition

A P system with active membranes and cooperative rules, of initial degree $d \geq 1$, is a tuple

$$\Pi = (\Gamma, \Lambda, \mu, w_{h_1}, \ldots, w_{h_d}, R)$$

where

- Γ is an alphabet
- Λ is a finite set of labels
- μ is a membrane structure (represented as a rooted unordered tree) consisting of d membranes labeled by elements of Λ
- $w_{h_1}, \ldots, w_{h_d}$, with $h_1, \ldots, h_d \in \Lambda$, are multisets describing the initial contents of each of the d regions of μ
- R is a finite set of rules.

P Systems: Rules

- Cooperative **rewriting** rules:

$$[\, u \to v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-in** rules:

$$u\, [\,]_h \to [\, v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-out** rules:

$$[\, u \,]_h \to v\, [\,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative **weak division** rules:

$$[\, u \,]_h \to [\, v \,]_h [\, w \,]_h \text{ for } h \in \Lambda \text{ and } u, v, w \in \Gamma^\star$$

P Systems: Rules

- Cooperative **rewriting** rules:

$$[\, u \to v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-in** rules:

$$u\, [\,]_h \to [\, v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-out** rules:

$$[\, u \,]_h \to v\, [\,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative **weak division** rules:

$$[\, u \,]_h \to [\, v \,]_h [\, w \,]_h \text{ for } h \in \Lambda \text{ and } u, v, w \in \Gamma^\star$$

P Systems: Rules

- Cooperative **rewriting** rules:

$$[\, u \to v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-in** rules:

$$u\, [\ ]_h \to [\, v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-out** rules:

$$[\, u \,]_h \to v\, [\ ]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative **weak division** rules:

$$[\, u \,]_h \to [\, v \,]_h [\, w \,]_h \text{ for } h \in \Lambda \text{ and } u, v, w \in \Gamma^\star$$

P Systems: Rules

- Cooperative **rewriting** rules:

$$[\, u \to v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-in** rules:

$$u\, [\ ]_h \to [\, v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-out** rules:

$$[\, u \,]_h \to v\, [\ ]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative **weak division** rules:

$$[\, u \,]_h \to [\, v \,]_h [\, w \,]_h \text{ for } h \in \Lambda \text{ and } u, v, w \in \Gamma^\star$$

P Systems: Rules

- Cooperative **rewriting** rules:

$$[\, u \to v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-in** rules:

$$u\, [\ ]_h \to [\, v \,]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative communication **send-out** rules:

$$[\, u \,]_h \to v\, [\ ]_h \text{ for } h \in \Lambda \text{ and } u, v \in \Gamma^\star$$

- Cooperative **weak division** rules:

$$[\, u \,]_h \to [\, v \,]_h [\, w \,]_h \text{ for } h \in \Lambda \text{ and } u, v, w \in \Gamma^\star$$

Outline

Evolutionary Algorithms (EAs)

- Draw inspiration from *biological evolution* to solve (optimization) problems
- Distinction between inner representation (*genotype*) and actual solution (*phenotype*)
- *Mapping* function to move from genotype to phenotype
- *Fitness* measure (computed on the phenotype) to quantify the performance of a solution

Evolutionary Algorithms (EAs)

- Draw inspiration from *biological evolution* to solve (optimization) problems
- Distinction between inner representation (*genotype*) and actual solution (*phenotype*)
- *Mapping* function to move from genotype to phenotype
- *Fitness* measure (computed on the phenotype) to quantify the performance of a solution

