

**APPLICATION OF COMPUTATIONAL
INTELLIGENCE METHODS TO IN-CORE FUEL
MANAGEMENT**

**T.C. BURHANETTİN KURULU
DOKÜMANTASYON MERKEZİ**

Ph.D. Thesis by
Adem ERDOĞAN, M.Sc.

(301942001)

104264

Date of submission : 3 July 2001

Date of defence examination : 25 October 2001

Supervisor (Chairman): Prof. Dr. Melih GEÇKİNLİ *Melih Geçkinli* 07.11.2001

Members of the Examining Committee Prof.Dr. Şarman GENÇAY (İ.T.Ü.) *Şarman Gençay* 15.11.2001

Prof.Dr. Atilla ÖZGENER (İ.T.Ü.) *Atilla Özgener* 15.11.2001

Prof.Dr. Burhanettin CAN (M.Ü.) *Burhanettin Can* 14.11.2001

Assoc. Prof.Dr. Serhat ŞEKER (İ.T.Ü.) *Serhat Şeker* 15.11.2001

NOVEMBER 2001

PREFACE

In this study, a computer program package has been developed which supports the in-core fuel management activities for pressurized water reactors, generates and recommends an optimum-loading pattern to ensure safe and efficient reactor operation.

I wish to express my appreciation to my adviser, Prof. Dr. Melih Geçkinli, for his continued interest and guidance throughout the course of this study.

I would also like to thank all the members of the Nuclear Engineering Department, Çekmece Nuclear Research and Training Center, for their support and encouragement.

Finally, I want to thank my wife for her patience and help in preparing this thesis.

November, 2001

Adem ERDOĞAN

CONTENTS

ABBREVIATIONS	v
LIST OF TABLES	vi
LIST OF FIGURES	vii
LIST OF SYMBOLS	iv
SUMMARY	xi
ÖZET	xii
1. INTRODUCTION	1
2. OPTIMUM LOADING PATTERN	6
2.1 Comparing Refueling Schemes	10
2.2 Optimization	11
3. REVIEW OF OPTIMIZATION METHODS	13
3.1 Dynamic Programming	13
3.2 Heuristic Search	14
3.3 Simulated Annealing	17
3.4 Direct Search	19
3.5 Binary Exchange	19
3.6 Genetic Algorithms	20
3.7 Hybrid Search	20
4. CORE CALCULATIONS	22
4.1 Cross Sections	25
4.2 Global Core Calculations	26
5. ARTIFICIAL NEURAL NETWORKS	31
5.1 Introduction	31
5.2 Neural Networks	32
5.3 The Back-propagation Algorithm	33
5.3.1 Generating Output	35
5.3.2 Training	36
5.3.3 Factors Affecting Learning	41
6. PREDICTING CORE POWER AND K_{EFF} USING NEURAL NETWORK	43
6.1 The Neural Network Structure	43

6.1.1 Training Patterns	45
6.1.2 The Results	49
7. OPTIMUM LOADING PATTERNS: SIMPLE RULE-BASED METHOD	56
7.1 Results	60
8. GENETIC ALGORITHMS	64
8.1 Fundamentals of The Algorithm	65
8.2 Applications	67
8.2.1 Traveling Salesman	68
8.3 Theory	69
9. OPTIMUM LOADING PATTERNS: GENETIC ALGORITHM	74
9.1 Crossover	77
9.2 Mutation	78
9.3 Results	82
9.3.1 Verification of the Search Procedure	82
9.3.2 Calculations Using 7-Assembly Types	85
10. THE XCORE PROGRAM	90
11. CONCLUSIONS AND DISCUSSIONS	95
REFERENCES	100
APPENDIX A	108
APPENDIX B	111
APPENDIX C	113
RESUME	116

ABBREVIATIONS

ANN	: Artificial neural network
BOC	: Beginning of cycle
BWR	: Boiling Water Reactor
DNBR	: Departure from nucleate boiling ratio
EOC	: End of cycle
GA	: Genetic Algorithm
MCHFR	: Minimum critical heat flux ratio
PWR	: Pressurized Water Reactor



LIST OF TABLES

	<u>Page No</u>
Table 6.1. Fuel assemblies defined for Almaraz NPP	43
Table 6.2. Fuel assemblies and their encoded neural network values	45
Table 6.3. The results of the position effect methods	53
Table 6.4. The time required for the calculation of a single loading pattern.....	55
Table 7.1. Comparison of the results of the two simple rule-based random methods.....	61
Table 7.2. Results of the simple rule-based random method	62
Table 9.1. Fuel assembly types and their rank numbers	75
Table 9.2. The best 10 of all loading patterns and a sample GA result	83
Table 9.3. GA search results for the simplified case.....	85
Table 9.4. Archived results of the sample calculations with the genetic algorithm.....	86
Table 9.5. A PWR benchmark vs. GA results	89

