Enzyme Computation

Computing the way proteins do

Jaime-Alberto PARRA-PLAZA
Computer Science, Avispa Group, Pontificia Universidad Javeriana
Cali, Colombia; jparra@javerianacali.edu.co

Jaime VELASCO-MEDINA
Bionanoelectronics, Universidad del Valle
Cali, Colombia; jvelasco@univalle.edu.co

and

Eduardo CAICEDO-BRAVO
Perception and Intelligent Systems, Universidad del Valle
Cali, Colombia; ecaicedo@univalle.edu.co

ABSTRACT

It is presented enzyme computation, a computational paradigm based on the molecular activity inside the biological cells, particularly in the capacity of proteins to represent information, of enzymes to transform that information, and of genes to produce both elements according to the dynamic requirements of a given system. The paradigm explodes the rich computational possibilities offered by metabolic pathways and genetic regulatory networks and translates those possibilities into a distributed computational space made up of active agents which communicate through the mechanism of message passing. Enzyme computation has been tested in diverse problems, such as image processing, species classification, symbolic regression, and constraints satisfaction. Also, given its distributed nature, an implementation in dynamical reconfigurable hardware has been possible.

Keywords: Bioinspired systems, computational agents, computational models, cytocomputation, distributed computation, parallelism.

1. INTRODUCTION

It is very well known the enormous success that bioinspired approaches have obtained in solving problems such as optimization, classification, searching, or constraint satisfaction [1]. However, it is also a fact, and a concern for most of the researchers in the field, the evident dependence with respect to the right choice of values for a set of uncorrelated parameters for this success to come [2]. Nonetheless, when turning to Nature for new inspirations it is evident that it has already solved this problem, so, a closer look at how this has been accomplished can be useful for trying to minimize the impact of these parameters in the relative success of the technique [3].

It is in this spirit that enzyme computation is proposed, as a framework for solving problems inspired in the processes that are performed inside the cytoplasm of biological cells through the interaction of proteins and enzymes and among these and genes. The enzymatic approach is part of a major concept, the cytocomputational paradigm [4]. Currently, it has three parts, all of them concurrent: a linear genotype, which produces resources (proteins and enzymes) on demand; a set of proteins, which hold incoming and temporal data and that are arranged in a regular geometric space, able to perform local computation; and a set of enzymes acting upon proteins to perform transformations. In designing the framework, explicit parameters requiring careful tuning have been avoided. Instead, each part regulates and is regulated by the others, resembling the metabolic pathways and the genetic regulatory networks found in biological cells. The fitness of a given representation is tested according to the patterns presented in the current problem and this establishes if that particular component becomes specialized, generic, or silenced.

A solution to a problem runs in this way: first, it is necessary to set genes for the resources to be allocated, been these initial information, i.e, state, data, and functions. All of these can be added or eliminated at any time, as the computational space is naturally concurrent and distributed. Second, genes express enzymes, which process proteins if possible. The enzyme selected for a given protein is assigned according to a matching matrix, which is updated from previous solvers in order to gain insight into common patterns.

If the computational space gets stuck, this becomes a pressure for silenced genes to express or for active genes to change, following a variety of evolutionary processes, but without dependency on representation or position, i.e., each gene expresses a whole, functional product, and it is the interactions among these products that evolves. For that, every product is tagged with a concentration factor and a shape value, and evolution implies fitting these two numbers; in this way, as the process advances, products become part of disjoint sets. It is this specialization that is able to generate meaningful patterns and to minimize the dependency with respect to preset parameters.

This paper is organized as follows: section 2 gives the biological support. Section 3 presents the computational model. Section 4 shows several illustrative examples. Section 5 offers the conclusions and future work.

2. BIOLOGICAL CONCEPTS

All living beings on Earth share the same biochemical machinery present in every one of their constituent cells. Inside a cell, three macromolecules outstand as the main actors when modeling its computational powers (see Figure 1): genes, proteins, and enzymes. Although enzymes are also proteins, their active role as protein transformers justifies making this explicit distinction, leaving the name protein only for those macromolecules exhibiting very low capacities of transformation [5].

