Advances in Engineering Software 28 (1997) 581–592
© 1997 Elsevier Science Limited. All rights reserved
Printed in Great Britain
0965-9978/97 $17.00+0.00

PII: S0965-9978(97)00040-9

A novel neural/symbolic hybrid approach to heuristically optimized fuel allocation and automated revision of heuristics in nuclear engineering

Hava T. Siegelmann[a], **Ephraim Nissan**[b,*] **& Alex Galperin**[c]

[a]*Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel*
[b]*School of Computing and Mathematical Sciences, The University of Greenwich, Wellington St, Woolwich SE18 6PF, UK*
[c]*Department of Nuclear Engineering, Ben-Gurion University, Beer-Sheva, Israel*

(Received 29 February 1996; accepted 27 May 1997)

FUELCON is an expert system for optimized refueling design in nuclear engineering. This task is crucial for keeping down operating costs at a plant without compromising safety. FUELCON proposes sets of alternative configurations of allocation of fuel assemblies that are each positioned in the planar grid of a horizontal section of a reactor core. Results are simulated, and an expert user can also use FUELCON to revise rulesets and improve on his or her heuristics. The successful completion of FUELCON led this research team into undertaking a panoply of sequel projects, of which we provide a meta-architectural comparative formal discussion.

In this paper, we demonstrate a novel adaptive technique that learns the optimal allocation heuristic for the various cores. The algorithm is a hybrid of a fine-grained neural network and symbolic computation components. This hybrid architecture is sensitive enough to learn the particular characteristics of the 'in-core fuel management problem' at hand, and is powerful enough to use this information fully to automatically revise heuristics, thus improving upon those provided by a human expert. © 1997 Elsevier Science Limited.

1 INTRODUCTION

Nuclear engineering is an industrially significant domain,[1] prominent in the broader area of power generation and supply. Our FUELCON project, started in the mid-1980s, has thus far yielded an expert system that has inaugurated a paradigm among computer tools for its specific task. This task is to provide a good fuel reload configuration for when a reactor is shut down periodically for refueling. Reload design is also called, more broadly, in-core fuel management. Because of the way it affects the effectiveness of fuel utilization—and how long the reactor could operate effectively before it has to be shut down again—reload design has a major incidence on costs at nuclear power plants. ''Individuals responsible for fuel management are among the most skilled in the nuclear industry and their time is always at a premium. It is therefore not surprising that AI techniques, both expert systems and pattern recognition, are being applied to assist with refuelings.''[2] (p. 40).

FUELCON is already a working system that can be applied industrially.[3–13] During the last two or three years, we have been trying to upgrade the reasoning capabilities of the tool. This has led to an array of sequel projects; we report on one of these in this paper. We are going to focus, here, on the contribution of neural computation, on top of symbolic manipulation as carried out by FUELCON, for the purposes of not only producing good reload designs but also improving upon the human expert's heuristics that guide automated design at the symbolic computation level.

Indeed, in FUELCON, heuristic domain knowledge is applied to the generation of hundreds of alternative fuel allocation patterns per session, this plurality being itself a peculiar asset. From simulation results downstream (which

*Author to whom all correspondence should be addressed.

are done by using a simulator called NOXER), the very few best solutions stand out, thus creating a shift from just local optima as achieved by other methods of computer-assisted reload design to what FUELCON enables us to achieve: namely, global optimality in the solution space as per a given policy of search, the latter being embodied in the ruleset loaded during the given session.

Note that heuristics are used. Therefore, when looking for optimality, it is understood that the optimum is with respect to the space of solutions as delimited by the heuristics, which embody a strategy. How well it performs is relative to the best result published in the domain literature, or from the record of previous sessions. This way, the tool is also very important for evaluating strategies, and for detecting and exploring the potential of new or underexplored strategies.

Each one of the allocation patterns (ie fuel configurations) thus generated prescribes how to allocate units of fuel, of different kinds, inside the square cases of the (geometrically and symmetrically schematized) core of a particular nuclear reactor. Which relevant criteria of fuel allocation are selected when solving the problem at hand depends on the type of reactor, the features of the individual plant, its current state and the solver's expertise. In turn, the cumulated knowledge body of the speciality, as in its operational form of prescribing problem-solving steps, reflects—at a deeper level—models of reactor physics, as well as of the economics of nuclear plant operation.

One crucial aspect of in-core fuel management is that fuel allocation, and the circumstances motivating the details of selection, cannot be fully predefined before the moment comes for shutting down a plant and refueling it. Even though rough forecasts are possible, they are not robust enough in terms of post-optimality: unforeseen discrepancies with respect to expectations are all too likely to undermine the convenience and even the very safety of any configuration resulting from pre-shutdown guesses. It is only at shutdown that the reliable design of the new allocation is possible.

We have seen that, in FUELCON, a ruleset is applied to generate families of good fuel configurations in the reactor core. A practitioner may just rely on FUELCON to propose solutions from which s/he can pick the best according to the visualized results of the simulation. The domain expert, however, can afford to be more ambitious: FUELCON is a testbench for the human expert heuristics. The expert is challenged to significantly improve not just the configurations but his or her own heuristic rules as well. The results of one given iteration of the expert system are simulated by a separate component, the examination of whose results in turn prompts the human expert to manually improve upon the ruleset that was previously formulated, also manually, by the same or another expert.

The main contribution of neural revision of the ruleset—the main sequel project we describe in this paper—consists

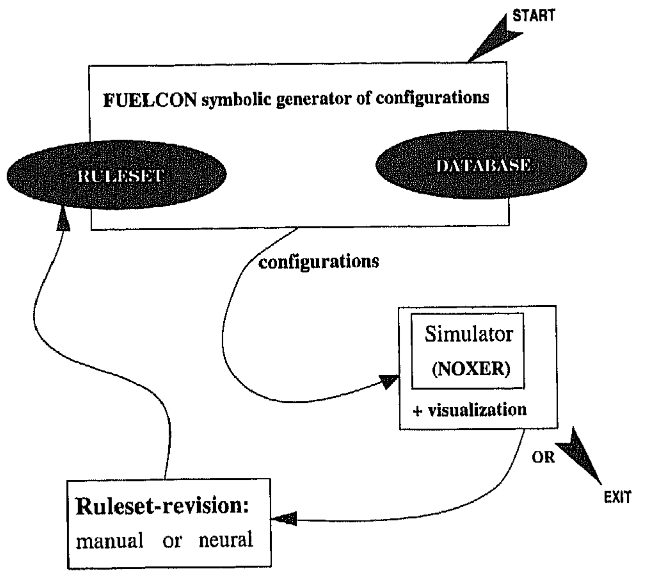

Fig. 1. Phases in the integrated operation loop of using the tool.

