Engineering with Computers (1997) 13: 1-19
© 1997 Springer-Verlag London Limited

Engineering with Computers

Upgrading Automation for Nuclear Fuel In-Core Management: from the Symbolic Generation of Configurations, to the Neural Adaptation of Heuristics

Ephraim Nissan[1], Hava Siegelmann[2], Alex Galperin[3] and Shuky Kimhi[4]

[1,2] Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, Israel; [3,4] Department of Nuclear Engineering, Ben-Gurion University, Beer-Sheva, Israel

Abstract. *FUELCON is an expert system in nuclear engineering. Its task is optimized refueling-design, which is crucial to keep down operation costs at a plant. FUELCON proposes sets of alternative configurations of fuel-allocation; the fuel is positioned in a grid representing the core of a reactor. The practitioner of in-core fuel management uses FUELCON to generate a reasonably good configuration for the situation at hand. The domain expert, on the other hand, resorts to the system to test heuristics and discover new ones, for the task described above. Expert use involves a manual phase of revising the ruleset, based on performance during previous iterations in the same session. This paper is concerned with a new phase: the design of a neural component to carry out the revision automatically. Such an automated revision considers previous performance of the system and uses it for adaptation and learning better rules. The neural component is based on a particular schema for a symbolic to recurrent-analogue bridge, called NIPPL, and on the reinforcement learning of neural networks for the adaptation.*

Keywords. Allocation; Design; Downtime; Expert systems; Machine learning; Neural networks; Nuclear engineering: in-core fuel management; Refueling; Reload

Correspondence and offprint requests to: Ephraim Nissan, [1] School of Computing and Mathematical Sciences, The University of Greenwich, Wellington Street, Woolwich, London SE18 6PF, UK. (Bitnet: E.Nissan@greenwich.ac.uk).
[2] Currently at the Department of Industrial Engineering and Management, The Technion, Haifa, Israel. (Bitnet: iehava@ie.technion.ac.il).
[3] Currently at the Department of Nuclear Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel. (Bitnet: alexg@bgumail.bgu.ac.il).
[4] Currently at the Kernforschungszentrum Karlsruhe, Institut für Neutronenphysik und Reaktortechnik, Postfach 3640, D-76021 Karlsruhe, Germany.

1. Introduction

FUELCON is an expert system in an industrially significant domain: nuclear engineering. The system's task is to provide a good fuel-reload configuration (i.e. refueling design), and thus to indirectly achieve minimization for the duration of the very costly shut-down periods at nuclear plants. FUELCON is already a working system that as is, can be applied industrially [1-5]. We report on current achievements of the project, and on new developments that upgrade the reasoning capabilities of the tool.

In FUELCON, peculiar heuristic domain-knowledge is applied to the generation of a very great number of configurations of how to allocate units of fuel of different kinds, inside the cases of the (geometrically and symmetrically schematized) core of the nuclear reactor. The type of reactor, the features of the individual plant, and its current state, along with the cumulated knowledge corpus of the specialty, as reflecting (at a deeper level) models of reactor physics, determine the selection of the relevant criteria of fuel allocation. These cannot be fully predefined, before the moment comes for shutting down a plant. Therefore, the design of the new allocation is possible only at this moment.

In FUELCON, a ruleset is applied to generate families of good configurations of fuel in nuclear reactor cores. Whereas the practitioner may be satisfied with the output, the domain expert is challenged to optimize not just the configurations, but his or her own heuristic as well: the results of one given iteration of the expert system are simulated by another component, which prompts the human expert to (manually) improve on the ruleset previously (manually) formulated.

In this work, we design a device for automated ruleset-revision. Because of the peculiar architecture of FUELCON, which involves a certain operation-loop, success with the device proposed is tantamount to achieving a full automation of the discovery process, at the task considered (apart from the initial formulation of the ruleset, as provided by the expert).

The automation of revision is accomplished by using neural networks learning algorithms, which tune the rules to yield better configurations, based on previous performance. For this aim, we translate the rules into a neural network using a particular technique: NIPPL is a language and translation schema defined by Siegelmann [6] for transforming rulesets into neural networks. Here, we use this schema to obtain translation from symbolic rules to analogue network. The application is not trivial at all, as many questions related to the neural structure and the learning still remain difficult.

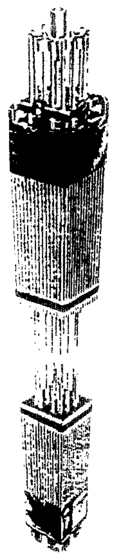


Fig. 1. An assembly of fuel rods. It fits in a position within the grid that constitutes the core of a nuclear reactor.

2. Preliminaries

As our work encompasses four subjects: fuel management, the system FUELCON, neural networks, and learning algorithms, we shortly describe some related background for each.

2.1. Preliminaries of Fuel Management in Nuclear Power Plants

1. Nuclear fuel is loaded into the core of a nuclear reactor. Fuel comes in fuel assemblies, i.e. packages of 200–250 fuel rods. These are 350 cm long (see Fig. 1). Fuel assemblies are inserted vertically in the core, in a grid of positions. In fact, whereas the core actually is in three dimensions, the core geometry is usually represented as a grid in the plane: Fig. 2 shows a planar schema of a reactor core.

For the purposes of designing the allocation of fuel, it is enough to reason on just one slice out of this table of perpendicular square cases: a symmetry of one eighth is typical (see Fig. 3). In this core geometry, important regions for reasoning about are, for example, diagonals, or then the central region of the core. (In the figure, it is the upper tip of the slice.)

2. Among the fuel rods, there is liquid coolant: a major (and, in traditional fuel management, simpler) kind of plant employs, as coolant, pressurized water. Pressure is needed to prevent the water from boiling. However, there are steam generators on the borders: their role is to act as heat exchangers. Of course, not all of the thermal energy thus produced can be exploited as electrical power, e.g. about 3000 thermal MW would yield about 1000 electric MW.

As the nuclear plant produces energy, the fuel assemblies become depleted. Typically, annually (but the length of such cycles is somewhat variable), plants are shut down, and fuel is reallocated into the core.[1] According to the degree of their partial depletion, one distinguishes between different types of assemblies. (There exist also other parameters setting a difference.) There is fresh fuel, and then one-, twice-, and thrice-burned fuel. The latter kind is discharged from the core, and no longer reused in the next interval of operation. It is replaced with fresh fuel.[2] Heuristics of allocation of units of the various kinds of fuel into the core geometry, take into account the difference between the various kinds, in terms of the degree of depletion. For example, one heuristic includes the avoidance of allocating fresh fuel at the centre of the core, to avoid too high a local power density

[1] An exception is such reactors that do not practice reload in batches of fuel, as there are no downtime periods for refueling; replacement is continuous. As said by Cochran and Tsoulfanidis [7]: 'Canadian deuterium uranium (CANDU) reactors [...] can operate with natural uranium as fuel. [...] Another unique characteristic of CANDU is that it can be replaced on line, i.e., without shutting down' (p. 5).

[2] In a *pressurized water reactor* (PWR), i.e. the kind of reactors we are concerned with, fuel replaced per year is 1/3. The fraction is different in other kinds of reactors: in *boiling water reactors* (BWR), it is 1/4, whereas in *high-temperature gas reactors* (HTGR), it is just 1/6. Cochran and Tsoulfanidis [7] define major types of reactors (Ch. 1) and give values for that parameter (p. 4). In CANDU reactors, replacement is continuous, not a fraction of a batch of fuel: cf. previous footnote.

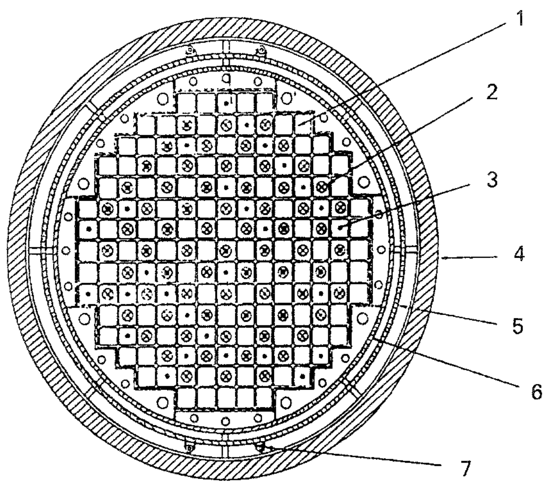


Fig. 2. A planar schema of the core of a typical nuclear reactor. 1, The position of a single fuel-assembly; 2, a control rod assembly location; 3, an in-core instrument location; 4, the reactor vessel; 5, the thermal shield; 6, the core barrel; 7, a surveillance specimen holder tube. Understanding these detailed notions is unnecessary to make sense of the text.

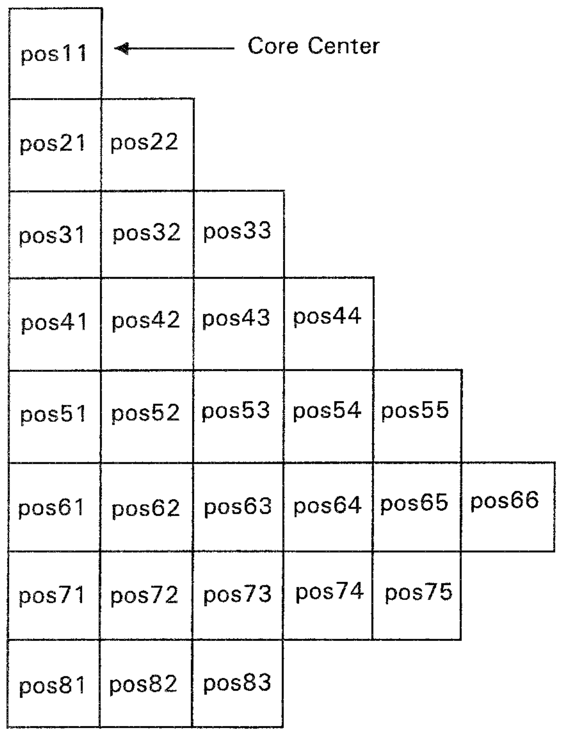

pos11
Core Center
pos21 pos22
pos31 pos32 pos33
pos41 pos42 pos43 pos44
pos51 pos52 pos53 pos54 pos55
pos61 pos62 pos63 pos64 pos65 pos66
pos71 pos72 pos73 pos74 pos75
pos81 pos82 pos83

Fig. 3. A slice, in one-eighth core symmetry, of the schema of a core. The positions in the core (i.e. the cases in the grid) are named after the coordinates inside this slice. The position-identifiers, as naming each case in this figure, are those in use in FUELCON.

distribution, which would be problematic for cooling purposes.

