Solving Combinatorial Problems with Networks of Genetic Processors[1]

Marcelino Campos, José M. Sempere

Abstract: *Recently, a new model of computation that is inspired by genetic operations over strings such as mutation and crossover has been proposed. Networks of Genetic Processors (NGPs) are highly related to previously proposed models such as Networks of Evolutionary Processors (NEPs) and Networks of Splicing Processors (NSPs). NGPs are computationally complete and several complexity measures have been proposed to evaluate their computing power with restricted resources (mainly, the time and the number of processors in the network). In this work we evaluate NGPs in an experimental approach. We have selected a NP-complete decision problem, the* Hamiltonian Cycle Problem*, and we have solved different instances with the proposed model of computation. Our aim is to prove that the selected problem (and all NP problems) can be solved in polynomial time with NGPs. In this case, our experiments show that the problem can be solved in linear time with a fixed number of processors for a given size of the problem.*

Keywords: *Networks of biologically-inspired processors, Combinatorial Problems, Complexity.*

Introduction

Natural Computing [KR, 2008] is a research area that looks to biology and biochemistry nature in order to propose new models of computation and algorithms. The experiment performed by Adleman [Adleman, 1994] in which he solved a combinatorial problem using only DNA strands and enzymes was a milestone that triggered the proposals of new models of computation based on the biomacromolecules and living cells behavior. So, the Networks of Bio-inspired Processors (NBPs) have processors that operate over multisets of strings by applying string operations inspired by biology. The communication and the filtering of the new strings among the processors are the two remaining ingredients that make this moldel a complete model of computation (that is, in many cases they are equivalent to Turing machines). There have been different proposals of NBPs: the Networks of Evolutionary Processors (NEPs) [CMMS, 2003] operate with point mutation over strings by applying substitution, deletion and insertion of new symbols. The Networks of Splicing Processors (NSPs) [MMM, 1949] work with splicing rules inspired by DNA recombination with context. In our model, the Networks of Genetic Processors (NGPs) [CS, 2012], the processors can apply mutation by substitution and splicing with empty context (full crossover). It has been proved that NGPs are computationally complete and they can solve any NP problem with a polynomial number of computing steps which can be applyied in polynomial time over the size of the input instance. In this work, we use NGPs to solve the Hamiltonian Cycle Problem (HCP) that is a well known NP-complete decision problem. So, any combinatorial decision problem that belongs to NP can be solved by reducing it to the HCP and then applying our proposal to obtain a solution.

The structure of this work is as follows: first, we introduce basic concepts on formal language theory and computation that are used in some definitions. Then, we define the Networks of Genetic Processors and we introduce a complexity measure to evaluate the time complexity of the model. In the next section, we define The Hamiltonian Cycle Problem and we propose a NGP to solve it. We show the results of the experiments that we have carried out and, finally, we provide some conclusions and some guidelines for future work.

Basic concepts and notation

[1]Work partially supported by the Ministry of Science and Innovation under coordinated research project TIN2011-28260-C03-00 and research project TIN2011-28260-C03-01.

In the following we will introduce some basic concepts about language theory from [HU, 1979] and computational complexity from [DK, 2000].

An *alphabet* is a finite set of elements named *symbols*. A *string* is any ordered finite sequence of symbols. The empty string is denoted by ε and it is defined as the string with no symbols. The infinite set of all the strings defined over a given alphabet V will be denoted by V^*. Given the string $x \in V^*$, we denote the minimal subset $W \subseteq V$ such that $x \in W^*$ by $alph(x)$. Given the string $x \in V^*$, we denote the set of segments of x by $seg(x)$, and it is defined as the set $\{\beta \in V^* : x = \alpha\beta\gamma$ with $\alpha, \gamma \in V^*\}$. Obviously, given any string $x \in V^*$ the set $alph(x)$ is a subset of $seg(x)$. A language defined over an alphabet V is a subset of strings of V^*.

Given the alphabet V, a *mutation rule* $a \to b$, with $a, b \in V$, can be applied over the string xay to produce the new string xby (observe that a mutation rule can be viewed as a substitution rule).

A *crossover operation* is an operation over strings defined as follows: Let x and y be two strings, then $x \bowtie y = \{x_1y_2, y_1x_2 : x = x_1x_2$ and $y = y_1y_2\}$. Observe that $x, y \in x \bowtie y$ given that we can take ϵ to be a part of x or y. The operation can be extended over languages as $L_1 \bowtie L_2 = \bigcup_{x \in L_1, y \in L_2} x \bowtie y$. Obviously, for any language L, $L \bowtie L$ is well defined.