of enabling this 'nobler' cognitive task, of improving upon the previous ruleset, to be automated. The initial formulation of the ruleset is still to be provided by the expert; what follows, in both the traditional FUELCON and this sequel project, is an operation loop throughout various steps (and the respective software components): see Fig. 1. We shall show the outline of how to automate ruleset revision: in a sense, this success is tantamount to virtually achieving a full automation of the discovery process by which heuristics are adapted and refined in the loop. We added the 'missing link' by replacing, in the architecture, a manual step with an automated one. Neural learning algorithms tune the rules to yield better configurations, based on performance in previous iterations as assessed in the simulation phase downstream of the generator of fuel configurations. For the purposes of the neural learning phase, we designed a component transforming the rules into a neural network using a particular technique embodied in the *NIL* language and translation schema[14] (also called *NEL*), defined by one of the present authors. The general use of *NIL* is for transforming symbolic algorithms and structures—and, for our present purposes, rulesets—into neural networks. We use this schema to obtain translation from symbolic rules to analogue network. Learning then takes place on the neural-network equivalent to the rules, and revision is performed on this representation.

Not only is the approach novel for the domain of application: it is not trivial from the viewpoint of artificial neural networks (ANN) and integrated symbolic/neural hybrid architectures. In terms of the classification of hybrids as proposed by Goonatilake and Khebbal,[15] it is a *function-replacing hybrid* that we have in the extended FUELCON architecture (the other two categories being *inter-communicating hybrids* and *polymorphic hybrids*). In this paper, we are also going to define a semi-formal framework for discussing pools of heterogeneous components as integrated into alternative, possibly hybrid architectures. This approach, we believe, may also prove useful for engineering computing and hybrid system design.

In this paper, we first discuss reactor and fuel management, then describe the version of FUELCON that only carries out symbolic manipulation (conventional AI, non-neural computation). The architectural options of the sequel projects are then discussed in a formal notation, and then we present the neural revision of the ruleset. Finally, we briefly outline sequel projects.

2 THE PROBLEM: REFUEL DESIGN

The domain of the project described, FUELCON, is *in-core fuel management*,[16] at nuclear power plants of the conventional kind, i.e. pressurized water reactors (PWR). Another paper[17] presents a review of the domain and of the problem. We recapitulate very briefly here. Nuclear fuel comes in assemblies of rods. Single assemblies are inserted in a grid, this being a planar horizontal section of the reactor core. In practice, it is usual to consider just a slice of the grid, in one-eighth symmetry. For example, let it be this array of positions:

*pos*11					
*pos*21	*pos*22				
*pos*31	*pos*32	*pos*33			
*pos*41	*pos*42	*pos*43	*pos*44		
*pos*51	*pos*52	*pos*53	*pos*54	*pos*55	
*pos*61	*pos*62	*pos*63	*pos*64	*pos*65	*pos*66
*pos*71	*pos*72	*pos*73	*pos*74	*pos*75	
*pos*81	*pos*82	*pos*83			

Position *pos*11 is the centre of the core. Positions *pos*22, *pos*33, *pos*44, *pos*55, *pos*66 are the core's diagonal. In this example, positions *pos*81, *pos*82, *pos*83, *pos*74, *pos*66 are the periphery of the core. The first column in the array is the core's main axis.

Reactors are operated in cycles of about one year at a time, and are then (at EOC, i.e., *end of cycle*) shut down for inspection and *refueling*. During operation, fuel becomes gradually depleted up to a certain degree reached at EOC, depending on the fuel assembly's position in the grid and on the assembly's record of previous use (*fresh fuel*, *once burnt* or *twice burnt*, according to the number of previous cycles of utilization being zero, one, or two). A fuel assembly is discarded after three cycles, or if a defect is found. Because of the cost of fuel and of transportation, it is important to make efficient use of the available pool of fuel assemblies.

The task of the human practitioner in the role of the in-core fuel manager, and of tools, such as FUELCON, that are intended to assist him or her, is to determine a safe, efficient arrangement of the available fuel assemblies in the grid. This arrangement is called a *fuel pattern*, a *fuel configuration* or a *reload design*. Owing to forecasts not being robust (i.e., exceeding sensitivity with respect to postoptimality), only at EOC is it possible to obtain the exact problem that has to be solved and then to design the new configuration in detail. Whereas some tools—either from operations research, or expert systems—try to improve on a given solution by switching places in the grid (i.e., by *shuffling*) and then propose one satisfactory solution (which is the case of an expert system prototype of IntelliCorp),[18–20] the FUELCON expert system generates a multitude of solutions from scratch, according to a given strategy that is embodied in heuristic rules, that can be tuned to improve performance.

3 FUELCON: THE EXPERT SYSTEM

3.1 The process of generating configurations

There is a standard, general solution method resorted to by human fuel managers to solve the in-core fuel management problem, which in turn can be conceived as per the following metacode:

```
                  sorted pool           configuration
       ''SORT''  ----------->  ''INSERT'' -------------> ''EVALUATE''
WHERE:
        ''SORT''     IS: sort(WHAT: ALL(fuel-assemblies); IN: fuel-pool);
        ''INSERT''   IS: insert(WHAT: fuel-assembly ; IN: ALL(core-positions));
        ''EVALUATE'' IS: evaluate(configuration).
```

Fuel is sorted for reload according to some physical feature. Then a candidate configuration is constructed by inserting (on paper) one fuel assembly per core position until the core is filled. (One practical way to do that may be to take a ready-made solution out of the plant's record, or of one's experience, or of some real-case study published in the specialized literature, even though such a solution is unlikely to be admissible for the problem at hand at this plant at this moment in time.) This yields a candidate configuration that has to undergo evaluation. This, in turn, is achieved by simulating it by means of simulation software; i.e., by simulating how the core reactor would behave during the next power production period were it to be reloaded as per the configuration at hand: would this involve an unacceptably high local power density? If not, then you have a satisfactory solution. Otherwise, try to modify the configuration at hand by *shuffling*; i.e., generate a new configuration by means of a binary (or more complicated) exchange of assemblies among adjacent (or non-adjacent) core positions.

Unlike all previous tools for reload design—and unlike most current tools even now that almost a decade has elapsed since the FUELCON project started—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). Among knowledge-based tools, that was the way that, e.g., the IntelliCorp tool was intended to work.

In contrast, FUELCON is not fed an input configuration but, instead, 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 by the given ruleset that generates them and which, in turn, embodies heuristics reflecting a given generic conception. Some heuristics are fairly general; other rules embody options. For example, to achieve uniform power distribution and thus maximal fuel depletion, the following general operational criterion is known (along with its explanation): "If every assembly has the same loading of uranium, then it can be shown theoretically that the resulting power profile will be cosine shaped. Therefore, achievement of a uniform power distribution requires that assemblies with less than the core-wide average fuel loading be placed at the core center and those with more at the core periphery."[2] (p. 40).

