Networks of Bio-Inspired Processors.

José M. Sempere
Research Group on Computation Models and Formal Languages
Departmento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Networks of Bio-Inspired Processors.

1. General ingredients and components
2. Networks of Evolutionary Processors (NEPs)
3. Networks of Splicing Processors (NSPs)
4. Networks of Genetic Processors (NGPs)
5. Towards a full general model

Networks of Bio-Inspired Processors.

A NBP is a computational model which

… is inspired by biological aspects (darwinian evolution, DNA recombination, etc.)

… is computationally complete (it has the computation power of a Turing machine)

… is parallel and distributed

… is universal (allows the interpretation of NBPs as source data)

… solves NP-complete problems in “polynomial” time

Networks of Bio-Inspired Processors. An introduction

	P systems	NBPs
Computationally complete and universal	OK	OK
Parallel and distributed	OK	OK
Works with strings	OK	OK
Hardware implementations	OK+KO	OK + KO
Works with multisets of data	OK	OK
Software simulators	OK	OK + KO
In vitro/in vivo implementations	KO	KO
Efficient solutions to hard problems	OK	OK

From NEPs to P Systems → Evolutionary P systems (Mitrana, Sempere 2009)

From P Systems to NBPs → Open problem

Networks of Bio-inspired Processors

Some bioinspired operators over strings and languages

Insertion Insert a symbol into a string — aaaaa ➔ aabaaa

Deletion Delete a symbol from a string — aabaaa ➔ aaaaa

Substitution (mutation) Substitute a symbol into a string — aaaaa ➔ aabaa

Splicing Splicing rules $r=(u_1\#u_2\$v_1\#v_2)$ — r=(a#a$b#b) (abcdaa,bbabcd) ➔ (abcdababcd,ba)

Crossover Full massive splicing with empty context — aa ▹◃bb ➔ λ, bb, abb, aabb, ab, aab, …

Hairpin completion Hairpin completion from folded strings

Superposition Complementarity completion from double stranded strings

loop and double loop recombination DNA recombination based on gene assembly

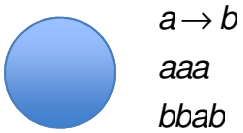
inversion, duplication and transposition DNA fragments modification (operations on substrings)

… etc, etc.

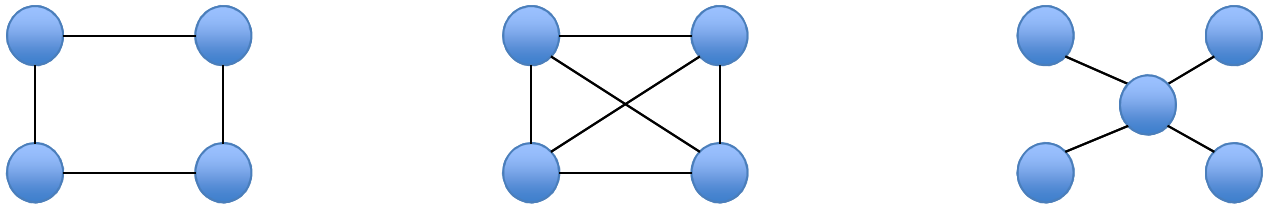
Networks of Bio-inspired Processors

The ingredients to define a Network of Bioinspired Processors

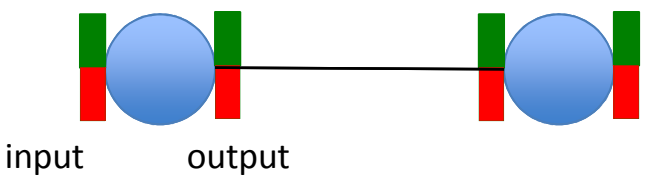
A finite set of processors that apply operations over strings which have been inspired by biomolecular functions and operations in the nature. The processors work with a multiset of strings.

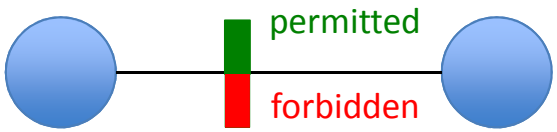


A connection topology between processors in the form of a network.



A set of (input/output) filters which can be attached to the processors or to the connections.





Networks of Evolutionary Processors

Networks of Bio-Inspired Processors

Accepting Networks of Evolutionary Processors

An evolutionary processor over *V* is a 5-tuple *(M,PI,FI,PO,FO)*,where:

Either $M \subseteq Sub_V$ or $M \subseteq Del_V$ or $M \subseteq Ins_V$
The set M represents the set of evolutionary rules of the processor.

PI,FI $\subseteq$ V are the input permitting/forbidding contexts of the processor

PO,FO $\subseteq$ V are the output permitting/forbidding contexts of the processor
(with PI $\cap$FI= $\varnothing$ and PO $\cap$ FO=$\varnothing$)

We can define the following predicated for the filters

$$rc_s(z,P,F) \equiv [P \subseteq alph(z)] \wedge [F \cap alph(z) = \varnothing]$$

$$rc_w(z,P,F) \equiv [alph(z) \cap P = \varnothing] \wedge [F \cap alph(z) = \varnothing]$$

Networks of Bio-Inspired Processors

Accepting Networks of Evolutionary Processors

$$\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$$

where

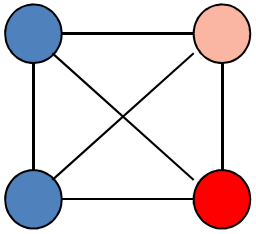
V and U are the input and network alphabets
$G=(X_G,E_G)$ is an undirected graph without loops
$N: X_G \rightarrow EP_U$ associates an evolutionary processor to every node in G
$\alpha: X_G \rightarrow \{l,r,*\}$ associates an action mode to every node (Hybrid networks)
$\beta: X_G \rightarrow \{s,w\}$ associates a filter predicate to every node
x_I, x_O are the input and output nodes

Networks of Bio-Inspired Processors

Accepting Networks of Evolutionary Processors

$$\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$$

How does the network work ?