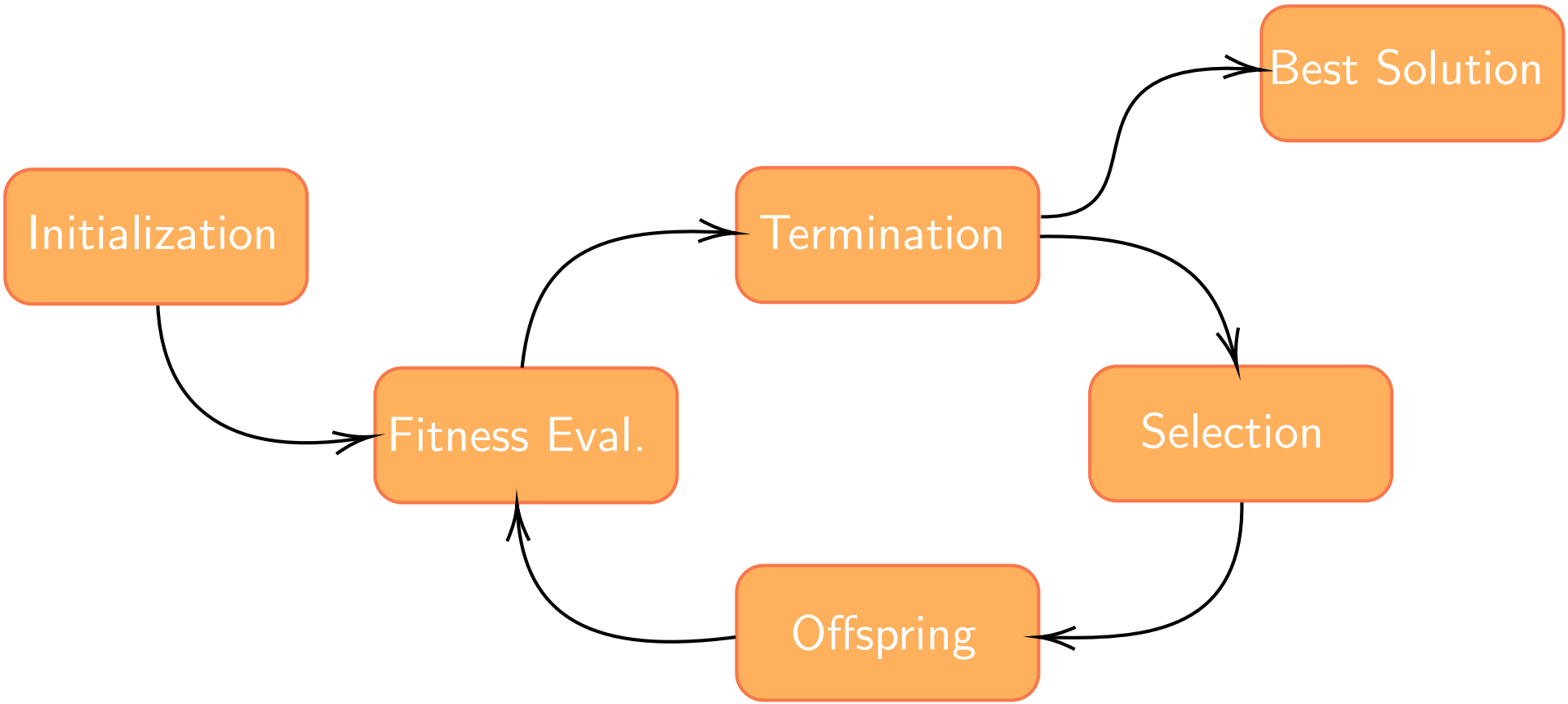
Evolutionary Algorithms (EAs)

- Draw inspiration from *biological evolution* to solve (optimization) problems
- Distinction between inner representation (*genotype*) and actual solution (*phenotype*)
- *Mapping* function to move from genotype to phenotype
- *Fitness* measure (computed on the phenotype) to quantify the performance of a solution

Evolutionary Algorithms (EAs)

- Draw inspiration from *biological evolution* to solve (optimization) problems
- Distinction between inner representation (*genotype*) and actual solution (*phenotype*)
- *Mapping* function to move from genotype to phenotype
- *Fitness* measure (computed on the phenotype) to quantify the performance of a solution

Evolutionary Cycle



Grammatical Evolution (GE)

Definition

An EA that can evolve strings (programs) of any language defined by a Context-Free Grammar (CFG).