Let P and F be two disjoint subsets of an alphabet V, and let $w \in V^*$. We define the predicates $\varphi^{(1)}$ and $\varphi^{(2)}$ as follows

1. $\varphi^{(1)}(w, P, F) \equiv (P \subseteq alph(w)) \wedge (F \cap alph(w) = \emptyset)$ (*strong predicate*)
2. $\varphi^{(2)}(w, P, F) \equiv (alph(w) \cap P \neq \emptyset) \wedge (F \cap alph(w) = \emptyset)$ (*weak predicate*)

We can extend the previous predicates to act over segments instead of symbols. Let P and F be two disjoint sets of finite strings over V, and let $w \in V^*$. We extend the predicates $\varphi^{(1)}$ and $\varphi^{(2)}$ as follows

1. $\varphi^{(1)}(w, P, F) \equiv (P \subseteq seg(w)) \wedge (F \cap seg(w) = \emptyset)$ (*strong predicate*)
2. $\varphi^{(2)}(w, P, F) \equiv (seg(w) \cap P \neq \emptyset) \wedge (F \cap seg(w) = \emptyset)$ (*weak predicate*)

In the following, we work with this extension over segments instead of symbols. We can use single symbols depending on the definition of P and F. The construction of these predicates is based on random-context conditions defined by the sets P (*permitting contexts*) and F (*forbidding contexts*). Let V be an alphabet and $L \subseteq V^*$, and let $\beta \in \{(1), (2)\}$, we define $\varphi^{\beta}(L, P, F) = \{w \in L | \varphi^{\beta}(w; P, F)\}$.

In the following, we introduce some basic concepts of computational complexity theory. A *decision problem* is a (mathematically defined) problem where the answer is the affirmation or negation of a predicate over the parameters of the problem. A deterministic algorithm is an algorithm that given a particular input, it always produce the same output. A nondeterministic algorithm is an algorithm that can exhibit different behaviors on different runs. A nondeterministic algorithm for a decision problem has two different phases: the guessing phase and the checking phase. In the guessing phase a possible solution is created randomly without complexity cost. In the checking phase the algorithm checks if this possible solution is a good real solution. The time complexity in nondeterministic algorithms is measured only by the checking phase. The complexity class P contains all the decision problems that can be solved by deterministic algorithms in polynomial time. The complexity class NP contains all the decision problems that can be solved by nondeterministic algorithms in polynomial time. It is a classical and open problem to know whether $P = NP$ or not.

Accepting Networks of Genetic Processors

Now, we provide the basic concepts and definitions for the Networks of Genetic Processors as they were introduced in [CS, 2012]. In the same work, a formal proof of the computational completeness and the relationships between NGPs and Parallel Genetica Algorithms are showed.

Let V be an alphabet. A genetic processor N over V is defined by the tuple $(M_R, A, PI, FI, PO, FO, \alpha, \beta)$, where:

- M_R is a finite set of mutation rules over V
- A is a multiset of strings over V with a finite support and an arbitrary large number of copies of every string.
- $PI, FI \subseteq V$ are finite sets with the input permitting/forbidding contexts
- $PO, FO \subseteq V$ are finite sets with the output permitting/forbidding contexts
- $\alpha \in \{1, 2\}$ defines the function mode with the following values
  - If $\alpha = 1$ the processor applies mutation rules
  - If $\alpha = 2$ the processor applies crossover rules and $M_R = \emptyset$
- $\beta \in \{(1), (2)\}$ defines the type of the input/output filters of the processor. More precisely, for any word $w \in V^*$ we define an input filter $\rho(w) = \varphi^{\beta}(w, PI, FI)$ and an output filer $\tau(w) = \varphi^{\beta}(w, PO, FO)$. That is, $\rho(w)$ (resp. $\tau(w)$) indicates whether or not the word w pass the input (resp. the output) filter of the processor. We can extend the filters to act over languages. Thus, $\rho(L)$ (resp. $\tau(L)$) is the set of words of L that can pass the input (resp. output) filter of the processor.

A Network of Genetic Processors (NGP) of size n is defined by the tuple $\Pi = (V, N_1, N_2, ..., N_n, G, \mathcal{N})$, where V is an alphabet, $G = (X_G, E_G)$ is a graph, N_i $(1 \leq i \leq n)$ is a genetic processor over V, and $\mathcal{N} : X_G \to \{N_1, N_2, ..., N_n\}$ is a mapping that associates the genetic processor N_i to each node $i \in X_G$.