(I) Evolutionary steps

$$C_i \Rightarrow C_{i+1}$$

- *Every rule that can be applied is massively applied*
- *No competition between rules. All the rules are applied by using different copies*

(II) Communication steps

$$C_i \vdash\!\!\rightarrow C_{i+1}$$

- *Every processor sends all the filtered strings to its neighbours*
- *Every processor receives and stores filtered strings*
- *Strings that are sent but not received are lost*

(III) Network at work

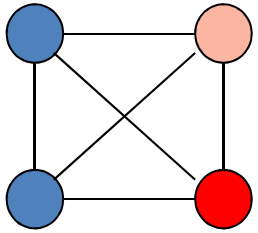
$$C_0 \Rightarrow C_1 \vdash\!\!\rightarrow C_2 \Rightarrow C_3 \vdash\!\!\rightarrow C_4 \ldots$$

Networks of Bio-Inspired Processors

Accepting Networks of Evolutionary Processors

$$\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$$

Accepted language



1. There exists a configuration in which the set of words existing in the output node x_o is non-empty. (halting and accepting computation)
2. There exist two consecutive identical configurations. (halting and rejection computation)
3. It works forever.

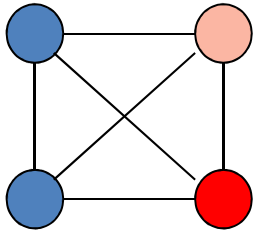
$L(\Gamma)=\{w \in V^* : \text{the computation of } \Gamma \text{ on w is an accepting one}\}.$

Networks of Bio-Inspired Processors

Generating Networks of Evolutionary Processors

$$\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$$

Generated language



1. No specific halting configuration
2. The output node x_o collects a (possibly infinite) set of words

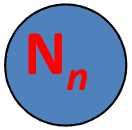
$G(\Gamma)=\{w \in V^* : w$ eventually enters into the output node$\}$.

Networks of Bio-Inspired Processors

Building oracles with (A/G)NEPs

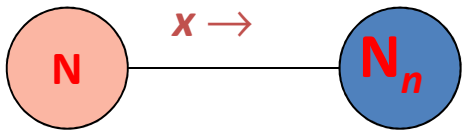
$$\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$$

Let L_{oracle} be a language (the oracle language)

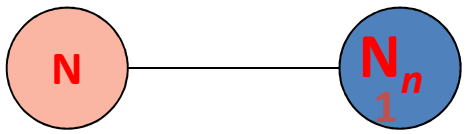


$PI_n = L_{oracle}\#$ $A_n = \{ 0 \}$

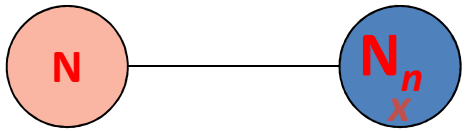
$PO_n = \{0,1\}$ $M_n = \{ a \to \varepsilon, \# \to 1, \varepsilon \to 0 \}$



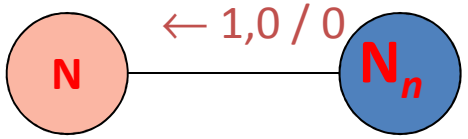
(1) **N** sends the query ***x#***



(3) $\mathbf{N}_n$ transforms the query ***x#*** into **1**



(2) If ***x#*** passes the input filter then ***x#*** $\in$ $L_{oracle}\#$



(4) $\mathbf{N}_n$ sends the answer

Networks of Bio-Inspired Processors

GNEPs completeness and beyond Turing's limit

Theorem (*Castellanos, Martín-Vide, Mitrana, Sempere* 2003)
Every RE language can be generated by a complete NEP with 5 processors.

Theorem (*Castellanos, Martín-Vide, Mitrana, Sempere* 2003)
Every RE language can be generated by a star NEP with 5 processors.

Theorem (*Castellanos, Martín-Vide, Mitrana, Sempere* 2003)
Every RE language can be generated with a ring NEP with 6 processors.

The Arithmetic Hierarchy

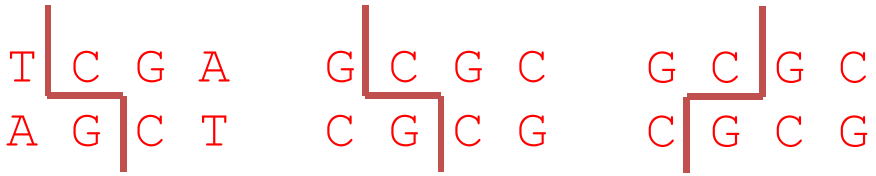
$\Sigma_0 = \text{REC} \quad \Sigma_{n+1} = \text{A-r.e. con A} \in \Sigma_n$
$\Pi_n = \text{co-}\Sigma_n \; \Delta_n = \Sigma_n \cap \Pi_n$

Theorem
Every language in the Arithmetic Hierarchy can be generated by a complete/star/ring NEP.
(here the number of processors depends on the network topology and the oracle language)

Networks of Splicing Processors

DNA Recombination and Splicing

```
5’ – CCCCCTCGACCCCC – 3’
3’ – GGGGGAGCTGGGGG – 5’

5’ – AAAAAGCGCAAAAA – 3’
3’ – TTTTTCGCGTTTTT – 5’

5’ – TTTTTGCGCTTTTT – 3’
3’ – AAAAACGCGAAAAA – 5’
```