- Genotype → bit string (or integer string, considering 8 bits for conversion)
- Phenotype → string of the language
- Mapping → use of the modulus operator (%)
- Fitness → depends on the problem

Grammatical Evolution (GE)

Definition

An EA that can evolve strings (programs) of any language defined by a Context-Free Grammar (CFG).

- Genotype $\rightarrow$ bit string (or integer string, considering 8 bits for conversion)
- Phenotype $\rightarrow$ string of the language
- Mapping $\rightarrow$ use of the modulus operator (%)
- Fitness $\rightarrow$ depends on the problem

Grammatical Evolution (GE)

Definition

An EA that can evolve strings (programs) of any language defined by a Context-Free Grammar (CFG).

- Genotype $\rightarrow$ bit string (or integer string, considering 8 bits for conversion)
- Phenotype $\rightarrow$ string of the language
- Mapping $\rightarrow$ use of the modulus operator (%)
- Fitness $\rightarrow$ depends on the problem

Grammatical Evolution (GE)

Definition

An EA that can evolve strings (programs) of any language defined by a Context-Free Grammar (CFG).

- Genotype $\rightarrow$ bit string (or integer string, considering 8 bits for conversion)
- Phenotype $\rightarrow$ string of the language
- Mapping $\rightarrow$ use of the modulus operator (%)
- Fitness $\rightarrow$ depends on the problem

Grammatical Evolution (GE)

Definition

An EA that can evolve strings (programs) of any language defined by a Context-Free Grammar (CFG).

- Genotype $\rightarrow$ bit string (or integer string, considering 8 bits for conversion)
- Phenotype $\rightarrow$ string of the language
- Mapping $\rightarrow$ use of the modulus operator (%)
- Fitness $\rightarrow$ depends on the problem

Context-Free Grammar (CFG)

Definition

A CFG is defined as a tuple $\mathcal{G} = (N, T, s, P)$, where N is the set of *non-terminal* symbols, T is the set of *terminal* symbols (with $N \cap T = \emptyset$), $s \in N$ is the starting symbol, and P is the set of *production rules*.

Example: a CFG for mathematical expressions.

$$\langle \text{expr} \rangle ::= (\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle) \mid \langle \text{num} \rangle \mid \langle \text{var} \rangle$$
$$\langle \text{op} \rangle ::= + \mid - \mid * \mid /$$
$$\langle \text{var} \rangle ::= x \mid y$$
$$\langle \text{num} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Context-Free Grammar (CFG)

Definition

A CFG is defined as a tuple $\mathcal{G} = (N, T, s, P)$, where N is the set of *non-terminal* symbols, T is the set of *terminal* symbols (with $N \cap T = \emptyset$), $s \in N$ is the starting symbol, and P is the set of *production rules*.

Example: a CFG for mathematical expressions.

$$\langle \text{expr} \rangle ::= (\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle) \mid \langle \text{num} \rangle \mid \langle \text{var} \rangle$$
$$\langle \text{op} \rangle ::= + \mid - \mid * \mid /$$
$$\langle \text{var} \rangle ::= x \mid y$$
$$\langle \text{num} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Genotype-phenotype mapping

Production rules P are the core of the mapping: after selecting s as starting symbol for the phenotype, the mapping proceeds using them to replace non-terminal symbols with other symbols until the string is only composed of terminal symbols.

1. The next unused value g_i of the genotype is taken (and marked as used)
2. The value is divided by the number of possible replacement options $|R_s|$
3. The remainder of the division j is used to select the new symbol (e.g., 0 corresponding to the first substitute, and so on)

Genotype-phenotype mapping

Production rules P are the core of the mapping: after selecting s as starting symbol for the phenotype, the mapping proceeds using them to replace non-terminal symbols with other symbols until the string is only composed of terminal symbols.