We distinguish two types of Networks of Genetic Processors: The *accepting* one and the *generating* one. In the accepting case, the network will be denoted by ANGP and it has two distinguished processors, the *input* and the *output* processors, N_{input} and N_{output} respectively. A *configuration* of an ANGP $\Pi = (V, N_1, N_2, ..., N_n, G, \mathcal{N})$ is defined by the tuple $C = (L_1, L_2, ...L_n)$, where L_i is a multiset of strings defined over V for all $1 \leq i \leq n$. A configuration represents the multisets of strings which are present in any processor at a given moment (remember that every string appears in an arbitrarily large number of copies). The initial configuration of the network is $C_0 = (A_1, A_2, ...A_n)$. Observe that since the input string w is allocated in the input node, $A_{input} = \{w\}$, while the output node is empty, so $A_{output} = \emptyset$.

Every copy of any string in L_i can be changed by applying a *genetic step* in accordance with the mutation or crossover rules associated with the processor N_i. Formally, we say that the configuration $C_1 = (L_1, L_2, ...L_n)$ directly changes into the configuration $C_2 = (L'_1, L'_2, ...L'_n)$ by a genetic step, written as $C_1 \Rightarrow C_2$, if L'_i is the multiset of strings obtained by applying the mutation or crossover rules of N_i to the strings in L_i. An arbitrarily large number of copies of each string is available in every node. Therefore, after a genetic step, one gets an arbitrarily large number of copies of any string, which can be obtained by using all possible mutation or crossover rules associated with that node. By definition, if L_i is empty for some $1 \leq i \leq n$, then L'_i is empty as well. In a *communication step*, each processor N_i sends all copies of the strings to all the processor connected to N_i according to G, provided that the strings are able to pass its output filter of N_i. In addition, it receives all copies of the strings sent by any processor connected to N_i according to G, provided that they can pass its input filter. Formally, we say that the configuration C' is obtained in one communication step from configuration C, written as $C \vdash C'$, iff

$$C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))) \text{ for all } x \in X_G \quad (1)$$

Let $\Pi = (V, N_1, N_2, ..., N_n, G, \mathcal{N})$ be an ANGP. A *computation sequence* in Π will be a sequence of configurations $C_0, C_1, \cdots$, where C_0 is the initial configuration of Π, $C_{2i} \Rightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$ for all $i \geq 0$. This sequence must be maximal (no other sequence follows from the previous one).

We will consider that a sequence of configurations is finite whenever at least one of the following two conditions holds:

1. The output node contains at least one string. That is, if N_k is the designated output node then $L_k \neq \emptyset$. In this case we say that the network accepts the input string.

2. In a genetic step the operations cannot be applied (the strings at every processor do not change) and no string is received or transmitted in the next communication step. After two consecutive genetic or communication steps, the configuration of the network does not change.

The language accepted by an Accepting Network of Genetic Processors is the set of input strings such as the network halts with at least one string in the output processor. Observe that, from the definitions given above, any ANGP is a deterministic device and we can predict the network behavior from a given input configuration.

We can establish the following time complexity measure for the ANGP model: Let us consider an ANGP R and the language L accepted by R. 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. Observe that this measure is not defined whenever the machine does not halt or it repeats two nonacceptance consecutive configurations. This measure, as defined above, fully satisfies Blum's axioms for abstract complexity measures [Blum, 1967]. In addition, we can define the partial function $Time_R : \mathbb{N} \to \mathbb{N}$ as follows:

$$Time_R(n) = max\{Time_R(x) : x \in L(R), |x| = n\} \tag{2}$$

If we take an integer function $f : \mathbb{N} \to \mathbb{N}$ then we can define the following language class provided that there exists an ANGP R that holds the property required in the definition

$$\begin{aligned} Time_{ANGP}(f(n)) = \{L : \text{There exits an } ANGP, R \text{ and a natural number } n_0 \\ \text{such that } L = L(R) \text{ and } \forall n \geq n_0 \ \ Time_R(n) \leq f(n)\} \end{aligned} \tag{3}$$

Finally, for a set of integer functions $\mathbb{C}$ we define

$$Time_{ANGP}(\mathbb{C}) = \bigcup_{f \in \mathbb{C}} Time_{ANGP}(f(n)) \tag{4}$$

We will consider the function set $poly$ as the set of integer polynomial functions, and we denote $Time_{ANGP}(poly)$ by $PTime_{ANGP}$. It has been proven in [CS, 2012] that $NP \subset PTime_{ANGP}$. So, all NP problems can be solved by ANGPs in polynomial time. In the following section, we select a NP-complete problem in order to test the practical implications of the previous result.