TaqI SciNI HhaI

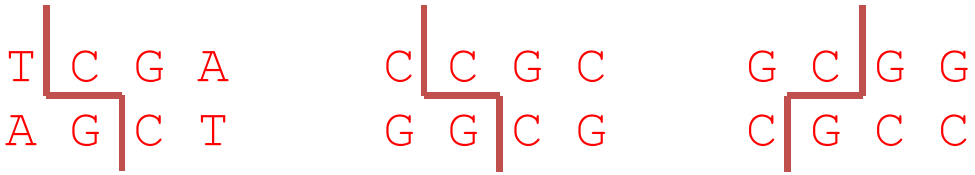
double strands restriction enzimes (endonuclease)

```
5’ – CCCCCT   CGACCCCC – 3’
3’ – GGGGGAGC   TGGGGG – 5’

5’ – AAAAAG    CGCAAAAA – 3’
3’ – TTTTTCGC    GTTTTT – 5’

5’ – TTTTTGCG    CTTTTT – 3’
3’ – AAAAAC    GCGAAAAA – 5’
```

ligases

```
5’ – CCCCCTCGCAAAAA – 3’
3’ – GGGGGAGCGTTTTT – 5’

5’ – AAAAAGCGACCCCC – 3’
3’ – TTTTTCGCTGGGGG – 5’
```

Splicing over strings (Types I and II)



TaqI SciNI HhaI

Patterns (T, CG, A) (C,CG,C) — Class I; (G,CG,G) — Class II

$w_1 = w'_1\ u_1\ x_1\ v_1\ w''_1$ $p_1 = (u_1\ ,\ x_1\ ,\ v_1)$

$w_2 = w'_2\ u_2\ x_2\ v_2\ w''_2$ $p_2 = (u_2\ ,\ x_2\ ,\ v_2)$

The splicing only occurs if p_1 and p_2 are of the same class and $x_1 = x_2$

$z_1 = w'_1\ u_1\ x_1\ v_2\ w''_2$

$z_2 = w'_2\ u_2\ x_2\ v_1\ w''_1$

Splicing over strings (Types I and II)

$w_1 = w'_1\ u_1\ x_1\ v_1\ w''_1$ $\quad p_1 = (u_1\ ,\ x_1\ ,\ v_1)$ $\quad z_1 = w'_1\ u_1\ x_1\ v_2\ w''_2$

$w_2 = w'_2\ u_2\ x_2\ v_2\ w''_2$ $\quad p_2 = (u_2\ ,\ x_2\ ,\ v_2)$ $\quad z_2 = w'_2\ u_2\ x_2\ v_1\ w''_1$

The patterns (p_1, p_2) can be denoted as $(u_1, u_2; u_3,u_4)$ or as the string $u_1\#u_2\$u_3u_4$.

Let r=$u_1\#u_2\$u_3u_4$ be an splicing rule, then we can define the following operations

Type I splicing operation

$(x,y) \vdash_r z$ iff $x=x_1u_1u_2x_2$,
$y=y_1u_3u_4y_2$,
$z=x_1u_1u_4y_2$,

Type II splicing operation

$(x,y) \models_r (z,w)$ iff $x=x_1u_1u_2x_2$,
$y=y_1u_3u_4y_2$,
$z=x_1u_1u_4y_2$,
$w= y_1u_3u_2x_2$

H schemes

$\sigma = (V, R)$ where

V an alphabet
$R \subseteq V^*\#V^*\$V^*\#V^*$ a set of splicing rules

If R belongs to the family of languages L then σ is of type L

$\forall\ L \subseteq V^*$

$$\sigma_1(L) = \{\ z \in V^* : (x,y) \vdash_r z,\ \ x,y \in L,\ r \in R\ \}$$

$$\sigma_1(x,y) = \{\ \ z \in V^* : (x,y) \vdash_r z,\ r \in R\ \}$$

$$\sigma_1(L) = \bigcup_{x,y \in L} \sigma_1(x,y)$$

Language classes denoted by the H schemes
(the noniterative case)

$\sigma = (V, R)$

$$S_1(L_1,L_2) = \{ \ \sigma_1(L) : L \in L_1, R \in L_2 \}$$

L_1 is closed under splicing of type L_2 if $S_1(L_1 , L_2) \subseteq L_1$

Lemma **For all the families of languages L_1 , L_2 , L'_1 , L'_2 such that $L_1 \subseteq L'_1$ and $L_2 \subseteq L'_2$ the inclusion $S_1(L_1 , L_2) \subseteq S_1(L'_1 , L'_2)$ holds.**

Language classes denoted by the H systems
(the noniterative case)

L_1 \ L_2	FIN	REG	LIN	CF	CS	RE
FIN	FIN	FIN	FIN	FIN	FIN	FIN
REG	REG	REG	REG, LIN	REG, CF	REG, RE	REG, RE
LIN	LIN, CF	LIN, CF	RE	RE	RE	RE
CF	CF	CF	RE	RE	RE	RE
CS	RE	RE	RE	RE	RE	RE
RE	RE	RE	RE	RE	RE	RE

$S_1(L_1,L_2)$

Language classes denoted by the H schemes (the iterative case)

$$\sigma = (V, R) \qquad L \subseteq V^*$$

$$\sigma_1(L) = \{\ z \in V^* : (x,y) \vdash_r z,\ \ x,y \in L, r \in R\ \}$$

$$\sigma^0{}_1(L) = L$$
$$\sigma^{i+1}{}_1(L) = \sigma^i{}_1(L) \cup \sigma_1(\sigma^i{}_1(L)), \quad i \geq 0$$

$$\sigma^*{}_1(L) = \bigcup_{i \geq 0} \sigma^i{}_1(L)$$

$$H_1(L_1,L_2) = \{\ \sigma^*{}_1(L) : ,\ \ L \in L_1, R \in L_2\ \}$$

Language classes denoted by the H systems
(the iterative case)

L_1 \ L_2	FIN	REG	LIN	CF	CS	RE
FIN	FIN, REG	FIN, RE	FIN, RE	FIN, RE	FIN, RE	FIN,RE
REG	REG	REG, RE	REG, RE	REG, RE	REG, RE	REG, RE
LIN	LIN, CF	LIN, RE	LIN, RE	LIN, RE	LIN, RE	LIN, RE
CF	CF	CF, RE	CF, RE	CF, RE	CF, RE	CF, RE
CS	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE
RE	RE	RE	RE	RE	RE	RE