1. The next unused value g_i of the genotype is taken (and marked as used)
2. The value is divided by the number of possible replacement options $|R_s|$
3. The remainder of the division j is used to select the new symbol (e.g., 0 corresponding to the first substitute, and so on)

Genotype-phenotype mapping

Production rules P are the core of the mapping: after selecting s as starting symbol for the phenotype, the mapping proceeds using them to replace non-terminal symbols with other symbols until the string is only composed of terminal symbols.

1. The next unused value g_i of the genotype is taken (and marked as used)
2. The value is divided by the number of possible replacement options $|R_s|$
3. The remainder of the division j is used to select the new symbol (e.g., 0 corresponding to the first substitute, and so on)

Genotype-phenotype mapping

Production rules P are the core of the mapping: after selecting s as starting symbol for the phenotype, the mapping proceeds using them to replace non-terminal symbols with other symbols until the string is only composed of terminal symbols.

1. The next unused value g_i of the genotype is taken (and marked as used)
2. The value is divided by the number of possible replacement options $|R_s|$
3. The remainder of the division j is used to select the new symbol (e.g., 0 corresponding to the first substitute, and so on)

Mapping example

$$g = 231\ 15\ 133\ 142\ 178\ 224$$

i	g_i	$\lvert R_s \rvert$	j	w	Phenotype p
0	231	3	0	0	⟨**expr**⟩
1	15	3	0	0	(⟨**expr**⟩ ⟨op⟩ ⟨expr⟩)
2	133	3	1	0	((⟨**expr**⟩ ⟨op⟩ ⟨expr⟩) ⟨op⟩ ⟨expr⟩)
3	142	10	2	0	((⟨**num**⟩ ⟨op⟩ ⟨expr⟩) ⟨op⟩ ⟨expr⟩)
4	178	4	2	0	((2 ⟨**op**⟩ ⟨expr⟩) ⟨op⟩ ⟨expr⟩)
5	224	3	2	0	((2 * ⟨**expr**⟩) ⟨op⟩ ⟨expr⟩)
0	231	2	1	1	((2 * ⟨**var**⟩) ⟨op⟩ ⟨expr⟩)
1	15	4	3	1	((2 * y) ⟨**op**⟩ ⟨expr⟩)
2	133	3	1	1	((2 * y) / ⟨**expr**⟩)
3	142	10	2	1	((2 * y) / ⟨**num**⟩)
					((2 * y) / 2)

⟨expr⟩ ::= (⟨expr⟩ ⟨op⟩ ⟨expr⟩) | ⟨num⟩ | ⟨var⟩
⟨op⟩ ::= + | - | * | /
⟨var⟩ ::= x | y
⟨num⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Outline

Inferring P Systems with GE

Scenario

We observe $\mathcal{C}$, the set of subsequent configurations pairs (C_i, C_{i+1}) of the P system, for $0 \leq i < n$, and we want to apply GE to find the ruleset whose application would give rise to the sequence $C_0, \ldots, C_n$.

To apply GE we need to define

- A CFG to express the ruleset of a P system
- The fitness of a ruleset

Inferring P Systems with GE

Scenario

We observe $\mathcal{C}$, the set of subsequent configurations pairs (C_i, C_{i+1}) of the P system, for $0 \leq i < n$, and we want to apply GE to find the ruleset whose application would give rise to the sequence $C_0, \ldots, C_n$.

