Mathware & Soft Computing 6 (1999) 5-32

A Hybrid Evolutionary Approach to Intelligent System Design

Amr Badr*, Ibrahim Farag** & Saad Eid***
*Dept. Computer Science, Inst. Statistical Studies & Res. (ISSR), Cairo Univ. *e-mail: ruaab@rusys.EG.net*
** Dept. of Computer Science at ISSR, Cairo Univ.
*** Dept. of Elect. Eng., College of Engineering , Cairo Univ.

Abstract

The problem of developing a general methodology for system design has always been demanding. For this purpose, an evolutionary algorithm, adapted with design-specific representation data structures is devised. The representation modeling the system to be designed, is composed of three levels of abstraction: the first, is an 'abstract brain' layer- mainly a number of competing finite state machines, which in turn control the second level composed of fuzzy Petri nets; the third level constitutes the component automata of the Petri nets. Several mutation operators have been developed acting on that representation. The function of these operators is controlled by a 'stochastic L-system'- representing the chronological application of design tasks. The Petri nets' layer receives tokens from the abstract brain layer and serves in modeling synchronous and asynchronous behaviour. The framework is specifically suited to what is called 'distributed design'- design of communicating systems. For that purpose, a methodology has been developed and implemented/ tested in the system GIGANTEC: Genetic Induction for General Analytical Non-numeric Task Evolution Compiler.

Keywords. Genetic Algorithms- Evolutionary Algorithms- Finite State Machines- Petri Nets- Fuzzy Sets- Symbolic Computing- Model Design- State Spaces- Mutation Operators- Search- Exploration- Stochastic Context Sensitive Grammar- Stochastic L-system.

1 Introduction

The quest for the development of a general framework specifically suited for linguistic large-scale designs has always been a problem. Genetic algorithms have been known for their application in design problems. A genetic algorithm is a model of machine learning which derives its behaviour from a metaphor of some of the mechanisms of *evolution* in nature. This is done by the creation within a machine of a *population* of individuals represented by *chromosomes*, in essence, a

set of character strings that are analogous to the chromosomes of our DNA. The individuals in the population then go through a process of *simulated evolution*. Genetic algorithms are used in a number of application areas. An example would be multi-dimensional optimization problems in which the character string of the chromosome can be used to encode the values for the different parameters being optimized. In practice, a genetic model of computation is implemented by having arrays of bits or characters to represent the chromosomes [Gol89]. Simple bit manipulation allow the implementation of 'crossover', 'mutation' and other operators. Evolutionary algorithms are broader in scope. In fact, any genetic algorithm with a chromosome representation other than the bit string can be termed an evolutionary algorithm [Mic96]. Other examples of evolutionary algorithms are evolutionary programming, evolution strategies, classifier systems and genetic programming.

It is evident that the majority of applications tackled so far were directed toward the 'numerical' optimization scheme, constrained or unconstrained. Design problems are no difference. Systems tackling the symbolic side of the design has always been scarce. The problem considered in the system **GIGANTEC** is the evolution of a symbolic design plan. This is comparable to Genetic Programming techniques. [Koz92] [Koz94]. This conforms with the principle of "Less Numerics-More Intelligence" [Wan93] [Lim94]. Evolutionary algorithms have been utilized for state space *search* and *exploration*. Exploration is search in ill-defined state spaces. The work presented is novel in the method and way it tackles the design problem. A *methodology* of design is forwarded. The designer should provide a 'context sensitive L-system' that defines the behavioural interaction between abstract system tasks. As mentioned before, there are three levels of abstraction: the finite state machine, the Petri nets and the component automata. The tokens generated by the *finite state machines* (FSMs) abstract layer are forwarded into a *petri net dispatcher* (PND). The PND in turn distributes & selects the potential sub-petri net to fine-tune the abstract task represented by that token.

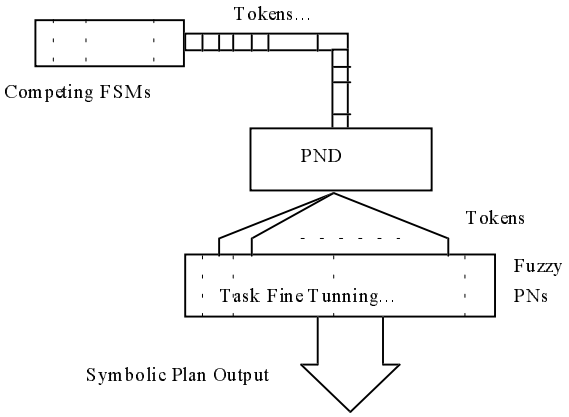


Figure 1

Methods and techniques of *object-oriented programming* are utilized to their utmost. Each FSM is an instance by itself. Each petri net is a class by itself where

synchronous and asynchronous actions are built into the class. Synchronous and asynchronous actions are modeled by means of intra- and inter- concurrency within a Petri net, or between Petri nets.

2 Problem Definition

Mathematically speaking, the problem can be formulated as follows: an intelligent design, represented algorithmically by a finite state machine, is intended to complete a given task:

$$T = \{t_1, t_2, ..., t_m\}$$

where $t_1, ..., t_m$ are subtasks which compose T. Each subtask is represented by a Petri net and is defined by a set of performance specifications to be fulfilled by that subtask:

$$X = [x_1, x_2, ..., x_n]$$

The state variables describing the current status of an intelligent design are given by:

$$S_{t_i} = \{s_{i1}, s_{i2}, ..., s_m\}$$

where $t_i = t_1, t_2, ..., t_m$

$$S_{t_i} = \text{ states for task } t_i$$

and s_{ij} indicates the performance specification for the state variable x_j in the subtask t_i such that $x_j \subset s_{ij}$ indicates that specification $\{ij\}$ has been met. A potential function describes the cost of task T in terms of its subtasks, and can be defined as:

$$pot(T, x) = v_T$$

where $v_T = \Psi(v_{t_i}) = \Psi[pot_{i=1}^{m}(t_i, s_{t_i})]$
and Ψ is a problem specific function which is usually a composite one and evaluated according to conflict matrices, cubes or hypercubes used to determine the fittest finite state machine on basis of a conflict between FSM_i and FSM_j among the population. This will be clarified in later sections and in the experiments.

The intelligent design has a set of available low-level designs:

$$D = \{D_1, D_2, ..., D_r\}$$

where r is the population size and each design D_i is composed of sub-designs d_{ij} associated with the m subtasks of T: $D_i = \{d_{i1}, d_{i2}, ..., d_{im}\}$
Each D_i is represented by a finite state machine. Each d_{ij} is represented by a Petri net or a Petri net component automaton. The intelligent design problem is defined as the intelligent selection of the optimum design D_{opt} from D given a specified task T. The optimal design D_{opt} is defined as the design that maximizes the 'potential' of the system, and the probability that the task T is completed successfully, such that:

$$Pr(T) = pot(T, x) = \text{ maximum}$$

where Pr(T) is the probability of a selected design to complete the required task. As mentioned, this probability is determined by a problem-specific function Ψ which is in turn evaluated according to conflict matrices.

The design itself, made up of m components, is represented mathematically as follows:

$$D_i = FSM_i \xrightarrow{\text{[Token (s)]}} \begin{bmatrix} f_1(\overline{\mu}_1, \overline{\lambda}_1, PN_1, CA_1) \\ \text{M} \\ f_j(\overline{\mu}_j, \overline{\lambda}_j, PN_j, CA_j) \\ \text{M} \\ f_m(\overline{\mu}_m, \overline{\lambda}_m, PN_m, CA_m) \end{bmatrix}$$

where $\overline{\mu}_j$ and $\overline{\lambda}_j$ are the input and output points of the jth component sub-design (the jth Petri net) referred to as 'semaphores' in the formal description of the representation data structures. The CA_j are the 'component automata' of PN_j.

3 System's Functionality.

Due to the large variations in design problems and internal representation, **GIGANTEC** was implemented as a compiler for the code generation of both 'GA operators' and 'representation' depending on a description of the design problem. The system is composed of two main components:

1. A problem *independent* part (**GIGANTEC**)- which receives the problem's directives and description and generates the problem specific source code for design.

2. A problem *dependent* part- which is the source code generated by **GIGANTEC**, to be compiled by a C++ compiler, which once executed, carries out the design analysis specific to the problem.

The function of the system can be easily visualized. Inputs to the system constitute an L-system of stochastic context sensitive grammar (SCSG) that acts as a directive to the operation of the GA operators to be generated; and a problem description part. The problem description part defines the resolution/decomposition of tasks into sub-tasks, and the interactions between tasks both synchronously and asynchronously.

The inputs are passed to the **GIGANTEC** compiler, which in turn generates source code for the *representation* and *genetic operators* for that problem. The problem representation data structures are finite state machines, petri nets and ADT tokens. Generated representation inherits from base classes, which are common to all problems' definitions. Base classes, common to all problems, are also provided for genetic operators to inherit from.