Fuel depletion reduces the potency of the fuel. The relevant parameter is the neutron multiplication factor, symbolized as K. It is defined as the number of neutrons created per one neutron destroyed. To achieve continuous power production, an adequate chain reaction is necessary. This is expressed by saying that the reactor core has to be kept critical. For this, $K = 1$ is the minimal value that it is strictly necessary to maintain, but practically, a higher value of K is maintained, i.e. we need a certain excess of criticality. This is meant to sustain reasonably long inter-refueling intervals.

3. Fuel management includes two domains: the management of fuel to be acquired and in store is a fairly complex domain; instead, the area of our own project is *in-core fuel management.* The (in-core) fuel management problem is the problem, for the (in-core) fuel manager, of designing the reload, i.e. of determining the configuration of fuel in the core, in order to start a new operation period at the nuclear plant.

Downtime periods at plants are costly: they typically take a few weeks per year; just consider the case of a plant that produces approximately one million dollars of electricity per day.

It is legitimate to wonder: couldn't the fuel manager do his reload design job during the previous operation cycle? This would allow arrival at the downtime period with a ready design, thus cutting down the inactivity period of the plant.

The experienced engineer can forecast power profiles for the core, according to the design implemented at the beginning of the current operation cycle. Realistically, a forecast for the end of that cycle (EOC) will not match the actual state of the reactor, because of unforeseen reasons, notwithstanding a flexibility window. Differences would be such that were a design prepared for the next cycle based on forecasts for the end of the current cycle, that solution would not be robust enough to fit the actual situation at EOC (see Fig. 4).

A solution has to be devised by customizing it for the given plant: plants have their own individuality, because of their configuration. Moreover, the way plants operate may vary from place to place because regulations are different in different countries (e.g. it happens that a country forbids the use of plutonium in fuel, as its proliferation could subserve military purposes).

Even if we are to refer to the same plant, predictions for the next cycle cannot be valid, as, for example, unpredictable weather conditions can cause a shift in energy consumption (for the purposes of domestic and institutional climatization of interiors). Therefore, there is a shift in energy supply requirements at a given plant subserving the given community.

As to time-dependent variability, it depends, at a

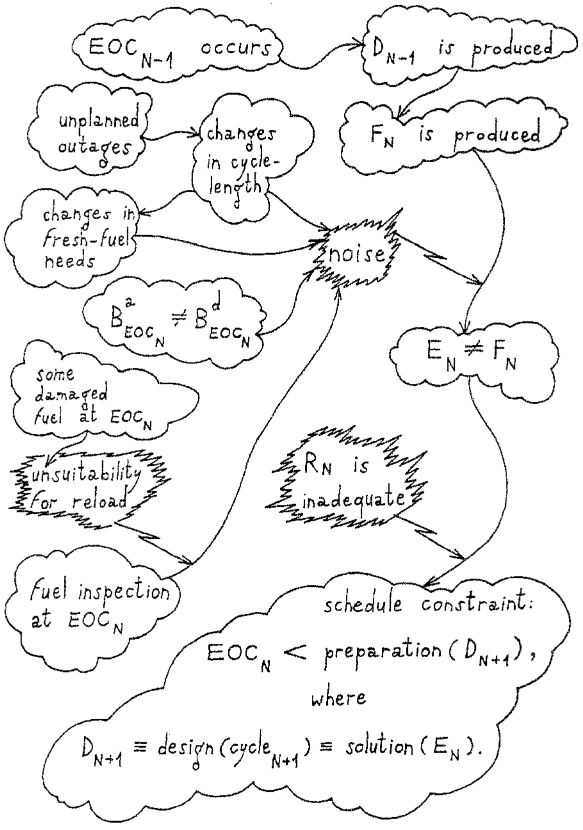

EOC_{N-1} occurs
D_{N-1} is produced
unplanned outages
changes in cycle-length
F_N is produced
changes in fresh-fuel needs
noise
$B^a_{EOC_N} \neq B^d_{EOC_N}$
$E_N \neq F_N$
some damaged fuel at EOC_N
unsuitability for reload
R_N is inadequate
fuel inspection at EOC_N
schedule constraint: $EOC_N <$ preparation (D_{N+1}), where $D_{N+1} \equiv$ design $(cycle_{N+1}) \equiv$ solution (E_N).

Fig. 4. Why the next-cycle refueling design is unavailable before downtime periods: a list of reasons and their mutual dependency. Key:

$B^a_{EOC_N}$: actual fuel-burnup cumulated values at EOC_N.

$B^d_{EOC_N}$: designed fuel-burnup cumulated values at EOC_N.

D_N: design of the in-core fuel-allocation for the *N*th operation-cycle.

E_N: actual end-of-cycle state of the reactor core, once the *N*th operation-cycle is completed.

EOC_N: the end-of-cycle of the *N*th operation cycle at the given plant.

F_N: forecasts for the state of the reactor core at the end of the *N*th operation-cycle at the given plant.

R_N: robustness of solutions for the fuel-management problem (i.e. of such configurations that may be designed for fuel-reload into the core) as devised for F_N.

plant, also on the availability of fuel in store. Another kind of time-variability, this one applying to longer spans, is due to political realities: in a given country, regulations are sometimes modified through political intervention, as prompted by public pressure (e.g. in the aftermath of widely reported accidents).

4. Let us consider the standard practice of the engineer who is responsible for designing the configuration according to which the reactor core is going to be refueled at the next EOC. This procedure is shown in Fig. 5.

This procedure (unlike what we do in our FUELCON expert system) embodies such a best-first search that starts from an initial candidate, and follows with a series of trial-and-error correction steps of a local nature. The suitability of best-first search stems from the very nature of the procedure. The fuel manager makes an effort to avoid potentially dangerous patterns, with regard to the local power peaking in the process of looking for the 'best' position, within the core grid, for each fuel assembly. Heuristics do exist, and the more experienced the fuel manager is, the more s/he is able to carry out 'mentally computed' evaluations, corresponding to a heuristic model of reactor core physics.

Of course, the solution thus obtained must be simulated. Not only that, standard simulations are mandated by current legislation and plant manufacturers' guidelines. An expert fuel manager can predict roughly, without detailed calculations, a power profile across a core, and even local power spikes. However, power density distributions in the core depend on physical properties of different materials, which make up the reactor core itself, and these properties are described by complicated functions of energy and space. An accurate evaluation of the spatial power distribution involves a series of intensive calculations (to be performed by a simulator) solving a space–energy dependent Boltzmann equation.

5. Figure 6 illustrates three different classes of computer tools for assisting the fuel manager in devising configurations for refueling. A survey of extant tools is beyond our present scope [8]. Because of relevance to the present work, we mention an expert system prototype, that looks for just one solution by modifying a given configuration.

At IntelliCorp, the prototype was developed (using the KEE shell) of an interactive fuel-shuffling knowledge-based system. Petschhat *et al.* [9] described it focusing on the application, whereas Faught [10] stressed the discussion of knowledge-engineering aspects (see also Rothleder *et al.* [11], that addresses nuclear engineers).

As far as we know, that particular project did not proceed beyond the early prototyping phase. There exist, at the present state of the art, other projects that reached a more advanced phase, e.g. in simulated annealing as applied to the in-core fuel-management problem. (Kropaczek and Turinsky [12] described FORMOSA, a program that combines GPT and optimization by simulated annealing.) We are mentioning the IntelliCorp expert system in particular, because like our own project, it adopted rule-based expert systems as a technology.

The IntelliCorp system handles differently two major classes of plants (pressurized water reactors,