The Hamiltonian Cycle Problem

In order to test NGPs as decision problem solvers, we have selected a well known combinatorial problem: the Hamiltonian Cycle Problem (HCP). This problem belongs to Karp's list of NP-complete problems that were published in his landmark paper on NP-completeness [Karp, 1972]. In addition, the same problem was selected by Adleman to give a proof of concept for DNA computing solving [Adleman, 1994]. The HCP can be defined as follows: In graph theory, a *path* is a sequence of vertices such that from each of hem there is an edge to the next vertex in the sequence. A Hamiltonian cycle is a path that includes every vertex of the graph exactly once, and there exists an edge from the last vertex in the sequence to the first one. The (undirected) Hamiltonian Cycle Problem is the problem of determining whether, given an (undirected) graph, a Hamiltonian cycle exits.

Proposal of an ANGP to solve the HCP

Before we start to define an ANGP to solve the HCP, we must propose an encoding of the input graph instance. Given that the size of the graph (the number of vertices it has) is unbounded, and the input alphabet of the ANGP is finite, we will encode each vertex as a string of symbols. In order to make our experiments in an easy way, we will consider only those graphs that have at most 50 vertices. Observe that for those graphs with a number of vertices bigger than 50, we can provide the same solution that we are carrying out. The encoding of vertices to strings have a linear time complexity, so the results will not be affected by this aspect.

Given a graph $G = (D, E)$ where D is a set of n vertices, with $n \leq 50$, and E is a set of edges, we propose the following ANGP R:

$R = (V, N_0, N_1, N_2, ..., N_{49}, N_c, N_{pc}, N_{out}, K_{53}, f)$, where $V = \{0, 1, 2, ...49\} \cup \{0', 1', 2', ...49'\} \cup \{\#, \#', 0^*\}$, where K_{53} is a star graph with 53 nodes. The graph have two connected central nodes N_c and N_{pc} that are connected with the rest of nodes $N_i : 0 \leq i \leq 49$. The output node N_{out} is only connected to N_{pc}.

The processors are defined as follows:

For every vertex $q \in D$
$N_q = (\{\# \rightarrow q'\}, \emptyset, \{p\# : (p, q) \in E\} \cup \{0^*\# : (0, q) \in E\}, \{q\}, \{q'\}, \emptyset, 1, (2))$

$N_c = (\emptyset, \{\#'\}, \{n' : 0 \leq n \leq 49\}, \{\#, \#'\}, \{n' : 0 \leq n \leq 49\} \cup \{n : 0 \leq n \leq 49\} \cup \{\#'\#', 0^*\}, \emptyset, 2, (2))$

$N_{pc} = (\{q' \rightarrow q : 0 \leq q \leq 49\} \cup \{\#' \rightarrow \#\}, \{0^*\#\}, \{n'\#' : 0 \leq n \leq 49\}, \emptyset, \{n : 0 \leq n \leq 49\} \cup \{0^*\#\}, \{n' : 0 \leq n \leq 49\} \cup \{\#'\}, 1, (2))$

$N_{out} = (\emptyset, \emptyset, \{n : 0 \leq n \leq 49\}, \emptyset, \emptyset, \emptyset, 1, (1))$

The computation of the network starts in the N_{pc} processor, with the sequence $0^*\#$ given that we consider that the cycle starts at vertex 0. The $\#$ symbol marks the end of the string and it is used to add new vertices to the cycle. The sequence leaves N_{pc} and goes to processors $N_i : 0 \leq i \leq 49$ where $(0, i) \in E$. Every processor $N_i : 0 \leq i \leq 49$ has a rule that changes the $\#$ symbol for the symbol i (the name of the vertex). Then, the sequence goes to the processor N_c that is used to add the symbol $\#'$ at the end. This sequence goes to N_{pc} that cleans all the marks from all the symbols. The most interesting aspect of the sequence from N_{pc} is that the symbols before the $\#$ represents the last vertex from the path, that is the vertex j. In the next computation step, the sequences go only to the processors $N_i : 0 \leq i \leq 49$ where $(j, i) \in E$ (this process is carried out by the filters of each processor). The network repeats this process until the computation ends. If during the computation, a string gets a path with all the nodes, this string goes to the N_{out} processor, the computation ends and the answer to the problem is YES. On the other hand, if the graph does not have a Hamiltonian cycle there is a computation step where the strings cannot be communicated to any processor $N_i : 0 \leq i \leq 49$. Then, no string is received or transmitted and the configuration of the network does not change. So, the network halts and the answer to the problem is NO.

