NETWORKS OF EVOLUTIONARY PROCESSORS (NEP) AS DECISION SUPPORT SYSTEMS

Miguel Angel Díaz, Nuria Gómez Blas, Eugenio Santos Menéndez,
Rafael Gonzalo, Francisco Gisbert

Abstract: This paper presents the application of Networks of Evolutionary Processors to Decision Support Systems, precisely Knowledge-Driven DSS. Symbolic information and rule-based behavior in Networks of Evolutionary Processors turn out to be a great tool to obtain decisions based on objects present in the network. The non-deterministic and massive parallel way of operation results in NP-problem solving in linear time. A working NEP example is shown.

Keywords: Natural Computing, Networks of Evolutionary Processors, Decision Support Systems.

ACM Classification Keywords: F.1.2 Modes of Computation, I.6.1 Simulation Theory, H.1.1 Systems and Information Theory.

Introduction

There are many approaches to decision-making and because of the wide range of domains in which decisions are made; the concept of decision support system (DSS) is very broad. A DSS can take many different forms. In general, we can say that a DSS is a computerized system for helping make decisions. A decision is a choice between alternatives based on estimates of the values of those alternatives. Supporting a decision means helping people working alone or in a group gather intelligence, generate alternatives and make choices. Supporting the choice making process involves supporting the estimation, the evaluation and/or the comparison of alternatives. In practice, references to DSS are usually references to computer applications that perform such a supporting role [Alter, 1980].

Abbreviated DSS, the term refers to an interactive computerized system that gathers and presents data from a wide range of sources, typically for business purposes. DSS applications are systems and subsystems that help people make decisions based on data that is culled from a wide range of sources. For example: a national on-line book seller wants to begin selling its products internationally but first needs to determine if that will be a wise business decision. The vendor can use a DSS to gather information from its own resources (using a tool such as OLAP) to determine if the company has the ability or potential ability to expand its business and also from external resources, such as industry data, to determine if there is indeed a demand to meet. The DSS will collect and analyze the data and then present it in a way that can be interpreted by humans. Some decision support systems come very close to acting as artificial intelligence agents.

DSS applications are not single information resources, such as a database or a program that graphically represents sales figures, but the combination of integrated resources working together.

Using the mode of assistance as the criterion [Power, 2002] differentiates communication-driven DSS, data-driven DSS, document-driven DSS, knowledge-driven DSS, and model-driven DSS.

- A model-driven DSS emphasizes access to and manipulation of a statistical, financial, optimization, or simulation model. Model-driven DSS use data and parameters provided by users to assist decision

makers in analyzing a situation; they are not necessarily data intensive. Dicodess is an example of an open source model-driven DSS generator [Gachet, 2004].

- A communication-driven DSS supports more than one person working on a shared task; examples include integrated tools like Microsoft's NetMeeting or Groove [Stanhope, 2002].
- A data-driven DSS or data-oriented DSS emphasizes access to and manipulation of a time series of internal company data and, sometimes, external data.
- A document-driven DSS manages, retrieves and manipulates unstructured information in a variety of electronic formats.
- A knowledge-driven DSS provides specialized problem solving expertise stored as facts, rules, procedures, or in similar structures.

By incorporating AI techniques in a decision support system, we make that DSS artificially intelligent - capable of displaying behavior that would be regarded as intelligent if observed in humans. Artificially intelligent DSSs are becoming increasingly common. Perhaps the most prominent of these are expert systems, which support decision making by giving advice comparable to what human experts would provide.

This paper is focused on knowledge-driven DSS using Artificial Intelligence models. Networks of Evolutionary Processors have rules, facts, and collaboration among processors to generate a final decision based on evolutionary steps that take place in processors. Next section describes the computational model of Networks of Evolutionary Processors. And finally, an example is shown.