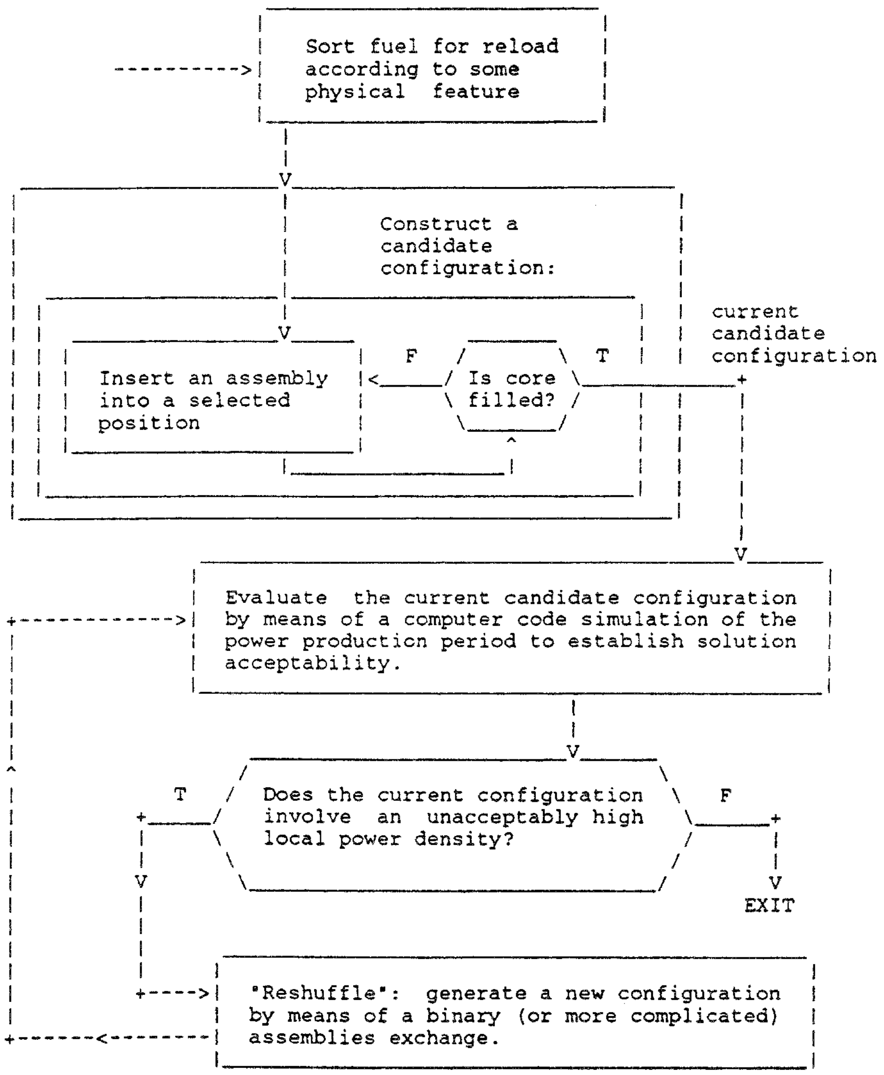

Sort fuel for reload according to some physical feature
Construct a candidate configuration:
Insert an assembly into a selected position
F
Is core filled?
T
current candidate configuration
Evaluate the current candidate configuration by means of a computer code simulation of the power production period to establish solution acceptability.
T
Does the current configuration involve an unacceptably high local power density?
F
EXIT
"Reshuffle": generate a new configuration by means of a binary (or more complicated) assemblies exchange.

Fig. 5. A standard solution method resorted to by human fuel-managers to solve the fuel management problem, i.e. to design how to refuel the reactor core. Unlike what we do with FUELCON, here it is best first search we have; it starts from an initial candidate configuration, which is gradually modified.

and such that use so-called burnable poisons), whereas in FUELCON, handling is substantially similar.

In shuffling, backtracking is expressed through binary exchanges made in the reload design: the positions of two fuel assemblies are switched. Such backtrack steps are aimed to correct local problems, e.g. excessively high power density of a specific fuel assembly. When it is the engineer that generates a candidate solution, the manual procedure consists of consecutive single placements (on the paper in front of the human fuel manager) of the fuel assemblies into one of the available core positions, until the core is filled. It is a heuristically guided depth-first strategy. Next, the manually obtained configuration is analysed (typically, by means of software), and, if found unsatisfactory, either the engineer or a shuffling system that s/he uses, modifies the solution by a binary exchange of assemblies. The basic configuration pattern is preserved, unless a large number of such binary exchanges is carried out. With such 'blinkers' on the *observability space* (the latter being the idiom from systems and control), local optimality is a prized target. With FUELCON, instead, we have been more ambitious.

2.2. Preliminaries of FUELCON

Unlike the IntelliCorp expert system, our own tool, the FUELCON expert system, does not simply assist the user in shuffling a configuration that s/he has provided as input (according to personal experience, or real case studies published in the domain literature). Indeed, FUELCON is not fed an input configuration, but instead it incorporates a replaceable ruleset, as formulated by a domain expert: the search is carried out, not for a single optimal solution, but for a set of alternative allocations (i.e. fuel configurations) grouped into families; these are typified

```
 ____________________________________
|                                    |
|     MANUAL  DESIGN  PACKAGES:      |
|====================================|
|                                    |
|     help the engineer to produce   |
|    reload configurations according |
|     to his or her own expertise:   |
|                                    |
|  _________________   ____________  |
| |                 | |            | |
| | in calculations | |            | |
| |  for analyzing  | | by producing| |
| |   the reactor-  | |  graphics  | |
| |   core physics  | |            | |
| |_________________| |____________| |
|              |              |      |
|__________    |  __________  |  ____|
               |              |
calculate      |              | visualize
               |              |
___________V______________V__________
|                                    |
|          EXPERT  SYSTEMS:          |
|====================================|
|                                    |
|   incorporate the design guidelines |
|   of reload designers, and propose |
|        good configurations:        |
|                                    |
|       either            or         |
|  _________________  _____________  |
| |                 ||             | |
| | by reshuffling: || by a broad  | |
| |   for just one  || search for  | |
| |     solution.   || solution-   | |
| |                 || families.   | |
| |   (IntelliCorp) || (FUELCON)   | |<----+
| |_________________||_____________| |     |  search for
|____________________________________|     |  (near-)
 ____________________________________      |  optimality
|                                    |     |  (local or
|       OPTIMIZATION  PACKAGES:      |-----+   global)
|====================================|
|                                    |
|   embody a traditional algorithmic |
|  approach; their evolution concerns: |
|                                    |
|  * the class of general optimization |
|    methods  (integer programming,  |
|    simulated annealing, etc.);     |
|  * the way the nuclear-physics     |
|    problem is formulated.          |
|____________________________________|
```

Fig. 6. Classes of computer tools for reload design. Arrows between boxes stand for shared roles. This classification is based on Galperin *et al.* [1] (Sec. 4) and Parks and Lewins [8].

by the given ruleset that generates them, and which, in turn, embodies heuristics reflecting a given generic conception. The given input situation is typified by the given reactor (whose core has a given geometry), and the time-dependent given pool of available fuel.

Let us describe the way configuration-families are generated. The expert system accesses, in the database, the geometry of the reactor core concerned, and the pool of fuel-assemblies, as being subdivided by type. The generation of the set of configurations is guided by the ruleset being consulted, and is *ex nihilo*, i.e. starting with an empty core.

The search, in FUELCON, is forward-oriented and breadth-first. There is no backtracking. According to a beam-search algorithm, a tree of configurations is developed, level by level, through partial configurations as intermediate stages. Each leaf in the tree corresponds to one full configuration, i.e. to a fully loaded core that fully exploits the available fuel. Each level in the tree is associated with a particular fuel assembly (out of the available pool kept in store), according to a predefined order as given in a loading sequence.

For efficiency reasons, the loading sequence is given: actually, it is part of the contribution of the domain expert, just as is the ruleset. However, basically, it would be easy to automate the generation of the loading sequence, too: those fuel-assemblies about whose class there are elimination rules, are put ahead in the loading sequence; the more numerous are the rules that affect the class of the assembly, the earlier the place has to be of that assembly within the loading sequence.

As to the rules in FUELCON, they represent two distinct types of knowledge: generic principles from physics, and local, specific knowledge suited to accommodate given situations. When an experienced user uses FUELCON, s/he is typically ambitious, as to the goals of optimization. Moreover, the researcher in the domain typically wishes to test new versions of a given ruleset s/he had formulated, in order to obtain improved configurations out of the family generated, or improved families of configurations. Then, an operation-loop in using FUELCON takes place, for the same given input problem. Each single iteration includes a generation phase, and an evaluation phase. Figure 7 illustrates the generation phase as during a single iteration. (The configuration-base is that part of the database that stores the collection of configurations being generated.)

Downstream of the rule-based generator of configurations (which is coded in Lisp), the output is fed to NOXER, a locally developed simulator. The results of NOXER are both visual and numeric. A 'cloud' of solutions is displayed in the cartesian plane of two parameters, within a 'window' of admissibility: this is a region laying under a horizontal line in the display. Its 'southwestern' corner is at the origin of the coordinates. In this admissible region, configurations on the right are more efficient than configurations on the left. The domain expert may wish to move the 'cloud' of solutions into a 'southeast' direction, in order to get several configurations that are both safe and very efficient. To do so, s/he manually revises the ruleset. This is the step that we have set to automate by means of a neural component, based on the NIPPL

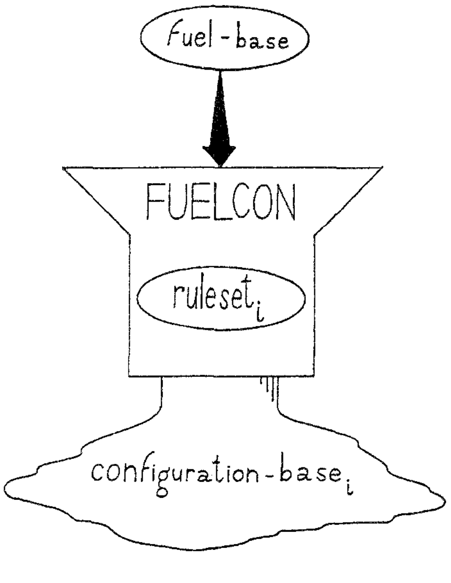


Fig. 7. A projection of a single iteration on the ruleset and database, from the viewpoint of the FUELCON generator of configurations as candidates for the fuel-reload to be implemented in the reactor core. The database contains the pool of output configurations.

language and symbolic-to-neural conversion schema. Figure 8 shows the loop of how the FUELCON/NOXER integrated tool is used.