To apply GE we need to define

- A CFG to express the ruleset of a P system
- The fitness of a ruleset

CFG for a P system ruleset

$$
\begin{aligned}
&\langle \text{ruleset} \rangle ::= \langle \text{ruleset} \rangle \cup \{ \langle \text{rule} \rangle \} \mid \{ \langle \text{rule} \rangle \} \\
&\langle \text{rule} \rangle ::= \langle \text{membrane} \rangle \ \langle \text{core} \rangle \\
&\langle \text{core} \rangle ::= \langle \text{multiset} \rangle \ \text{rewrite} \langle \text{multiset} \rangle \mid \\
&\qquad\qquad \langle \text{multiset} \rangle \ \text{sendin} \langle \text{multiset} \rangle \mid \\
&\qquad\qquad \langle \text{multiset} \rangle \ \text{sendout} \langle \text{multiset} \rangle \mid \\
&\qquad\qquad \langle \text{multiset} \rangle \ \text{division} \langle \text{multiset} \rangle \langle \text{multiset} \rangle \\
&\langle \text{multiset} \rangle ::= \langle \text{object} \rangle \cup \langle \text{multiset} \rangle \mid \langle \text{object} \rangle \\
&\langle \text{membrane} \rangle ::= \text{m1} \mid \ldots \mid \text{m}\Lambda \\
&\langle \text{object} \rangle ::= \text{o1} \mid \ldots \mid \text{o}\Gamma
\end{aligned}
$$

Mapping example

$g = 231\ 55\ 113\ 17\ 35\ 144$

⟨ruleset⟩ ::= ⟨ruleset⟩∪{⟨rule⟩} | {⟨rule⟩}
⟨rule⟩ ::= ⟨membrane⟩ ⟨core⟩
⟨core⟩ ::= ⟨multiset⟩ rewrite⟨multiset⟩ |
⟨multiset⟩ sendin⟨multiset⟩ |
⟨multiset⟩ sendout⟨multiset⟩ |
⟨multiset⟩ division⟨multiset⟩⟨multiset⟩
⟨multiset⟩ ::= ⟨object⟩∪⟨multiset⟩ | ⟨object⟩
⟨membrane⟩ ::= m1 | ... | mΛ
⟨object⟩ ::= o1 | ... | oΓ

i	g_i	$\lvert R_s \rvert$	j	w	p
0	231	2	1	0	⟨**ruleset**⟩
1	55	1	0	0	⟨**rule**⟩
2	113	10	3	0	⟨**membrane**⟩⟨core⟩
3	17	4	1	0	m2 ⟨**core**⟩
4	35	2	1	0	m2 ⟨**multiset**⟩ sendin ⟨multiset⟩
5	144	10	4	0	m2 ⟨**object**⟩ sendin ⟨multiset⟩
6	231	2	1	1	m2 o3 sendin ⟨**multiset**⟩
7	55	10	5	1	m2 o3 sendin ⟨**object**⟩
					m2 o3 sendin o4
					$\{o_3\}\ [\ ]_{m_2} \rightarrow [\ \{o_4\}\ ]_{m_2}$

Fitness

It is necessary to introduce a fitness to quantify how well the rules inferred by the evolutionary algorithm approximates the observed behavior.

For $i = 0, \ldots, n$, we define the fitness of a ruleset as the distance between:

- C_{i+1}
- $\tilde{C}_{i+1}$, obtained from C_i applying the ruleset under evaluation.

The membrane structure of a *P system* is a rooted unordered tree
$\rightarrow$ define a distance inspired by the *edit distance* between labeled trees.

Edit distance between membrane structures

The edit distance between two membrane structures is measured by counting the minimum number of operations required to transform one into the other.

Fitness

It is necessary to introduce a fitness to quantify how well the rules inferred by the evolutionary algorithm approximates the observed behavior.

For $i = 0, \ldots, n$, we define the fitness of a ruleset as the distance between:

- C_{i+1}
- $\tilde{C}_{i+1}$, obtained from C_i applying the ruleset under evaluation.

The membrane structure of a *P system* is a rooted unordered tree
$\rightarrow$ define a distance inspired by the *edit distance* between labeled trees.

Edit distance between membrane structures

The edit distance between two membrane structures is measured by counting the minimum number of operations required to transform one into the other.

Fitness

It is necessary to introduce a fitness to quantify how well the rules inferred by the evolutionary algorithm approximates the observed behavior.