4 Schematic Model

The generated source (representation/ operators) in its executable version functions as follows. The outer layer is the global evolutionary algorithm which acts upon the *abstract brain layer* (FSMs) by problem specific evolutionary operators. The abstract brain layer generates 'tokens' to be received by the inner layer of

'decomposable' petri nets. Each token has a specific petri net to receive it. The token represents an 'abstract entity' (a high-level ADT) that identifies the respective Petri net. This specific petri net act as a simulator of the abstract task's functional behaviour. The petri nets are decomposable in the sense that they can be partitioned into *component automata* [Bat93]. Component automata represent the inner-most 'discrete' and 'non-decomposable' task.

The schematic decomposition of the problem-*dependent* part is shown in the following diagram:

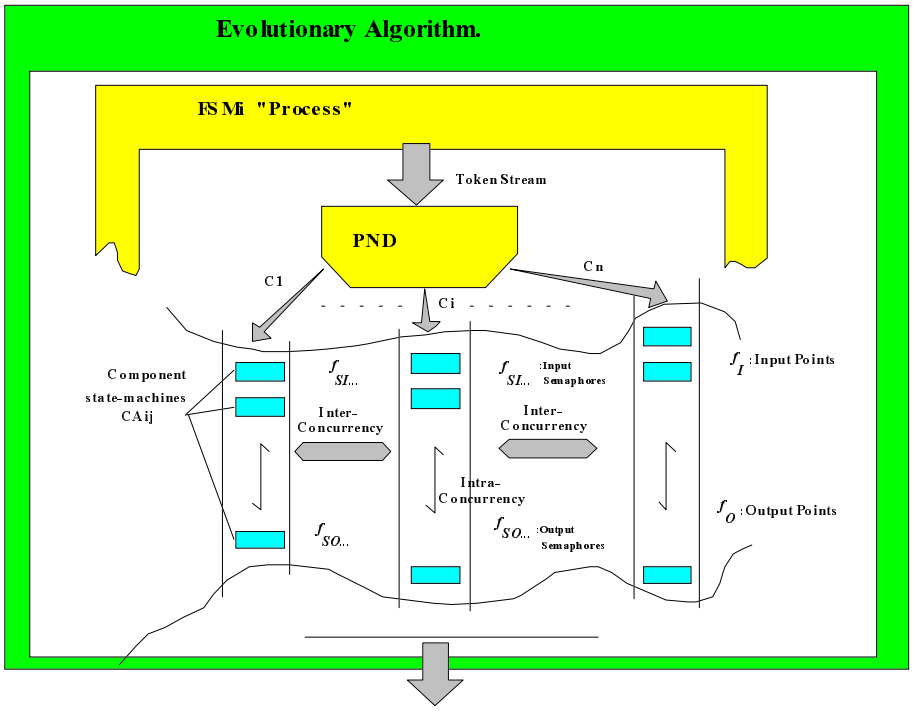


Figure 2

The terminology used in the above diagram is verified in the next section.

5 Representation Data Structures

5.1 The Abstract Brain.

The abstract brain data structure is represented by a population of finite state machines. A finite state machine has a finite number of states and produces outputs on state transitions after receiving inputs. A finite state machine (FSM) M can be defined as the quintuple:

$M = (I, O, S, \delta, \lambda)$

where I: a finite and nonempty set of input symbols

O: a finite and nonempty set of output symbols
S: a finite and nonempty set of states
$\delta : S \times I \rightarrow S$ is the state transition function
$\lambda : S \times I \rightarrow O$ is the output function

When the machine is in a current state s in S and receives an input a from I it moves to the next state specified by $\delta(s, a)$ and produces an output given by $\lambda(s, a)$. [Lee96].

The finite state machine in this definition is fully specified in a sense that at a state and upon an input there is a specified next state by the state transition function and a specified output by the output function. Otherwise, the machine is partially specified. At certain states with some inputs, the next states or outputs are not specified. Also, the machine defined is deterministic. At a state and upon an input, the machine follows a unique transition to a next state. Otherwise, the machine is non-deterministic. The machine may follow more than one transition and produce different outputs accordingly.

5.2 Fuzzy Petri Nets

5.2.1 Coordination Structures.

The coordination structures' basic component is the Petri Net (PN). A Petri net consists, as opposed to the previous definition of a net, of a finite set of places P, a finite set of transitions T, an input function I, and an output function O. A Petri net is thus defined as the quadruple:

$N = (P, T, I, O)$.

where $I : T \rightarrow P^{\infty}$ is the input function, a mapping from transitions to bags of places.

$O : T \rightarrow P^{\infty}$ is the output function, a mapping from transitions to bags of places.

The idea of the Petri net transducer (PNT) introduced by Wang and Saridis [Wan88] is borrowed. The PNT is defined as:

$$PNT = (N, \Sigma, \Delta, \sigma, \mu, F)$$

The PNT is basically a language translator that translates one language to another. The controller of the translation is $N = (P, T, I, O)$, a Petri net. μ is the initial marking of N. Σ is the input alphabet representing output tasks. σ is the translation mapping from $T \times (\Sigma \cup \{\lambda\})$ to finite sets of Δ^* (where λ is the empty string and Δ^* is the set of all finite length strings over Δ). F is the set of final markings. Thus a model can be formulated to the controlling coordination structures, analogous to [Wan88] and [Wan91], composed of a dispatcher D, and a set of coordinators C. The dispatcher D is defined as a PNT,

$$D = (N_d, \Sigma_o, \Delta_o, \sigma_d, \mu_d, F_d)$$

with the Petri Net,

$$N_d = (P_d, T_d, I_d, O_d)$$

The coordinators are defined as $C = \{C_1, C_2, ..., C_n\}$, $n \geq 1$, where each coordinator C_i is a PNT,

$$C_i = (N_c^i, \Sigma_c^i, \Delta_c^i, \sigma_c^j, \mu_c^i, F_c^i)$$

with the Petri Net,

$$N_c^i = (P_c^i, T_c^i, I_c^i, O_c^i).$$

The connection points,

$$F = \bigcup_{i=1}^{n} \{f_I^i, f_{SI}^i, f_O^i, f_{SO}^i\}$$

are the input point, input semaphore, output point and output semaphores of C_i.

5.2.2 Fuzziness and ADT Tokens

Tokens represent the *entities* marking the Petri Net, whether initial, or current or final. In the following definition, tokens are formulated as an algebraic ADT; the token is defined as the triple,

$Tok = (Nom, D, M)$,

where Nom: Name of the token

D : Data parts of the token

M : Methods

The token Tok is Π-*respecting*, that's to say, has a strong relationship with the Petri net partition. A token with a fuzzy membership, is defined as,

$$TokF = (Tok, F, \mu_c, R)$$

where Tok : is the token

F : is the fuzzy sets

μ_c : is the current PN marking

R : is the response schedule.

The representation using fuzzy tokens provide the respective entities with fuzzy sets that readily model data available only in numeric form into descriptive forms. Thus, fuzzy tokens were selected rather than crisp ones. Tokens forwarded to the Petri Net Dispatcher (PND) by the abstract brain layer, are re-directed to the appropriate PN. Tokens are defined as an algebraic ADT with fuzzy memberships (implemented as a C++ class). Fuzziness in the token is fired according to three schemes [Cao96],

1- A Global Scheme,

2- Current Marking of the PN,

3- A Local Scheme.

A simple example is the following PN,

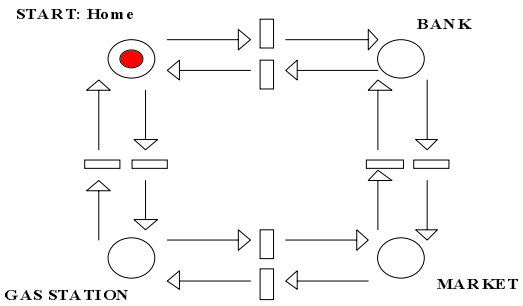


Figure 3

A suitable token for the given PN can be '*Money*' with fuzzy memberships: *Low, Medium*, and *High*. For example, amount of money can be 'medium' at 'Home', 'High' at 'BANK', etc... i.e. The ADT token is sensitive to the current marking of the PN.

5.2.3 Parallelism.

Analogous to the definitions in [Eng91]- Definition-1 and [Bat95]- Definition-1, a parallel system (object-oriented) can be formulated as the tuple,

$$Par_{int\ er} = \{D, C, F, In, obj, mod)$$

where D : Dispatcher structure
C : Coordinator PNs
F : Connection points
In : Initial Configuration
obj : is an object function which associates an object with each coordinator.
mod : is a mode function which specifies the state of $C_i \in \{unborn,\ alive,\ dead\}$.
This accounts for inter-object concurrency. Namely, concurrency between different threads, each representing an object- PN.
Intra-object concurrency, within a PN, (note: a PN is decomposable to CA) can be formulated as the tuple,

$$Par_{int\ ra} = (C_i, CA_i, F_i, In_i)$$

where C_i = the ith PN Coordinator
CA_i= the component automata of C_i
F_i = connection points of CA_i
In_i= Initial configuration of the PN.