Now, let us consider Fig. 9. It illustrates the subcomponents of the ruleset and the database in FUELCON. The ruleset includes *elimination rules* (that are never revised, and that prevent the generation of forbidden configurations), and *preference rules* (which are subject to revision, and that typify the search-subspace). The database of FUELCON includes a subdatabase of fuel types and units, and a

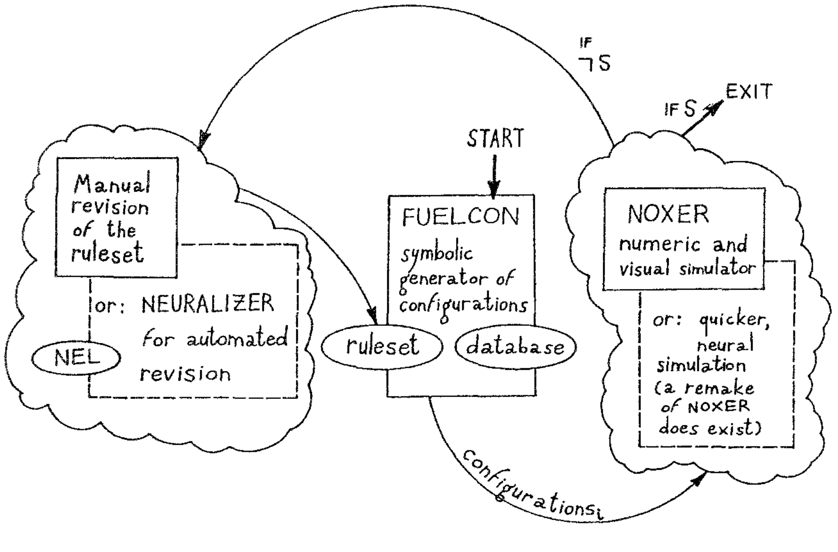

IF ¬S
IF S
EXIT
START
Manual revision of the ruleset
or: NEURALIZER for automated revision
NEL
FUELCON symbolic generator of configurations
ruleset
database
NOXER numeric and visual simulator
or: quicker, neural simulation (a remake of NOXER does exist)
Configurations$_i$

Fig. 8. The integrated operation-loop. Configuration-set generation and simulation are followed by ruleset revision. S stands for the statement: $\exists c$, $c \in Configurations_i \bigwedge Evaluation(Simulation(c)) \geq Goal_i$. The index in $Goal_i$ reflects the fact that the user will often set higher goals from iteration to iteration.

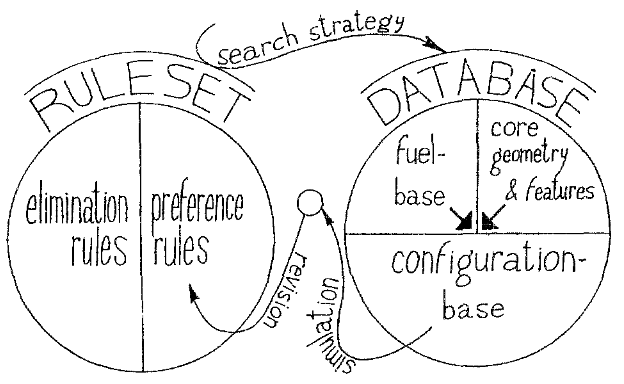


Fig. 9. The subcomponents of, and relationship between, the ruleset and the database in FUELCON.

component that describes the geometry and the features of the reactor core (as per the symmetric slice considered). The two combine into the third subcomponent of the database: the collection of configurations being developed.

Figure 10 illustrates the way a single iteration in the operation loop (shown in Fig. 8) affects the configuration-base and the ruleset. In other terms, Fig. 10 shows how the loop modifies parts inside the components for which the two circles in Fig. 9 stand.

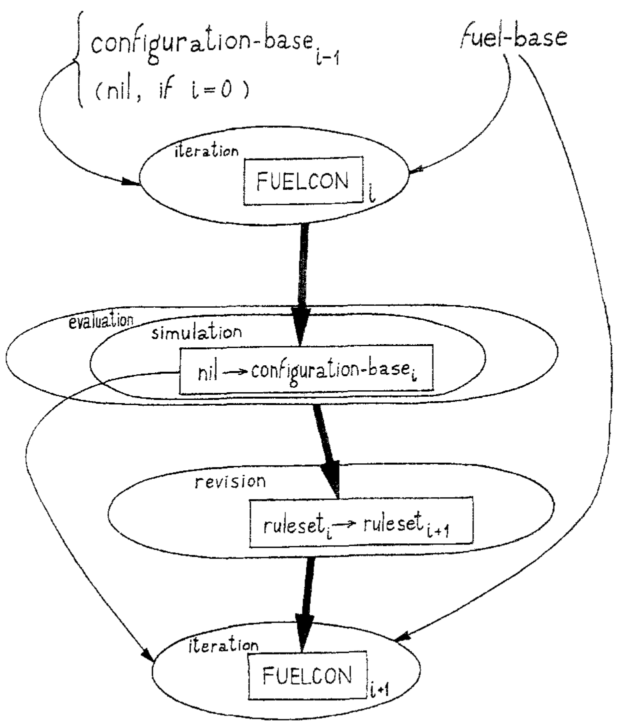


Fig. 10. A projection (more detailed than in Fig. 7) of a single iteration of the operation loop of the global system, on the macro-components of the database, and on the ruleset.

Evaluation, downstream of generation, is by NOXER, and then manually, by the user, or automatically, by our neural component, as he/she/it checks the output of NOXER: such evaluation has to be conducted on the output sets of full-load configurations, as we have seen. Improvement is checked by comparing the visual simulations of successive iterations, that each includes full generation, instead of backtracking before the core is filled. Such an ergonomic choice, in the way the operation of FUELCON was conceived, is better than the following alternative, that could be conceivably put forth, as based on our previous description of our project: evaluating partial solutions, i.e. comparing pairs of such configurations that were only partially loaded. Such a course of action would be difficult and generally rather inconclusive, if we were to perform such comparisons on the basis of given hypotheses. We have tried to avoid the basic deficiency of shuffling systems, a disadvantage because the starting point of the search is somewhere in the solution space. Such an early decision deprives the user of the advantage of a broad search: s/he has to make the first guess, and that guess is decisive for the outcome of the session.

2.3. Preliminaries of Neural Networks

Artificial neural networks provide an appealing model of computation. Such networks consist of an interconnection of a number of parallel agents, or *neurons*. Each of these receives signals as inputs, computes some simple function, and produces a signal as output, which is in turn broadcast to the successive neurons involved in a given computation. Some of the signals originate from outside the network and act as inputs to the whole system, while some of the output signals are communicated back to the environment and are used to encode the end result of the computation.

The study of recurrent networks has many different motivations. They constitute a very powerful model of computation, they are capable of approximating rather arbitrary dynamical systems, and this is of use in adaptive control and signal processing applications [13–15] and, most importantly for us, they constitute a powerful tool of automatic learning.

The classical approaches of computer science and artificial intelligence are based on understanding and explaining key phenomena in a discrete, symbolic manner. These approaches have the limitations of human understanding, and more seriously, they can not change or adapt by observing their own performance or additional data.

Neural networks, on the other hand, estimate input–output functions. They are trainable dynamical systems which learn by observing a training set of input–output pairs. In speech processing applications and language induction, recurrent net models are used as identification models, and they are fitted to experimental data by means of a gradient descent optimization (the so-called *backpropagation* technique) of some cost criterion [16–20]. Unlike statistical estimators, they estimate functions without assuming a mathematical model for the dependence of the output on the input.

We focus on recurrent neural networks, that is, networks of simple processors where the architecture allows for feedback loops. Each processor's state is updated by an equation of the type

$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}x_j(t) + \sum_{j=1}^{M} b_{ij}u_j(t) + c_i\right), \quad i = 1, \ldots, N \quad (1)$$

where N is the number of processors and M is the number of external input signals. Oftentimes, the function σ is the classical sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

Sometimes, however, we prefer it to be the linear-saturated function:

$$\sigma(x) := \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \le x \le 1 \\ 1 & \text{if } x > 1 \end{cases} \quad (3)$$

A subset of the N processors, say $x_{i_1}, \ldots, x_{i_l}$, are the output processors; they are used to communicate the outputs of the network to the environment.

The input–output map of the network depends upon the constants (also called weights) a, b, c. Learning, or adapting, is thus the process of fitting the constant so that the network computes what is required. Such fitting can be done numerically, as the network is a dynamical system.

2.4. Preliminaries of Learning Algorithms

There are various techniques for adapting the constants, depending on the architecture of the network, on the type of task required, and on the type of information that is available to learn from. Some of the learning algorithms are based on learning in biological neural networks, such as the Hebb rule; others are totally numerical, such as gradient descent techniques.

The learning algorithms can roughly be described as belonging to one of three main methodologies. The classical neural network learning paradigms are the *learning with teacher* approaches. The assumption is that a set of pairs of (*input*, *desired-output*) of the neural network are provided and the network adapts itself to comply with them. The main technique for adapting the network is the gradient descent, also called *backpropagation.*

The second methodology is *unsupervised learning.* Here, no teacher provides the output of the network, only input strings are taken (sampled) from some large input set, and the network is to classify them by similarities. The most common network of this type is the Kohonen self-organizing map.

The third methodology is the one we adopted. It is called *reinforcement learning* [21–23]. This one is commonly used in control applications. In reinforcement learning problems, it is common to think explicitly of the network as a controller in an environment (see Fig. 11). The environment supplies the inputs to the network, receives its output, and then provides the reinforcement signal. This signal gives no hint of what the right output should be, but evaluates how good the current output is. It is therefore important to have some source of randomness in the network so that the space of possible outputs can be explored. The output units are thus governed by the standard stochastic rule:

$$\text{Prob}\,(S_i = b) = \sigma_\beta(h_i) = \frac{1}{1 + \exp(2\beta h_i)} \quad (4)$$

where $h_i = \sum_j \omega_{ij} V_j$ is the input net to the neuron, that is, the linear combination of the values of the neurons and possibly the external input. We first have to define the error δ_i^μ, which is the error in the output unit i when the input to the network is the pattern μ.

Assume that the score r^μ of the input pattern μ is binary. The desired binary output D_i of the ith output neuron is then well defined: S_i for $r^\mu = 1$ and $-S_i$ for $r^\mu = -1$. The error in the output neuron can then be easily computed by

$$\delta_i^\mu = D_i - \langle S_i^\mu \rangle$$

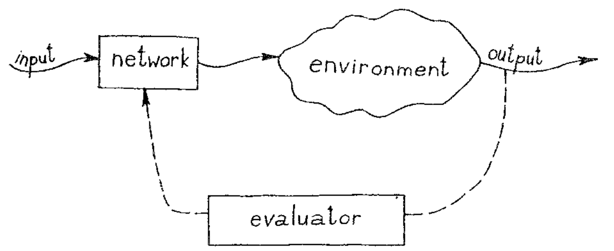


Fig. 11. A schema of reinforcement learning for adapting neural networks.

where $\langle S_i^\mu \rangle$ is the average of the ith output unit for input μ. If the score value r^μ is in the range [0, 1] then, by Barto and Jordan [24],

$$\delta_i^\mu = r^\mu[S_i - \langle S_i^\mu \rangle] + (1 - r^\mu)[-S_i - \langle S_i^\mu \rangle]$$

The update rule of the weights is

$$\Delta\omega_{ij} = v(r^\mu)\delta_i^\mu V_j^\mu$$

where v is a coefficient deciding the amount of adjustment. This rule only explains how to adapt the weights of the output units. We can, however, propagate the error δ and adjust the other weights by the backpropagation technique.