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.

This is how FUELCON generates families of configurations. The expert system has a database, subdivided in parts that, respectively, contain the structure of the core geometry of the particular reactor concerned, data on the pool of available fuel assemblies (which are subdivided by type), and the partial configurations in the process of being generated.

The set of configurations is generated concomitantly, *ex nihilo* (i.e, starting with an empty core), and then considering one fuel assembly at a time. Given that particular assembly, in each one of the partial configurations generated thus far there generally are various alternative empty positions where that assembly may be placed. In this way a tree of configurations is developed, level by level, through partial configurations as intermediate stages, up to the terminal level at which the core is completely filled. The search is forward oriented and breadth-first, and there is no backtracking. Each leaf in the tree corresponds to one full configuration, i.e., to a fully loaded core that fully exploits the available fuel.

The search can be conceptualized as a hierarchy—a search-tree—and each level in the tree is associated with a particular fuel assembly (out of the available pool kept in store); but in particular, for efficiency reasons, this is done according to a predefined order as per a loading sequence. The latter is given just as the rules in the ruleset are given: both the ruleset and the loading sequence are provided by the domain expert. Yet, 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.

On the other hand, this paper discusses, in Section 6, the basics of automating ruleset revision in a neural sequel project of FUELCON. The present section provides a necessary recapitulation of the way FUELCON is configured without the addition of the neural component, the latter being the topic of the rest of the paper. For the purposes of the discussion developed there, let us consider in particular the way the ruleset is structured in FUELCON.

3.2 The ruleset of FUELCON and NOXER

The ruleset in FUELCON consists of two different kinds of rules: elimination rules, which are mandatory and must apply in all sessions mainly to enforce safety; and optional rules, which reflect a policy and allow one to concentrate search in promising regions of the solution space.[3] From a different perspective, we can also say that rules in FUELCON represent two distinct types of knowledge: generic principles from reactor physics, and local, specific knowledge suited to accommodate given situations.

As already mentioned in the Introduction, FUELCON can assist the practitioner, but is even more useful as a testbench for the domain expert. A middle ground is that when an experienced user uses FUELCON, he or she is likely to be ambitious beyond what is yielded by just running the initial ruleset. If such a user is experienced enough (in the task, and in using FUELCON in particular), adjustments to the ruleset are called for, to try to get a new family of configurations that includes such a solution that is better that the best one yielded thus far. Moreover, the expert may wish to test new heuristics that would, e.g., zoom in on certain promising leads detected in the outcome of the previous run, or get denser families, or move the family as a whole into a safer or more efficient area in the cartesian plane of two parameters in which the results of simulation are visualized. In this way, manual revisions of the ruleset fit in an operation loop in using FUELCON when trying to solve the same given input problem. Each single iteration includes a generation phase, an evaluation phase, and a ruleset revision phase: see Fig. 1.

Downstream of the rule-based generator of configurations

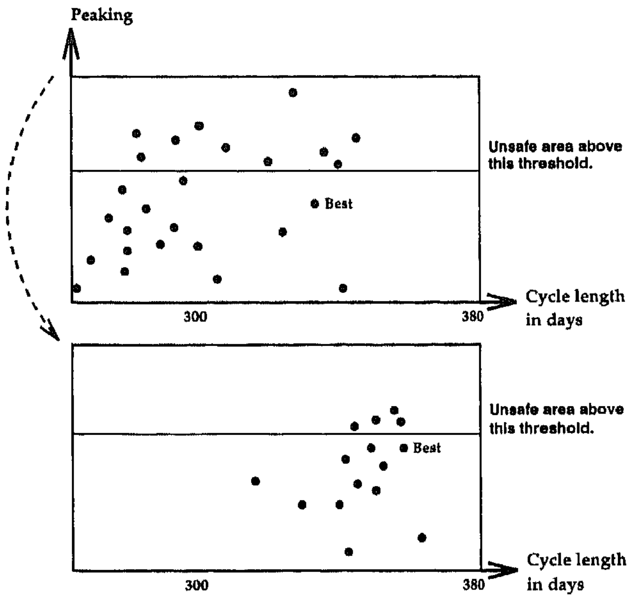

Fig. 2. The 'cloud' of solutions moves into a better region after the ruleset has been revised.

(which is coded in Lisp), the output is fed to NOXER, a locally developed simulator. The results of NOXER are both visual and numeric. At each iteration in the loop, the family of configurations generated is processed by NOXER and displayed as a 'cloud' of solutions (one dot per configuration) in a cartesian plane as mentioned. In this plane, there is a 'window' of admissibility: it is a region lying 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 'northeast' direction (provided it is under the safety threshold) in order to get several configurations that are both safe and very efficient: see Fig. 2. The means to achieve this are to revise the ruleset appropriately. This is the step of revision that we have set to automate by means of a neural component and symbolic-to-neural conversion schema.

4 SEQUEL PROJECTS: A META-ARCHITECTURAL DISCUSSION

In Fig. 1, we outlined a loop of how to operate our tool. The flow is through an architecture of components: the rule-based generator of fuel configurations, the simulator (either NOXER, or an alternative) and ruleset revision, which in turn is to be performed either manually or by means of a neural component.

The sequel projects we have been developing since the completion of FUELCON explore alternative paradigms in order to provide a rich pool of improved components. These components are either complementary (as shown in Fig. 1) or alternative. An AND/OR tree is an adequate formalism for discussing the various options: see Fig. 3.

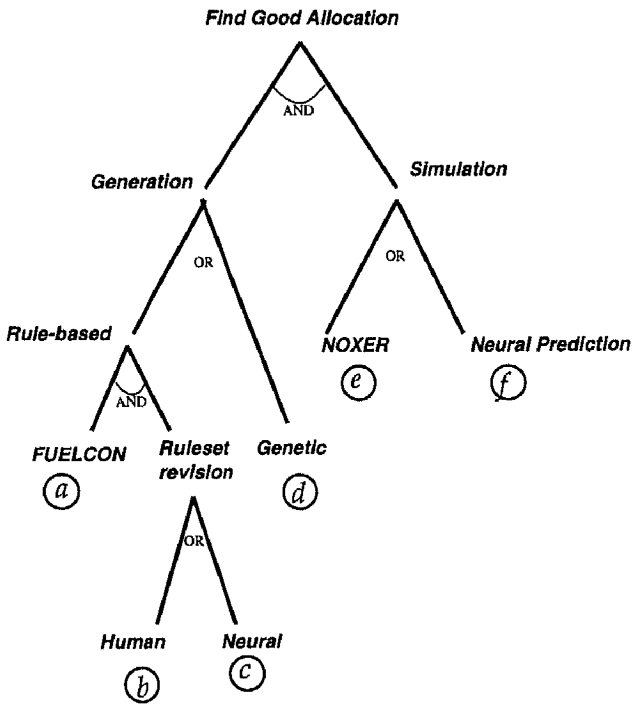

Fig. 3. Meta-architectural AND/OR tree.