$H_1(L_1,L_2)$

Extended H Systems

$\sigma = (V, R)$ is an H scheme $\quad\quad$ $L \subseteq V^*$ is a language

$\gamma = (V, L, R)$ is a H system

$$L(\gamma) = \sigma^*_1(L)$$

$\gamma = (V, T, A, R)$ is an extended H system

V is an alphabet
$T \subseteq V$ is an alphabet of terminal symbols
$A \subseteq V^*$ is a set of axioms
$R \subseteq V^*\#V^*\$V^*\#V^*$ is a set of splicing rules

$$L(\gamma) = \sigma^*_1(A) \cap T^*$$

$$EH_1(L_1,L_2) = \{\ L(\gamma) : A \in L_1, R \in L_2 \}$$

Language classes denoted by the extended H systems

L_1 \ L_2	FIN	REG	LIN	CF	CS	RE
FIN	REG	RE	RE	RE	RE	RE
REG	REG	RE	RE	RE	RE	RE
LIN	LIN, CF	RE	RE	RE	RE	RE
CF	CF	RE	RE	RE	RE	RE
CS	RE	RE	RE	RE	RE	RE
RE	RE	RE	RE	RE	RE	RE

$EH_1(L_1,L_2)$

Extended H systems with permitting contexts

$\gamma = (V, T, A, R)$ is an extended H system

R is a finite set of 3-tuples in the form

$p = (r; C_1, C_2)$ $r = u_1\#u_2\$u_3\#u_4$

$C_1, C_2 \subseteq V^*$ (finite)

$(x,y) \models_p (z,w)$ iff $(x,y) \models_r (z,w)$

every element in C_1 appears in x

every element in C_2 appears in y

$L(\gamma) = \sigma^*_2(A) \cap T^*$

sets of permitting contexts C_1 and C_2

Lemma: $RE \subseteq EH_2(FIN,pFIN)$

set of axioms A

Splicing processors

Choudhary & Krithivasan, 2007

A splicing processor over V is a 8-tuple $(M,S,A,PI,FI,PO,FO,\beta)$,where:

M is a set of splicing rules with permitting context
S is a finite set of strings over V
A is a finite set of axioms over V
PI,FI $\subseteq$ V are the input permitting/forbidding contexts of the processor
PO,FO $\subseteq$ V are the output permitting/forbidding contexts of the processor
(with PI $\cap$ FI= $\varnothing$ and PO $\cap$ FO= $\varnothing$)
$\beta \in$ {(1),(2)} defines the input/output filter

We can define the following predicates for the filters

$$rc_{(1)}(z,P,F) \equiv [P \subseteq alph(z)] \wedge [F \cap alph(z) = \varnothing]$$

$$rc_{(2)}(z,P,F) \equiv [alph(z) \cap P \neq \varnothing] \wedge [F \cap alph(z) = \varnothing]$$

Splicing processors

Manea, Martín-Vide & Mitrana, 2005

A splicing processor over V is a 6-tuple (S,A,PI,FI,PO,FO),where:

S is a finite set of splicing rules over V
A is a finite set of auxiliary words over V
PI,FI $\subseteq$ V are the input permitting/forbidding contexts of the processor
PO,FO $\subseteq$ V are the output permitting/forbidding contexts of the processor
(with PI $\cap$ FI= $\varnothing$ and PO $\cap$ FO= $\varnothing$)

We can define the following predicates for the filters

$$rc_{(1)}(z,P,F) \equiv [P \subseteq alph(z)] \wedge [F \cap alph(z) = \varnothing]$$

$$rc_{(2)}(z,P,F) \equiv [alph(z) \cap P \neq \varnothing] \wedge [F \cap alph(z) = \varnothing]$$

Networks of Splicing Processors (NSPs)

Choudhary & Krithivasan, 2007

A NSP of size n is a tuple $(V,N_1,N_2, \ldots, N_n,G)$,where:

V is an alphabet
N_i is the i*th* splicing processor
G is an undirected graph without loops (the underlying topology of the network)

- The configuration of the network consists of the strings at every processor (excluding the axioms for the splicing rule)
- The network evolves as in the Networks of Evolutionary Processors (NEPs) with *splicing* steps and *communication* steps
- There exists an output processor which collects the strings as the product of a computation sequence
- The network halts whenever no splicing operation can be carried out and no string can be communicated

Accepting Networks of Splicing Processors (ANSPs)

Manea, Martín-Vide & Mitrana, 2005

An ANSP is a 9-tuple $(V,U,<,>,G,N,\alpha,x_I,x_O)$,where:

V,U are the input and network alphabets
$<,> \in U\backslash V$ are special symbols
$G=(X_G,E_G)$ is an undirected graph without loops (the underlying topology of the network)
$N: X_G \rightarrow SP_U$ associates to each node in the graph a splicing processor over U
$\alpha: X_G \rightarrow \{(1),(2)\}$ defines the type of filter at every processor
$x_I,x_O \in X_G$ are the input and output processors

- The configuration of the network consists of the strings at every processor
- The network evolves as in the Networks of Evolutionary Processors (NEPs) with *splicing* steps and *communication* steps
- The input processor initially holds the string to be analyzed
- The network halts whenever: (1) a string enters into the output processor (accepting computation) or, (2) There exists two identical configurations obtained either in consecutive splicing steps or in consecutive communication steps (not an accepting computation)

(A)NSPs are computationally complete

Choudhary & Krithivasan, 2007

Theorem. Each recursively enumerable language can be generated by a complete NSP of size two where the splicing rules are of type regular.

Simulate a type 0 Chomsky grammar which works in the same way as the EH_2(FIN,pFIN) system

(A)NSPs are computationally complete

Manea, Martín-Vide & Mitrana, 2005

Theorem. For any Turing machine M there exists an ANSP that accepts exactly the same language as M does.

