Telepathic SNP A-Machine

Ammar Adl†
Computer Science Department, Faculty of Computers and Information, Cairo University

Summary
The idea of telepathy has been always tackled as a paranormal phenomenon, and when some efforts were introduced in pursuit for explanation, the extended mind notion with morphic fields appeared, and a tendency for dealing with the matter outside the physical brain was the case. This paper is an attempt to study whether it is possible to extract a parallel computational model from such phenomena, and to propose another perspective explaining telepathy from inside the brain itself. The use of SNP Systems principles and their derivatives especially the SNP A-Machine will be the path to follow. A software was developed to simulate the environment providing a set of experiments to induce some factors as a simulation of the process.

Keywords:
SNP Systems, SNP A-Machine, Telepathy, Parallel Computational Models.

1. Introduction

A Spiking Neural P system is a class of P systems inspired by the functioning of neural networks, and the ways they use to exchange signals through their specialized junctions called chemical synapses. Go to [2] for more details. SNP System is a construction of a networked membranes hosting a multi-set of objects and being in a certain state according to which objects are dealt with. The communication channels among different cells are specified in advance and correspond to axons in neural cells. To consider a spiking neural P system - recalling from [2] - of degree $m \geq 1$, in the form:

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0),$$

Where:

(i). $O = \{a\}$ is the singleton alphabet (a is called *spike*);
(ii). $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$

Where:

a) $n_i \geq 0$ is the *initial number of spikes* contained by the cell;
b) R_i is a finite set of *rules* of the following two forms:
   (i) $E/a^r \rightarrow a; t$, where E is a regular expression over O, $r \geq 1$, and $t \geq 0$;
   (ii) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that $a^s \notin E/a^r \rightarrow a; t$ of type (1) from *Ri; L(E)* for any rule.

(iii). $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (synapses among cells);
(iv). $i_0 \in \{1, 2, \ldots, m\}$ *indicates the output neuron.*

First type of rules are the *firing rules*: provided that the contents of the neuron, is introduced by the regular expression E, and there are r spikes are consumed, the neuron is fired, and it produces a spike which will be sent to other neurons after t time units, considering the usage of a global clock all across the system, identifying the time for the whole system, hence the functioning of the system could be set as a synchronized model. There are two actions that take place in a single step: firing and spiking. A neuron fires when using a rule $E/a^r \rightarrow a; t$, this is only if the neuron contains n spikes and $a^n \in L(E)$ and $n \geq r$. The regular expression E represents the contents of the neuron. Here, at the level of a single neuron computation is in sequential mode, i.e. a single rule is to be fired at each step. Still, the maximal parallelism is at the level of the whole system, in the sense that in each step all neurons which can evolve (use a rule) have to do it. For spiking, the use of a rule $E/a^r \rightarrow a; t$ in a step q means firing in step q and spiking in step $q + t$. That is, if $t = 0$, then the spike is produced immediately, in the same step when the rule is used. If $t = 1$, then the spike will leave the neuron in the next step, if we consider that t is represented by a time interval $t = \{0, \ldots t_n\}$, then moving from $0 \rightarrow t_n$ *in time* will be simulating a recalling factor (r) and moving from $t_n \rightarrow 0$ *in time* will be simulating a forgetting process by a forget factor (f). In the time between firing a rule and producing a spike, the neuron is assumed to be building up for next firing stage (the refractory period); so it will not be able to accept any more incoming spikes, this much like going into a short hibernation state. [2]

2. The SNP A-Machine

As in [1] SNP A-Machine is a SNP system with one working neuron. A neuron is considered a living cell, this leads to the idea that it encapsulates the features of a specialized computational model inside, and based on the fact that if the same inputs are fed to the neuron it produces the same outputs and that some types of neurons deliver different outputs over processing time, these deliver more evidences on the neuron containing some kind of memory, and a learning mechanism. [2], refer to

Manuscript received May 5, 2013
Manuscript revised May 20, 2013

[1], for a detailed structure and design of the machine. Also refer to [4], for the single neuron discussion.