The root of the tree is the global task of solving the in-core fuel management problem. To carry it out, we need both the generation of reload configurations and their simulation. In turn, for the latter we already have the NOXER program; however, one of our sequel projects (still in its early stages) aims at providing an alternative, based on neural networks, that needs to be reliable but far more time effective.

On the other hand, for the subtree rooted in 'Generation' we have results to show for all options. An alternative to rule-based generation is being explored in a sequel project based on genetic algorithms: see Section 7. In contrast, the generation of reload configurations by means of a ruleset requires the extant FUELCON to be combined with ruleset revision that in turn is performed either manually or as neurally automated. It is on the latter option that we are going to focus in the rest of this paper. Manual revision also has interesting features: a body of ergonomic knowledge about how to use FUELCON in that way has been developed.[3]

Now, let us develop a convenient notation to be able to refer to the various options concisely and precisely. To define such a notation, we need both the AND/OR tree of Fig. 3 and the following abstract conceptualization. Consider the six dashed boxes in Fig. 4. Let these be components that together constitute a pool from which to select those components that are to be included in an actual architecture. Such an architecture is shown in Fig. 5, where control flow is through components α, δ and ζ, in that order. Let us express that sequence by means of the

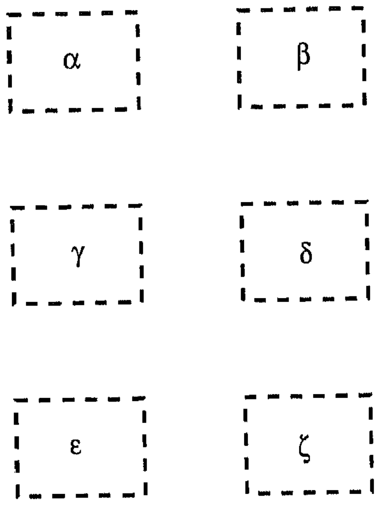

Fig. 4. A general schema of a pool of components for potential architectures.

noncommutative operator 'concatenation', symbolized by O. This way, the actual architecture selected in Fig. 5 (as opposed to potential ones) is $\alpha \bigcirc \delta \bigcirc \zeta$.

We apply this to the AND/OR tree of Fig. 3. The only candidates for concatenation are terminal nodes. Moreover, not all potential strings in any order are eligible. Historically, throughout our project, we have been developing architectures in the following order:

1. Architecture $a \bigcirc e \bigcirc b$, that is to say, the output of rule-based FUELCON, is simulated by NOXER, with manual ruleset revision downstream. If we assume that separately, somewhere, a constraint is stated on the actual order in which components are invoked in the control flow, then it is legitimate to decouple this constraint from the way we select strings of terminals from the AND/OR tree. Thus, let us reformulate the string in the order the terminals appear in Fig. 3. This yields the concatenated string $a \bigcirc b \bigcirc e$.

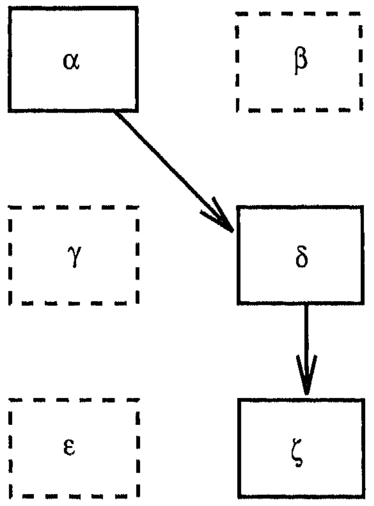

Fig. 5. An actual architecture, $a \bigcirc \delta \bigcirc \zeta$, selected from the pool of components as in Fig. 4.

2. $a \bigcirc c \bigcirc e$. This is one of the architectures we are investigating by integrating symbolic and neural computation: actually, it is the architecture, based on reinforcement learning, to which we are going to devote the bulk of our remaining discussion in this paper.
3. $d \bigcirc e$, this being the genetic system that Jun Zhao has developed in London as a doctoral project under the FUELCON 'umbrella': see Section 7.
4. $a \bigcirc c \bigcirc f$. This is a hybrid symbolic/neural architecture that we will not consider yet. It is based on the *distal teacher* paradigm of neural computation.

5 THE WAY NEURAL NETWORKS ARE APPLIED

Artificial neural networks are a computational tool with the ability to adapt to environments and learn from experience. The networks consist of interconnections of parallel agents; each agent computes a simple function, but the network as a whole constitutes a powerful model.[21–23] As opposed to classical AI techniques, the networks do not require lists of rules or entail reasoning. Networks are trainable dynamical systems which learn by observing a training set of input–output pairs, usually by gradient-descent optimization techniques (e.g., backpropagation). As for their general structure, networks are considered as nonparametric statistical models that are capable of approximating rather arbitrary functions.[24,25] Much effort in using them was demonstrated in the last decade, in particular in the areas of signal processing, adaptive control, speech and image recognition and time series prediction.[26,27]

Arguably, the most appealing characteristic of neural networks is that they can generalize—and thus respond well—not just on the training set, but rather on future behaviour that is not observable. Thus, it is important that networks are not being trained to give as perfect a match as possible to their training set, but only up to some level of precision so as to allow for good generalization. There are various techniques of how to do this well and not overfit the given data.[27] Failing to allow neural networks such an ability to generalize undermines the very reason and withholds the benefit of using them, as can be seen in some older, not so successful, applications reported in the literature. In the present project, we made special efforts to ensure that the generalization property is adequately possessed by the system.

5.1 Symbolic dynamics in neural processing

The incorporation of symbolic dynamics in neural networks may considerably enhance the network performance in practice. This idea seems to have been put forward by several researchers.

Psychologists have long been noting the fact that when a person learns a new task, that person does not start *tabula rasa* by learning to control his or her motor system, to talk, and so on, but rather uses all previous components and routines s/he has learned over the years, with emphasis on the combination of these as part of finding a solution to a new problem. Neural networks as a tool for engineering can also improve in both their running time and the richness of their behaviour by using a variety of pre-learned components.

Much advantage can be gained by allowing networks to use any piece of information available about the problem, or a way of solving it, for applications that range between fully structured problems (for which a full symbolic analysis exists) and those problems, termed 'random' by Abu-Mostafa,[28] that depend entirely on training examples.[29] In all such cases, the use is recommended of possibly available intelligent hints, as opposed to learning from scratch which for practical purposes is clearly inefficient because it ignores the partial information that may already exist. Exploiting this source makes for a better and faster adaptive learning machine.