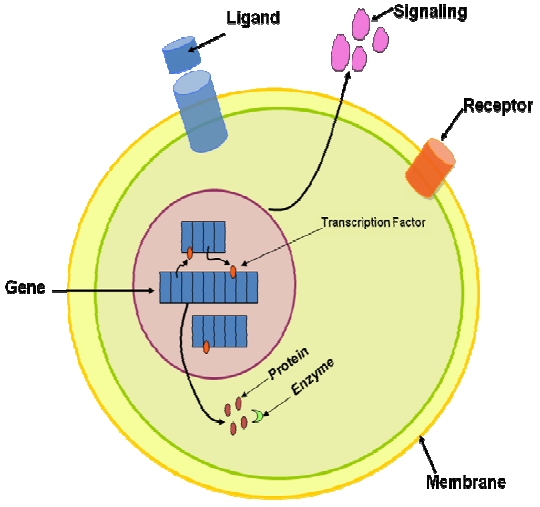


Figure 1. Protein interactions in a cell.

Three structures are clearly identifiable in Figure 1: nucleus, cytoplasm, and membrane. Nucleus stores the genetic code as a collection of genes. A gene normally is in charge of producing a protein or an enzyme, and its activity is regulated by special proteins called transcription factors (TF). Cytoplasm is the place where proteins and enzymes interact. Enzymes transform proteins modifying their properties. Membrane holds another kind of special proteins, receptors, which become specialized in reacting with other special proteins, ligands. Finally, some proteins may leave the cell and interact with other cells.

Gene structure

A gene in cytocomputation is composed, at least, of four parts: promoter, operator, exon, and intron (see Figure 2.) Promoter and operator make up the header of the gene and are intended to allow its regulation from other genes. Exons and introns conform the body of the gene and are in charge of producing resources either enzymes or proteins. In fact, products are only generated from exons; introns are a kind of disabled exons, susceptible to become active through mutations. In the biological system, exons do not directly produce proteins, but an intermediate called messenger RNA (mRNA). The intermediate generated by introns, called small nuclear RNA (snRNA), acts as another level of regulation and biologists are just starting to discover its principles [6-7].

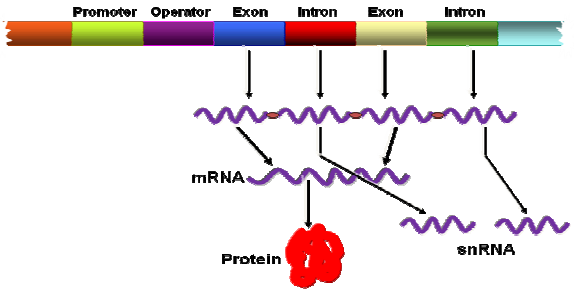


Figure 2. Gene structure

Gene regulation

Genes regulate each other through an interaction between cis elements and trans elements. Cis elements are the ones abovementioned, promoters and operators, which are constituents of every gene. Trans elements are a special kind of proteins, called transcription factors (TF), released as a product from the exons of genes. Transcription factors produced by a gene may regulate the activity of any gene, including the one that generated them [8-9].

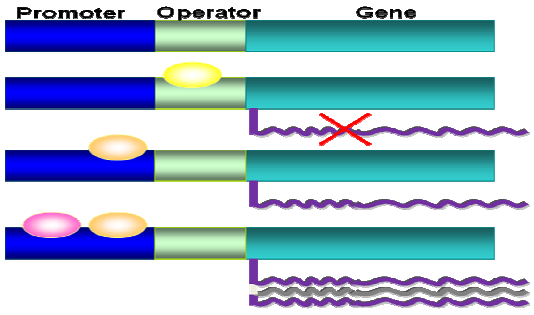


Figure 3. Gene regulation

The activity of a gene depends on the state of its promoter and operator. The promoter enables the gene's activity whereas the operator inhibits it. In order for a gene to be

operational, it is necessary that its promoter be bound to its TF. Two strategies exist in biological genes: promoter's regulation is either all-or-nothing or proportional. The former occurs when the TF must be all present, resembling the control performed by a logical AND gate; the latter happens when TF act as facilitators, working in an incremental fashion, i.e., the more TF present, the more product will be released by the gene (see Figure 3 bottom.). The control established by the operator follows a similar pattern: it can be either digital or analogical. In the first case, TF are as the inputs to a logical OR gate and the presence of any of them disable the gene. In the second case, the activity of the gene is reduced proportionally to the number of TF present in the operator [10].