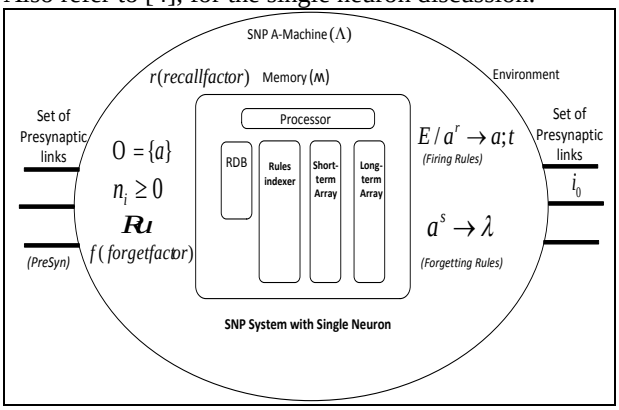

Fig.1.The SNP A-Machine structure. [1]

Definition.1.

An SNP A-Machine unit of degree m, m ≥ 1, is a construct: [1]

$\Lambda = (O, PreSyn, f_1,\ldots, f_r, r_1,\ldots, r_c, N, Ru, \mathcal{M}, PostSyn)$,

Where,

(i) *O = {a} is the alphabet (the object a is called spike);*

(ii) *PreSyn ⊆ {1, 2, . . . , m}×{1, 2, . . . , m} with (i, i) ∉ PreSyn for* $1 \leq i \leq m$ *(input synapses); where:*

(iii) $\mathcal{M}$ *is a dynamic array based sub-system.*

(iv) *N = {n}, ni ≥ 0; is the initial number of spikes contained by the machine;*

(v) *Ru is a finite set of rules {Rs, R, F} of the following forms:*

1. *Rs where:*
   1. $E/a^r \rightarrow a$; *t. E is a regular expression over O, r ≥ 1, and t ≥ 0; (firing Rule).*
   2. $a^s \rightarrow \lambda$, *for some s ≥ 1, with the restriction that* $a^s \notin L(E)$ *for any rule* $E/a^r \rightarrow a$; *t of type (1) from Rsi; (forgetting Rule).*
2. *R =* $\{Rr_{i1}, \ldots, Rr_{il_i}\}$, *for each i ∈ {1,... , m} and j ∈ {1,... ,* l_i*}, where:*
   $E/a^k \rightarrow (a^m, r)$; *t is a rule with* ***recalling*** *factor, k ≥* $m_{ij} \geq 0$ *and* $r_{ij} \geq 0$. *The sequence f =* $(r_1, r_2, \ldots, r_c)$ *is a finite sequence of natural numbers called the recalling sequence where* $r_1 = k$ *and* $r_c \geq 0$. *Inspired by rules in* [8].
3. *F =* $\{Rf_{i1}, \ldots, Rf_{il_i}\}$, *for each i ∈ {1,... , m} and j ∈ {1,... ,* l_i*}, where:*
   $E/a^k \rightarrow (a^m, f)$; *t is a rule with* ***forget*** *factor, k ≥* $m_{ij} \geq 0$ *and* $f_{ij} \geq 0$. *The sequence f =* $(f_1, f_2, \ldots, f_r)$ *is a finite sequence of natural numbers called the decaying (forgetting) sequence where* $f_1 = k$ *and* $f_r \geq 0$. *Inspired by rules in* [8].

– *PostSyn ⊆ {1, 2,…, m}×{1, 2,…, m} with (i, i) ∉ PostSyn for* $1 \leq i \leq m$ *(indicates the output neuron postsynaptic link.);*

3. The Learning Model

Referring to rule selection and invocation mechanisms discussed in [1], [8], and that if the choice of a firing path leads to the success with a higher probability than the choice of another, then the device computation process being refined by every passing time unit. Recalling from [1] the learning machine is a tuple of an SNP A-Machine, an input vector for the machine, a learning schema or function, a learning time interval, a threshold for recalling and forgetting a certain set of rules, a learning curve slope (*Lr*):

Definition.2.