LIST OF FIGURES

	<u>Page No</u>
Figure 2.1 : Loading pattern representation in two-dimensional configuration space.....	6
Figure 4.1 : A supercell representation of a fuel assembly.....	26
Figure 5.1 : The biological neuron and its artificial model.....	33
Figure 5.2 : A Feed-forward neural network with one hidden layer.....	35
Figure 5.3 : Output from a single node.....	37
Figure 5.4 : The sigmoid function.....	37
Figure 6.1 : A sample of loading pattern for Almaraz NPP.....	44
Figure 6.2 : Similar loading patterns obtained by binary exchange operation.....	46
Figure 6.3 : Network learning curve.....	49
Figure 6.4 : The number of iterations versus α (until $E < 0.05$).....	50
Figure 6.5 : The number of iterations versus η (until $E < 0.05$).....	51
Figure 6.6 : Number of iterations versus number of hidden layer nodes.....	52
Figure 6.7 : Maximum power fraction error versus number of hidden layer nodes..	52
Figure 6.8 : Maximum power fraction: generated output versus target values.....	54
Figure 6.9 : Effective multiplication factor: generated output versus target values..	54
Figure 7.1 : Initial fuzzy membership function for the outer region of the core.....	58
Figure 7.2 : Reactor core inner/outer regions.....	58
Figure 7.3 : The flow chart of the simple rule-based method.....	59
Figure 7.4 : $\mu(x)$ modification for sample cases.....	61
Figure 7.5 : $\mu(x)$ and fuel assembly distribution for an optimum loading pattern ($p_{\max}=1.285$, $k_{\text{eff}}=1.033$).....	63
Figure 9.1 : The loading pattern and the one-dimensional representation.....	75
Figure 9.2 : A sample chromosome representing the loading pattern of Figure 9.1.....	75
Figure 9.3 : A selected block from a loading pattern for the crossover operator.....	77
Figure 9.4 : A cycle of crossover operation.....	79
Figure 9.5 : A sample mutation operation.....	80
Figure 9.6 : The flow chart of GA in fuel pattern selection.....	81
Figure 9.7 : Patterns of the genetic algorithm results.....	84
Figure 9.8 : Average fitness value of the population along generations	87
Figure 9.9 : Maximum fitness value of the population along generations	87
Figure 9.10 : Average power fraction of the population.....	88
Figure 9.11 : Minimum power fraction in the population.....	88
Figure 10.1 : The main window of <i>Xcore</i>	91
Figure 10.2 : Program modules.....	92
Figure 10.3 : A sample <i>XCore</i> window that lists GA selected loading patterns.....	93
Figure A.1 : Maximum normalized power as a function of burnup.....	109
Figure B.1 : Power distribution calculated by the neural network.....	111

Figure B.2 : Neural network errors for power distribution 112

Figure C.1 : $\mu(x)$ and fuel assembly distribution for an optimum loading pattern... 113

Figure C.2 : $\mu(x)$ and fuel assembly distribution for an optimum loading pattern... 114

Figure C.3 : $\mu(x)$ and fuel assembly distribution for an optimum loading pattern... 115



LIST OF SYMBOLS

D	: Diffusion coefficient (m)
E	: The error function of the neural network
G	: Average energy released per fission (MEV/fission)
k_{eff}	: Effective multiplication factor
k_{∞}	: Infinite medium multiplication constant of a fuel assembly
N_{FA}	: Total number of fuel assemblies in the core
N_g	: Total number of energy groups
NP_j	: Normalized power fraction for fuel assembly j
o_k	: Generated output vector of the neural network
P_j	: Thermal power produced by fuel assembly j (W)
$\bar{P}$	: Average power produced per fuel assembly (W)
p_{max}	: Maximum power fraction of a loading pattern
P_t	: Total thermal power (W)
q'_{max}	: Maximum linear heat generation (W/m)
q'''_{max}	: Maximum power density (W/m ³)
$q'''(r)$	: Power produced per unit volume at position r (W)
$\underline{r}$	: Space vector
t_k	: Target output vector of the neural network
T	: Temperature (°K)
V_{FA}	: Volume of fuel assemblies
x_i	: Input vector to the neural network
y_j	: Output vector of the neural network
α	: Momentum parameter for the neural network
β_j	: Neural network input value for fuel assembly j
χ	: Fission spectrum
η	: Learning constant for the neural network
ϕ_{kj}	: Weights for the neural network (between hidden and output layers)
λ	: Neuron gain constant for the sigmoid function
$\mu(\mathbf{x})$	: Fuzzy membership function
ν	: Average number of neutrons produced per fission
σ	: Microscopic cross section (m ²)
ω_{ij}	: Weights for the neural network (between input and hidden layers)
Φ	: Neutron flux
Σ_f	: Macroscopic fission cross section (m ⁻¹)