Protein structure

A protein is made up of amino acids (aminos for short), attached each other through chemical bonds (see Figure 4.) An amino can set connections not only to its immediate neighbors, but to distant ones, and in this way a protein is folded in a tridimensional shape. Another possibility for an amino is becoming phosphorilated: this happens when a residue is attached to it, modifying its behavior. Both alternatives are used in the cytocomputational model [11].

Figure 4. Protein structure

Enzyme processing

Modifications to a protein are sometimes motivated by the protein itself, but can be also forced externally through the participation of an active agent: an enzyme. An enzyme has the capacity to alter the internal structure of a protein, using either the changing in bonds or the adding of phosphate residues (see Figure 5.) In this case, the enzyme is like an active agent, and the protein is a kind of passive component. After being attacked by an enzyme, a protein may suffer different transformations (see Figure 6): broke up in two or more new proteins, joined to another or other proteins, modified in structure (components added or deleted), modified in functionality (components changed.)

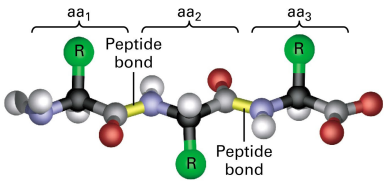


Figure 5. Protein with residues

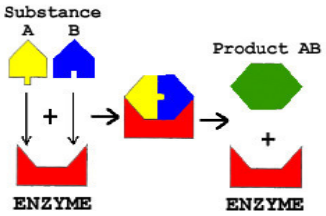


Figure 6. Enzyme activity

Pathways and networks

A protein can suffer successive transformations through its exposition to different enzymes. Because through any of these modifications a protein may become the source product for itself, many self-regulating scenarios are possible. In the cytoplasm, metabolic pathways are created when a cascade of successive transformations produces a set of proteins from a unique protein. In the same way, after modification, a protein may become a TF, allowing for regulation of an entire process. Even, a protein may regulate the rate of its own production. Genetic regulation networks occurs when a set of genes regulate each other through TF created inside the own set [12-13].

3. COMPUTATIONAL MODEL

Enzyme computation is built on the base of modeling sets of agents which interact among them following the rules present in metabolic pathways and genetic regulatory networks. In enzyme computation, problems are represented as records, holding the required information for the system to solve them. Figure 7 shows one of such records, written in Oz code [14-15]. *Input* holds the original information of the problem. *Output* is the place where the system will put the transformations and results obtained. *Process* is in charge of carrying out the transformations required, and a *finish* statement describes what condition must be met to end the process and what action to take.

```
c(
   input:Input
   output:Output
   finish:'cond'(condition:FinishCondition
                 action:FinishAction)
   process:Process
)
```

Figure 7. Problem descripcion

This record is received by a *solver,* entity in charge of creating, activating, and sensing the computational space which will solve the problem described. The Solver takes the problem and creates the structures necessary for solving it. Figure 8 shows an example for the case that the problem will be solved through a set of cooperative agents resembling the aminos of a protein.

```
fun {Solver Problem}
   DiffType = {Label Process}          Status
   ProteinInitState = c(input:Input    output:Output
            finish:c(finishCondition:FinishCondition
                     finishAction:FinishAction)
                     agent:Protein    status:Status)
   Protein = {AgentProtein ProteinInitState}
   Amino = {Tuple.make c {Record.width Input}}
in
   for I in 1..{Record.width Input} do
      InitState = c(mypos:I myval:Input.I
         neighs:DiffNeighs.DiffType
         agent:Amino    protein:Protein process:Process)
   in
      Amino.I = {AgentAmino InitState}          end
   for I in 1..{Record.width Input} do
      {Send Amino.I start}                      end
    Status
end
```

Figure 8. Solver descripcion

Once the threads are allowed to work, the enzyme system will attack the proteins created, modifying them until one of two conditions matches: the finishing condition is met or the computational space becomes stable (no more message traffic). The proper condition will be informed through the *status* flag, which can be sensed by other processes for synchronization purposes.

4. EXAMPLES

This section will present several examples extracted for different fields solved by enzyme computation. They are intended to show the main characteristics of the paradigm when dealing with real problems.