The use of symbolic hints is becoming crucial and mandatory when a problem has to be solved under constraints. In such cases one is to force some behaviour and allow for learning and adaptation only under the flexibility that does not violate the requirements. If such a symbolic inclusion of mandatory constraints were not available, one would have to neglect neural learning and allow for the strict symbolic algorithms.

We will not review here all work done in this field, but will describe briefly three main methodologies. Abu-Mostafa[28,29] introduced a methodology of incorporating hints into networks. His idea is to train the network with two kinds of examples: the actual example of the training set, and virtual examples that are artificially made to express the symbolic hints. Both examples are learned by the same numerical learning method, such as backpropagation. Abu-Mostafa expounds his idea as a nice, clean paradigm that he has applied in the financial domain.

Another line of work concentrates on using neural nets to 'refine expert knowledge' or fix rules and heuristics; the neural learning is limited, so that extracting the refined rules from the knowledge-based neural networks is still an available capability. This is the approach adopted by, among others, Towell and Shavlik.[30]

Symbolic dynamics was also suggested to assist networks in learning complex tasks by introducing the external modularity of well understood subtasks to the network or fixing the initial architecture. Classical neural net learning is used for simple tasks and on small subnetworks, while the modularity of the system is based on external analysis. This allows for using previous networks (or modules) in a bottom-up programming fashion, for incorporating mandatory constraints on the initial architecture, and for applying pre-defined hints. The most advanced such scheme is probably the neural language NIL, described in previous work by Siegelmann.[14] (It has also been referred to by the name NEL or NIPPL.) We make use of this scheme in our application.

5.2 The schema of NIL

NIL is a procedural parallel high level programming language. It includes all the control flow statements of the language Pascal, but is richer in its possible types and expressions. NIL defines a real-time software, or in particular a reactive software; that is, its input and output channels are open and thus it communicates constantly with its environment. NIL, however, is not only a real-time parallel and easy-to-use programming language. Its main appeal for us resides in its associated compiler. It long used to be the case that a compiler of a programming language was a software that translates the language into a machine executable code. The NIL compiler is very different from existing compilers, as its target representation is a neural network.

Note that the original conception of neural networks is as a hardware computing architecture, even though they have come to be simulated on ordinary computers. In contrast to the latter kind, the network (if thought of as a hardware model) does not allow for specialized parts of the ('hardware') architecture that store commands or handle fixed and temporary variables; there are no specialized registers or hardware-based program counters to indicate the command under current execution. In practice, however, neural networks have been simulated on conventional computers, even using a conventional operating system platform. To avoid confusion, this must be kept in mind.

The network has a totally homogeneous structure: there is some number of neurons; at each tick of the clock, all neurons are updated with new values. Thus, a network step consists of a parallel execution of assignments. In the networks resulting from our NIL compiler, the program commands, variables and all software, along with the registers, counters and other parts of computer hardware, are all translated into the same structure of neurons in a network. Although including all pre-known hints or programs, the target architecture is able to adapt and learn, just like all other neural architectures.

There are many ways of translating a rule or a program into a network. Our compiler does not translate into an optimal network in the sense of its smallest possible size or minimal structure. On the contrary, the NIL compiler translates to a network that is not very small but is homogeneous in structure. This is done for practical reasons. The network obtained by the translation scheme is to serve as an initial architecture for further learning, one that includes all previous knowledge. If the minimal size network was chosen, the network would not generalize well to other properties to be learned later, but rather will demonstrate overfitting to the initial knowledge. A homogeneous, larger network, on the other hand, has a structure that enables it to continue and learn.

Having these properties, NIL is of a solid practical use to include previously known knowledge. This is in contrast to Abu-Mostafa's treatment of hints,[29] that can be acquired during the learning process, but for which there is never any assurance that they are going to be fully represented and that subsequent computation would actually adopt them and adhere to them. When NIL translates mandatory constraints, they will not be changed during the later training phases, as opposed to heuristic rules that can be tuned later on. Other current applications of NIL include an adaptive controller to roast coffee beans and a tool to improve simulations of fighter planes.

In our current project, the neural network constitutes a meta-program that controls the search for good fuel allocations in a large search space. The use of NIL allows having an initial architecture that includes the mandatory rules (in practice, safety rules) that will never be changed, as well as heuristic rules of search. During the operation, the network is being changed and the heuristic rules can be tuned or changed, or even new methods can be learned. The advanced network that appears after learning from experience cannot be well described in terms of clear rules, except for the mandatory rules that are uncompromisingly specifiable by the network. This freedom of having a controller that is not fully described by rules introduces flexibility and a substantial improvement in terms of performance.

6 NEURAL RULE REVISION

In the previous, entirely symbolic version—a conventional expert system—of the generator, i.e., the FUELCON component that by executing the ruleset generates a family of in-core fuel configurations,[3] the mode of use intended for expert users (as opposed to less ambitious practitioners) involves a manual stage downstream. In the ruleset, indeed, a particular search heuristic is embodied in the optional rules of the ruleset (i.e. the policy, not the safety-related rules). Once run on the problem at hand, the current strategy represented as a ruleset yields a large set of configurations that are then simulated numerically and graphically. Then—and this is the manual phase—the expert user looks at the performance of the simulated configurations as visualized, and may choose to revise the ruleset, either in order to improve the results for the particular problem or as means for the domain expert to probe and improve on the strategy. Then the revised ruleset is run again. Thus, the operation mode is a loop.

The neural/symbolic sequel project described here reflects the realization that it is feasible to fully automate rule revision (the 'missing link' in automating the loop: the step that was still manual) and thus, provided that we have an initial ruleset, to ensure that the entire process of fuel allocation is virtually automated.

As previously stated, neural networks can be utilized, when correctly designed, to learn and adapt. They thus have the potential to serve as a general optimizer of fuel allocation that will adapt differently to various cores and various fuel rods available. One could design a direct neural optimizer that will find the best allocation, like Hopfield networks[31] or Grossberg networks.[32] As the search space of the allocation problem is large, those neural optimizers will not do well; it is presently common knowledge that neural networks do not scale well to optimization.[26] We chose rather to adapt a smaller scale setup in order to solve the same allocation problem: we will learn and adapt the search heuristic along with its parameters and then apply the chosen search to calculate the preferred allocation. This application generalizes well and provides good solutions.