5.3 Component Automata

Each Petri Net is supposed to be decomposable into its component automata (CA). A definition can be formulated, analogous to [Bat94] as follows:

$$CA = (N, \Pi, \mu)$$

where $N = (P, T, I, O)$ is a PN as defined before

Π : is a partition of P into disjoint classes $\Pi_1, \Pi_2, ..., \Pi_m$ such that $\forall i (1 \leq i \leq m)$

μ : is the initial marking of N where $\mu \subseteq P$ such that $\forall i \in [1, ..., m], |\Pi_i \cap \mu| = 1$.

The nets generated by the classes of Π are called elementary sub-nets (or component automata) of N.

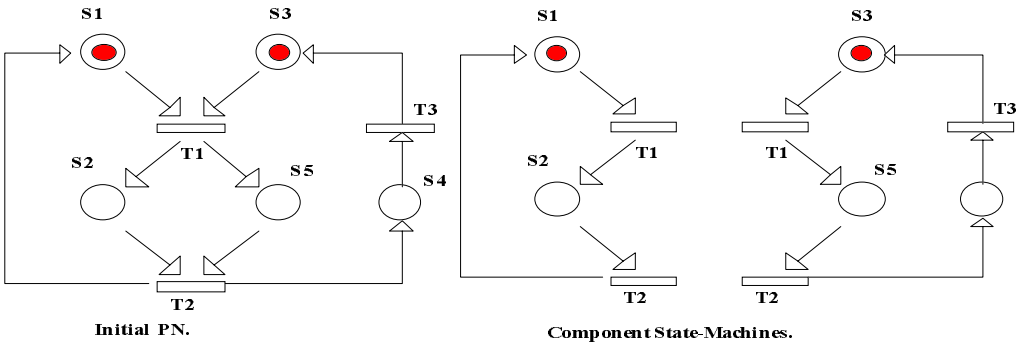


Figure 4

The mathematical definition of component automata of a Petri Net, have been previously given in a preceding section. The initial marking of the PN,

$$\mu = \{s_1, s_3\}$$

The net can be partitioned into,

$$\begin{aligned} \Pi_1 &= \{s_1, s_2\} \text{ and} \\ \Pi_2 &= \{s_3, s_4, s_5, \} \end{aligned}$$

Given that $\mu \subseteq P$, such that $\forall i \in [1, 2]$, $|\prod_i \cap \mu| = 1$, where in this case,

$$\begin{aligned} |\Pi_1 \cap \mu| &= |\{s_1, s_2\} \cap \{s_1, s_3\}| = |\{s_1\}| = 1 \text{ and} \\ |\Pi_2 \cap \mu| &= |\{s_3, s_4, s_5\} \cap \{s_1, s_3\}| = |\{s_3\}| = 1 \end{aligned}$$

The gates of communication among the partition Π are the transitions T1 and T2. This is an example of intra-object concurrency; where the original PN is created as an object instance operating in a thread of its own. The 'synchronous' communication here, is through the gates T1 and T2.

6 The Evolutionary Algorithm

The global evolutionary algorithm [Bad98] is the outermost layer acting upon the abstract brain layer (of FSMs). A FSM is shown in the diagram below.

A single FSM is shown. Its source code is generated by **GIGANTEC**, and is inherited from a base FSM class. The code is problem dependent and incorporates within, the operation of mutation operators. Embedded also, is the code for the

problem states/ transitions and input/ output symbols of the FSM. Several mutation operators are generated, which are, in their broad sense, described as an: add node, delete node, alter initial state, mutate output symbol, mutate transition.

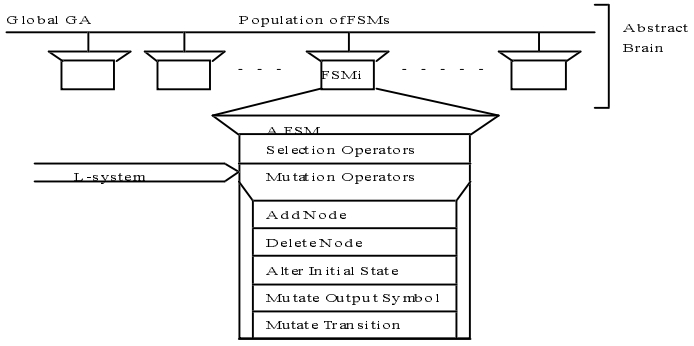


Figure 5

6.1 L-systems and Context Sensitive Grammar

All operators are guided by context sensitive grammar. Compare with Fogel's operators which are not grammar guided [Fog92]. For example, given the following grammar:

1: S $\rightarrow$ AAA
2: A > A $\rightarrow$ [B, C]
3: A $\rightarrow$ B
4: B < C $\rightarrow$ B
5: C > C $\rightarrow$ B
6: A < C $\rightarrow$ B
7: B > A $\rightarrow$ CB

Is resolved (in a broad sense!) as follows:

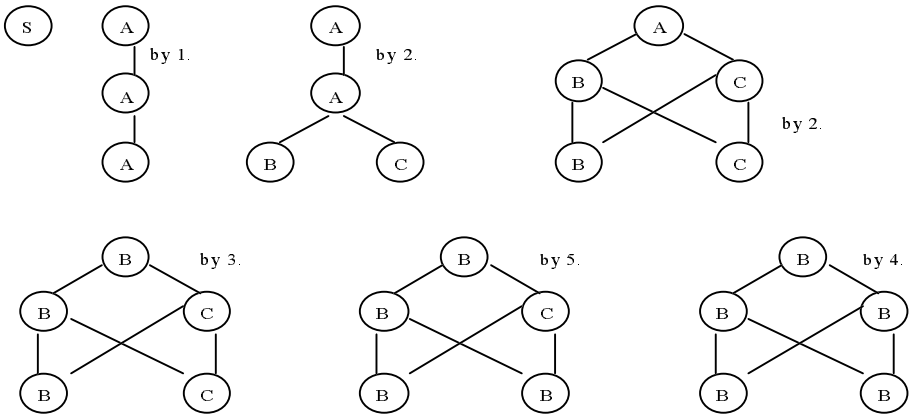


Figure 6

An alternative can be:

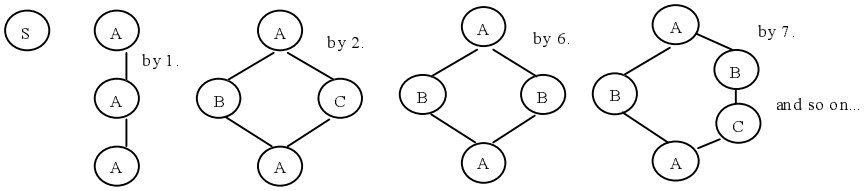


Figure 7

Each FSM represents a program description (algorithm) for a particular design. The population of FSMs compete together to elect the fittest design amongst the population. For the global evolutionary algorithm to function properly, the population size must be a small number.

6.2 Mutation Operators

The mutation operators were designed to work on the proposed data structure for the EA based on state-machines. The section on L-systems and context sensitivity is a prerequisite to this section. The stochastic context sensitive L-system is used as a 'controller' to all mutation operators used in **GIGANTEC**. There are five main mutation operators developed:

1- Change output symbol.
2- Change transition.
3- Add state
4- Delete state
5- Alter initial state.

These operators are dependent on that developed by David Fogel in his excellent thesis [Fog92]. However, they differ largely in the method of their evolution- they are context sensitive; they are L-system guided, and they are stochastic. Following, are pseudo-descriptions for these operators.

Mutation-1. Change Output Symbol.

1- Pick a state randomly.
2- Pick randomly an input symbol.
3- Change output symbol with a random alternative symbol.
4- For the newly generated status:
a- Apply pattern matching algorithm until a contradiction is reached.
{contradiction to the contextual rules of the L-system provided.}
b- If YES, then exhaust 3. then 2. then 1. in sequence.
c- If NO, then FINISH.