Cooperative computation

There exist many problems which are parallel by definition but are usually seen as sequential because of the way they are traditionally solved. Consider for instance the problem of finding the maximum value in a vector of numbers. Any programmer will almost instantly conceive a solution implying a loop and a store. The store will hold the current maximum, and the loop will be used for comparing each number against such current maximum. When reaching the last position of the vector, the store will hold the global maximum value. Although this approach is simple, it hides the real characteristics of the problem, which disallows any attempt to take advantage of a particular disposition of the numbers, i.e., the traditional approach always requires to traverse the entire vector no matter the maximum value is already at the first position [16].

In enzyme computation the time required to find the solution is directly proportional to the *disorder* of the vector, it is to say, the more local maxima exist, the more is the time necessary to find the global maximum. On average, the time required is in the order of *NLogN,* being *N* the size of the vector and *Log* the logarithm base 2. In an extreme situation, if the maximum is in the middle of the vector, the solution can be obtained in only one step. Figure 9 shows a typical scenario and the successive transformations performed on the protein until it gets stable and holding the final solution. Observe how fast unhelpful information is discarded in every stage. The process followed is that every amino compares its value against those of its neighbors. Any amino having a neighbor greater than it will exclude itself from the protein. At any time, only those aminos that are greater or equal to both neighbors will be alive. Comparisons continue until no more bonds are removed, which indicates that a stable state has been reached and the solution lies in the surviving aminos.

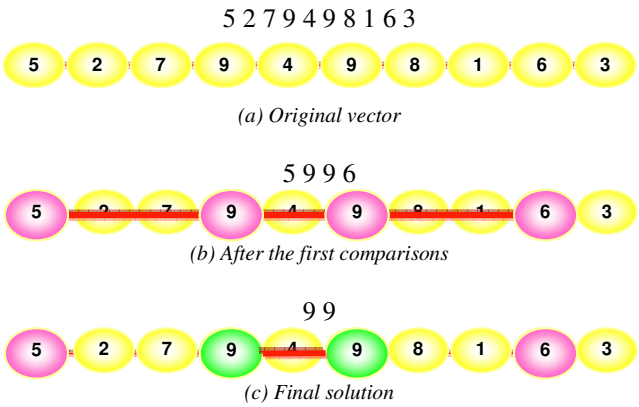


Figure 9. Problem solved by protein computation

Propagating computation

Consider finding the shortest path between two nodes in a directed graph. The entire graph is associated with a protein, where each node is an amino and its vertices are the chemical bonds attached to that amino. The process starts by phosphorilating the starting and ending nodes. The starting node then proceeds by exciting its neighbor bonds. Every time an amino senses activity at any incoming bond it enters an unstable state which is resolved when all these bonds become excited. At this point, the amino selects the least-valued bond and propagates its value to its outcome neighbors. At the same time, the strength of the winning bond is increased. When the ending node is reached, it no longer propagates new information and the computational space becomes stable, finishing the process. The path of interest can then be retrieved by visiting the aminos and traversing through their strongest bonds (see Figure 10, where the shortest path is S-a1-b2-E.)

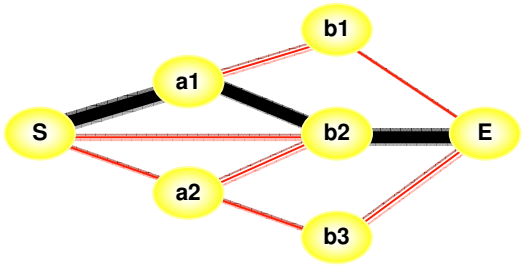


Figure 10. Problem solved by propagating computation

The process follows in this way: after the node S is excited, it excites its three output bonds, propagating their strengths to nodes a1, a2, and b2. Nodes a1 and a2 has only one input bonds, therefore they propagate that value and return to its stable state, but node b2 has more inputs and that requires entering a waiting state until all inputs become known. Every node will in turn receive input information, and will either wait for the rest of its nodes or propagate the weakest value to its output, strengthening that connection. When the ending node is solved, it will not propagate further information; instead, it will traverse back through the strongest connections until getting the starting node, time when the complete shortest path information will be available.