A Learning SNP A-Machine is: [1]

$\Lambda L = (\Lambda, X_{input}, L, T, Th, Lr)$, where:

(i) *Λ is an SNP A-Machine.*

(ii) $X_{input} = \{x_1,\ldots, x_n\}$ *is a finite set of inputs of Λ.*

(iii) *L: Z → Z is a function from the set of integer numbers onto the set of integer numbers. The learning function.*

(iv) $T = \{t_1,\ldots, t_n\}$ *is a finite set of time stamps attached to the inputs for Λ.*

(v) ***Th*** *= {rth, fth} is a set of two values for recall and forget values thresholds.*

(vi) *Lr is a positive value called the learning rate.*

4. The Telepathic Process

The telepathy phenomenon studied here is just an *assumption*, in order to try defining and abstracting the concept, in its simplest case we can put it as *"Having a shared idea, feeling or even doing a certain action at the same time, or almost at the same moment among two or more actors"*. There is no physical contact, defined

distance, or any type of communication channels in this very case. Unlike the representation in [5] which is introducing a physical world or evidences existing among agents, so as a shared mind will exist.

(i) The environment
The factors affecting an actor are the constructing blocks for the "world" that telepathy will occur in; these factors can be categorized under the following

a. Shared experiences
For two or more actors, to construct a global extended mind, an assumed experience shall exist; this is not a shared culture among actors. Yet can be considered as a global shared memory among all of them, it is a *telepathic memory;* data inside memory is built along a learning curve.

b. External stimuli
Social interaction, coincident situations, and other accidental events like threats and fear, can make a certain idea to be initiated due to such stimulating factors as a logical consequence.

c. Internal stimuli
The way an actor is exposed to some inner biological structure or developments, or past encounter to some events, might be another set of affecting elements, this might be observed in flocks of birds, school of fish, but this is not limited to the same species rule [6]. It is a matter of close interaction, with spending "enough" shared time, to start the *telepathic sequence*.

(iii) The need
This can be seen as a joint between the latter two types of stimuli, need can happen due to some external, internal or both of these elements. When taking place also it is assumed to trigger a certain sequence of actions that might initiate a *telepathic sequence*

(iv) The time.
This factor is one of the most important key players in the argument, it is occupying the last ring in the telepathic chain of reactions, without a shared mind timing window, a certain idea or feeling can exist among a number of actors, yet will not fall into the telepathic set of actions. Time is the carrier that transforms a certain set of ordinary actions, feelings, or ideas occurring among some set of actors, into a *telepathic sequence.*

(v) The state of shared mind
The above assumed playing factors were a trial to describe the most obvious elements in the phenomena, when combined by time, an initial state of sharing is evolving, which can be called a shared mind, and at some break point in time a certain *telepathic sequence* can be triggered showing some similarity among different actors behaviors, may be with slightly different paths of execution, yet the accumulative results can be very close.

(vi) The state of decay
When the time window of the sequence is beginning to narrow, a new state of forgetting can emerge, it is a state of losing that moment in time when the actions synchronization is being lost among all playing actors, this can be described as a fading memory, or loss of factors causing telepathy at a very moment in time, the decaying sequence doesn't mean to lose all data, it can be fed back into the shared experience stage to be recalled once again, like using a long-term memory.

5. An Abstract Telepathic Model

With the previously assumed factors, a need for abstracting the process arises:
A Telepathic Model *TL* is a construct of the form:

$TL = (E, T, T_{seq}, M_{th})$, *Where,*

(i) $E = \{Exp, L_r\}$ *the telepathic environment, where,*
- *a. Exp, the set of experiences =* (N, T_{mem}),
  - *N, the set of stimuli =* $\{ Ext_{input}, Int_{input} \}$, *representing the "need", where Ext and Int =* $\{ ex_1,..., ex_n \}$, $\{ ix_1,..., ix_n \}$ *are finite sets of external and internal inputs respectively.*
  - T_{mem}, *is the telepathic memory.*
- *b.* L_r *, is the learning function, representing the number of hits found in memory, and r is a positive value called the learning rate.*