Explanation. Best done by an example. Given the following 'part' of a FSMi,

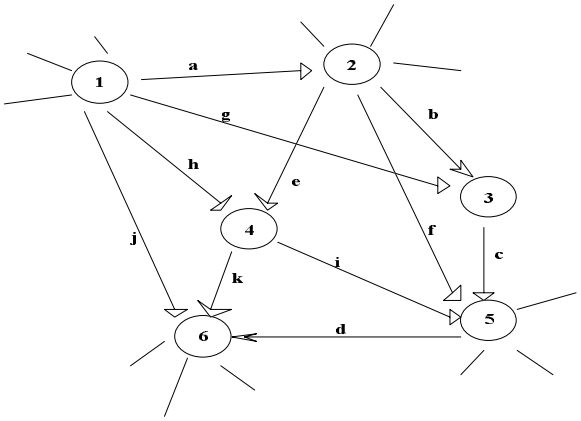


Figure 8

Consider the symbols shown to be the output symbols. Pick the link 2-4 with output symbol e. Transform $e \rightarrow e'$. If aei is a pattern that should be maintained, then the mutation is reset (rejected). Also, the same applies if aek is another pattern that is violated.

Mutation-2. Change Transition.

1- Pick a state randomly.

2- Pick randomly an input symbol.

3- Pick randomly a new output state.

4- For the newly generated status:

a- Apply pattern matching algorithm until a contradiction is reached.
{contradiction to the contextual rules of the L-system provided.}

b- If YES, then exhaust 3. then 2. then 1. in sequence.

c- If NO, then FINISH.

Explanation. Best done by an example. Given the following 'part' of a FSMi,

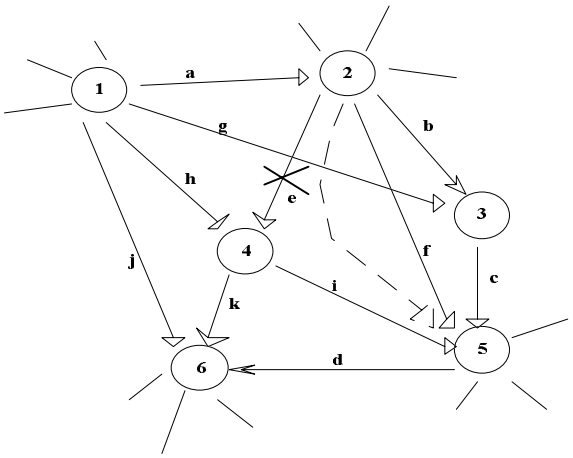


Figure 9

The state selected randomly is state 2, with the randomly selected input symbol at link 2-4. The new output state is 5. if *aei* is a pattern that should be maintained, then the mutation is reset (rejected). Also, the same applies if *aek* is another pattern that is violated. Also if *aed* is a contradiction, then mutation is reset. etc...

Mutation-3. Add State.

1- Generate a new state.
2- Pick random old states to point to the new state.
3- Generate for each input/ output symbols according to rewriting rules of L-system if exists, else generate input/output symbols randomly.
4- For the newly generated status: the random case
a- Apply pattern matching algorithm until a contradiction is reached.
b- If YES, goto 2. for this specific path.
c- If NO, then FINISH.

Explanation. Best done by an example. Given the following 'part' of a FSMi,

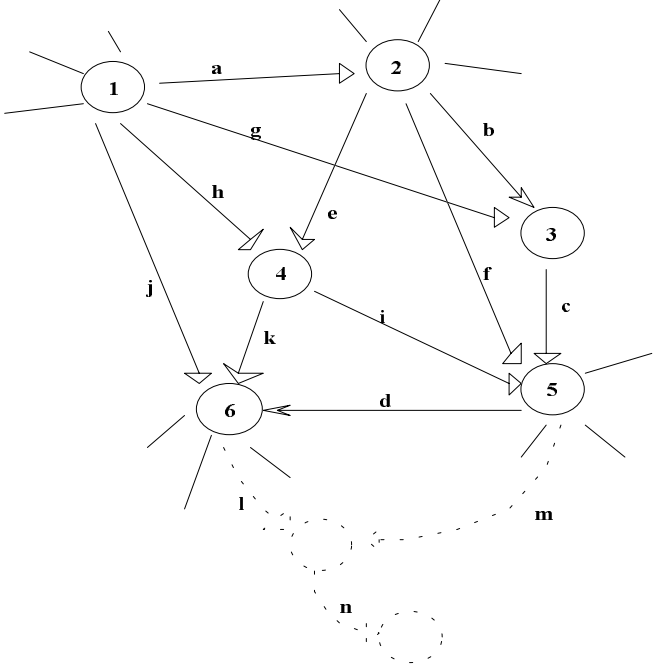


Figure 10

The newly generated patterns, for example *hkln* and *abcmn* and so on, should conform with the L-system rewriting rules.

Mutation-4. Delete State.

1- Pick a random state to delete.
2- Delete all transitions to this state.
3- For the newly generated status:
a- Apply pattern matching algorithm until a contradiction is reached.

b- If YES, restore old state and goto 1.

c- If NO, then FINISH.

Explanation. Best done by an example. Given the following 'part' of a FSMi,

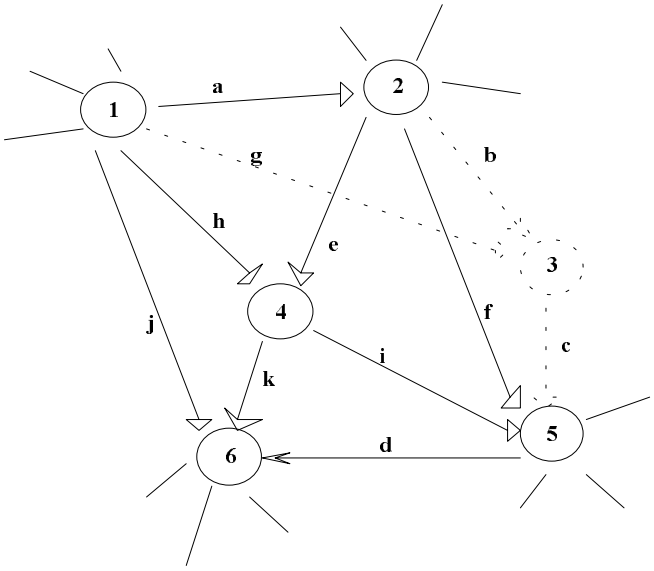


Figure 11

If state 3 is the one to be deleted, then, if *abcd* and *gcd* are patterns, then they should not be violated, and mutation is reset.

Mutation-5. Alter Initial State.

1- Pick a random state.

2- If it is specified as one of the initial states in L-system! then FINISH, else

goto 1.

A pattern matcher, the Aho-Corasick machine is utilized in these mutation operators. The Aho-Corasick machine is a Finite State Machine used for pattern matching. This machine lends itself to automatic construction- according to the provided context sensitive L-system. It is used to match multiple patterns in parallel. This machine is developed by A. Aho and M. Corasick [Aho75]. The machine takes a set of (possibly overlapping) patterns to be matched and producing a finite state machine that can be used to match any of the patterns. A deterministic machine was devised for speed purposes.

6.3 System Evaluation

A particular design representation is evaluated by carrying out a conflict process (competition) between an FSMi and FSMj ($j \neq i$ and j : 1..*population-size*).

Conflict matrices, cubes or hypercubes are constructed to provide conflict-indices to be accumulated, as shown in the algorithm EvalFSM below:

Algorithm. EvalFSM.

begin
  for i = 1 to population_size
  begin
    for j = 1 to population_size
    begin
      -check i not equal to j, if YES continue, else break to next iteration.
      - run conflict between FSMi and FSMj
      begin
        for n = 1 to contest_length
          fitness(FSMi) = fitness (FSMi) +
            conflict_index [i][j];
      end
    end
  end
end
Scan population i: 1 → population_size for FSMk, the fittest individual;

The concept of the conflict-index is borrowed from game theory. When applied in fuzzy terms, it is analogous to fuzzy associative memories (FAMs).

An example of a conflict matrix:

			FSMi				
			Variable k				
		....	...	L	M	H	....	
		L						
FSMj	Variable k	M			Conflict-index			
		H						
		....						

6.4 The Algorithm

The proposed algorithm is shown, in an abstract form, as follows:

Algorithm PEA: Proposed Evolutionary Algorithm.

begin
  **initialize population $P(t)$ - by applying L-system rules and Add-node mutation operator.*
  **for gen = 1 to generation_size do*
  begin

- Evaluate $P(t)$ by competition between FSMs (EvalFSM algorithm).
- Sort population $P(t) \xrightarrow{sort} P'(t)$
- for i = 1 to population_size do
begin
- Apply roulette (SUS) to to select individual η_i.
- Mutate η_i- according to mutation operators' probabilities.
end
- Pass the new population P"(t) to next generation $P(t) = P"(t)$;
end
end.

As shown, the PEA utilizes components such as the EvalFSM algorithm, for evaluating fitnesses, the Stochastic Universal Sampling (SUS) [Bak87] algorithm with elitist strategy and the five mutation operators discussed before.

7 A Methodology of Design

The research proposes a *methodology* for design. A sequence of system-independent *framework* of *steps* steers the design wheel. The theoretical concepts of each step, are explained in previous sections. The framework is summarized as follows:

1- Identify the chronological constraints- in terms of a stochastic L-system rewriting rules.