Simulate the movements of a Turing machine with a number of processors that linearly depends on the size of the alphabet and states of the Turing machine

(A)NSPs are computationally complete

Manea, Martín-Vide & Mitrana, 2005

Theorem. For any ANSP Γ, accepting the language L, there exists a Turing machine M that accepts the same language L.

The nondeterministic Turing machine associates every state to a node in the ANSP. The splicing rules and evolution strings are nondeterministically chosen. Whenever the Turing machine enters into the state which is associated to the output node, then it halts and accepts the input word.

Manea, Martín-Vide & Mitrana, 2007

Introducing time complexity measures

We consider an ANSP Γ with the input alphabet V that halts on every input. The time complexity of the computation

$$C_0(x), C_1(x), C_2(x), \ldots, C_m(x)$$

of Γ on x is denoted by **$Time_\Gamma(x)$** and equals m.

The time complexity of Γ is the partial function from N to N,

$$\mathbf{Time_\Gamma(n) = max\{Time_\Gamma(x) \mid |x| = n\}.}$$

For a function f : N → N we define

$Time_{ANSP}(f(n)) = \{L \mid L = L(\Gamma)$ for an ANSP Γ with $Time_\Gamma(n) \leq f(n)$ for some $n \geq n_0\}$.

$$PTime_{ANSP} = \bigcup_{k \geq 0} Time_{ANSP}(n^k)$$

Complexity issues

Manea, Martín-Vide & Mitrana, 2007

Complexity results

Proposition. If $L \in NP$ then $L \in PTime_{ANSP}$.

Proposition. If $L \in PTime_{ANSP}$ then $L \in NP$.

$PTime_{ANSP} = NP$

Complexity issues

Manea, Martín-Vide & Mitrana, 2007

Introducing space complexity measures

The length complexity of the computation

$C_0(x), C_1(x), C_2(x), \ldots, C_m(x)$

of Γ on x is denoted by **$\text{Length}_\Gamma(x)$** and equals to

$\max\{ |w| : w \in C_i(x) : 1 \leq i \leq m\}$.

The length complexity of Γ is the partial function from N to N,

$\text{Length}_\Gamma(n) = \max\{\text{Length}_\Gamma(x) \mid |x| = n\}$.

For a function f : N → N we define

$\text{Length}_{ANSP}(f(n)) = \{L \mid L = L(\Gamma)$ for an ANSP Γ with $\text{Length}_\Gamma(n) \leq f(n)$ for some $n \geq n_0\}$.

$$PLength_{ANSP} = \bigcup_{k \geq 0} Length_{ANSP}(n^k)$$

Complexity issues

Manea, Martín-Vide & Mitrana, 2007

Complexity results

Proposition. If $L \in$ PSPACE then $L \in PLength_{ANSP}$.

Proposition. If $L \in PLength_{ANSP}$ then $L \in$ PSPACE.

$PLength_{ANSP}$ = PSPACE

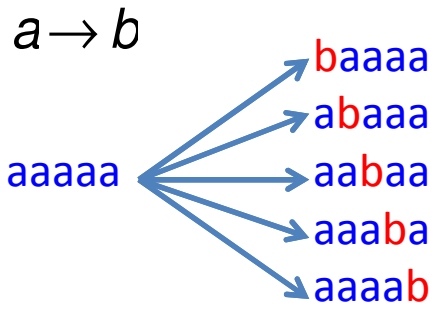
Networks of Genetic Processors

Networks of Genetic Processors

From ANEPs and ANSPs to Accepting Networks of Genetic Processors (ANGPs)

Substitute evolutionary operations or splicing rules by

(a) Mutation operations



(b) Crossover between strings

$x \triangleright\!\triangleleft y = \{ x_1y_2, y_1x_2 : x=x_1x_2, y = y_1y_2 \}$

	cd	cd	cd
ab	cd,ab	d,cab	λ,cdab
ab	acd, b	ad,cb	λ,cdb
ab	abcd, λ	abd,c	ab,cd

Networks of Genetic Processors

From ANEPs and ANSPs to Accepting Networks of Genetic Processors (ANGPs)

Some important remarks:

(1) NEPs with only substitution (mutation) processors are not computationally complete
(2) NSPs with empty contexts (*crossover*) are not computationally complete

Combine mutation and crossover to …

(1) achieve computation completeness
(2) Connect Networks of Bio-Inspired Processors with Genetic Algorithms

Networks of Genetic Processors

Accepting Networks of Genetic Processors (I)

A genetic processor over V is a 5-tuple $(M_R, A, PI, FI, PO, FO, \alpha, \beta)$, where:

- MR is a finite set of mutation rules over V (a → b)
- A is a multiset of strings over V with finite support and an arbitrary large number of copies of every string
- PI,FI $\subseteq$ V* are finite sets of input permitting/forbidding contexts
- PO,FO $\subseteq$ V* are finite sets of output permitting/forbidding contexts
- $\alpha \in$ {(1),(2)} defines the function mode such that

  (1) means only mutation operations
  (2) means crossover operations (MR = $\varnothing$)

- $\beta \in$ {(1),(2)} defines the filter predicates

(1) $rc_s(z; P, F) \equiv [P \subseteq seg(z)] \wedge [F \cap seg(z) = \varnothing]$
(2) $rc_w(z; P, F) \equiv [seg(z) \cap P \neq \varnothing] \wedge [F \cap seg(z) = \varnothing]$

Networks of Genetic Processors

Accepting Networks of Genetic Processors (II)

ANGP of size *n* is a tuple $\Gamma = (V, N_1, N_2, \ldots, N_n, G, N)$

where V is an alphabet

$G=(X_G,E_G)$ is an undirected graph without loops

N_i $(1 \leq i \leq n)$ is a genetic processor over V

$N: X_G \rightarrow \{N_1,N_2,\ldots,N_n\}$ associates a genetic processor to every node in the graph

How does the network work ?