(ii) T, *the global model timing =* $\{t_1,...,t_n\}$ *is a finite set of global clock ticks (timestamps) attached to the inputs for E.*

(iii) T_{seq}, *is the telepathic sequence, representing the output of the system*

(iv) $M_{th} = \{r_{th}, f_{th}\}$ *is a set of two values for recall and forget values thresholds. The r here represents the value that's when r>0 an initial state of telepathic sequence is of higher probability, and f represents the state of decay.*

6. Mapping TL to SNP A-Machine

Using the description in [1], a mapping between the original SNP A-Machine and the new *TL* model can be set as follows:

(i) The already existing rules inside the neuron. (The ones inside the long term memory) will correspond to Int_{input} in the new model.

(ii) The initial rules configuration is set by biological evolution and genetic structures, constructing also the set of internal stimuli Int_{input}.

(iii) The machine's sub-system (Λ) of the predefined structured three arrays, the Indexer, Rules Store, and a Synchronizer:

- The indexer will be holding the rules indexes; to be consulted at first time input is sent to the neuron, to check if there are any matching results from previous computation.
- The second array will be representing the short-term memory. And the third array will be representing the long-term memory, both will be representing T_{mem}.

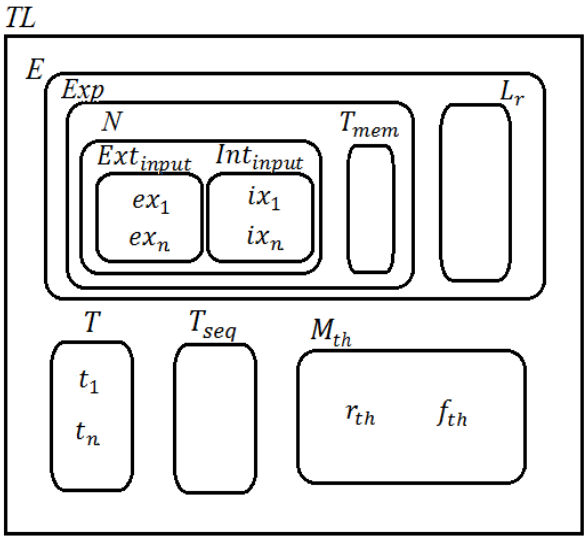

Fig.2. The Abstract Telepathic Model (TL).

Listing.1: Fire Search Job Algorithm.

```
PROCEUDRE (FIRE_SEARCH_JOB)

BEGIN
// try to match the searched value with the array item value of that job
IF (searchValue = array item value) // if value found
STOP (all running jobs) // stop other parallel search jobs, stop all threads.
RETURN TRUE // return true to indicate that value found
ELSE
IF (other search jobs are running) // there are still some search jobs running
THEN WAIT. // wait till other jobs finished its search
ELSE
RETURN FALSE // all jobs finished search and value was not matched with any of them.
END
```

Listing.2: Machine Telepathic Parallel Search Algorithm.

```
PROCEUDRE
(TELEPATHIC_SEARCH {memoryType, SearchValue})
BEGIN
    DEFINE array. // load the chosen array in.
    DEFINE event = notFound // Found or Not Found
                    flag, set to its default value.

    // Determine the array type to be searched, and load it.
    IF (memoryType = short-term memory) THEN
        LOAD array = short-term memory array.
    ELSE IF (memoryType = long-term memory) THEN
        LOAD array = long-term memory array.
    ELSE
        LOAD array = rules database array.
    END IF.

// If no items in the array, then search result is false.
    IF (array = empty), THEN HALT, RETURN FALSE,
END IF

// Waiting for parallel search jobs results to determine the final search result.
    WHILE (search jobs are running)

// Fire a job for each item in the array to be searched, process is in parallel.
        FOREACH array.item (i).value
        PARALLEL BEGIN // start a thread and do the
                            following call in parallel.
            event=CALLPROCEDURE (
            FIRE_SEARCH_JOB
            {searchValue, array.item(i).value }).
            IF (event = found) THEN HALT, RETURN
TRUE END IF //a match was found, return true.
        END FOR
        PARALLEL END // end of the parallel section.

    END WHILE

    IF (event = notFound) THEN //search jobs did not find any match.
    RETURN FALSE. //a match was not found, return false.

END
```