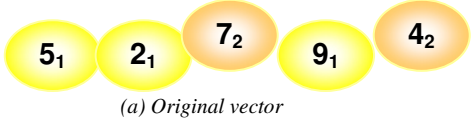
Phosphorilating computation

Be the task of sorting a vector of numbers. Using enzyme computation, the job is done in a distributed fashion, where each number is associated with a protein and all proteins are injected into the cytoplasm, becoming susceptible to be transformed by enzymatic catalysis. Initially, all proteins are created with a common phosphate residue, for instance the value 1. Enzymes choose randomly pairs of proteins and determine possible modifications in the phosphate residues based on this condition:

if protein1.val >= protein2.val
then protein1.phospho ++
else protein2.phospho ++

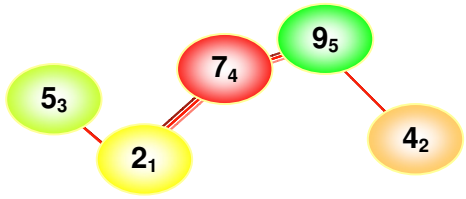
Enzymes only act upon proteins holding the same phosphate group, therefore, after some time, a snapshot of the process will show several computational spaces, each one with several proteins on it (see Figure 11.) Computation stops when the base space, the one with phosphate value 1, collapses to a single component. At this time, every space should have only one component and the sorted version of the vector can be retrieved by taking each element from the top space to the bottom.

5 2 7 9 4



(a) Original vector

(b) After 5-7 and 2-4 comparisons



(c) Final solution

Figure 11. Problem solved by phosphorilating computation

Competitive computation

Be the task of finding the roots of a polynomial. This task requires exploring a space solution in order to incrementally obtain a more accurate solution until satisfying a maximum error. One way of using genetic agents to obtain faster convergence is defining a candidate root as composed of three floating point numbers:

Root = Ss(Is.Fs)*10^Ps +*
Sm(Im.Fm)*10^Pm +*
Sl(Il.Fl)*10^Pl*

Where S=sign, I=integer part, F=fractional part, P=exponential power.

There exist three genes in charge of these three parts: small, medium, and large (corresponding to the subscripts s, m, and l in the above expression.) A regulation network is established among these genes. The idea is that gene *l* explores large portions of the space solution, whereas gene *m* works on a more localized portion of it, and gene *s* is dedicated to fine tuning the answer. Figure 12 shows a polynomial with four roots: -1000, -250, 5, and 820. Three candidate roots are also shown, represented by a circle, a square, and a triangle, commanded by their large, medium, and small genes respectively. It is to say; the closer a root is to an answer, the more control the small gene will have on the other genes. Once a root is found, its associate gene will tend to exclude other genes from searching in this area, provided that there are no multiple roots.

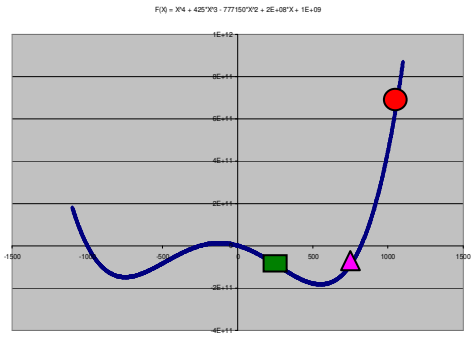


Figure 13. Competitive computation

Diffusion computation

Some problems have dynamic information that can not be easily represented as aminos or genes. Consider the case of a group of autonomous robots executing a collecting task. Each robot explores the working space looking for specific items to carry to some predefined place. In order to make the whole process more efficient, it would be desirable some kind of collaboration among the robots in order to reduce repetitive efforts. Moreover, it would be interesting to have some predictive capacity to assist in the decision of what type of robots to deliver.

Be a scenario like the one shown in Figure 14, where the rectangles represent storage places, the small circles are the supplies and the medium and big circles are two types of robots. A medium robot can carry one unit whereas a big robot can hold three items. Several behaviors are available for a robot such as moving straight, wandering, staying close the storage places, or passing its items to another robot. Each behavior is controlled by a gene, and regulation is performed through transcription factors produced by other robots and by the environment (storage places and supplies). Simulations show that several societies evolve. In one case, big robots stay closer to the stores while small robots wanders and deliver items to them. In another case, large quantities of supplies put in line promote offspring composed entirely by big robots moving straight.