The method described in this section is the composition of $a \bigcirc c \bigcirc e$ from Fig. 3. The ruleset at the heart of the heuristic search is translated into a neural network, using the NIL scheme. By the use of the symbolic search FUELCON and the core simulator NOXER, the network considers its performance, just as the person does, and tunes and changes its heuristic. The mandatory part of the rules is not to be changed. It is important to note that we still allow the expert to interfere if he insists on doing that, and to add or change rules by teaching hints from virtual examples, as was suggested by Abu-Mostafa. Thus far, this does not look necessary, but it is likely that practice will resort to that option. Providing such an option seems anyway to be a plus in terms of credibility when it comes to trying to convince a human domain expert to accept the very notion of automatic revision.

6.1 The method of adaptation

As the network governs the heuristic search, it can be thought of as a closed-loop adaptive controller that acts in the physical environment of the nuclear core and controls the search (see, e.g., Wang *et al.*[33] on adaptive control). One may wonder how we can adapt the neural network that represents such a controller. This question is indeed more than legitimate: a classical approach to neural network adaptation is based on observing the input/output behaviour of the network and training it with a training set of pairs of input and desired output. These typical learning approaches are called supervised learning. Our network does not have such a setup. We do not know the exact rules or meta-heuristic for a given nuclear core, we have to optimize and learn it, but we cannot do it using simple examples.

What can we use for learning? We can apply the search that is implied by the network, receive allocations and feed back to the network a grade that describes how good the allocation is or, equivalently, how appropriate was the strategy revision enacted by the network. Using this grade, the network makes probabilistic changes to its structure to search for better possibilities. There are various adaptive algorithms that can do such tasks with some level of success. We chose the one that we found the most appropriate: the method of *reinforcement learning*.

In reinforcement learning, it is common to think explicitly of the network as a controller in an environment: see Fig. 6. The environment provides the inputs to the network, receives its output, and then provides the

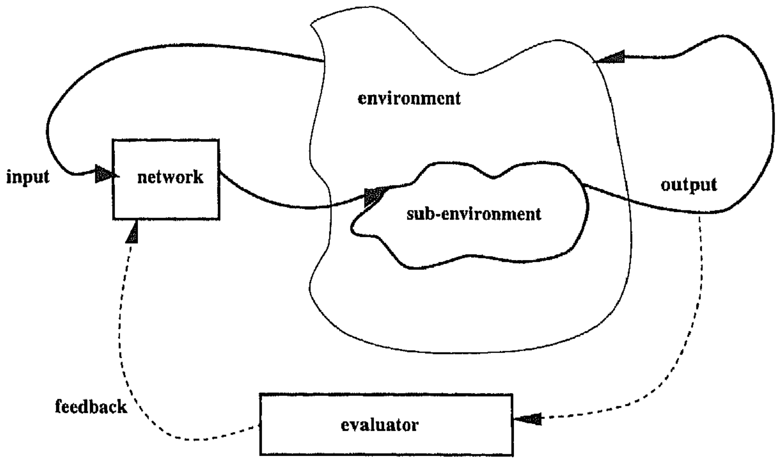

Fig. 6. Reinforcement learning.

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:

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

where h_i is the input net to the neuron, that is, the linear combination of the values of the neurons and possibly the external input.

By a standard analogy of neural network modelling, learning by reinforcement learning can be viewed as similar to learning by a teacher, where the output is supervised and feedback is given. The main difference is that the error in the output units for a given input pattern is no longer equal to the value (*DesiredOutput* – *TrueOutput*), but rather that the error of the output units has to be calculated in a more subtle manner.

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

$$\delta_i^\mu = D_i - \ < S_i^\mu >$$

where $< S_i^\mu >$ is the average of the ith output unit for input μ. In our case the score value r^μ is not binary, but rather may range in the continuum [0, 1]; we use the formula proposed by Barto and Jordan:[34]

$$\delta_i^\mu = r^\mu[S_i - \ < S_i^\mu >] + (1 - r^\mu)[-S_i - \ < S_i^\mu >]$$

From here on, the reinforcement learning algorithm corresponds to classical backpropagation.

6.2 An example of rule translation

Let us exemplify a rule for network conversion: consider two different rules (rules 5 and 8 in a ruleset from an example we discussed in a previous paper[3]):

5. *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 to 8 in the core.*
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 to 6) that position whose distance from the centre of the core is minimal.*

The first rule here, i.e., rule 5, is an elimination rule. It prevents placing high-burnup assemblies adjacently to each other, and embodies the heuristic policy of devising a so-called *checkerboard* pattern in respect of burnup levels: this policy alternates high-burnup and fresh or low-burnup assemblies to get an even distribution. The second rule, i.e., rule 8, is a preference rule that is meant to prune the space of solutions, as opposed to mandatory rules imposed in all sessions and dictated by safety considerations. In FUELCON, the distinction between the two kinds of rules is incorporated at the ergonomic level, in the sense that when using the system, and eventually when revising the ruleset on need, the expert user knows he should not touch the rules concerned with safety but only manipulate the other subset of rules. It is ultimately immaterial whether, in the implementation, two arrays or just one correspond to the two subsets of rules, but separation is preferable in terms of transparency. When it comes to automating the revision of the subset of preference rules, the program simply needs not touch the subset of mandatory rules. It only has to do with the preference rules.

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

```
A = record of [ burnt-once,
                burnt-twice,
                burnup,
                kind,
                position,
                ··· ]
```

and a position S which is represented as a record:

```
S = record of [ row,
                column,
                radius,
                assembly,
                ··· ]
```

The rules can be written as NIL functions. The function of rule 5 receives as input the record A and the position s, and decides whether the position contradicts the rule. The function of rule 8 receives an assembly and returns a position or the value 0. In the following functions, we write the reserved words of NIL in boldface and the predicates in italics. Lines are numbered successively.

```
1.  Function rule-5 ($A$, $S$): Boolean;
2.  var $i$: Integer, flag: Boolean;
3.  Begin
4.     $i = 0$;
5.     flag = Good-position;
6.     If
7.        ($A$.burnt-twice) $\wedge$ ($A$.burnup $>$ 20500)
8.     then
9.        Repeat
10.          $i = i + 1$;
11.          If
12.             ($neighbor(S, i)$)$\wedge$($i$.assembly.burnt-twice)
13.          then
14.             flag = Bad-position;
15.       Until
16.             (flag = Bad-position) $\vee$ ($p = 20$);
17.    rule-5 = flag
18. End;
```

This program carried out its task by scanning all of the positions in the one-eighth slice of the reactor core. In 1994, two students of Siegelmann and Nissan at Bar-Ilan University (Ramat-Gan, Israel) pointed out that a more efficient solution obtains if redundancy is eliminated: their alternative version just checked adjacent positions. Needless to say, the relevance of such programming fine points is pervasive in software development.

```
1.  Function rule-8 ($A$): Integer;
2.  var $i, p, v$: Integer;
3.  Begin
4.     $v = \infty$;
5.     $p = 0$;
6.     If
7.        $A$.burnt-once
8.     then
9.        For $i = 1$ to 20 do
10.          If
11.             rule$_1(A, i)\wedge$rule$_2(A, i)\wedge$rule$_3(A, i)_4(A, i)\wedge$
12.             rule$_5$ $(A, i)$ $\wedge$ rule$_6$ $(A, i)$ $\wedge$ ($i$.radius $< v$)
13.          then
14.             $p = i$;
15.             $v = i$.radius;
16.    rule-8 $= p$
17. End;
```

Each 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 is to be decided upon, depending on the particular application at hand.

