Information Sciences 265 (2014) 189–197

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Networks of polarized evolutionary processors ☆

Pedro P. Alarcón [a], Fernando Arroyo [b], Victor Mitrana [c,a,*]

[a] Department of Organization and Structure of Information, University School of Informatics, Polytechnic University of Madrid, Crta. de Valencia km. 7, 28031 Madrid, Spain
[b] Department of Languages, Projects and Computer Information Systems, University School of Informatics, Polytechnic University of Madrid, Crta. de Valencia km. 7, 28031 Madrid, Spain
[c] Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, 010014 Bucharest, Romania

ARTICLE INFO

Article history:
Received 10 November 2012
Received in revised form 2 May 2013
Accepted 29 December 2013
Available online 7 January 2014

Keywords:
Network of polarized evolutionary processor
Problem solver
3-Colorability problem
Common Algorithmic Problem

ABSTRACT

In this paper, we consider a new variant of networks of evolutionary processors which seems to be more suitable for a software and hardware implementation. Each processor as well as the data navigating throughout the network are now considered to be polarized. While the polarization of every processor is predefined, the data polarization is dynamically computed by means of a valuation mapping. Consequently, the protocol of communication is naturally defined by means of this polarization. This new variant is investigated here as a problem solver. We propose solutions based on networks of polarized evolutionary processors to two computationally hard problems, namely the "3-colorability problem", and a more general one known as the "Common Algorithmic Problem". Our solutions are uniform (they work for all instances of the same size) and time efficient (they work in linear time).

1. Introduction

Networks of evolutionary processors (NEPs) form a class of highly parallel and distributed computing models inspired and abstracted from the biological evolution. Informally, a network of evolutionary processor consists of a virtual (complete) graph in which each node hosts a very simple processor called evolutionary processor. By an evolutionary processor we mean a mathematical construction which is able to perform very simple operations inspired by the point mutations in DNA sequences (insertion, deletion or substitution of a single base pair). By an informal parallelism with the natural process of evolution, each node may be viewed as a cell having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node processor, which is specialized just for one of these evolutionary operations, acts on the local data and then local data becomes a mobile agent which can navigate in the network following a given protocol. Only that data which is able to pass a filtering process can be communicated. This filtering process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies.

It is worth mentioning that NEPs resemble a pretty common architecture for parallel and distributed symbolic processing, related to the Connection Machine [9] which was defined as a network of microprocessors in the shape of a hypercube. Each

☆ A preliminary version of this paper has appeared in 16th Annual Conference on Advances in Knowledge-Based and Intelligent Information and Engineering Systems – KES 2012, IOS Press 2012, 807–815.

* Corresponding author at: Department of Organization and Structure of Information, University School of Informatics, Polytechnic University of Madrid, Crta. de Valencia km. 7, 28031 Madrid, Spain. Tel.: +34 666317217.
E-mail addresses: pedrop.alarcon@eui.upm.es (P.P. Alarcón), farroyo@eui.upm.es (F. Arroyo), victor.mitrana@upm.es (V. Mitrana).

0020-0255/$ - see front matter © 2014 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.12.036

microprocessor was very simple, processing one bit per unit time. Also it is closely related to the tissue-like P systems [14] in the membrane computing area [16].

NEPs as language generating devices and problem solvers have been considered in [1,15], respectively. They have been further investigated in a series of subsequent works. NEPs as accepting devices and problem solvers have been considered in [13]; later on, a characterization of the complexity classes **NP**, **P**, and **PSPACE** based on accepting NEPs has been reported in [11]. Universal NEPs and some descriptional complexity problems are discussed in [10]. The reader interested in a survey of the main results regarding NEPs is referred to [12].

Software implementations of NEPs have been reported, see, e.g., [3,4,7], most of them in JAVA. They encountered difficulties especially in the implementation of filters. The main idea to simulate the non-deterministic behavior of NEPs has been to consider a safe-thread model of processors, that is to have each rule and filter in a thread, respectively. Clearly the threads corresponding to the filters are much more complicated than those associated with the evolutionary rules. Configuration changes in a NEP are accomplished either by a communication step or by an evolutionary step, but these two steps may be realized in any order. This suggests that evolution or communication may be chosen depending on the thread model of processor [4]. The input and output filters are implemented as threads extending the `Runnable` interface. Therefore a processor is the parent of a set of threads, which use all objects from that processor in a mutual exclusion region. When a processor starts to run, it starts in a cascade way the rule threads and filter threads. As one can see, the filters associated with processors, especially if there are both input and output filters, seem to be hardly implementable. Consequently, it would be of interest to replace the communication based on filters among processors by another protocol. A first attempt was to move filters from each node to the edges between the nodes, see, e.g., [5]. Although this variant seems to be theoretically simpler, the attempts towards an implementation have encountered similar difficulties due to the fact that the filters associated with edges are similar to those associated with nodes.

To this aim, this work proposes a new variant of NEP and discusses the potential of this variant for solving hard computational problems. The main and completely new feature of this variant is the valuation mapping which assigns to each string an integer value, depending on the values assigned to its symbols. Actually, we are not interested in computing the exact value of a string, but just the sign of this value. By means of this valuation, one may metaphorically say that the strings are electrically polarized. Thus, if the nodes are polarized as well, the strings migration from one node to another through the channel between the two cells seems to be more natural and easier to be implemented.

We consider here networks of polarized evolutionary processors (NPEP) as problem solvers. When discussing the potential of a given NPEP to solve a problem one may say that one designs a NPEP-algorithm. We propose a NPEP-algorithm for a well-known **NP**-complete problem, namely the "3-colorability" problem, that requires a linear time in the size of the given instances. Further on, we consider another **NP**-complete problem, namely the "Common Algorithmic Problem", which is not a decision problem anymore, and solve it by means of an efficient NPEP-algorithm.

2. Preliminaries

We start by summarizing the notions used throughout this work. An *alphabet* is a finite and non-empty set of symbols. The cardinality of a finite set A is written $card(A)$. Any finite sequence of symbols from an alphabet V is called *string* over V. The set of all strings over V is denoted by V^* and the empty string is denoted by ε. The length of a string x is denoted by $|x|$ while $alph(x)$ denotes the minimal alphabet W such that $x \in W^*$. Furthermore, $|x|_a$ denotes the number of occurrences of the symbol a in x.

A homomorphism from the monoid V^* into the monoid (group) of additive integers $\mathbf{Z}$ is called *valuation* of V^* in $\mathbf{Z}$.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ and $ab \neq \varepsilon$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V are denoted by Sub_V, Del_V, and Ins_V, respectively. Given a rule σ as above and a string $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma(w) = \begin{cases} \{ubv : \ \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{otherwise.} \end{cases}$
- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma(w) = \begin{cases} \{uv : \ \exists u, v \in V^* \ (w = uav)\}, \\ \{w\}, \text{otherwise.} \end{cases}$
- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then $\sigma(w) = \{uav : \ \exists u, v \in V^* \ (w = uv)\}$.

Note that the action mode of an evolutionary rule applied to a string w: it returns the set of all strings that may be obtained from w depending on the position in w where the rule was actually applied.

For every rule σ and $L \subseteq V^*$, we define the action of σ on L by $\sigma(L) = \bigcup_{w \in L} \sigma(w)$. Given a finite set of rules M, we define the action of M on the string w and the language L by:

$$M(w) = \bigcup_{\sigma \in M} \sigma(w) \text{ and } M(L) = \bigcup_{w \in L} M(w),$$

respectively.