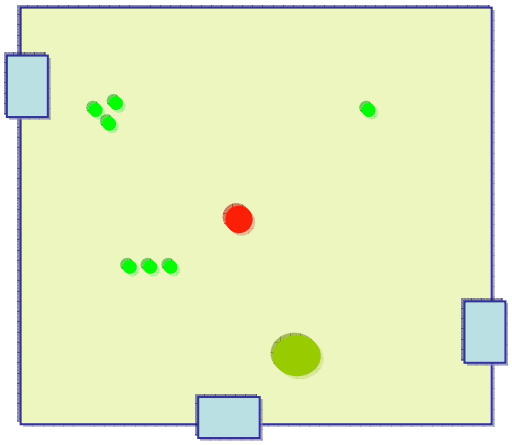


Figure 14. Cell computation

5. CONCLUSIONS AND FUTURE WORK

It was presented a computational paradigm based upon the molecular activity inside biological cells. The paradigm captures the way enzymes and genes work, allowing for a distributed computational space where flexible agents are able to modify and been modified in order to adapt themselves to changes in the input information.

Several examples where shown describing the versatility of the paradigm. Proteins permit to represent complex structures of information with different levels of interconnections. Enzymes allow modeling different transformation processes. Metabolic pathways let the designer describe spatial and temporal steps in more complex transformations. Promoters and operators allow fine tuning cooperative and competitive distributed systems. Finally, genetic regulatory networks provide a platform to solve problems requiring iterative and evolutionary approaches.

Future work will be devoted to extend the applicability of enzyme computation to more complex tasks such as control of paramedic services in a big city and lexicographical assistance in natural language processing.

6. ACKNOWLEDGMENTS

This work was supported in part by Colciencias, the Colombian Science Foundation, through a doctorate scholarship.

7. REFERENCES

[1] D. Floreano and C. Mattiussi, *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. MIT Press, 2008.

[2] S. Grand, *Creation: life and how to make it*. Harvard University Press, 2000.

[3] J. A. Parra-Plaza and J. Velasco, *"CytoComputation: a new computational paradigm inspired by biomolecular models"* Doctorate Thesis Proposal, Universidad del Valle, Cali, Colombia, 2007.

[4] J. A. Parra-Plaza, *"Cytoelectronics: a computational intelligence paradigm based upon the retrovirus dynamics"* in ICCI 2005 International Congress on Computational Intelligence, Montería, Colombia, 2005.

[5] P. F. Cook and W.W. Cleland. *Enzyme Kinetics and Mechanism.* Garland Science, 2007.

[6] J. Mattick, *"Challenging the dogma: the hidden layer of non-protein-coding RNAs in complex organisms*", in BioEssays, Vol. 25, 930-939, Oct 2003.

[7] J. Knight, "Gene regulation: Switched on to RNA", *Nature*, Volume 425, No 6955, 2003.

[8] E. H. Davidson and D. H. Erwin. "Gene Regulatory Networks and the Evolution of Animal Body Plans", *Science* 10 February 2006: Vol. 311. No. 5762, pp. 796 – 800, 2006.

[9] J. Knight, "Gene regulation: Switched on to RNA", *Nature*, Volume 425, No 6955, 2003.

[10] J. C. Bongard and R. Pfeifer. "Evolving Complete Agents Using Artificial Ontogeny", in *Morpho-functional Machines: The New Species (Designing Embodied Intelligence)*, F. Hara and R. Pfeifer Edds. Springer-Verlag, pp. 237-258, 2003.

[11] J. A. Parra-Plaza, *"A-Communities: an agent-based platform for cytocomputation"* in ICCI 2006 International Congress on Computational Intelligence, Bogota, Colombia, 2006.

[12] J. D. Watson, T. A. Baker, S. P. Bell, A. Gann, M. Levine, and R. Losick. *Molecular Biology of the Gene.* Benjamin Cummings, 2007.

[13] H. Lodish, A. Berk, C. A. Kaiser, and M. Krieger, *Molecular Cell Biology*, W. H. Freeman, 2007.

[14] P. Van Roy, *Concepts, Techniques, and Models of Computer Programming*. MIT Press, 2004.

[15] www.mozart-oy.org

[16] D. Le M'etayer, *"Higher-order multiset programming"*, in Proc. of the DIMACS workshop on specifications of parallel algorithms, American Mathematical Society, Dimacs series in Discrete Mathematics, 1994.