To demonstrate, we write down the dynamic update of the neurons in a recurrent implementation of rule 5. The translation of the second function is similar, but care has to be taken in the function calls to the functions rule$_1$ up to rule$_6$. There will be a neuron for each variable and a temporary variable, as well as for the distributed representation of the program counter (i.e., line number). The function includes the variables i and flag, as well as the rule 5 itself. In addition, each expression implies an expression variable (and possibly some temporary variables as well). The program counters are pc$_1$ to pc$_{18}$.

- **The variables:**

1. The variable i is changed in lines 4 and 10. We can write its substitutions in the general formula of

$$i = 0 \cdot \mathrm{pc}_4 + (i+1)\cdot \mathrm{pc}_{10} + i(1 - \mathrm{pc}_4 - \mathrm{pc}_{10})$$

2. The function variable rule-5 = flag$\cdot$pc$_{17}$.

- **The expression variables:**

3. The expression of line 7 requires two temporary variables:

$$v_7 = \sigma(\mathrm{pc}_7 + \mathrm{A.burnt\text{-}twice} + v_{7,1})$$

where $v_{7,1}$ tests whether the burnup is high and will be set with program counter 6.

4. The Boolean expressions of lines 12 and 16 are similar.

- **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:

$$\mathrm{pc}_i = \mathrm{pc}_{i-1} \text{ for}$$
$$i = \{2, 3, 4, 5, 6, 7, 9, 11, 12, 14, 15, 16, 18\},$$

$$\mathrm{pc}_8 = \sigma(\mathrm{pc}_7 + \mathrm{v}_7 - 1)$$

$$\mathrm{pc}_{10} = \sigma(\mathrm{pc}_9 + \mathrm{v}_{\mathrm{loop}} - 1), \quad \mathrm{v}_{\mathrm{loop}} = \sigma(\mathrm{pc}_{16} - \mathrm{v}_{16})$$

$$\mathrm{pc}_{13} = \sigma(\mathrm{pc}_{12} + \mathrm{v}_{12} - 1)$$

$$\mathrm{pc}_{17} = \sigma(\mathrm{pc}_{16} + \mathrm{v}_{16} - 1)$$

The resulting network is the cyclic interconnection describing these equations.

7 GENETIC ALGORITHMS AS AN ALTERNATIVE FOR GENERATING RELOAD CONFIGURATIONS

In November 1996, Jun Zhao successfully discussed at the University of Greenwich a doctoral dissertation on FUELGEN.[35] This project, supervised *in loco* by Brian Knight, Ephraim Nissan and Alan Soper and, from abroad, by Alex Galperin (with Eve Siegelmann also consulting), consists of a remake of FUELCON made by resorting to genetic algorithms. Its output is simulated by resorting to NOXER. Results were obtained, for given case studies, that to our knowledge are better than the results published in the literature, using any method. FUELGEN will be described in separate papers.

Let it suffice, here, to point out that, *a posteriori*, it can be seen that the incorporation of intelligent technologies paradigms within the FUELCON umbrella project reflects the successive emergence of expert systems, then neural networks, then genetic algorithms. It is because of what they are best at, that neural networks have been positioned in the project in the function we have been considering in this paper. Genetic algorithms, instead, are very appealing for the reload pattern generation function. An anonymous referee appears to think the same when stating: "It seems to me that the logical way to solve this problem is by GAs."

As an alternative to the generation of allocations by means of a rule-based approach, FUELGEN generates allocations via genetic algorithms. The use of genetic algorithms for the generation of fuel positioning in the reactor core fits the paradigm $d \bigcirc f$ according to Fig. 3. In FUELGEN, there is no use for any heuristic rule, any intervention of human reasoning within the framework we have been discussing in this paper, or any problem modelling. Allocations are generated—first randomly, and then by a sequence of crossover and mutation steps. The fitness of the allocation is measured for all allocations considered, and the algorithm probabilistically biases towards better configurations by means of reproduction.

Care is taken to prevent the algorithm from selecting such allocations that are forbidden by the safety-related mandatory rules. (Of course, safety is never entrusted to that level of processing. After generating the configurations, simulation prunes out unsafe solutions; moreover, at an actual plant one more level of simulation is to be carried out, with standard software as mandated by regulations binding by law. On safety, see e.g. Gittus *et al.*,[36] which discusses safety characteristics, as understood, of fuel and reactors in the European Community.)

8 CONCLUDING REMARKS

Fuel reload design is a crucial task in the operation and economics of nuclear power plants. We described FUELCON, an expert system that automates reload design. Then we discussed one of the sequel projects that augments the expert system with a neural component. We integrate neural processing into the architecture for the purposes of automating the revision of the ruleset that embodies fuel reload strategies.

With FUELCON, it is not just the discovery of fuel allocation patterns that is carried out. FUELCON also enables the human expert to explore or discover the potential of new strategies, whereas the neural component reconceptualizes in terms of automation that same stage of the operation mode of FUELCON. We briefly announced the completion of FUELGEN, a tool based on genetic algorithms which fits within the FUELCON umbrella project. The architectural alternatives within this umbrella project have been described, in this paper, in terms of an AND/OR tree.

ACKNOWLEDGEMENTS

We wish to thank Dr Shuky Kimhi, whose doctoral project (under Galperin's supervision) involved the transformation of Galperin and Nissan's early prototype of FUELCON into a complete tool, fully tested on real-case studies. This paper, in turn, mainly describes the sequel project—developed by Siegelmann and Nissan in 1994–95—for augmenting the expert system with neural computation, automating ruleset revision. The paper was written by Siegelmann and Nissan in equal parts. All three authors are equally responsible for this paper. The sequel project based on genetic algorithms, referred to in Section 7, is the subject of Jun Zhao's recently completed doctoral project at the University of Greenwich under the supervision of the FUELCON team.