(I) Genetic steps

$C_i \Rightarrow C_{i+1}$

- *Every rule that can be applied is massively applied*
- *No competition between rules. All the rules are applied by using different copies*

(II) Communication steps

$C_i \vdash C_{i+1}$

- *Every processor sends all the filtered strings to its neighbours*
- *Every processor receives and stores filtered strings*
- *Strings that are sent but not received are lost*

(III) Network at work $C_0 \Rightarrow C_1 \vdash C_2 \Rightarrow C_3 \vdash C_4 \ldots$

SAME ACCEPTANCE CRITERION AS IN THE EVOLUTIONARY CASE

Networks of Genetic Processors

Theorem: ANGPs are computationally complete

From deterministic Turing machines .... to ANGPs

$M = (\Sigma, \Gamma, Q, \delta, q_0, B, Q_f)$

$R = (V, N_c, N_1, \ldots, N_n, N_{out}, \hat{K}, f)$

instantaneous description $\alpha q a \beta$

encoded instantaneous description $q\alpha\$a\beta F$

movement to the right $\delta(q,a)=(p,b,R)$

dedicated processor N_{qaR} $\begin{matrix} q \to p \\ \$ \to b' \\ a \to \overline{\$} \end{matrix}$ strong PI = {q,$a}

movement to the right $\delta(q,a)=(p,b,L)$

a couple of dedicated processors for every c

N_{qacL1} $\begin{matrix} q \to p \\ \$ \to \underline{c} \\ a \to b' \end{matrix}$ strong PI = {q,$a} N_{qacL2} $c \to \overline{\$}$ strong PI = {p,c$\underline{c}$b'}

visit to a new rightmost cell

a couple of dedicated processors

$\alpha q B$

N_B crossover with #BF N_{B2} restores #BF

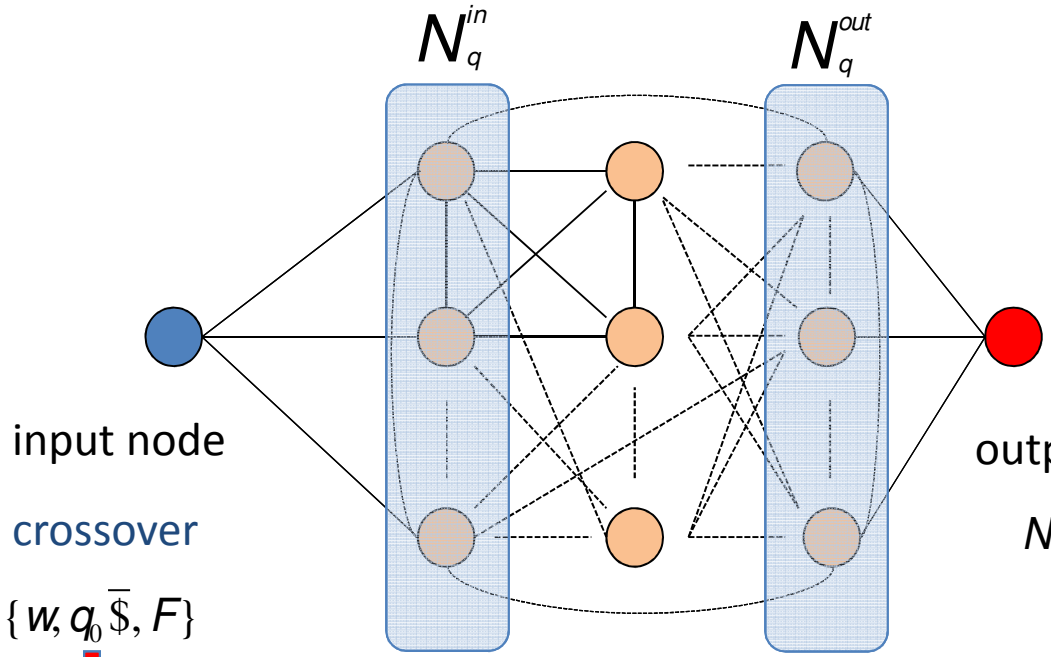
$q\alpha\$BF \in q\alpha\$F \triangleright\!\triangleleft \#BF$

Networks of Genetic Processors

Theorem: ANGPs are computationally complete

$$\Gamma = (V, N_1, N_2, \ldots, N_n, G, N)$$

The network topology



Networks of Genetic Processors

Theorem: ANGPs are computationally complete

A similar simulation for non-deterministic Turing machines

$M = (\Sigma, \Gamma, Q, \delta, q_0, B, Q_f)$

instantaneous description $\alpha q a \beta$

movement to the right

$(p,b,R) \in \delta(q,a)$

movement to the right

$(p,b,L) \in \delta(q,a)$

$R = (V, N_c, N_1, \ldots, N_n, N_{out}, \hat{K}, f)$

encoded instantaneous description $q\alpha\$a\beta F$

dedicated processor N_{qapbR} $q \to p$, $\$ \to b'$, $a \to \overline{\$}$ strong PI = {q,$a}

a couple of dedicated processors for every c

$N_{qapbcL1}$ $q \to p$, $\$ \to \underline{c}$, $a \to b'$ strong PI = {q,$a} $N_{qapbcL2}$ $c \to \overline{\$}$

strong PI = {p,c$\underline{c}$b'}

Networks of Genetic Processors

Looking to the computational complexity

Let us consider an ANGP R and the language L accepted by R, then the time complexity of the accepting computation of R if x is given as an input string is denoted by $Time_R(x)$ and it is defined as the number of steps (both communication and evolutionary ones) such that the network R halts on x in an acceptance mode.