Networks of Evolutionary Processors

A network of evolutionary processors of size n is a construct NEP = $(V, N_1, N_2, \ldots, N_n, G)$, where V is an alphabet and for each $1 \leq i \leq n$, $N_i = (M_i, A_i, PI_i, PO_i)$ is the i-th evolutionary node processor of the network. The parameters of every processor are:

- M_i is a finite set of evolution rules of one of the following forms only:

  $a \prod b$, $a, b \in V$ (substitution rules)

  $a \prod \varepsilon$, $a \in V$ (deletion rules)

  $\varepsilon \prod a$, $a \in V$ (insertion rules)

  More clearly, the set of evolution rules of any processor contains either substitution or deletion or insertion rules.
- A_i is a finite set of strings over V. The set A_i is the set of initial strings in the i-th node. Actually, in what follows, we consider that each string appearing in any node at any step has an arbitrarily large number of copies in that node, so that we shall identify multisets by their supports.
- PI_i and PO_i are subsets of V^*representing the input and the output filter, respectively. These filters are defined by the membership condition, namely a string $w \in V^*$ can pass the input filter (the output filter) if $w \in PI_i$ ($w \in PO_i$).

$G = (N_1, N_2, \ldots, N_n, E)$ is an undirected graph called the underlying graph of the network [Paun, 2002] [Paun, 2000]. The edges of G, that is the elements of E, are given in the form of sets of two nodes. Kn denotes the complete graph with n vertices. By a configuration (state) of an NEP as above we mean an n-tuple $C = (L_1, L_2, \ldots, L_n)$\$, with $L_i \subseteq V^*$ for all $1 \leq i \leq n$. A configuration represents the sets of strings (remember that each string

appears in an arbitrarily large number of copies) which are present in any node at a given moment; clearly the initial configuration of the network is $C_0 = (A_1, A_2, \ldots, A_n)$.

A configuration can change either by an evolutionary step or by a communicating step. When changing by an evolutionary step, each component L_i of the configuration is changed in accordance with the evolutionary rules associated with the node i. When changing by a communication step, each node processor N_i sends all copies of the strings it has which are able to pass its output filter to all the node processors connected to N_i and receives all copies of the strings sent by any node processor connected with N_i providing that they can pass its input filter [Manea, 2006] [Martin, 2005] [Garey, 1979].

Theorem 1. A complete NEP of size 5 can generate each recursively enumerable language.

Theorem 2. A star NEP of size 5 can generate each recursively enumerable language.

Theorem 3. The bounded PCP can be solved by an NEP in size and time linearly bounded by the product of K and the length of the longest string of the two Post lists.

Para ver esta película, debe disponer de QuickTime™ y de un descompresor TIFF (LZW).

Figure 1.- A sample Network of Evolutionary Processors

An Example

Opportunities for building business expert systems abound for both small and large problems. In each case, the expert system is built by developing its rule set. The planning that precedes rule set development is much like the planning that would precede any project of comparable magnitude within the organization. The development process itself follows an evolutionary spiral composed of development cycles.

Each cycle picks up where the last ended, building on the prior rule set. For a developer, the spiral represents a continuing education process in which more and more of an expert's reasoning knowledge is discovered and formalized in the rule set. Here, each development cycle was presented in terms of seven consecutive stages. Other characterizations of a development cycle (involving different stages or sequences) may be equally valuable. Many aspects of traditional systems analysis and project management can be applied to the development of expert systems.

Rule set development is a process of discovery and documentation. Research continues in search of ways of automating various aspects of the process. It would not be surprising to eventually see expert systems that can assist in this process -- that is, an expert system that "picks the mind" of a human expert in order to build new expert systems. Until that time comes, the topics discussed in this chapter should serve as reminders to developers of expert decision support systems about issues to consider during the development process.

As it stands, rule-based systems are the most widely used and accepted AI in the world outside of games. The fields of medicine, finance and many others have benefited greatly by intelligent use of such systems. With the

combination of rule-based systems and ID trees, there is great potential for most fields. Here it is an example of a rule-base expert system.

Table 1.- Rule-based decision support system applied to medical diagnosis (snapshoot)

```
Assertions
A1: runny nose
A2: temperature=101.7
A3: headache
A4: cough

Rules
R1:    if   (nasal congestion)
            (viremia)
       then diagnose (influenza)
            exit

R2:    if   (runny nose)
       then assert (nasal congestion)

R3:    if   (body- aches)
       then assert (achiness)

R4:    if   (temp >100)
       then assert (fever)

R5:    if   (headache)
       then assert (achiness)

R6:    if   (fever)
            (achiness)
            (cough)
       then assert (viremia)
```

It is necessary to define the underlying graph In order to simulate previous example with a Network of Evolutionary Processors. First of all, an initial processor containing the assertions and basic rules will forward important information to a second processor, which is in charge of a given disease, and forward its result to a container processor. This process can be shown in figure 2.