For more background on networks and reinforcement learning, see, for example, the textbook by Hertz *et al.* [25].

3. An Example of a Ruleset and Manual Revision

3.1. The Initial Ruleset

Let us exemplify a configuration-generating ruleset. In one real-case study, the following initial ruleset was used [1, 2]:

1. *Don't load any fresh fuel-assembly in any such position that its distance from the centre of the core is shorter than the distance of POS44 therefrom.*
   (Distance is an integer number: a number of positions separating the given position from the centre, *not* the distance on the paper, by which, say, measuring a case diagonally yields a longer distance than measuring the case along one of its four sides.) This rule is meant to prevent the loading of fresh assemblies into the innermost region of the core; and this, in order to make it less likely that configurations are generated that would yield too high local power densities in the innermost region of the core.
2. *Don't load a fresh assembly in such a position that is adjacent to another position where there is another assembly of the same kind, except when one of those two positions is in a corner position, that is, except when one of those two positions is adjacent along two of its sides to the reflector* (i.e. the water that surrounds the fuel-assemblies in the core).
   This rule, which prevents placing two fresh assemblies side by side, has the same purpose as Rule 1. Indeed, placing two fresh assemblies adjacently when none of them contains rods of burnable poison, would lead, in most cases, to an increase in local power density at beginning-of-cycle (BOC), if the region considered is not on the periphery of the core.
3. *Don't place any twice-burned assembly in any of the positions of the eighth row and of positions POS74, POS75, and POS66.*
   This rule, which prevents placing high-burnup assemblies in the outermost region of the core, fits into the general strategy adopted, which is 'from the outside, inside' in principle, that is, that more power is expected to go on the periphery, and that eventually allows one to replace some fresh assemblies with low-burnup already used assemblies.
4. *Don't load a twice-burned assembly in such a position that is adjacent to another position where there is another twice-burned assembly, if the positions considered are comprised in rows 5–8 in the core.*
   This rule, which prevents placing high-burnup assemblies adjacent to each other, is intended to direct the process towards the generation of such reload configurations that fit into the category typified by a checkboard pattern (in respect of burnup levels).
5. *Don't load any such twice-burned assembly that has a very high value of cumulated burnup (over 20500 MWd/t), adjacently to a position containing a twice-burned assembly.*
   This rule is in line with Rule 4. Rule 5 is meant to prevent the concentration of such assemblies that have a high burnup (corresponding to a low value of K, the neutron multiplication factor), this time (as opposed to Rule 4) in any region of the core, and this in order to prevent the formation of 'hollows' in respect of power density.
6. *Don't load any twice-burned assembly in any position belonging to any of rows 2, 3 or 4, if more than one position adjacent thereto does contain a twice-burned assembly.*
   This rule is meant to prevent a concentration of high-K fuel in the innermost region of the core: this is an important region, in terms of neutron flow. Rule 6 is intended to prevent the formation of configurations where local power densities would exceed the threshold allowed. Moreover, Rule 6 is intended to convey the generation process into producing burnup checkboard configurations.
7. *If it is a twice-burned assembly that is currently being considered, then choose for it (from amongst those positions that were not forbidden by Rules 1–6) that position whose distance from the centre of the core is minimal.*
   (Cf. Rule 3.)
8. *If it is a once-burned assembly that is currently being considered, then choose for it (from amongst those*

positions that were not forbidden by Rules 1–6) that position whose distance from the centre of the core is minimal.

Rule 8, along with Rule 7, and along with the given order of the loading list, is intended to direct the generation process into producing such configurations that the lower the burnup value of the assembly, the more important for neutron flow the region is where that assembly's position has been selected.

3.2. Elimination Rules versus Preference Rules

Rules 1–6 are elimination rules, whereas Rules 7 and 8 are not mandatory from the physical viewpoint, but are preference rules meant for pruning, and are enacted last. A constraint was imposed, that the size of the space of solutions must not exceed about one thousand solutions at any moment in the generation process. What this constraint affects most is the possibility to optimize the way partly-burned assemblies are reloaded. However, two main considerations were retained:

- Experience teaches that, for a problem and general policy of the kind considered, the influence, on the fuel cycle length, of how partly-burned assemblies are ordered inside the core, is smaller than the influence thereon of how the fresh assemblies are placed.
- Preference rules, along with the constraint on the size of the solution space, allow for a shorter processing time. (However, when we switched supporting systems, generation time became much faster, measured in seconds, and thus less of a problem. On the other hand, we have already seen that the real bottleneck is in the time required for performing the simulations on the solutions obtained: it is this that advises in favour of wavering the requirement that the absolute optimum be found, in favour of good local optima.)

3.3. Ruleset Revision Steps

Once results were obtained and simulated, the domain experts spotted which positions in the core configurations caused the maximal power density, which led to modifications in the ruleset, as follows.

- Rule 2, about adjacent fresh-fuel assemblies, was modified to allow adjacency even when one of the positions is adjacent to the reflector on just one side (instead of, as before, when adjacency to the reflector is on two sides, that is, in corner positions). Such a modification leads to an increase in the number of fresh-fuel configurations generated.
- One more rule was formulated that is concerned with once-burned assemblies, and it is an exclusion rule. It is as follows:

  Rule 9: Don't place any once-burned assembly in any position adjacent to a fresh assembly, if the position considered for loading the once-burned assembly belongs to any of rows 2, 3 or 4 in the core.

  Indeed, the analysis of the configurations generated by the first step indicated that the situation excluded by the new Rule 9 caused an increase in local power density, in the region involved, to a range of values between 1.4 and 1.5. As the threshold assumed is 1.4, it was suitable to have a specific rule preventing this kind of situation.

Now, the generator was run again; simulation led to further revision, of both the ruleset, and the given loading sequence.

The gradual improvement of solutions can be seen in the three parts of Fig. 12. In the upper display, no configuration in the family generated is admissible, as none falls within the admissibility window, i.e. the region under peaking = 40.

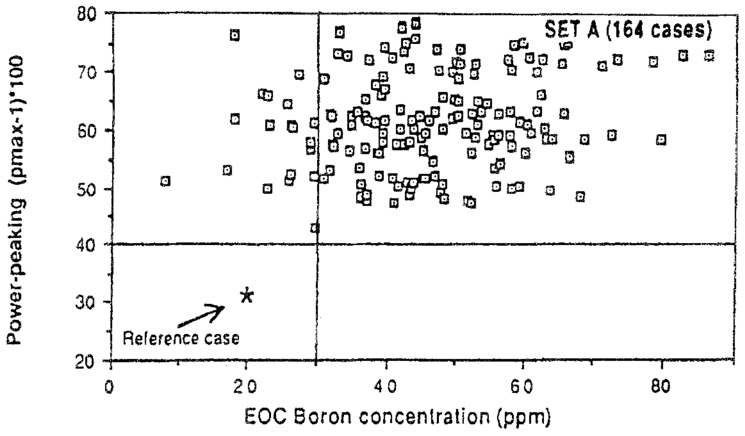
The middle display, instead, features some admissible solutions, of which, moreover, several are good, because, inside the admissibility window, they are up (with high peaking, though below the threshold), and on the right (with high boron concentration at end-of-cycle). Then, in the third display, we can see that the family itself is improved, as it tends to concentrate closer to the admissibility region, and in a higher percentage than before within its window; moreover densly on the right-hand side (the better part) of the admissibility window.

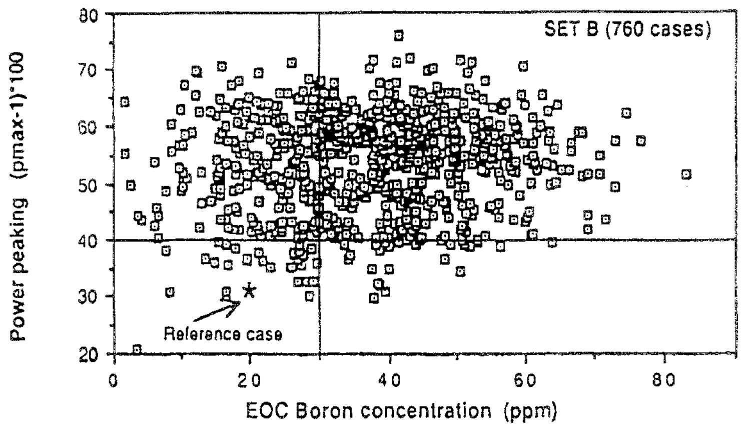
In Fig. 13, a sample output configuration is shown. Fuel assemblies are allocated to the positions of the reactor core, that here is shown as in a one-eight symmetry, which, in turn, is the symmetry for which the reasoning was carried out.

3.4. Contribution of the Neural Component

It is especially because of its positioning within the architecture, that the neural component we are adding does contribute to the automation of discovery. Indeed, the neural component, which we are going to discuss in the following two sections, is inserted in the architecture with the role of ruleset-reviser (rather than, for example, the simulator).