2- Analyze the system to identify *input* variables and *output* variables.

3- Identify *dependent* and *independent* variables, and exclude those that can be computed from other variables by equations, or other means.

4- Verify the different *components* on basis of *discrete* tasks.

5- Identify the variables associated with each component and associate a *token* with each variable.

6- Select the *range* of application of each variable and define fuzzy sets for each variable according to its relevance.

7- Identify the *inter-relationships* between each component and the other- in terms of synchronization- whether synchronous or asynchronous.

8- Construct a Petri Net for each component- if relevant.

9- Define a synchronization scheme between the constructed Petri nets, on basis of inter-relationships defined previously. {Inter-synchronization}

10- Subdivide component PNs on basis of an intra-synchronization scheme- concurrent and communicating actions within a particular PN component.

11- Identify a starting action for the system.

12- Define an objective function (potential) for the evaluation of designs.

13- Identify component functions of the overall design potential.

14- Design *Fuzzy Associative Memories* (FAM) or a *Conflict Matrices* (or hyper-cubes) relating the interactions between variables for potential component functions (Need an expert in the respective field!)

15- Design an overall output function relating component potential functions.

16- Select the number of generations, size of population, and conflict period.

17- Run the system to generate designs.

8 Experiments

8.1 Experiment (1): Design of a Medical Diagnosis System- The Cerebro-spinal Fluid Circulation.

The four ventricles and the subarachnoid space, formed by the cerebral meninges (membranes which cover the brain and spinal cord, and which enclose the cerebrospinal fluid (CSF)), contain a watery fluid. As this fluid also appears in the spinal compartments, it is called the cerebrospinal fluid (CSF) or liquor cerebrospinalis. The fluid serves as a buffer that reduces the effect of skull impact on the brain. The CSF also serves as a heat buffer and takes care of the disposal of certain waste products. Secretion from the choroid plexus (nerve junctions), on the walls of the first and second ventricle, is the main source of CSF formation. CSF is also formed in the third and fourth ventricle and in the spinal cord. [Wij93] and [Dun92].

Under normal conditions, when there is free communication of CSF between the cranial and spinal compartments, the processes of CSF formation and CSF absorption are in equilibrium. Then the pressure is equal in all compartments. When this equilibrium is disturbed, the craniospinal system must compensate for changes in volume. The parameters concerned, should provide information on the rate of CSF formation and absorption as well as on the storage capabilities of the system. The absolute values of these parameters, however, depend largely on the assumptions made in the mathematical relationships.

The basis of the CSF circulation model is given by the equilibrium existing between CSF formation, CSF absorption and changes in total craniospinal volume. This equilibrium exists under all circumstances within the craniospinal system,

$$F_i - F_o - F_{cs} = 0$$

where F_i = CSF formation rate (ml/ hr)

F_o = CSF absorption rate (ml/ hr)

F_{cs} = rate of changes in CSF volume (ml/ hr)

The exact relationship between CSF formation and the pressure gradient across the choroid plexus is not known. It is assumes that CSF formation is linearly dependent on the pressure gradient between the arteries in the choroid plexus and the cerebral ventricles,

$$F_i = (P_a - P)/R_i \qquad \{Eqn - 1\}$$

where P_a = arterial pressure in the choroid plexus (mmHg)

P = pressure in the cerebral ventricles (mmHg) = ICP

ICP = Intra- Cerebral Pressure

R_i = resistance to formation of CSF

Since the arterial pressure in the choroid plexus is difficult to measure, it is often replaced by the systemic arterial pressure (SAP).

The CSF absorption rate is generally accepted to be linearly related to the pressure difference between the subarachnoid space and the dural sinuses, if and so long as the ICP exceeds the dural sinus pressure,

$$F_o = (P - P_d)/R_0 \text{ for } P \geq P_d \qquad \{Eqn - 2\}$$

$$F_o = 0 \text{ for } P < P_d$$

where P_d = sinus pressure (mmHg)

R_o = resistance to absorption of CSF or outflow resistance

The storage capabilities of the system are deduced from the volume-pressure relationship,

$$P = P_0 + P_1 \exp(E_1 V_e) \qquad \{model - 1\}$$

where $P = ICP$ (mmHg)

P_1, P_o = constant pressure terms (mmHg)

E_1 = elastance coefficient

V_e = elastic volume, change in total craniospinal volume with respect to equilibrium volume

boundary conditions: $E_1 > 0$ so $P > 0$ and $P_o < P$

We can derive,

$$dP/dV = E_1(P - P_o)$$

$$\text{and } dP/dt = (dV/dt).(dP/dV) = F_{cs}.E_1.(P - P_o)$$

The model can be tested and parameters calculated by adding known external flow to the physiological flows. The 'continuous liquor infusion test' will be implemented in this model. The infusion rate is chosen so that a new equilibrium pressure is reached within a few minutes,

$$F_{ex} = const$$

A pressure independent CSF formation can be formulated by,

$$P = R_0 F_{ex} + R_0 F_1 + P_d = ICP \qquad \{model - 2\}$$

A pressure dependent CSF formation can be formulated by,

$$P = \frac{R_i R_o F_{ex}}{R_i + R_o} + \frac{P_a R_o + P_d R_i}{R_i + R_o} \qquad \{model - 3\}$$

A suitable L-system to model such a behaviour, can be a one that allows any sequence of ICPs in equal probabilities:

Rewriting rules	Probability
S → ICP(L)	0.33
S → ICP(M)	0.33
S → ICP(H)	0.33
L < L → L	0.33
L < L → M	0.33
L < L → H	0.33
M < L → L	0.33
M < L → M	0.33
M < L → H	0.33
H < L → L	0.33
H < L → M	0.33
H < L → M	0.33

Table 1

and so on. This L-system simplifies matters by allowing all sequences to be possible. A more rigid analysis can enforce other sequences according to the experience of the physician.

The fig. below shows the Petri net modeling the formation, absorption and storage of the cerebrospinal fluid.

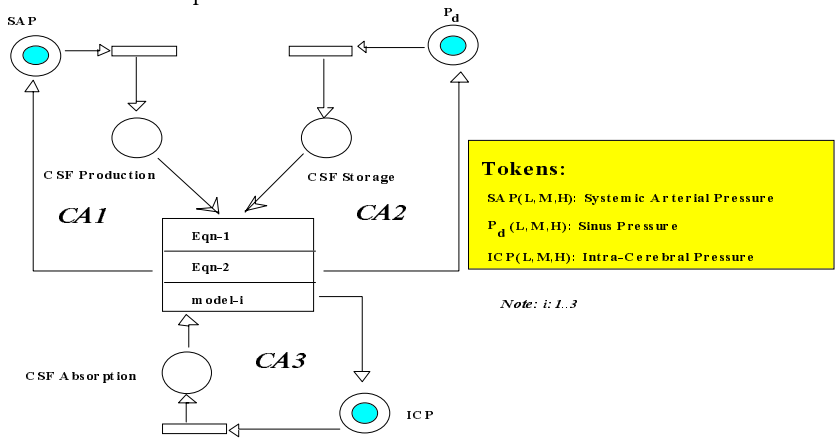


Figure 12

The partition $\Pi = \{CA_1, CA_2, CA_3\}$ with three component state-machines, each with a token. The tokens are SAP, P_d, and ICP. Each token has three fuzzy sets: Low, Medium and High. The input variables are defined to be SAP (Systemic Arterial Pressure) and P_d (Sinus Pressure). Each was given its range in its fuzzy sets,

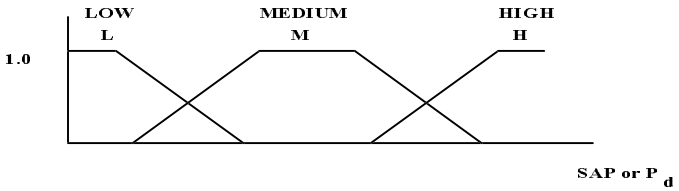


Figure 13

The output variable is ICP (Intra-Cerebral Pressure). The system was provided with a FAM modeling the interaction between SAP and P_d.