Para ver esta película, debe disponer de QuickTime™ y de un descompresor TIFF (LZW).

Figure 2.- Network of Evolutionary Processors Architecture sample

Para ver esta película, debe disponer de QuickTime™ y de un descompresor TIFF (LZW).

Figure 3.- Network of Evolutionary Processors Architecture for medical diagnosis

Previous simple example can be extended to a more general diagnosis system as detailed in figure 3. There exists one processor or even more processors in charge of a located diagnosis problem such as: influenza, migraines, heart diseases, etc… These local diagnosis processors can communicate each other to auto complete information diagnosis. Finally, each result of diagnosis processors is sent to an information processor that can combine multiple diagnoses or just show them.

Configuration of a Network of Evolutionary Processors

Next table shows an XML file with the NEP initial configuration corresponding to the medical diagnosis shown in table 1. There are three processors (see figure 2): assertion process (name = 0), specific diagnosis processor (name = 1) and diagnosis information processor (name = 2). Objects travel trough these processors until a final diagnosis is present in the last one. Obviously, this is a simple example, but according to figure 3 the NEP architecture could be complicated in order to obtain a more sophisticated diagnosis. Main idea is to put some assertions in one or more processors and then let them evolve using evolution steps (rules application) and communication steps.

This configuration file is parsed into JAVA objects and a separate thread for each processor is created; also each rule and filter are coded as threads in order to keep the massive parallelization defined in the theoretical model of Networks of Evolutionary Processors.

Table 2.- NEP initial configuration corresponding to table 1

```
<?xml version="1.0"?>
<NEP>
   <processor>
      <name>0</name>
      <object>runny nose</object>
      <object>high temperature</object>
      <object>headache</object>
      <object>cough</object>
      <rule>
         <antecedent>
            <object>runny nose</object>
         </antecedent>
         <consequent>
            <object>nasal congestion</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>body-aches</object>
         </antecedent>
         <consequent>
            <object>achiness</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>high temperature</object>
         </antecedent>
         <consequent>
   <processor>
      <name>1</name>
      <rule>
         <antecedent>
            <object>nasal congestion</object>
            <object>viremia</object>
         </antecedent>
         <consequent>
            <object>influenza</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>fever</object>
            <object>achiness</object>
            <object>cough</object>
         </antecedent>
         <consequent>
            <object>viremia</object>
         </consequent>
      </rule>
      <inputfilter>
         <object>nasal congestion</object>
         <object>fever</object>
         <object>achiness</object>
         <object>cough</object>
      </inputfilter>
      <outputfilter>
         <object>influenza</object>
```

```
            <object>fever</object>
         </consequent>
      </rule>
      <rule>
         <antecedent>
            <object>headache</object>
         </antecedent>
         <consequent>
            <object>achiness</object>
         </consequent>
      </rule>
      <inputfilter>
      </inputfilter>
      <outputfilter>
         <object>nasal congestion</object>
         <object>fever</object>
         <object>achiness</object>
         <object>cough</object>
      </outputfilter>
   </processor>
      </outputfilter>
   </processor>
   <processor>
      <name>2</name>
      <inputfilter>
         <object>influenza</object>
      </inputfilter>
      <outputfilter>
      </outputfilter>
   </processor>
   <conn>
      <from>0</from>
      <to>1</to>
   </conn>
   <conn>
      <from>1</from>
      <to>2</to>
   </conn>
</NEP>
```

Note that connections among processors are defined in a unidirectional way, but any type of connection can be expressed in the XML configuration file in order to have a more complex underlying graph in the network architecture.

Simulation Results

Results concerning simulation of NEP configuration in table 2 can be seen in table 3. **Processor 2:3** , which is the output processor, has the object **influenza**, desired result. This object is generated in **Processor 1:2** using rules inside it. NEP behavior is totally non-deterministic since rules, filters and processors run together in parallel. This example only uses substitution rules, neither insertion nor deletion rules are coded.