$$Time_R(n) = \max\{Time_R(x) : x \in L(R), |x| = n\}$$

$Time_{ANGP}(f)$={L: There exists an ANGP, R, and a natural number n_0 such that L=L(R) and for all $n \geq n_0$, ($Time_R(n) \leq f(n)$)}

$$Time_{ANGP}(C) = \bigcup_{f \in C} Time_{ANGP}(f)$$

$$Time_{ANGP}(poly) \equiv PTime_{ANGP}$$

Theorem: $NP \subset PTime_{ANGP}$

Open Problem: $PTime_{ANGP} \subset ?$

Networks of Genetic Processors

Other variants of Networks of Genetic Processors

Generating Networks of Genetic Processors (GNGPs)

No input processor
The output processor collects the generating language
The halting criterium is the repetition of two consecutives configuration

Theorem: Let L be a recursively enumerable language generated by a grammar G in Kuroda's Normal Form. Then, there exists a GNGP R such that

(1) R has 16 genetic processors

(2) R generates L

Optimizing Networks of Genetic Processors (ONGPs)

The input processor stores the instance of the problem P to be optimized according to f
The output processor collects the solution S such that, at anytime t,

$$S = \text{argmax/ min}(f, t) : (\forall t_i \leq t)(f(S_{t_i}) \leq / \geq f(S))$$

No explicit halting criteria
The processor filters can be substituted by integer functions and threshold values

Networks of Genetic Processors

From Networks of Genetic Processors to Parallel Genetic Algorithms

The main components of a Genetic Algorithm (or Evolution Program) are:

- A genetic representation for potential solutions to the problem
- A way to create an initial population of potential solutions
- An evaluation function that plays the role of the environment, rating solutions in terms of their "*fitness*"
- Genetic operators that alter the composition of the potential solutions
- Values for various parameters that the genetic algorithm uses(population size, probabilities of applying genetic operators,etc.)

Networks of Genetic Processors

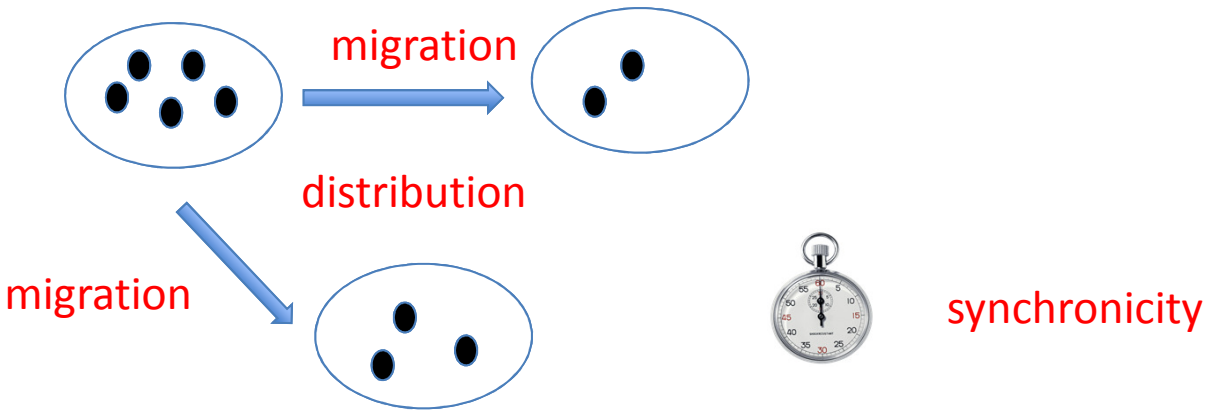
From Networks of Genetic Processors to Parallel Genetic Algorithms

The main ingredients to propose Parallel and Distributed Genetic Algorithm

The distribution of the individuals in different populations (master-slave, multiple populations or islands, fine-grained populations or hierarchical and hybrid populations) and the neighborhood topology (rings, *m,n*-complete, ladders, grids, etc.)

The synchronicity of the populations evolution and communication.

The migration phenomena: The migration rates (the percentage of individuals that migrate from one population to a different one),the migration selection (the selections of the individuals that migrate) and the migration frequency.



Networks of Genetic Processors

From Networks of Genetic Processors to Parallel Genetic Algorithms

From (Parallel) Genetic Algorithm as optimizers to acceptors

Acceptance Criterion I

Let *w* be an input string. We will say that a PGA accepts w if, after a finite number of iterations (operator applications, fitness selection and individuals migration), w appears in a predefined survival population.

Acceptance Criterion II

Let w be an input string. We will say that a PGA accepts w if, after a finite number of iterations (operator applications, fitness selection and individuals migration), a distinguished individual x_{yes} appears in a predefined survival population. We say that the PGA rejects the input string if, after a finite number of iterations (operator applications, fitness selection and individuals migration), a distinguished individual x_{not} appears in a predefined survival population or the PGA never finishes.

Theorem: Let D be a decision problem and L_D its acceptance language. D can be solved by a Parallel Genetic Algorithm with acceptance criterion I iff it can be solved with acceptance criterion II.

Networks of Genetic Processors

From Networks of Genetic Processors to Parallel Genetic Algorithms

Theorem: Parallel Genetic Algorithms with multiple-populations, synchronicity and full migration phenomena are computationally complete.

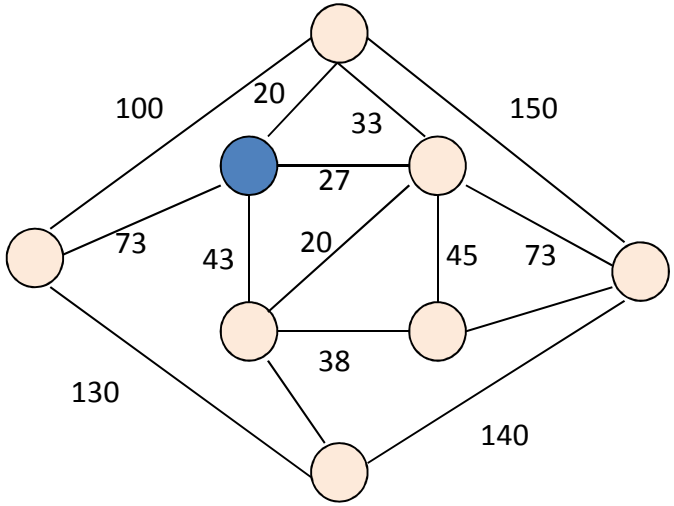
Open Problem I Is full migration phenomena really needed ?