The point is that FUELCON generates configurations by means of a ruleset, originally provided by the expert, and later revised by the expert. Revision is a 'noble' task, intellectually speaking, but we are trying to show that the heuristics of ruleset-revision





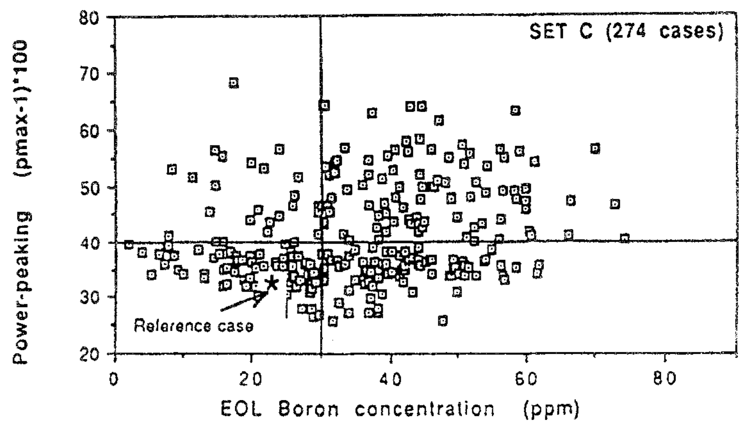
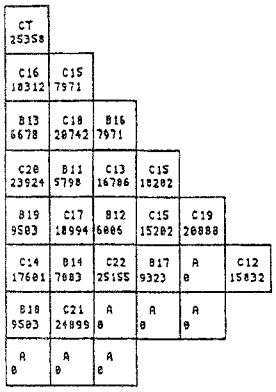


Fig. 12. An example of graphic display of the simulation of an output family of configurations ('dots'), evolving from iteration to iteration. In the quadrant shown, a horizontal line and a vertical line are drawn, that divide it into four regions. Those configurations falling in the two upper regions are forbidden, because of safety reasons. Configurations in the lower leftmost rectangle are less efficient than those in the also admissible region on its right. The more an admissible configuration is on the right, the more it is efficient. Therefore, the human expert wishes to have the 'cloud' of dots moved into a 'southeast' direction, during the next iteration. Another effect that the human expert may wish is 'zooming', i.e. a higher density in the cloud, a smaller spread in the family generated by the heuristics currently tested, as embodied in the ruleset and gradually improved from formulation to formulation.



CT 25358					
C16 10312	C15 7971				
B13 6678	C18 20742	B16 7971			
C20 23924	B11 5798	C13 16706	C15 10202		
B19 9503	C17 18994	B12 6006	C15 15202	C19 20888	
C14 17601	B14 7003	C22 25155	B17 9323	A 0	C12 15032
B18 9503	C21 24899	A 0	A 0	A 0	
A 0	A 0	A 0			

Fig. 13. A sample output configuration. To each position in the core, a fuel assembly is assigned, whose kind is indicated by the identifier that appears in the first row inside each case in the grid. Just a slice, in one-eighth core symmetry, of the core schema is shown.

can be, in turn, automated. Hence the substantiality of the specific contribution of the neural component, as outlined in this paper.

4. The Neural Component

The development of the neural network component, called NEURALIZER, is meant to achieve complete automation of the operation loop of our expert system. For this purpose, we translate the ruleset into a neural network, and cause it to change in such a way that would yield better configurations.

In the following, we are going to concentrate on two issues:

- the learning algorithm selected for carrying out the adaptation phase, and
- the translation scheme.

The network is constituted of a representation of the rules. The input of our network is the current state of the core of the nuclear reactor. As to the output of the network, it is the advice, which relies upon the rules, on how to build a new fuel configuration. The advice is forwarded into the environment that uses it to construct a configuration of the fuel assemblies in the core. After a few steps of consulting the network, the configuration is ready. This is the output of our composite system, within whose architecture the network is just one component. The configuration is evaluated by NOXER, and the evaluation is fed back to the network in order to have the rules tuned in such a way that would yield a better fuel configuration. Notice that we do not know the (*input*, *desired-output*) pairs of the network as in the supervised learning approach, but, rather, just the (*input*, *evaluation of the output of the environment*) pairs of the composition. The particular learning approach that complies with such knowledge is *reinforcement learning* [26].

It is important to note that the network includes two types of rules. The mandatory (also called elimination) rules, are mainly those intended to ensure safety, and must not be changed, while the optional (i.e. preference) rules may be reconsidered. We leave the mandatory part of the network unchanged, similarly to the way described by Jordan [27] for the approach of supervised learning with a distal teacher.

Our translation technique is general, and allows rules of not only propositional calculus, but of first-order logic as well. This is due to the recursion of the resulting network. We are going to use the *neural information processing programming language* (*NIPPL*) and the corresponding translation scheme, introduced by Siegelmann [6, 28]. NIPPL, also called NEL, is defined as a high level language which is rich enough to express any computer algorithm or rule-based system. The language combines features of both PASCAL and LISP in terms of the data structures and the flow control (including loops). Previously, NIPPL was a theoretical language only. Here we suggest bringing it into use. Let us briefly overview the language and its compiler.

NIPPL is a procedural, parallel language. It allows for the subprograms procedure and function. A sequence of commands may either be executed sequentially (*Begin*, *End*) or in parallel (*Parbegin*, *Parend*). There is a wide range of possible **data types** for constants and variables in NIPPL, including the simple types: Boolean, character, scalar type, integer, real, and counter (i.e. an unbounded natural number or 0); and the compound types: list (with the operations defined in LISP), stacks, sets, records and arrays. For each data type, there are a few associated predefined functions, e.g. *Isempty*(stack), *In*(element, set) and *Iszero*(counter).

Expressions will be defined on the different data types. Examples of expressions are:

1. $\sum_{i=1}^{7} c_i x_i$ for constants c and either real or integer values of the variables x_i.
2. (B_1 And B_2) **Or** ($x > \frac{1}{2}$) for Boolean values B_1, B_2 and an integer value x.
3. **Pred** and **Succ** of an element e of a finite ordered type T returns another element of the same type.
4. **Chr** operates on an integer argument and returns a character.

Statements of NIPPL must include atomic statements (e.g. assignments, procedure calls, I/O statements), sequential compound statements (*Begin*, *End*), parallel compound statements (*Parbegin*, *Parend*), flow control statements which include both conditional (e.g. *If-then*, *If-then-else*, *case*, and *cond*) and repetition statements (such as *while* and *repeat*).

Instead of describing the compiler, we show one example of translation [6]. Let M and N be values in $[0, 1]$ and let B be a Boolean expression. The conditional statement

If (B) **then** $x = M$
else $x = N$

can be executed by the network shown in Fig. 14, as follows:

$$x_1(t) = \sigma(M + B - 1)$$

$$x_2(t) = \sigma(N - B)$$

$$x_3(t + 1) = \sigma(x_1(t) + x_2(t))$$

The neuron x_1 attains the value $\sigma(M)$ when $B = 1$. As σ is the linear-saturated function of equation (3), and M is assumed to lie in the range $[0, 1]$,

$$x_1(t) = \sigma(M) = M$$

When $B = 0$, $x_1(t) = \sigma(M - 1) = 0$. The neuron x_2 computes $\sigma(N - 1) = 0$ for $B = 1$, and $\sigma(N) = N$ for $B = 0$. Summing the above two values into x_3 results in

$$\sigma(M + 0) = M \quad \text{for } B = 1,$$

$$\sigma(0 + N) = N \quad \text{for } B = 0$$

as desired.

To synchronize the update, an *If* statement requires two sub-statement counters: one for the first update level, c_1, and one for the second update, c_2. The full update for the *If* statement is thus:

$$x_1^+ = \sigma(M + B + c_1 - 2)$$

$$x_2^+ = \sigma(N - B + c_1 - 1)$$

$$x_3^+ = \sigma(x_1 + x_2 + c_2 - 1)$$

The update equations of the counters are omitted.

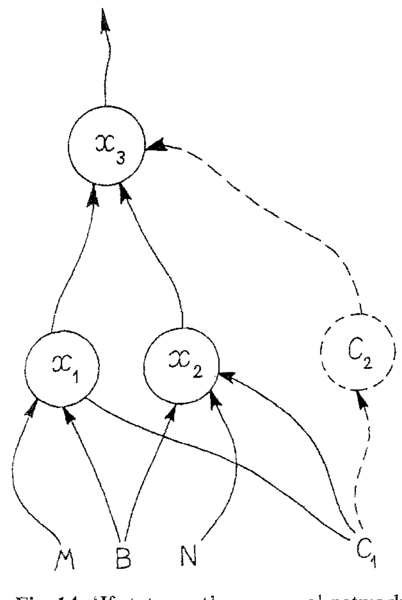


Fig. 14. 'If statement' as a neural network.

5. Rule Translation: An Example

We next illustrate how to construct a network out of the rules of FUELCON. Consider the second rule of Section 3.1:

Don't load a fresh assembly in such a position that is adjacent to another position where there is another assembly of the same kind, except when one of those two positions is in a corner position.