Pd \ SAP	L	M	H
L	F1	F2	F3
M	F4	F5	F6
H	F7	F8	F9

Figure 14

The F_i's are Petri nets of the structure shown before. This PN is decomposable into its component state-machines, each modeling a particular process. This is a perfect example of intra-concurrency within a Petri net, which exactly matches reality; as the CSF production, storage and absorption, all take place in parallel! The component state-machines are shown more detailed below,

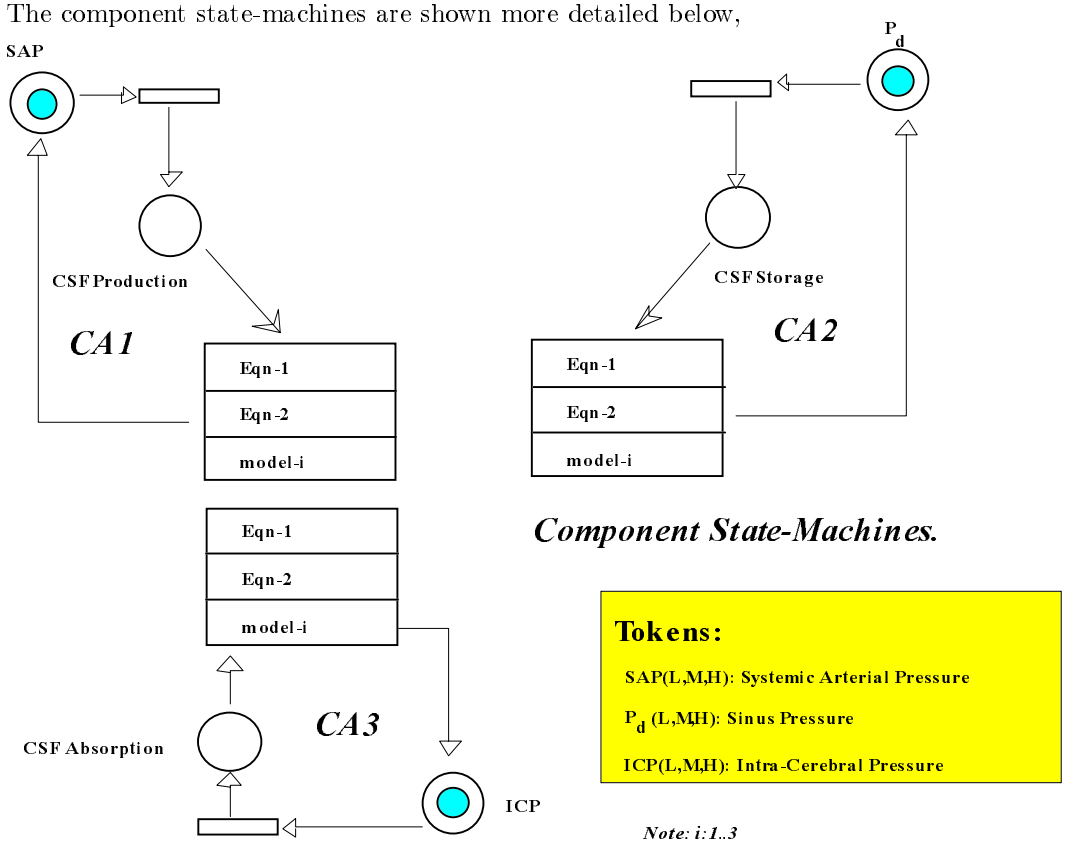


Figure 15

For the purpose of fitness evaluation of the suitability of the generated FSMs, a hypercube of weights, modeling the interaction between FSMi and FSMj- containing all variables SAP, P_d and ICP of each FSM is provided.

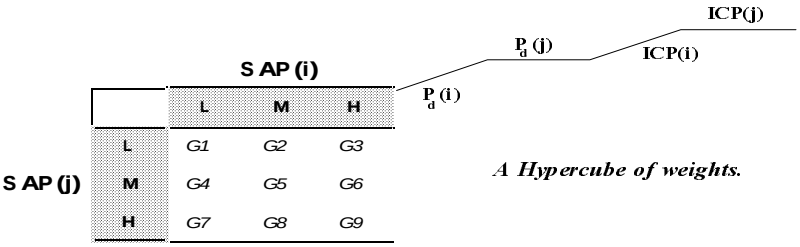


Figure 16

The dimension of the *hypercube* is equal to 6. The entries Gi are to be put according to the experience of the physician. An annotated branch of the resultant FSM is shown below, with input and output variables.

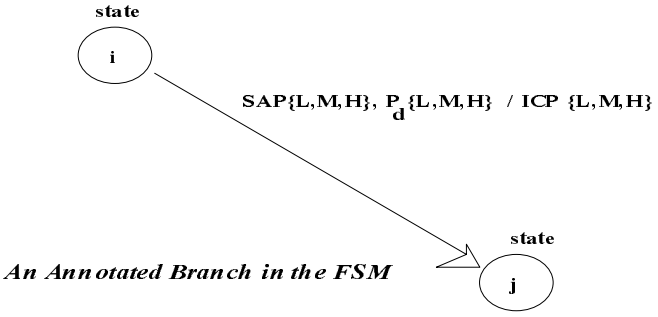


Figure 17

An evolved FSM is shown below:

Input: SAP/Pd

Output: ICP>Next State

state	L/L	L/M	L/H	M/L	M/M	M/H	H/L	H/M	H/H
0	L>8	M>13	M>4	M>0	L>10	M>14	L>8	H>6	H>15
1	L>7	H>6	M>12	M>8	L>8	M>10	M>8	M>1	H>6
2	M>14	H>10	L>2	L>8	L>9	L>3	M>1	H>13	M>11
3	L>4	M>8	H>0	H>5	M>11	M>5	H>12	M>5	M>0
4	M>6	M>4	H>3	M>7	L>7	L>2	M>11	M>2	H>12
5	L>15	L>14	M>6	M>4	L>0	L>10	M>5	M>13	M>5
6	M>4	M>15	H>6	L>0	M>5	M>15	M>7	H>13	H>3
7	M>4	M>0	M>15	L>4	L>5	H>8	L>0	M>3	H>10
8	M>8	M>6	M>14	L>6	M>5	L>12	H>5	H>4	M>8
9	L>14	L>10	M>13	M>8	H>11	L>1	M>5	M>0	M>5
10	L>14	L>4	H>10	L>8	L>5	M>0	M>15	M>5	M>4
11	M>0	H>12	L>9	L>4	L>6	L>2	H>14	H>15	H>11
12	L>8	L>1	H>11	L>1	M>15	H>4	H>2	M>10	H>4
13	L>8	M>12	L>15	L>15	M>0	L>2	M>14	H>0	H>14
14	M>12	M>0	M>4	M>7	L>2	M>2	M>9	M>2	M>10
15	L>10	L>5	H>5	L>11	L>10	M>0	M>8	H>10	M>3

Table 2

Note: In cases of diagnosed high ICP, surgery may afford relief.

8.2 Experiment (2): Design of a Control System for Effluent Substrate Concentration in an Activated Sludge Process.

Nitrification is the process of ammonia oxidation by specialized organisms, called *nitrifiers*. Their growth rate is much slower than that of the heterotrophic organisms which oxidize organic carbon, and they can be washed out of the reactors by the sludge wastage stream. In an activated sludge system, when the organic load is high, then the high biomass growth rates require high waste rates. Nitrification will not be possible under these conditions because the concentration of nitrifiers will become very low. (see fig. below) [Dun92]

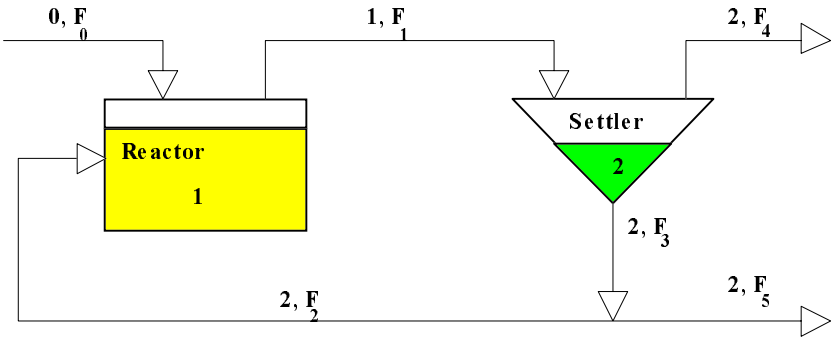


Figure 18

The dynamic balance equations can be written for all components around the reactor and around the settler. The settler is simplified as a well-mixed system with the effluent streams reflecting the cell separation.

Reactor		
Organic Substrate Balance	$\frac{dS_1}{dt} = \frac{1}{V_1}\left(F_0 S_0 + F_2 S_2 - F_1 S_1 - \frac{R_1 V_1}{Y_1}\right)$	Eq-R1
Ammonia Substrate Balance	$\frac{dA_1}{dt} = \frac{1}{V_1}\left(F_0 A_0 + F_2 A_2 - F_1 A_1 - \frac{R_2 V_1}{Y_2}\right)$	Eq-R2
Heterotrophic Organisms Balance	$\frac{dO_1}{dt} = \frac{1}{V_1}\left(F_2 O_2 - F_1 O_1 - R_1 V_1\right)$	Eq-R3
Nitrifying Organisms Balance	$\frac{dN_1}{dt} = \frac{1}{V_1}\left(F_2 N_2 - F_1 N_1 - R_2 V_1\right)$	Eq-R4

Table 3

Similar equations can be presented for settler.