7. The Simulator Overview

The SNP A-Machine simulation software is developed using Dot Net C#, and is composed of a number of components combined together to simulate the rule firing idea and memory recall and forget behavior. It has some

rules and constraints to control the acceptance process. The simulator architecture consists of some layers; Presentation, Business logic, View controllers, Machine container, Resources manager, and Database layer. The core layer of the above ones is the (Machine Container). It contains the main components of the machine: Processing, Memory, Processing, Rules Database, and Indexer Unit. The business logic layer contains the controller that runs business workflows. It communicates with the machine container, and accesses its different components to achieve the simulation tasks. It also contains the model of data, and maps it using the resources manager to its physical data model.

The resources manager contains the XML serializer that serializes and de-serializes objects encapsulating the machine runs. It is used to load data from data layer and pass it as objects to the business logic layer.

The view controllers' layer is the one used by business logic layer to communicate with the presentation one. The view controllers take the data from business layer and format it in a presentation understandable objects that can be easily outputted to the screens. Also it collects data from screen and passes it to the business logic to start processing the orders of the events raised by the view. The final layer is the presentation one that is responsible for viewing the machine state and input forms to the user. It displays the forms, charts and grids holding the simulation runtime state to the user. Experiments were made to test different cases and configurations effect on the machine. Here is a list of them:

(i) Performance when learnt words are never forgotten from memory, this is like a permanent experience in the scope of the *TL model*.

(ii) Recall and forget factors difference with memory utilization, to represent the state of decay.

(iii) Memory hits against recall factor *r* and forget factor *f*, to measure how different time windows can affect the telepathic sequence.

(iv) Rule firing frequency and rule sorting inside the SNP A-Machine, showing the effects of the "*Need*" on the *TL model*.

The idea of simulation logic is to simulate the processing of incoming input patterns, reflecting the Ext_{input} and learning them through firing some set of built-in rules in the machine Int_{input} then saving these rules' learnt values in an easy access store T_{mem}.

Memory is a volatile store, and its volatility ratio is controlled by some configurations related to the learnt rules themselves. As each rule has a value for recalling and forgetting factors [1]. Incoming data pattern is first searched for in the memory buffers with its two types (short-term and long-term), and if found there, then the machine succeeds to recognize an already learnt pattern. If not found, a search for a rule begins, when found a Rule Firing sequence is started. And matching rule's output will be put into T_{mem}.

Also the order of firing rules (order of putting values inside memory) is kept in the indexer. The benefit of using indexer is to search rules in order where the least forgotten rule is the first one to be consulted next run. There are some constraints that limit and control the learning operation of the machine. Those constraints are set by the simulator configurations. Here is a list of different factors that control the learning process: Number of rules in the rules database, The Rule's recall factor *r*, The Rule's forget factor *f*.

Table 1: Experiment configurations description:

Name	Description
Word Length (Ext_{input})	Number of characters making the machine unit input.
Global Clock (T)	Number of clock ticks determining one machine run duration.
Global Clock Step	Duration between two clock ticks. (simulating one second)
Rules Number (Int_{input})	Number of rules in the machine rules database.
Forget Factor f Maximum Value	The maximum number of ticks a word stays in long-term memory.
Recall Factor r Maximum Value	The maximum number of ticks a word stays in short-term memory.
Spikes Trains Number	Number of continuous words' sections (with no spaces) through one run.
Fired Rules %	Rule is fired when a word matches its input pattern. Firing percentage of the spikes train words = (words matching rules' inputs / the spikes train) (E.g. 20/100 of the train words are generated from rules output).
Memory Hits %	Memory is hit when a word matches a fired rule output pattern. Hits percentage of the spikes train words = (words equal rules' outputs / the spikes train) (E.g. 20/100 of the train words are generated from rules output).
Frequency	Number of iterations of the experiment. (Number of runs)

Table 2: Experiment output description: (T_{seq})