The input to the network includes the new assembly, A, which is represented as a record.

```
A = record of [  burnt,
                 kind,
                 position,
                 ...       ]
```

The rule can be written as a NIPPL function that receives as input the record A and a position s, and decides whether the position contradicts Rule 2. In the following function, we write the reserved words of NIPPL in boldface and the predicates in italics. Lines are numbered successively.

```
 2. Function rule-2 ($A$, $s$): Boolean;
 2. var $p$: Integer, flag: Boolean
 3. Begin
 4.     $p = 0$
 5.     flag = Good-position;
 6.     If
 7.         (($A$.burnt = fresh) $\wedge$ ($\neg$corner($s$))
 8.     then
 9.         repeat
10.             $p = p + 1$;
11.             If
12.             (neighbor($s$, $p$) $\wedge$ $\neg$corner($p$) $\wedge$
                kind ($A$) = kind(assembly ($p$)))
13.             then
14.             flag = Bad-position;
15.         Until
16.             (flag = Bad-position) $\vee$ ($p = 20$);
17.     rule-2 = flag
18. End;
```

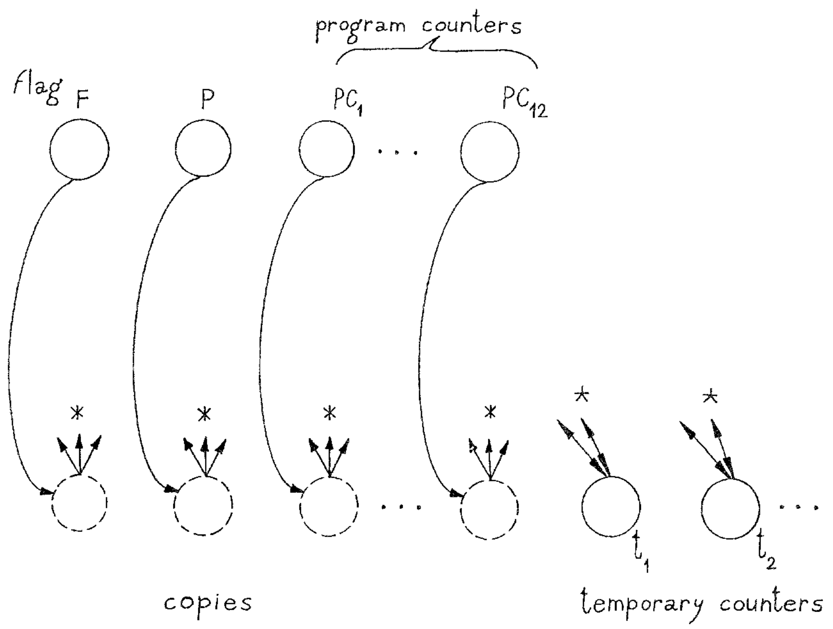


Fig. 15. The neural network corresponding to the sample rule whose translation and processing are discussed in the text. Circles in the upper row are variables. Cirlces on the left in the lower row are copies. Circles on the right in the lower row are temporary variables. The six-pointed star upon each copy indicates there is full connection to the upper row (i.e. the circle has arrows towards all of the circles in the upper row). The five-pointed star upon each temporary variable indicates there is full connection both to and from the upper row (i.e. the circle has bidirectional arrows connecting it to all of the circles in the upper row).

This program carries out its task by scanning all of the positions in the one-eight slice of the reactor core. This is redundant. An alternative version just checks adjacent positions.

We next demonstrate how to translate this little program into a network. This rule can be translated into either a simple feedforward network that tests the 20 positions simultaneously or into a recurrent network that tests them serially.

The first implementation requires more hardware but is fast and straightforward. The second one is cheaper in terms of hardware, and it scales to any number of positions. This tradeoff of hardware and time will be decided upon in the exact application. We choose to construct a recurrent network in this case, as to demonstrate the compilation of loops.

There will be a neuron for each variable and a temporary variable, as well as for the distributed representation of the program counter. The function includes the variables p and flag, as well as Rule 2 itself. In addition, each expression implies an expression variable (and possibly some temporary variables as well). The program counters are $pc_1, \ldots, pc_{18}$.

- **The variables:**
  1. The variable p is changed in lines 4 and 10. We can write its substitutions in the general formula of

$$p = 0 \cdot pc_4 + (p + 1) \cdot pc_{10} + p(1 - pc_4 - pc_{10})$$

  2. The function variable rule-2 = flag$\cdot pc_{16}$.
- **The expression variables:**
  1. The expression of line 7 required three temporary variables:

$$v_7 = \sigma(v_{7,1} - v_{7,2})$$

     where $v_{7,1}$ is a Boolean neuron for fresh assembly and $v_{7,2}$ tests whether the position is in the corner. These neurons will be set with program counter 6, and the neuron for v_7 sets with pc_7.
  2. Similar is the Boolean expression for line 12. Here $v_{12,1}$ to $v_{12,3}$ are set during the set of program counter 11, and the update $v_{12} = \sigma(v_{12,1} - v_{12,2} + v_{12,3} - 1)$ with 12.
  3. The Boolean expression of line 16 is translated similarly.

- **The program counters:** Each program counter (between 1 and 18) is associated with a neuron. The update of the counters decides the flow control. As the program is parallel, a few counters may take true values simultaneously. The update equations of the program counter neurons are given by:

$$pc_i = pc_{i-1} \quad \text{for } i = \{2, 3, 4, 5, 6, 7, 9, 11, 12, 14, 15, 16, 18\},$$

$$pc_8 = \sigma(pc_7 + v_7 - 1)$$

$$pc_{10} = \sigma(pc_9 + v_{\text{loop}} - 1), \quad v_{\text{loop}} = \sigma(pc_{16} - v_{16})$$

$$pc_{13} = \sigma(pc_{12} + v_{12} - 1)$$

$$pc_{17} = \sigma(pc_{16} + v_{16} - 1)$$

The resulting network is cyclic; see Fig. 15.

6. Symbolic and Neural Integration: A Comparison

The integration of expert systems and neural networks is an emerging area [29, 30]. The two areas are complementary. Sometimes they are overlapping alternatives. Hybrid systems with expert system and neural network components are just one option for integration; another active subsector is neural re-implementations, i.e. redoing extant symbolic reasoning systems as neural, through some interface or just by recoding.

A recently announced volume, Medsker [31], is devoted to hybrid neural and symbolic expert systems; the present authors have not had the opportunity to see it. The table of contents, as announced, features in Ch. 5 an application in nuclear engineering, to a different task: nuclear plant monitoring by means of a hybrid systems approach.

The strength of the approach embodied in the FUELCON/NIPPL integration is apparent, if we consider the more limited ambitions and capabilities of *CAPS*, a connectionist architecture for implementing extant rulesets coded in OPS5: 'CAPS supports a subset of the OPS5 language which includes variables, negation, conjunction, and disjunction of conditions. It uses a translation program to transform an OPS5 program into a limited interconnected, fully trained neural network' [32].

The advantage of NIPPL is that not only is it more expressive, but it also enhances clarity, by allowing one to code in a high-level language that is similar to conventional, parallel, or symbolic programming. Instead, CAPS is considered by its proponents just an interim solution, given the fact there are extant symbolic rulesets around: they do 'not advocate that future expert systems should be implemented in this manner. It is better to build future connectionist expert systems by using the adaptive nature of neural networks to bypass rules (as formed by a human expert) and let the network form its own internal rules (by learning from examples). The architecture is an interim measure for enhancing the performance of existing rule-based OPS5 programs by implementing them on connectionist networks' (*ibid.*).

For certain tasks, it would be desirable that once a symbolic ruleset is enhanced through a neural implementation, it could reformulate symbolic rules for the human expert to see. Neither NIPPL nor CAPS do that. The generality of NIPPL makes it suitable for having further components in a project such as FUELCON go neural; we are envisaging such a new project for the simulator: the extant simulator is a conventional code, not a ruleset.

CAPS supports dynamic variable bindings, and some of the OPS5 constructs. NIPPL is clearly superior, in respect of syntactic richness; actually, the application to FUELCON exploits just a small subset of NIPPL. However, because of the maximal generality goal of NIPPL as a high-level language, an interim tradeoff was necessary, and interconnection minimization was not pursued, at least not as a specific goal. CAPS, instead, 'addresses the hardware implementation issues by utilizing limitedly interconnected networks of a few basic unit types' (*ibid.*).

There is one more remark to make, about the appropriateness of the neural learning adopted in the FUELCON/NIPPL project. The rulesets that the human experts developed for FUELCON, are not large. Of about a dozen rules, one half or more are mandatory rules, that NIPPL should not modify. Just half a dozen or less rules are those concerned by the optimization effort. This makes the situation very satisfactory, for learning. Indeed, had we large rulesets to optimize, then learning could be expected to be very slow: the larger the ruleset, the slower the learning. Instead, ruleset size is no problem, for the application to FUELCON.

7. Conclusions

7.1. The novelty of the results reported in this paper shows up under different respects. FUELCON represents a definite improvement in the domain area: nuclear engineering. Indeed, both the practitioner and the researcher in the domain of in-core fuel management (as distinct from fuel storage management), are offered a tool that greatly increases their options at allocating fuel assemblies in the positions of the core of a nuclear reactor.

Visibility at handling the allocation problem, and, thus, the possibility to manoeuvre, are enhanced. It was usual, in the domain, to look for just one configuration per plant/cycle situation, as a solution of the fuel management problem. To put it bluntly, the relative lack of ambition featured by this goal reflected the opacity of the process of problem-solving, let alone optimization: looking for one solution was like the game by which a child is blindfolded, and has to attach the tail on the image of a donkey. Solutions used to be found, but without a clear view of alternatives. Several techniques are known that generate a single solution, i.e. by modifying a fuel-allocation devised for a different situation, and found in the literature in the framework of a real-case study. Operations research methods were applied, by some.

With FUELCON, not only the expert system generates families of alternative solutions *ex nihilo* (rather than a single one), based on a transparent formulation of rules of thumb by an expert, but we can actually see, on the graphic display, a simulation of the entire family: the single configurations are dots in a plane, so we can compare the merits and defects of the alternative allocations, and the domain expert is enabled to evaluate the heuristics, and revise them iteratively.

7.2. Furthermore, our project is also novel in the following respects, that are specific to the new stage we have been discussing in this paper. The cognitive process of revising the heuristics is, *prima facie*, a 'noble' task: it is, in the sense of its belonging to the competence of the expert, as compared to the novice, and of the researcher, as compared to the practitioner.

Yet, we can be ambitious in this regard, too. We can, because of our adoption of the neural-network paradigm, on top of symbolic reasoning and algorithmic simulation methods, as typifying the previous stages of the FUELCON project. We set to automate even the process of revising the heuristic rules. In FUELCON, the operation loop has admissible, good options for fuel allocation generated at single iterations, whereas the domain heuristics themselves improve from iteration to iteration. Because of this loop, the automation of revision is tantamount to automating the global discovery process.