Settler		
Organic Substrate Balance	$\frac{dS_2}{dt} = \frac{1}{V_2}(F_1S_1 - F_3S_2 - F_4S_2)$	Eq-S1
Ammonia Substrate Balance	$\frac{dA_2}{dt} = \frac{1}{V_2}(F_1A_1 - F_3A_2 - F_4A_2)$	Eq-S2
Heterotrophic Organisms Balance	$\frac{dO_2}{dt} = \frac{1}{V_2}(F_1O_1 - F_3O_2)$	Eq-S3
Nitrifying Organisms Balance	$\frac{dN_2}{dt} = \frac{1}{V_2}(F_1N_1 - F_3N_2)$	Eq-S4

Table 4

The equations for the flow rates are given below,

Recycle flowrate: $F_2 = F_0R$ Eqn-1
where R is the recycle factor

Reactor outlet flow: $F_1 = F_2 + F_0$ Eqn-2

Flow of settled sludge: $F_3 = \frac{F_1}{C}$ Eqn-3
where C is the concentration factor for the settler

Flow of exit substrate: $F_4 = F_1 - F_3$ Eqn-4

Flow of exit sludge wastage: $F_5 = F_3 - F_2$ Eqn-5

Growth rates for the two organisms,

$R_1 = \mu_1O_1$
$R_2 = \mu_2N_1$

The problem now is to devise a control system for the recycle to maintain a constant value of the effluent substrate concentration. The following variables were identified:

Input Variables:

F_0 = Input flow rate
N_1, N_2 = Concentration of Nitrifiers in Reactor and Settler
O_1, O_2 = Concentration of Heterotrophs in Reactor and Settler
μ_1, μ_2 = Specific Growth Rate of Heterotrophs and Nitrifiers

Output Variables:

A_1, A_2 = Ammonia Substrate Concentration in Reactor and Settler
S_1, S_2 = Organic Substrate Concentration in Reactor and Settler
Fuzzy sets {Low, Medium, High} were chosen for input and output variables,

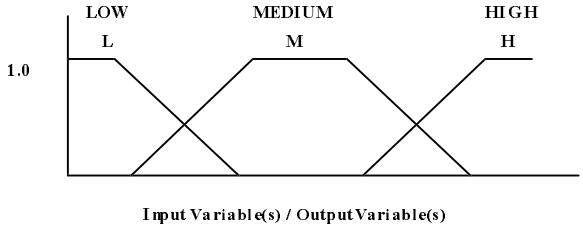
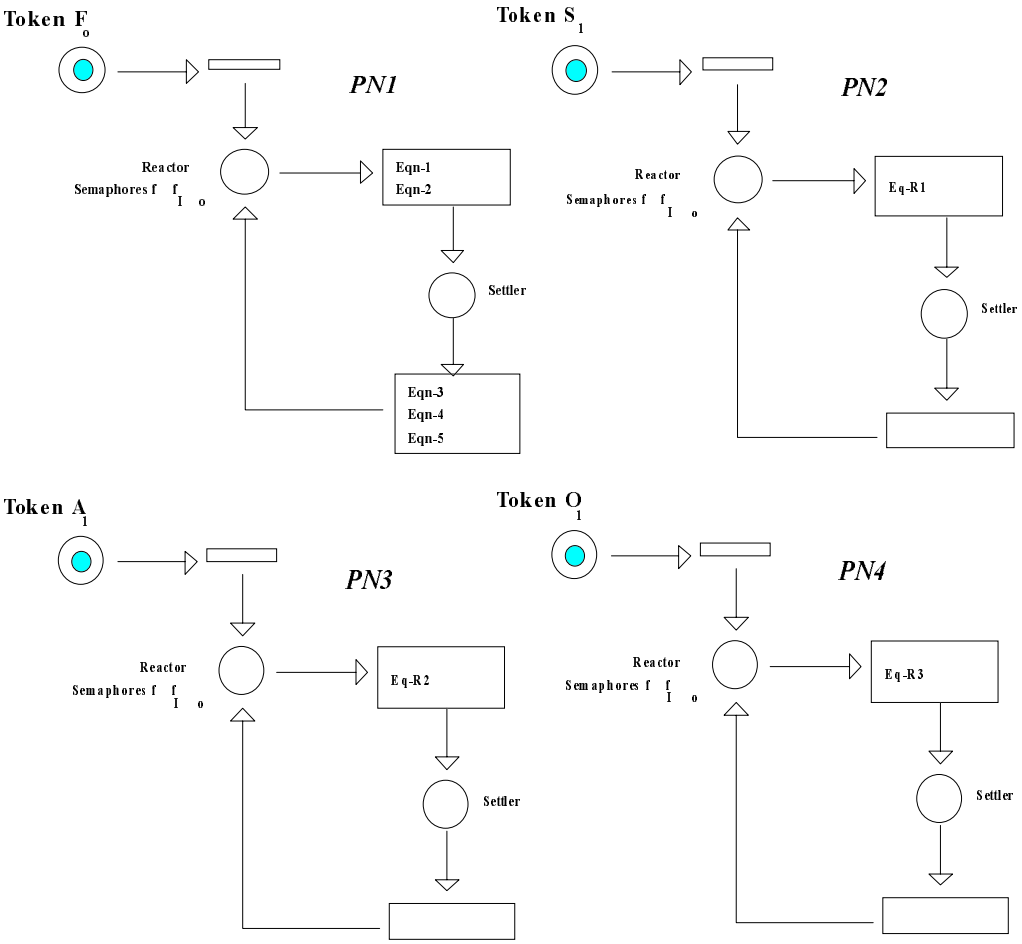


Figure 19

The following Petri Nets are devised to model the process:



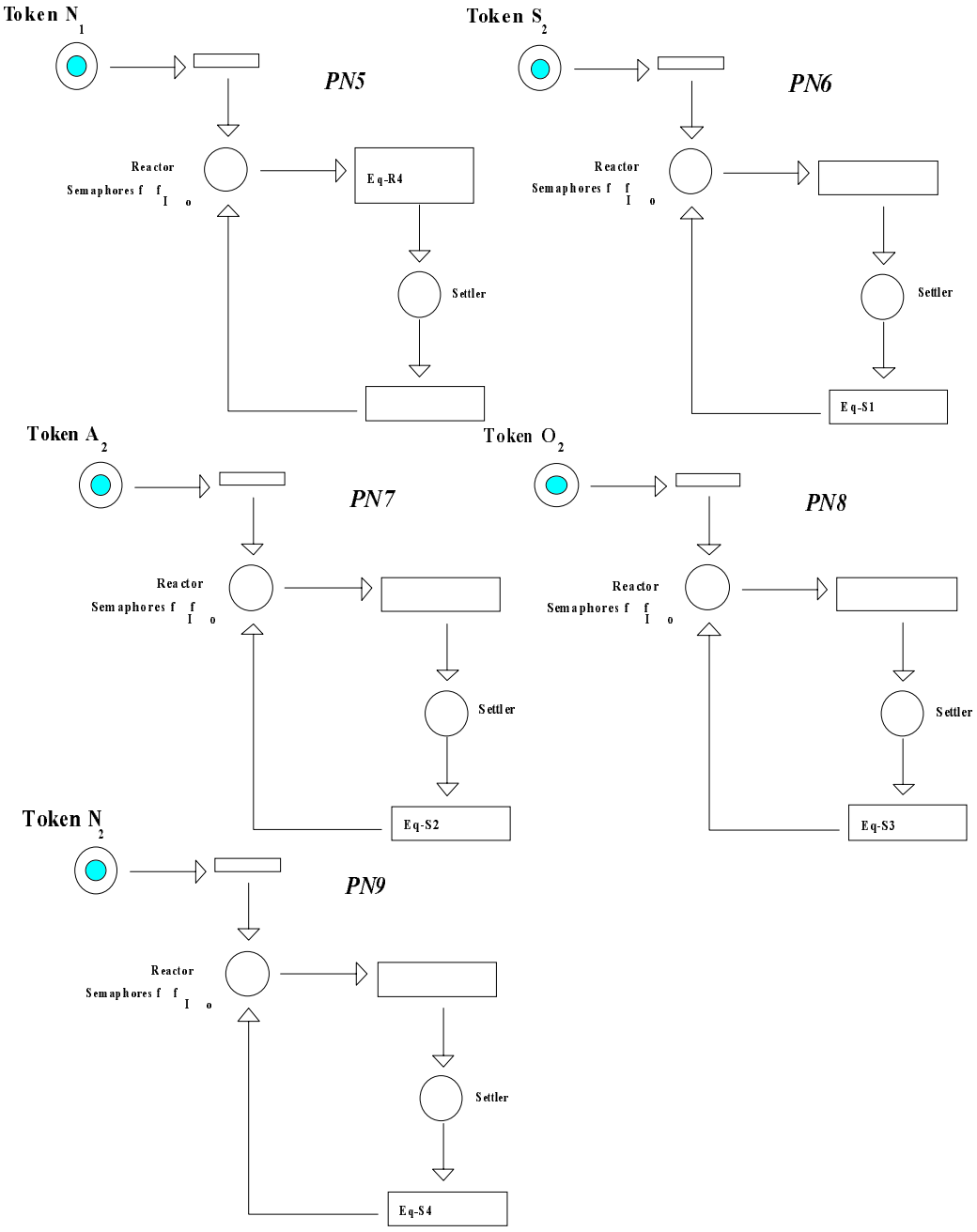


Figure 20

Thus, a model for fuzzy sub- Petri nets can be shown- A model of Inter-

Synchronization.