Name	Description
Avg. words found in short-term memory	Average number of words found in short-term memory for the whole experiment iterations.
Avg. words found in long memory	Average number of words found in long-term memory for the whole experiment iterations.
Avg. rules fired in Indexer	Average number of rules fired from the rules within the Indexer for the all experiment iterations.
Avg. rules fired in rules database	Average number of rules fired from rules database for the whole experiment iterations.
Avg. not found words	Average number of not found words for the whole experiment iterations.
Avg. success Percentage %	Average percentage of spikes train accepted words (found in memory, or matched rule input) whole experiment iterations.

In the machine's acceptance mode, it receives some inputs (spikes train) and tries to process them (identify the words in train using its set of rules) and learn their patterns, so that when another train comes, it can recognize it more efficiently and fast using its memory. The recognition ability of a machine is affected by the learning curve, and other properties as: learnt rules recall and forget factors, number of rules, and firing rate. Refer to [1] for more details about the machine processing mechanisms.

7.1. The Learning Curve

Learning curve explains machine learning through a specified run. When a word is put in memory (short-term or long-term), that word is learnt by the machine. Words lifetime in memory is dependent on recall and forget factors r and f of the rules. As long as word is in memory, the machine can memorize that word and it is still learnt. So the number of words in memory at some specific tick is an indication on how many words the machine has learnt from its total learning capacity (number of rules), this is to represent the building an experience for the machine.

7.2. Success Curve

Success curve represents the machine input acceptance. The word is accepted by the machine in two cases, if an input word matched a rule input and fired a rule, or the word was found in memory T_{mem}. So machine success value is the number of words (Ext_{input}) accepted by the machine during the specified ticks count. Success curve plots the number of words accepted against global clock ticks.

7.3. Machine state

Machine state is the representation of rules firing map and density through a specific run this is representing the internal processing of the machine while producing a *telepathic sequence*. Machine state is a graphical representation of rules firing, it is a grid with its columns and rows are rules and ticks in sequence. Through the machine run, if a word matched a rule input, it fires, and its value lasts in memory for the next ticks, the tick it fires in is recorded and highlighted on the state grid. The constraint of only one rule fires per tick makes the machine state row has either one black cell, or none. Continuing the machine run and rules firing, a map of black cells in the grid is constructed showing the locality of firing rules, its firing frequency and firing density.
frequency and firing density are high, but when rules number is big, frequency and density decreases for the same firing rate. By knowing other information as (recall factor *r* and forget factor *f*, much information can be extracted from the machine state, as:

Which rule is fired more frequently?
Depending on the firing percentage and the number of rules in the system, the density of firing is increased or decreased, as with little number of rules, the firing

(i) What is the active period through the run?
(ii) When does short-term memory gain a new item? And how much time an item stays in it longer than the recall factor *r* maximum value.
(iii) Is the rule fired from indexer (if it is the second fire or greater) or rules database?
(iv) The order of rules indexes in indexer.

7.4 Results and analysis

All experiments were conducted upon two SNP A-Machines in multiple iterations in order to check their performance under the Telepathic Process. When applying the same number of rules and a randomized external input trains, at some iteration we find that some patterns are almost similarly occurring at the same time window.
The following curves show that also under some similar configurations, randomized input streams, yet constrained by time, the accepted words, constructing the success curve, performance and the memory access are going similar at that very time window.