REFERENCES

1. Woodall, D. M., Nuclear power systems. In *The Engineering Handbook*, ed. R. C. Dorf. CRC Press, Boca Raton, FL, and IEEE Press, New York, 1995.
2. Bernard, J. A. & Washio, T., *Expert Systems Applications within the Nuclear Industry*. American Nuclear Society, La Grange Park, IL, 1989.
3. Galperin, A., Kimhi, S. & Nissan, E., 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*, 1993, **12**(4), 369–415.
4. Galperin, A. & Nissan, E., Applications of a heuristic search method for generation of fuel reload configurations. *Nuclear Science and Engineering*, 1988, **99**(4), 343–352.
5. Galperin, A., Kimhi, S. & Segev, M., A knowledge-based system for optimization of fuel reload configurations. *Nuclear Science and Engineering*, 1989, **102**.
6. Galperin, A. & Kimhi, S., Application of knowledge-based methods to in-core fuel management. *Nuclear Science and Engineering*, 1991, **109**, 103–110.
7. Kimhi, Y., A non-algorithmic approach to the in-core fuel

management problem of a PWR core. PhD dissertation, Nuclear Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel, 1992 (in Hebrew; English abstract).

8. Galperin, A., Exploration of the search space of the in-core fuel management problem by knowledge-based techniques. *Nuclear Science and Engineering*, 1995, **119**(2).
9. Galperin, A. & Nissan, E., Discovery as assisted by an expert tool: a refinement loop for heuristic rules in an engineering domain. In *Proceedings of the 16th Convention of Electrical and Electronics Engineers in Israel*, Tel-Aviv, 7–9 March 1989. IEEE, New York: Paper 1.4.3.
10. Nissan, E., Siegelmann, H. & Galperin, A., An integrated symbolic and neural network architecture for machine learning in the domain of nuclear engineering. In *Proceedings of the Conference on Pattern Recognition and Neural Networks*, within the 12th ICPR: *International Conferences on Pattern Recognition*, Jerusalem, 9–13 October 1994. IEEE Computer Society Press, New York, 1994.
11. Nissan, E., Siegelmann, H., Galperin, A. & Kimhi, S., Towards full automation of the discovery of heuristics in a nuclear engineering project, by combining symbolic and subsymbolic computation. In *Proceedings of the Eighth International Symposium on Methodologies for Intelligent Systems (ISMIS '94)*, ed. Z. W. Raś & M. Zemankova, Charlotte, North Carolina, 16–19 October 1994, pp. 427–436. Springer Verlag Lecture Notes in Artificial Intelligence, Vol. 869, Berlin, 1994.
12. Galperin, A., Kimhi, S., Nissan, E., Siegelmann, H. & Zhao, J., Symbolic and subsymbolic integration in prediction and rule-revision tasks for fuel allocation in nuclear reactors. In *Proceedings of the Third European Congress on Intelligent Techniques and Soft Computing (EUFIT '95)*, Aachen, Germany, 28–31 August 1995, Vol. 3, pp. 1546–1550. ELITE-Foundation, Aachen, 1995.
13. Galperin, A., Nissan, E. & Siegelmann, H., The hybrid-paradigm conception of ongoing or outlined sequel projects to FUELCON: a concentric attack on the nuclear-fuel on-line management problem. Workshop on Environmental and Energy Applications of Neural Networks. Richland, WA, 1995.
14. Siegelmann, H. T., On NIL: the software constructor of neural networks. *Parallel Processing Letters*, 1995, in press. Previous version: Neural programming language. Proceedings of the 12th National Conference on Artificial Intelligence (AAAI '94). Seattle, WA, 1994, pp. 882–887.
15. Goonatilake, S. & Khebbal, S. (eds), *Intelligent Hybrid Systems*. Wiley, Chichester, UK and New York, 1995.
16. Cochran, R. G. & Tsoulfanidis, N., *The Nuclear Fuel Cycle: Analysis and Management*. American Nuclear Society, La Grange Park, IL, 1990.
17. Nissan, E., Intelligent technologies for nuclear power systems: heuristic and neural tools. *Expert Systems with Applications*, in press.
18. Poetschhat, G. R., Rothleder, B. M., Faught, W. S. & Eich, W. J., Interactive fuel shuffle assistant graphics interface and automation for nuclear fuel shuffle with PDQ7. In *Proceedings of Topical Meeting on Advances in Fuel Management*, Pinehurst, NC, 2–5 March 1986. American Nuclear Society, La Grange, IL, 1986.
19. Faught, W. S., Prototype fuel shuffling system using a knowledge-based toolkit. Technical Report, IntelliCorp, Mountain View, CA, 1987.
20. Rothleder, B. M., Poetschhat, G. R., Faught, W. S. & Eich, W. J., The potential for expert system support in solving the Pressurized Water Reactor fuel shuffling problem. *Nuclear Science and Engineering*, 1988, **100**.
21. Siegelmann, H. T. & Sontag, E. D., On the computational power of neural networks. *Journal of Computer and Systems Science*, 1995, **50**(1), 132–150.
22. Siegelmann, H. T. & Sontag, E. D., Analog computation via neural networks. *Theoretical Computer Science*, 1994, **131**, 331–360.
23. Siegelmann, H. T., Computation beyond the Turing limit. *Science*, 1995, **268**(5210), 545–548.
24. Cybenko, G., Approximation by superpositions of a sigmoidal function. *Mathematical Control, Signals and Systems*, 1989, **2**, 303–314.
25. Hornik, K., Stinchcombe, M. & White, H., Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 1990, **3**, 551–560.
26. Haykin, S., *Neural Networks: A Comprehensive Foundation*. IEEE Press, New York, 1994.
27. Hertz, J., Krogh, A. & Palmer, R., *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, 1991.
28. Abu-Mostafa, Y. S., Learning from hints in neural networks. *Neural Computation*, 1995, **7**, 639–671.
29. Abu-Mostafa, Y. S., Machines that learn from hints. *Scientific American*, 1995, **272**(4), 64–69.
30. Towell, G. & Shavlik, J., Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 1993, **13**(1), 71–101.
31. Hopfield, J. J., Neurons with graded responses have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 1984, pp. 3088–3092.
32. Cohen, M. A. & Grossberg, S., Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man and Cybernetics*, 1983, **13**, 815–826.
33. Wang, H., Liu, G. P., Harris, C. J. & Brown, M., *Advanced Adaptive Control*. Pergamon, Oxford, 1995.
34. Barto, A. G. & Jordan, M. I., Gradient following without back-propagation in layered networks. In *Proceedings of the First IEEE International Conference on Neural Networks*, ed. Caudil & C. Butler, San Diego, CA, Vol. 2, pp. 629–636. IEEE, New York, 1987.
35. Zhao, J., An examination of the macro genetic algorithm and its application to loading pattern design in nuclear fuel management. PhD dissertation, Computer Science, University of Greenwich, London, 1996.
36. Gittus, J. H., Matthews, J. R. & Potter, P. E., Safety aspects of fuel behaviour during faults and accidents in pressurised water reactors and in liquid sodium cooled fast breeder reactors. *Journal of Nuclear Materials*, 1989, **166**(1/2), 132–159.