Open Problem II What is the role of crossover ?

Networks of Genetic Processors

Solving the Optimization Traveling Salesman Problem (I)

The Problem: There are *n* cities and connections between them. We have to find a path that starts and begins at a given city, visits any city with a minimal distance



G=(N,A)

Find C={1,2,...,n} such that $C = \text{argmin}(\sum_{i=1}^{n-1} A[C_i, C_{i+1}] + A[C_n, C_1])$

Networks of Genetic Processors

Solving the Optimization Traveling Salesman Problem (II)

- The strings in the processors are the secuences of nodes in a path
- The filters at the genetic processors are replaced by fitness functions (the sum of the distances in the path) and selection of the best
- The experiments replicate "*Solving Travaling Saleman Problem by Ant Colony Optimization Algorithm with Association Rule",* G. Shang, Z. Lei, Z. Fengting, Z. Chunxian. Third International Conference on Natural Computation (ICNC 2007)
- 30 cities defined through their coordinates

Maximum Population at any processor 10	**average**	**best**	**worst**
Genetic algotihms	852,99	675,57	982,83
Complete NGP with 7 processors	550,07	495,66	624,01
Linear NGP with 16 processors	528,52	485,71	601,6
Star NGP with 10 processors	512,18	484,25	545,11
Circular NGP with 13 processors	549,79	521,13	599,56

Networks of Genetic Processors

Solving the Optimization Traveling Salesman Problem (III)

Maximum Population at any processor 20	**average**	**best**	**worst**
Genetic algotihms	676,25	625,8	732,72
Linear NGP with 13 processors	502,35	428,28	553,18
Linear NGP with 16 processors	482,4	453,26	519,58
Linear NGP with 20 processors	503,03	447,66	576,3
Complete NGP with 20 processors	502,46	442,51	567,4
Linear NGP with 20 processors (+ one random generator every 3 processors)	491,07	436,95	541,59

Maximum Population at any processor 30	**average**	**best**	**worst**
Linear NGP with 20 processors	499,71	423,25	539,25
Complete NGP with 20 processors	496,01	457,65	540,99

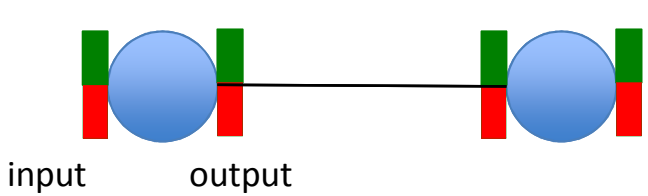
Networks of Bio-Inspired Processors

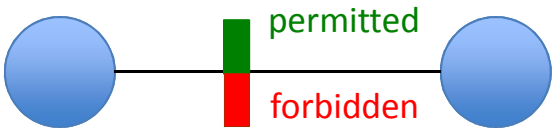
Towards a full general model …

First: Generalize the operations in the processors

Mutation processors Splicing processors Hairpin processors ---------- etc. etc.

Second: Generalize the filter positions





Networks of Bio-Inspired Processors

Towards a full general model …

A bio-inspired processor over V is a 5-tuple (op,PI,FI,PO,FO),where:

op is a biologically inspired operation over strings
PI,FI $\subseteq$ V are the input permitting/forbidding contexts of the processor

PO,FO $\subseteq$ V are the output permitting/forbidding contexts of the processor

- ***op* encapsulates the operation parameters**
- **PI,FI,PO and FO can be empty so the filters are attached to the connections**

Networks of Bio-Inspired Processors

Accepting Networks of Bio-Inspired Processors

$$\Gamma = (V, U, G, N, \beta, \gamma, x_I, x_O)$$

where

V and U are the input and network alphabets
$G=(X_G,E_G)$ is an undirected graph without loops
$N: X_G \rightarrow BP_U$ associate a bio-inspired processor to every node in G
$\beta: X_G \rightarrow \{s,w\}$ associates a filter predicate to every node
$\gamma: E_G \rightarrow 2^U \times 2^U$ associates a filter (P_e,F_e) to every edge in the graph
x_I, x_O are the input and output nodes

Networks of Bio-Inspired Processors

References

- Fernando Arroyo Montoro, Juan Castellanos, Victor Mitrana, Eugenio Santos, José M. Sempere. Networks of Bio-inspired Processors Triangle Vol. 7, pp 3-22. 2012

Networks of Evolutionary Processors

- J.Castellanos, Carlos Martín-Vide, Victor Mitrana, José M.Sempere. Networks of evolutionary processors. Acta Informatica 39, pp 517-529. 2003.

- M. Margenstern, V. Mitrana, M.J. Pérez-Jiménez. Accepting hybrid networks of evolutionary processors. In Proceedings of the International Meeting on DNA Computing, DNA 10, LNCS Vol. 3384, pp 235-246. Springer.2005.

Networks of Splicing Processors

- F. Manea, C. Martín-Vide, V. Mitrana. Accepting networks of splicing processors. In Proceedings of the First Conference on Computability in Europe, CiE 2005,LNCS Vol. 3526, pp 300-309. Springer. 2005.

- F. Manea, C. Martín-Vide, V. Mitrana. Accepting networks of splicing processors: complexity results. Theoretical Computer Science 371, pp 72-82. 2007.

Networks of Genetic Processors

- M. Campos, José M. Sempere. Accepting networks of genetic processors are computationally complete. Theoretical Computer Science Vol. 456, pp 18-29. 2012.