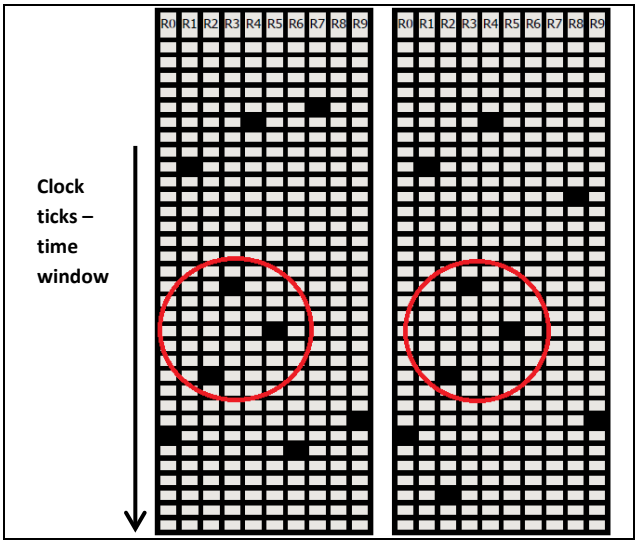

Fig.3. Machine state showing some telepathic sequence at a time window

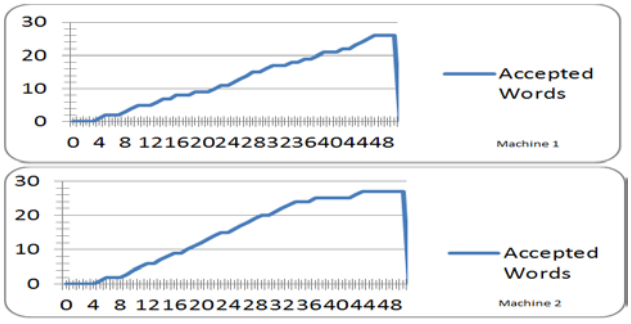

Fig.4. Success curves for the two machines.

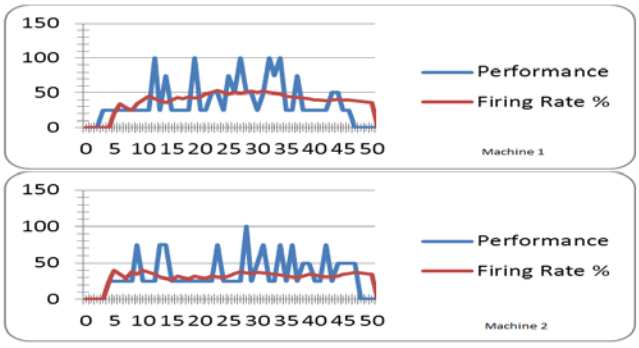

Fig.5. Performance curves for the two machines.

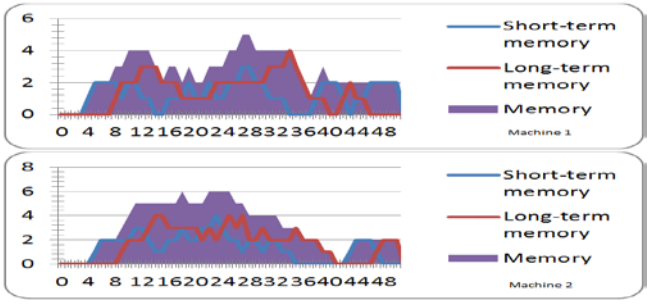

Fig.6. Memory Access curves for the two machines, showing a similar learning rate at some time window

8. Conclusions and Future work

From the experiments run over two SNP A-machines, it is found that when applying the telepathic model on such computational devices, some patterns appear to take place during the computational processing of incoming spikes (inputs) trains going into the system, the one thing obvious here, is that the two machines are not sharing any types of messages, or communication channels, which means that at some time window, and under some certain configurations a resulting value can occur simultaneously inside the two machines, this can be used as a parallel computational approach to tackle some problems. Set a telepathic processing environment, train the machines over time and expose them to similar criteria working factors, this might lead to some new types of solving algorithms with new types of performance. Also from a biological sense, and since SNP systems are derived from nature, simulating the cell computational models, after all, telepathy phenomenon itself might have some basic materialistic foundation inside the physical brain.