For $i = 0, \ldots, n$, we define the fitness of a ruleset as the distance between:

- C_{i+1}
- $\tilde{C}_{i+1}$, obtained from C_i applying the ruleset under evaluation.

The membrane structure of a *P system* is a rooted unordered tree
$\rightarrow$ define a distance inspired by the *edit distance* between labeled trees.

Edit distance between membrane structures

The edit distance between two membrane structures is measured by counting the minimum number of operations required to transform one into the other.

Fitness

It is necessary to introduce a fitness to quantify how well the rules inferred by the evolutionary algorithm approximates the observed behavior.

For $i = 0, \ldots, n$, we define the fitness of a ruleset as the distance between:

- C_{i+1}
- $\tilde{C}_{i+1}$, obtained from C_i applying the ruleset under evaluation.

The membrane structure of a *P system* is a rooted unordered tree
$\rightarrow$ define a distance inspired by the *edit distance* between labeled trees.

Edit distance between membrane structures

The edit distance between two membrane structures is measured by counting the minimum number of operations required to transform one into the other.

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   $\rightarrow$ cost = number of membranes added
2. **Removal** of a membrane and its content
   $\rightarrow$ cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   $\rightarrow$ cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   → cost = number of membranes added
2. **Removal** of a membrane and its content
   → cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   → cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   $\rightarrow$ cost = number of membranes added
2. **Removal** of a membrane and its content
   $\rightarrow$ cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   $\rightarrow$ cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   $\rightarrow$ cost = number of membranes added
2. **Removal** of a membrane and its content
   $\rightarrow$ cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   $\rightarrow$ cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   $\rightarrow$ cost = number of membranes added
2. **Removal** of a membrane and its content
   $\rightarrow$ cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   $\rightarrow$ cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   $\rightarrow$ cost = number of membranes added
2. **Removal** of a membrane and its content
   $\rightarrow$ cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   $\rightarrow$ cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   $\rightarrow$ cost = number of membranes added
2. **Removal** of a membrane and its content
   $\rightarrow$ cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   $\rightarrow$ cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   → cost = number of membranes added
2. **Removal** of a membrane and its content
   → cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   → cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Fitness

The **edit distance** is based on the following operations:

1. **Addition** of a membrane and its content
   → cost = number of membranes added
2. **Removal** of a membrane and its content
   → cost = number of membranes removed
3. **Change** of the objects contained in a membrane
   → cost = *Jaccard distance*

The Jaccard distance measures dissimilarity between two multisets A and B is defined as:

$$d_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

- value is comprised between 0 and 1
- $d_J(A, B) = 0$ if A and B are the same multiset
- $d_J(A, B) = 1$ when A and B have no objects in common

Outline

Conclusion

We introduced a Grammatical Evolution based approach for inferring the ruleset of a P system given some observations of its behavior.

The proposed approach could be relevant for:

- *P system* design tasks
- structure and operation identification.

In the future, we will assess the feasibility and the effectiveness of our method with a thorough experimental evaluation on various benchmark problems aimed at testing the ability to learn each type of rule.

Conclusion

We introduced a Grammatical Evolution based approach for inferring the ruleset of a P system given some observations of its behavior.

The proposed approach could be relevant for:

- *P system* design tasks
- structure and operation identification.

In the future, we will assess the feasibility and the effectiveness of our method with a thorough experimental evaluation on various benchmark problems aimed at testing the ability to learn each type of rule.

Conclusion

We introduced a Grammatical Evolution based approach for inferring the ruleset of a P system given some observations of its behavior.

The proposed approach could be relevant for:

- *P system* design tasks
- structure and operation identification.

In the future, we will assess the feasibility and the effectiveness of our method with a thorough experimental evaluation on various benchmark problems aimed at testing the ability to learn each type of rule.

Thanks!

Any unanswered question?

✉ giorgia.nadizar@phd.units.it

✉ gloria.pietropolli@phd.units.it