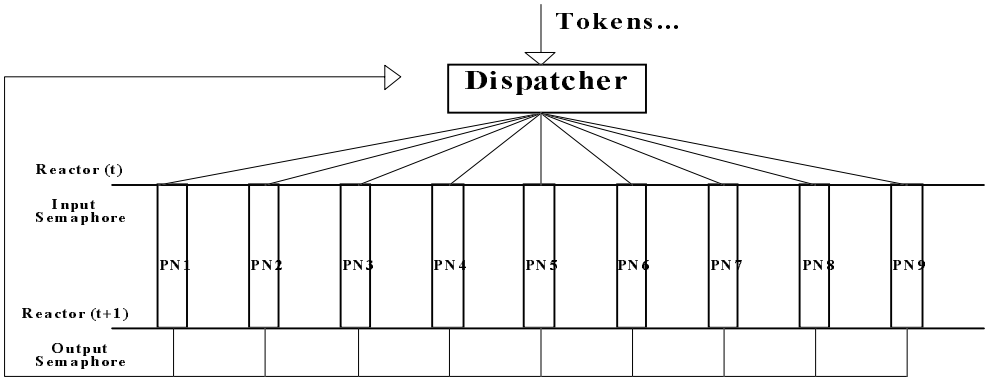


Figure 21

The competition between FSMs is carried out on basis of a contest length between FSMi and FSMj- and the FSM with the minimum change in substrate concentration (Ammonia, Organic)/ (Reactor, Settler) is selected. Thus,

$$\psi = \psi_1 \circ \psi_2 \circ \psi_3 \circ \psi_4$$

A cube for each ψ_i.

ψ_1 for :

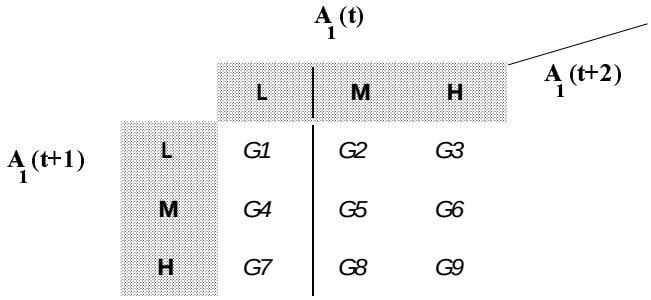


Figure 22

and so on up till ψ_4.

ψ_4 for S_2:

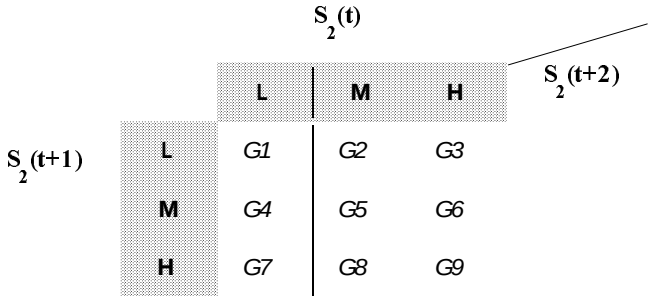


Figure 23

The final potential is taken as,
$\psi = \min \sum_i \psi_i$ over similar paths,

Again, for simplicity and ease of experimentation only, these were taken as constant numbers.

9 Conclusion

A novel design framework is presented that utilizes the power of GAs in search and exploration with the expressiveness and modeling capabilities of FSMs and PNs. A large potential of applications is expected to be modeled by such a system- all which have in common the characteristics of not having a predefined sequence of design steps and are distributed and communicating in nature.

The proposed evolutionary algorithm, together with its data structure, composed of finite state machines and fuzzy Petri nets, proved to be adequate in exploiting the search space of a particular design problem. The problem tackled here, is a linguistic one, with fuzzy variables. Numerical designs were not considered. The evolutionary algorithm has been utilized as a 'exploration' technique, rather than its optimization sense.

The main contribution of this research is considered as three-fold:

1- The introduction of a 'methodology' of design- which is an umbrella covering a large potential of applications.

2- The flexible data structure formulation (FSMs + FPNs), incorporating a large potential of application formalisms.

3- The adaptation of an evolutionary algorithm with specific operators for each instance of application-specific data structure formalisms.

These points were accomplished through a subdivision of application-dependent and application independent parts.

Design problems such as the design of 'monitoring' systems, 'control' systems, 'diagnostic' systems, and economical systems, are suitable for such a system. Thus, the design problem considered, is an *unperceived* one; that's to say, without a governing equation.

References

[Aho75] Aho, A.; Corasick, M. (1975). "Efficient String Matching: An Aid to Bibliographic Search", Communications of the ACM, v18, n6, pp333-340.

[Bad98] Badr, Amr. (1998). "A Hybrid Framework for Optimal System Design", PhD diss. Cairo Univ.

[Bak87] Baker, J. (1987). "Adaptive Selection Methods for Genetic Algorithms", in Proc. of the 2nd International Conference on Genetic Algorithms.

[Bat93] Battiston, E.; De Cindio, F. (1993). "Class Orientation and Inheritance in Modular Algebraic Nets", International Conference on Systems, Man, and Cybernetics, Conference Proceedings v2, pp717-723.

[Bat94] Battiston, E.; De Cindio, F.; Mauri, G. (1994). "A Class of Modular Algebraic Nets and its Support Environment", International Course on Petri Nets Notes, G. Rozenberg, C. Fernandez, M. Solar and V. Parada (Eds.), Editorial Universidad de Santiago.

[Bat95] Battiston, E.; Botti, O.; Crivelli, E.; De Cindio, F. (1995). "An Incremental Specification of a Hydroelectric Power Plant Control System using a Class of modulus Algebraic Nets", in De Michelis, G.; Diaz, M. (1995). Application and Theory of Petri Nets, pp84-102.

[Cao96] Cao, T.; Sanderson, A. (1996). Intelligent Task Planning using Fuzzy Petri Nets, Series in Intelligent Control and Intelligent Automation, v3, World Scientific.

[Dun92] Dun, I.; Heinzle, E.; Ingham, J.; Prenosil, J. (1992). Biological Reaction Engineering, VCH.

[Eng91] Engelfriet, J.; Leih, G.; Rozenberg, G. (1991). "Net based Description of Parallel Object-based Systems, or POTs and POPs", in de Bakker, J.; de Roever, W.; Rozenberg, G. (eds.). (1991). Proc. of School/Workshop in Foundations of Object-oriented Languages; LNCS 489, Springer Verlag.

[Fog92] Fogel, D. B. (1992). "Evolving Artificial Intelligence", PhD diss. UCSD.

[Gol89] Goldberg, D. (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley.

[Koz92] Koza, John. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection, Complex Adaptive Systems Series, MIT Press.

[Koz94] Koza, John. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs, Complex Adaptive Systems Series, MIT Press.

[Lee96] Lee, D.; Yannakakis, M. (1996). "Principles and Methods of Testing Finite State Machines: A Survey", Proc. of the IEEE, v84, n8.

[Lim94] Lima, P. U. (1994). "Intelligent Machines as Hierarchical Stochastic Automata", PhD thesis, Rensselaer Polytechnic Institute, Troy, NY.

[Mic96] Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs, 3rd Edn., Springer Verlag, New York.

[Wan88] Wang, F. Y.; Saridis, G. N. (1988). "A Formal Model for Coordination of Intelligent Machines using Petri Nets", in Proc. of the 3rd IEEE Int. Intell. Contr. Symp., Arlington, VA.

[Wan91] Wang, F. Y.; Kyriakopoulos, K. J.; Tsolkas, A.; Saridis, G. N. (1991). "A Petri-Net Coordination Model for an Intelligent Mobile Robot", IEEE Trans. on Systems, Man and Cybernetics, v21, n4.

[Wan93] Wang, F.; Saridis, G. (1993). "Task Translation and Integration Specification in Intelligent Machines", IEEE Trans. on Robotics and Automation, v9, n3.

[Wij93] Wijk, R.; Moller, D. (1993). Biomedical Modeling and Simulation on a PC, Springer Verlag.