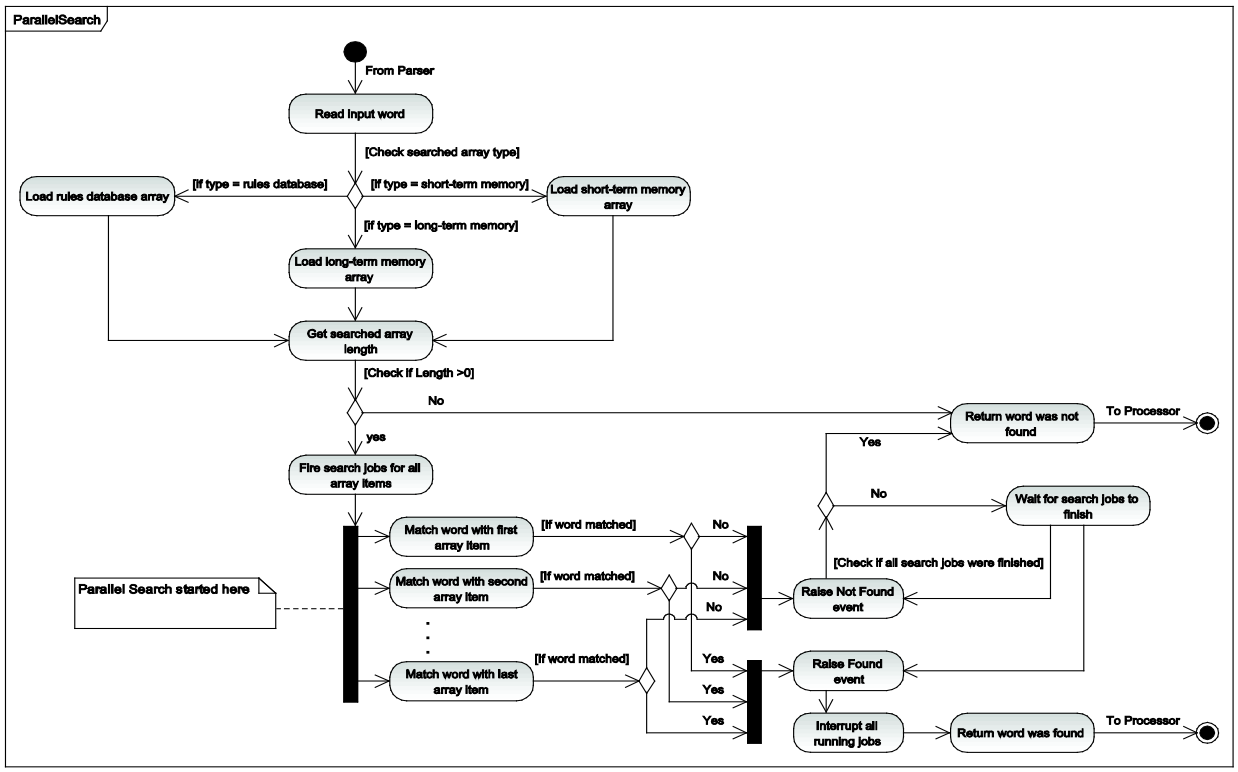

Fig.7. Telepathic parallel search logic flow.

References

[1] Spiking Neural P Systems with Memory Ammar Adl, Amr Badr, Ibrahim Farag, IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.10, October 2011.
[2] M. Ionescu, Gh. P˘aun and T. Yokomori: Spiking neural P systems. Fundamenta Informaticae, 71, 2-3, 279-308, 2006.
[3] Gh. P˘aun: Membrane Computing–An Introduction. Springer-Verlag, Berlin, 2002.
[4] W. Gerstner, W Kistler: Spiking Neuron Models. Single Neurons, Populations, Plasticity. Cambridge Univ. Press, 2002.
[5] Collective representational content for shared extended mind, Tibor Bosse a, Catholijn M. Jonker, Martijn C. Schut , Jan Treur, 2006 Elsevier, November 2005 .
[6] The Extended Mind, Rupert Sheldrake July-August 2003 issue of The Quest.
[7] P systems web page http://ppage.psystems.eu/.
[8] A First Model for Hebbian Learning with Spiking Neural P Systems. Miguel A. Guti´errez-Naranjo, Mario J. P´erez-Jim´enez Research Group on Natural Computing.2008

Ammar Adl received the Ph.D. degree in computer science from Cairo Univ. in 2012. His research interests include software design and architecture, soft, cellular, organic and Bio-computing.