Table 3.- Final configuration of NEP corresponding to described configuration on table 2

```
[----------
Processor 0 : 1
Rules: [[runny nose] --> [nasal congestion], [body-aches] --> [achiness], [high temperature] -
-> [fever], [headache] --> [achiness]]
Objects: [runny nose, high temperature, headache]
Output Filter: [nasal congestion, fever, achiness, cough]
Input Filter: []
----------
, ----------
Processor 1 : 2
Rules: [[nasal congestion, viremia] --> [influenza], [fever, achiness, cough] --> [viremia]]
Objects: [cough, fever, achiness, viremia, nasal congestion, influenza]
Output Filter: [influenza]
Input Filter: [nasal congestion, fever, achiness, cough]
----------
, ----------
Processor 2 : 3
```

```
Rules: []
Objects: [influenza]
Output Filter: []
Input Filter: [influenza]
----------
]
```

The great disadvantage is that a given NEP can only solve a given problem; if it is necessary to solve another problem (maybe a little variation) then another different NEP has to be implemented. The idea of learning tries to undertake such disadvantage proposing a model able to solve different kinds of problems (that is a general class of problems). Learning can be based on the self-organizing maps. There are a lot of open problems that need to be solved in order to show the computational power of this learning idea, but the possibility to compute NP-problems is promising apart from the massive parallelization and non-determinism of the model.

Conclusion

This paper has introduced the computational paradigm Networks of Evolutionary Processors. NEPs can be easily applied to Knowlede-driven Decision Support Systems due to the inherent rule-based behavior of NEPs. JAVA implementation of this model works as defined by the theoretical background of NEPs: massive parallelization and non-deterministic behavior.

Connectionists' models such as Neural Networks can be taken into account to develop NEP architecture in order to improve behavior. As a future research, learning concepts in neural networks can be adapted in a NEP architecture provided the numeric-symbolic difference in both models. NEPs can be considered universal models since they are able to solve NP-problems.

Bibliography

[Alter, 1980] Alter, S. L. Decision support systems: current practice and continuing challenges. Reading, Mass., Addison-Wesley Pub. (1980).

[Gachet, 2004] Gachet, A. Building Model-Driven Decision Support Systems with Dicodess. Zurich, VDF. (2004).

[Garey, 1979] M. Garey and D. Johnson. Computers and Intractability. A Guide to the Theory of NP-completeness. Freeman, San Francisco, CA, (1979).

[Manea, 2006] F. Manea, C. Martın-Vide, and V. Mitrana. All NP-problems can be solved in polynomial time by accepting networks of splicing processors of constant size. Proc. of DNA 12, in press. (2006).

[Martin, 2005] C. Martin-Vide and V. Mitrana. Networks of evolutionary processors: Results and perspectives. Molecular Computational Models: Unconventional Approaches. 78–114, Idea Group Publishing, Hershey. (2005).

[Paun, 2000] Paun G. Computing with Membranes. In: Journal of Computer and Systems Sciences, 61, 1. 108--143. (2000).

[Paun, 2002] Gh. Paun. Membrane Computing. An Introduction, Springer-Verlag, Berlin, (2002).

[Power, 2002] Power, D. J. Decision support systems: concepts and resources for managers. Westport, Conn., Quorum Books. (2002).

[Stanhope, 2002] Stanhope, P. Get in the Groove: building tools and peer-to-peer solutions with the Groove platform. New York, Hungry Minds. (2002).

Authors' Information

Miguel Angel Díaz – Dept. Organización y Estructura de la Información, Escuela Universitaria de Informática, Universidad Politécnica de Madrid, Crta. De Valencia km. 7, 28031 Madrid, Spain; e-mail: mdiaz@eui.upm.es

Nuria Gómez Blas – Dept. Organización y Estructura de la Información, Escuela Universitaria de Informática, Universidad Politécnica de Madrid, Crta. De Valencia km. 7, 28031 Madrid, Spain; e-mail: ngomez@dalum.eui.upm.es

Eugenio Santos Menéndez – Dept. Organización y Estructura de la Información, Escuela Universitaria de Informática, Universidad Politécnica de Madrid, Crta. De Valencia km. 7, 28031 Madrid, Spain; e-mail: esantos@eui.upm.es

Rafael Gonzalo – Dept. Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, 28660 Madrid, Spain; e-mail: rgonzalo@fi.upm.es

Francisco Gisbert – Dept. Lengujes y Sistemas Informáticos e Ingeniería del Software, Facultad de Informática, Campus de Montegancedo, 28660 Madrid, Spain; e-mail: fgisbert@fi.upm.es