Σ_s : Macroscopic scattering cross section (m^{-1})
 Σ_a : Macroscopic absorption cross section (m^{-1})
 Σ_R : Macroscopic removal cross section (m^{-1})



APPLICATION OF COMPUTATIONAL INTELLIGENCE METHODS TO IN-CORE FUEL MANAGEMENT

SUMMARY

In this study, a computer program package has been developed which supports the in-core fuel management activities for pressurized water reactors, generates and recommends optimum loading patterns to ensure safe and efficient reactor operation.

A search for an optimum fuel loading pattern must be conducted in the space of several core parameters such as power distribution, which is an excessively time consuming computational process. Global core calculation codes take a relatively long time to do the task. The time interval necessary for the iterative process was reduced by using an artificial neural network estimator for the calculations. In this way, it was possible to analyze more loading patterns in the same time interval and the probability of finding a desired optimum was increased.

As a case study, the core of the Almaraz Nuclear Power Plant of Spain, a pressurized water reactor, was modeled for the core calculation code system. The 2-group cross sections for the fuel assembly types were calculated and stored for later usage with the diffusion code. The fuel depletion effects were omitted.

2000 loading patterns were generated by placing fuel assemblies to random positions in the core, and for each pattern the power distribution and effective multiplication factor (k_{eff}) were calculated with the diffusion code. At the next stage, 500 of the loading patterns were introduced to the neural network as input data for the training process. The remaining 1500 patterns were used to validate the neural network implementation. It was shown that the neural network estimates the power distribution and the effective multiplication factor within acceptable error limits.

To complete the system, a loading pattern generator was developed. This module consists of a set of rules and an algorithm that places the fuel assemblies to core positions. The neural network estimated the power distribution and effective multiplication factor for the loading patterns that were generated. The patterns that have a maximum power fraction lower than, and a minimum effective multiplication factor higher than reference values were stored as candidate optimum patterns. At the last stage of the work, an alternative loading pattern generator based on genetic algorithm method was developed. In this method, an initial loading pattern is improved by applying the genetic operators to obtain the optimum. The loading patterns obtained from the simple rule-based and the genetic algorithm methods were compared, and the genetic algorithm was shown to be more successful.

It was seen that, it is possible to automate in-core fuel management activities by applying artificial intelligence techniques.

HESAPLAMALI ZEKA METODLARININ KALP İÇİ YAKIT YÖNETİMİNE UYGULANMASI

ÖZET

Bu çalışmada, basınçlı su reaktörleri için kalp-içi yakıt yönetiminde yardımcı olan, reaktörün güvenli ve verimli çalışmasını sağlayacak en uygun yakıt yükleme motiflerini belirleyen ve öneren bir bilgisayar program sistemi geliştirilmiştir.

En uygun yakıt yükleme motifinin belirlenmesinde motifleri karşılaştırmak için gereken güç dağılımı gibi reaktör kalp parametrelerinin hesaplanması, difüzyon denklemini çözen genel kalp hesabı kodları ile, uzun bir hesaplama zamanı almaktadır. Birçok tekrar gerektiren bu hesaplar bir yapay sinir ağı hesaplayıcısı ile yapılarak işlem süresi kısaltılmıştır. Böylece, daha çok yakıt yükleme motifini kısa bir zaman süresinde analiz etmek ve en uygunu bulma olasılığını artırmak mümkün olmuştur.

Bu amaçla, bu çalışma için örnek alınan İspanya'nın basınçlı su tipi Almaraz Nükleer Güç Reaktörü'nün kalp düzeni kullanılan kalp kod sistemi için modellenmiştir. Belirlenen farklı yakıt demeti tipleri için 2 gruplu tesir kesitleri hesaplandıktan sonra gerektiğinde kullanılmak üzere saklanmıştır. Yanma etkisi gözönüne alınmamıştır.