Experimental Results

We have considered only instances of graphs with Hamiltonian cycles. This is the case to measure the complexity in the worst case, given that the number of steps in the computation, for each graph, is the highest when it has a cycle. The reason for this is that the computation of the NGP consists in creating a Hamiltonian path. If the input graph does not have a Hamiltonian cycle the process ends before, and the number of steps is lower (this result has been observed empirically).

The graphs used in the experiments have a different number of vertices and edges. The table in the Figure 1 shows the results obtained by the experiments.

vertices	edges	Computation steps
5	8	40
5	10	40
5	12	40
10	18	80
10	20	80
10	22	80
15	28	120
15	30	120
15	32	120
20	38	160
20	40	160
20	42	160

Figure 1: Computation steps to solve The Hamiltonian Cycle Problem using Networks of Genetic Processors.

We can observe that the number of edges does not affect to the number of computation steps. The NGP can check all the edges at the same time because the edges are represented by the filters of every processor $N_i : 0 \leq i \leq 49$ and they work in pararell.

If n is the number of vertices, the number of computational steps used by the NGP to check if the graph have a Hamiltonian cycle is a most $8 * n$ ($O(n)$). In order to add one vertex to the path, the network makes 8 computation steps: one, in N_i, to change the $\#$ simbol to i', one to send the sequence from N_i to N_c, one to add $\#'$ at the end, one to go to N_{pc}, three to clean the sequence (two genetic steps plus one communication step), and one to return the sequence to N_j processor. If the graph has a hamiltonian cycle, the network repeats this process n times. So, the network performs at most $n * 8$ computation steps. This results gives a sign that HCP belongs to $Ptime_{ANGP}$ as it has been theoretically proved [CS, 2012].

Conclusions and Future Work

In a previous work [CS, 2012], we proposed a new model of computation, the Networks of Genetic Processors. In the same work we proved that the model is computational complete and we defined a time complexity measure for it.

In this work, we have carried out some experiments to observe the behavior of the NGPs to achieve the time complexity that has been theoretically proved. We have selected a NP-complete problem, the Hamiltonian Cycle Problem, and we have implemented an ANGP to solve it. The results of the experiments give a sign that the NGPs can solve NP problems in polynomial time (nevertheless, the NGPs work in an ideal parallel model and some practical aspects must be taken into account). For the Hamiltonian Cycle Problem, we can implement an ANGP where the computation steps depend only on the number of vertices (not on the number of the edges), and with a linear time complexity.

For our future research, we will consider other types of problems such as the optimization problems (i.e. the optimization Travelling Salesman Problem or the Knapsack Problem). For this case, we need to define a new kind of NGPs that work like a set of Parallel Genetic Algorithms. In addition, the implementation of NGPs in hardware to exploit the parallelism of the model will be considered too.

Bibliography

[KR, 2008] L. Kari, G. Rozenberg. The Many Facets of Natural Computing. *Communications of the ACM* Vol. 51 No.10, pp 72-83. 2008.

[Adleman, 1994] L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science* 226, pp 1021-1024. 1994.

[CS, 2012] M. Campos, J.M. Sempere. Accepting Networks of Genetic Processors are computationally complete. *Theoretical Computer Science*(2012), doi:10.1016/j.tcs.2012.06.028.

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

[Karp, 1972] R.M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum. pp. 85103

[MMM, 1949] 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.

[Blum, 1967] M. Blum. A Machine-Independent Theory of the Complexity of Recursive Functions. *Journal of the ACM*, Vol 14, No. 2. pp 322-336. 1967.

[HU, 1979] J.E. Hopcroft, J.E. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publishing Co. 1979.

[DK, 2000] D. Du, K. Ko. *Theory of Computational Complexity.* John Wiley & Sons, Inc. 2000.

Authors' Information

Marcelino Campos *Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia (Spain); e-mail: mcampos@dsic.upv.es*
Major Fields of Scientific Research: Formal Languages Theory, Computation and Learning, Bioinformatics.

José M. Sempere *Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia (Spain); e-mail: jsempere@dsic.upv.es*
Major Fields of Scientific Research: Formal Languages Theory, Computation and Learning, Bioinformatics.