An earlier stage of FUELCON, as in an AI perspective, is described in Galperin *et al.* [1]. The approach, in that paper, was to stress the integrated human/machine operation cycle, and the ergonomics of the manual phases in the discovery process. This includes considerations in the perspective of the expert critiquing systems paradigm [33]. It also includes an appreciation of the new body of knowledge acquired specifically about how to use FUELCON through manual interaction. However, those ergonomic considerations are no longer relevant, once we take the bold step of setting to completely automate the discovery process, by delegating ruleset-revision to a neural component, instead of to the human expert.

7.3. The state of research in neural computing features only a few papers about ruleset revision, that is, about converting rule-based systems into neural networks with the intention of correcting them or improving them. Towell *et al.* [34] had to deal with an expert system based on propositional calculus, and suggested transforming the original propositional domain theory into a neural network. The connection weights were elegantly adjusted in accordance with the observed examples using standard backpropagation techniques. Maclin and Shavlik [35] suggests the use of reinforcement learning for adaptation.

We list below three innovations of our work in terms of neural revision. First, our application is the first effort to affect real application by means of neural revision. It addresses a technically and economically challenging task in engineering, and its success can be expected to allow savings in the order of even millions of dollars, per nuclear reactor, per operation cycle. Secondly, although, in practice, most of the rules are propositional, rather than recurrent, our novel methodology of translating rules in both propositional and first order logic encompasses both the immediate practical translation as well as future applications to any complicated type of rule-based system.

The third, subtle, but very important point is the type of neural learning algorithm applied. Many learning algorithms exist and they are to be used in different settings. The classical neural network learning paradigms are the *learning with teacher* approaches. The assumption is that a set of pairs of input/output of the neural network are provided and the network adapts itself to comply with them. Our setting is more complicated. We do not know the input/output pairs of the network, but rather only the input and the evaluation of the output of the system which is composed of the neural network (i.e., the rules) and the environment that creates the configuration based on the output of the network. Two learning approaches have been developed to subserve such cases: the *reinforcement learning* approach [26] and the *learning-with-distal-teacher* approach [27]. These two methods differ by some subtle characteristics of the evaluation available: reinforcement learning is more general. It is the approach necessary for our own purposes in the project we have been discussing.

Previous phases of our project (the building blocks available at its inception, and a sample translation of one rule) were presented in two shorter papers [36, 37].

Acknowledgement

Hava Siegelmann wishes to thank Professor Eduardo Sontag, her advisor at Rutgers University: the original ideal of defining the NIPPL language is his. Ephraim Nissan is grateful to Professor Brian Knight, who has provided FUELCON with a haven for further development at the University of Greenwich. Further subprojects are ongoing [38]: with genetic algorithms (the doctoral

project of Mr Jun Zhao); and neural prediction equivalent to NOXER. The very first idea of FUELCON was Galperin's. Kimhi was his doctoral student. Galperin and Nissan developed the very first prototype. Nissan and Siegelmann devised the hybrid architecture.

References

1. Galperin, A.; Kimhi, Y.; Nissan, E. (1993). FUELCON: an expert system for assisting the practice and research of in-core fuel management and optimal design in nuclear engineering, Computers and Artificial Intelligence, 12, 369–415
2. Kimhi, Y. (1992) A non-algorithmic approach to the in-core fuel management problem of a PWR core, PhD dissertation, Department of Nuclear Engineering, Ben-Gurion University of the Negev (Beer-Sheva, Israel) (in Hebrew; English abstract)
3. Galperin, A.; Kimhi, Y. (1991) Application of knowledge-based methods to in-core fuel management, Nuclear Science and Engineering, 109, 103–110
4. Galperin, A.; Yimhi, Y.; Segev, M. (1989) A knowledge-based system for optimization of fuel reload configurations, Nuclear Science and Engineering, 102, 43
5. Galperin, A.; Nissan, E. (1988) Application of a heuristic search method for generation of fuel reload configurations, Nuclear Science and Engineering, 99 (4), 343–352
6. Siegelmann, H.T. (1994) Neural programming language, Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, Washington.
7. Cochran, R.G.; Tsoulfanidis, N. (1990) The Nuclear Reactor Cycle: Analysis and Management, American Nuclear Society, La Grange Park, IL
8. Parks, G.T.; Lewins, J.D. (1992) In-core fuel management and optimization: the state of the art, Nuclear Europe Worldscan, 12 (3/4), 41
9. Petschhat, G.R.; Rothleder, B.M.; Faught, W.S.; Eich, W.J. (1986) Interactive fuel shuffle assistant graphics interface and automation for nuclear fuel shuffle with PDQ7, Proceedings of the American Nuclear Society Topical Meeting on Advances in Fuel Management, Pinehurst, NC
10. Faught, W.S. (1987) Prototype fuel shuffling system using a knowledge-based toolkit, Technical Report, IntelliCorp, Mountain View, CA
11. Rothleder, B.M.; Petschhat, G.R.; Faught, W.S.; Eich, W.J. (1988) The potential for expert system support in solving the Pressurized Water Reactor fuel shuffling problem, Nuclear Science and Engineering, 100, 400
12. Kropaczek, D.J.; Turinski, P.J. (1991) In-core nuclear fuel optimization for Pressurized Water Reactors using simulated annealing, Nuclear Technology, 95, 9
13. Sontag, E.D. (1992) Neural nets as systems models and controllers, Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems, Yale University, New Haven, CN, pp. 73–79
14. Matthews, M. (1992) On the uniform approximation of nonlinear discrete-time fading-memory systems using neural network models, PhD dissertation, ETH No. 9635, E.T.H., Zurich
15. Polycarpou, M.M.; Ioannou, P.A. (1991) Identification and control of nonlinear systems using neural network models: design and stability analysis, Report 91-09-01, Department of EE/Systems, University of South California, Los Angeles
16. Cleeremans, A.; Servan-Schreiber, D.; McClelland, J. (1989) Finite state automata and simple recurrent networks, Neural Computation, 1, 372
17. Elman, J. L. (1990) Finding structure in time, Cognitive Science, 14, 179–211
18. Giles, C.L.; Miller, C.B.; Chen, D.; Chen, H.H.; Sun, G.Z.; Lee, Y.C. (1992) Learning and extracting finite state automata with second-order recurrent neural networks, *Neural Computation*, 4 (3), 393–405
19. Pollack, J.B. (1990) The induction of dynamical recognizers, Report 90-JP-Automata, Department of Computer and Information Science, Ohio State University
20. Williams, R.J.; Zipser, D. (1989) A learning algorithm for continually running fully recurrent neural networks, Neural Computation, 1, 270–280
21. Barto, A. G.; Singh, S. P. (1990) On the computational economics of reinforcement learning, Proceedings of the 1990 Connectionist Models Summer School, D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E Hinton (Editors), Morgan Kaufmann, San Mateo, CA
22. Narendra, K.; Thathacar, M. A. L. (1989) *Learning Automata: An Introduction*, Prentice-Hallm Engelwood Cliffs, NJ
23. Barto, A. G.; Anandan, P. (1985) Pattern recognizing stochastic learning Aatomata, *IEEE Transactions on System, Man, and Cybernetics*, **15**, 360–375
24. Barto, A.G.; Jordan, M.I. (1987) Gradient following without back-propagation in layered networks, Proceedings of the First IEEE International Conference on Neural Networks, San Diego, Vol. 2, pp. 629–636
25. Hertz, J.; Krogh, A.; Palmer, R.G. (1991) Introduction to the Theory of Neural Computation, Addison-Wesley, Reading, MA
26. Barto, A.G.; Sutton, R.S.; Watkins, C.J.C.H. (1991) Learning and sequential decision making, In Learning and Computational Neuroscience, M. Gabriel and J.W. Moore (Editors), MIT Press, Cambridge, MA
27. Jordan, M.I. (1992) Forward models: supervised learning with a distal teacher, Cognitive Science, 16, 307–354
28. Siegelmann, H.T. (1993) Foundations of recurrent neural networks, PhD Dissertation, Rutgers University, New Brunswick, New Jersey
29. Gallant, S.I. (1988) Connectionist expert systems, Communications of the ACM, 31 (2), 152–169
30. Medsker, L.R. Editor (1991) The Synergism of Expert System and Neural Network Technologies, special issue of Expert Systems with Applications, 2 (1)
31. Medsker, L.R. (1994) Hybrid Neural Network and Expert Systems, Kluwer, Dordrecht, The Netherlands
32. Bhogal, A.S.; Seviora, R.E.; Elmasry, M.I. (1991) Towards connectionist expert systems, Expert Systems with Applications, 2 (1), 3–14
33. Silverman, B.G. (1992) Survey of expert critiquing systems: practical and theoretical frontiers, Communications of the ACM, 35 (4), 106–127
34. Towell, G.G.; Shavlik, J.; Noordewier, M.O. (1990) Refinement of approximately correct domain theories by knowledge-based neural networks, Proceedings of the Eighth National Conference on Artificial Intelligence, p. 861
35. Maclin, R.; Shavlik, J.W. Incorporating advice into agents that learn from reinforcement (journal submission)
36. Nissan, E.; Siegelmann, H.; Galperin, A.; Kimhi, S. (1994) Towards full automation of the discovery of heuristics in a nuclear engineering project, by combining symbolic and subsymbolic Computation, Proceedings of the Eighth International Symposium on Methodologies for Intelligent Systems (ISMIS'94), Charlotte, N.C., Springer-Verlag, New York (Lecture Noter in Artificial Intelligence, Vol. 869), pp. 427–436

37. Nissan, E.; Siegelmann, H.; Galperin, A. (1994) An integrated symbolic and neural network architecture for machine learning in the domain of nuclear engineering, Proceedings of the Conference on Pattern Recognition and Neural Networks, within the 12th ICPR: International Conferences on Pattern Recognition, Jerusalem, IEEE, Computer Society Press, New York

38. Galperin, A.; Kimhi, S.; Nissan, E.; Siegelmann, H.; Zhao, J. (1995) *Symbolic and subsymbolic integration in prediction and rule-revision tasks for fuel allocation in nuclear reactors.* Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing (EUFIT'95), Aachen, Germany, (in press)