2000 adet yakıt yükleme motifi yakıt demetlerinin gelişi-güzel yerleştirilmesi yöntemi ile üretilmiş ve bunlara ait güç dağılımı ve çoğaltma sabiti (k_{eff}), difüzyon kodu kullanarak hesaplanmıştır. Bir sonraki aşamada, bu motiflerden 500'ü yapay sinir ağının eğitilmesi için kullanılmıştır. Kalan 1500 yükleme motifi ise, yapay sinir ağının başarısının testi için değerlendirilmiştir. Ağın, belirli bir hata sınırı içerisinde, güç dağılımı ve çoğaltma sabiti hesabını yapabildiği gözlenmiştir.

Sistemi tamamlamak amacı ile otomatik bir yakıt yükleme motifi üreticisi geliştirilmiştir. Bu modül, yakıt demeti tipine göre yerleştirmeyi düzenleyen birkaç kural ve bunları reaktör kalbine yerleştiren bir algoritmadan oluşmaktadır. Bununla üretilen yükleme motifleri için güç dağılımı ve çoğaltma sabiti yapay sinir ağı kullanarak hesaplanmıştır. Önceden belirlenmiş bir maksimum güç üretimi ve minimum çoğaltma sabiti kriterine uyan yükleme motifleri uygun seçenekler olarak sunulmak üzere saklanmıştır. Çalışmanın son aşamasında, alternatif bir yakıt yükleme motifi üreticisi olarak genetik algoritma yöntemini kullanan bir sistem geliştirilmiştir. Bu yöntemle, başlangıç olarak alınan bir yükleme motifi genetik operatörler ile geliştirilerek uygun yükleme motifleri elde edilmeye çalışılmaktadır. Daha sonra, genetik algoritma ve kuralla bağlı gelişi-güzel arama yöntemi karşılaştırılmış ve genetik algoritmanın başarısı gösterilmiştir.

Geliştirilen sistemle, kalp-içi yakıt yönetiminin otomatikleştirilmesinin yapay zeka teknikleri ile mümkün olduğu gösterilmiştir.

1. INTRODUCTION

This work demonstrates the use of computational intelligence techniques to pressurized water reactor (PWR) in-core fuel management activities. The goal is to find optimum fuel loading patterns that satisfy the reactor safety requirements, and lower the operating cost by reducing the fuel amount and increasing burnup, or elongating the time fuel is exposed.

A typical PWR (Pressurized Water Reactor) contains 100 to 200 fuel assemblies in its core. For the first burn-up cycle, the core is loaded with fresh fuels that have different enrichment values. These fuel assemblies are located according to a specific pattern to flatten the power distribution and to make better utilization. At the end of each cycle, about 1/3 of the assemblies are replaced with fresh ones. The initial loading pattern is to be verified with the licensed codes. For the succeeding cycles this is not necessary.

Traditionally, fresh fuel assemblies are located at the core periphery while the used and low enrichment assemblies are placed to the inner regions with respect to their k_{∞} . This procedure provides the required flat power distribution.

A more recent approach makes use of the “low-leakage” loading pattern. In this procedure, partially exposed fuels with low enrichment (reduced fissile material content) are located at the core periphery while fresh fuels go to the inner core regions. This reduces the irradiation-induced damage to the pressure vessel and improves neutron economy by reducing leakage. However, the radial power distribution is less flat, with possible density power peaks at fresh fuel assembly locations. Burnable neutron absorbers are used to flatten the power distribution to acceptable levels.

In addition to this function, burnable absorbers suppress initial excess reactivity and may help to control the moderator temperature coefficient (Petrovic, 1990).

As an optimization problem, in-core fuel management includes safety-related limitations as constraints while the cycle length is externally given in the planned power demand. Some of the safety-related factors are hot channel factor, moderator temperature coefficient and maximum fuel assembly burnup.

In most cases, it is not possible to deal with economic parameters directly to reduce the cost. Thus, the direct optimization of economic parameters is usually replaced by optimization of some other quantity related to reactor physics, like uranium utilization or cycle length.

The optimum solution is based on the decision of the following factors:

- Type of loading pattern
- Fresh fuel enrichment
- Number of fresh fuel assemblies
- Loading pattern
- Use of burnable absorbers.

Some of these variables may be fixed, like fuel enrichment. Actually, in most cases, factors other than “loading pattern” are determined before the final optimization is done. In this case, the problem is reduced to find the optimum loading pattern.

The optimization is performed stepwise. In the first step, the number of fuel assemblies and fuel enrichment are fixed. Next, a loading pattern is constructed. The power distribution is calculated for the given reactor core configuration. These steps are repeated until the power distribution complies with the restrictions.

The optimum loading pattern should allow a longer cycle length without compromising the integrity of the reactor core components.

During this iterative procedure, the loading pattern for each cycle influences succeeding ones. These multi-cycle effects have to be taken into account by performing the optimization over several cycles.

There is an important difference between a usual optimization problem and in-core fuel management optimization. Usually, for a given set of values of independent variables, value of target function is obtained by a simple substitution into some

formula. But, in the in-core fuel management case, for a given set of independent variables, finding the value of target function itself involves solving transport or diffusion equation many times. Therefore, the procedure of finding the right loading patterns is rather time-consuming.

The application of computational intelligence techniques to various engineering problems has been around for over 15 years. There is a large collection of examples of the application of artificial neural networks and genetic algorithms, as well as other techniques.

Uhrig (1991) and **Levine (1992)** discuss some potential applications of artificial neural networks to the operation of nuclear power plants. Artificial neural networks have been widely used in nuclear applications, such as, monitoring on-line stability of BWR's (**Tambouratsiz and Antonopoulos, 1999**), predicting neutron diffusion eigenvalues (**Lysenko et al., 1999**), analyzing nuclear power plant performance (**Guo and Uhrig, 1992**), predicting plant power (**Roh et al., 1991**) and monitoring feedwater flow rate and component thermal performance of PWR's (**Kavakhoğlu and Upadhyaya, 1994**). **Jouse and Williams (1993)**, **Ohga and Seki (1993)**, **Cheon and Chang (1993)**, **Barlett and Uhrig (1992)**, **Kim and Lee (1993)** and **Kim and Barlett (1994)** identified abnormal events and transients in power plants using neural networks.

Similarly, **DeChaine and Feltus (1995)**, **Toshinsky et al. (1999)**, **Chapot et al. (1999)**, **Toshinsky et al. (2000)**, **Pereira et al. (1999)** and **Francois and Lopez (1999)** applied genetic algorithms to the optimization of nuclear fuel loading patterns.

Fuzzy-rule based and knowledge based systems are other techniques that have been used in the optimization of fuel management (**Kim et al., 1993**), (**Galperin et al, 1989**) and (**Galperin and Kimhi, 1991**).

Also, other optimization methods have been applied successfully for the same problem. Examples include non-linear optimization by **Quist et al. (1999)**, linear optimization by **Sauar (1971)**, **Suzuki and Kiyose (1971)** and **Driscoll et al. (1990)**, direct optimization by **Suh and Levine (1990)**, simulated annealing method by

Kropaczek and Turinsky (1991), heuristic learning method by **Hoshino (1972)**, and backward diffusion method by **Yung-An et al. (1986)**.

Wade and Terney (1971), **Aragones et al. (1977)**, **Hoshino (1978)**, **Ahn and Levine (1985)**, **Sauer and Driscoll (1985)**, **Hobson and Turinsky (1986)**, **Kim et al. (1987)**, **Moon (1988)**, **Maldonado et al. (1995)**, **Van Geemert et al. (1998)**, **Turinsky (1999)** and **Shatilla et al. (2000)** applied various conventional optimization techniques to the search problem.

The organization of this work is as follows:

Chapter 2 discusses the processes involved in the in-core fuel management activities to set the scope of the problem. An optimum fuel-loading pattern is defined and several optimization methods are briefly explained in chapter 3.

The conventional calculations that are carried out for determining the core parameters are outlined in chapter 4. These include the estimations of fuel assembly constants and calculating the power distribution and effective multiplication factor. The Haling Principle that is widely used in the in-core fuel management process is reviewed in Appendix A.

To shorten the computation time, computational intelligence techniques are exploited to develop a robust system that finds acceptable solutions to the problem. Considering this point, it is attempted to make use of a properly trained artificial neural network as the estimator of the core parameters, namely, the power fractions generated by the fuel assemblies and the effective multiplication factor of the core. Chapter 5 outlines the foundation of the neural network architecture used in this work.

Chapter 6 explains the neural network implementation to predict core parameters. The encoding method that is developed for this work to convert a reactor core configuration into a vector form that can be fed to the neural network is detailed and the results of the application is discussed.

In the second part of the optimization process, outlined in chapter 7, a simple rule-based system is used to generate loading patterns. The rules that are defined to construct a loading pattern are listed. A mechanism that is developed to define a

flexible rule system is described. The results of the optimum loading pattern generation calculations are given. Later, the results and the shortcomings of the method are discussed.

A better solution to the problem of generating patterns that converge to the desired optimum is implemented by using genetic algorithms. Chapter 8 describes this algorithm and some theoretical background. The application of the method and its results are discussed in chapter 9. The encoding that is developed to define a *chromosome* is detailed and the re-ranking method is described. The techniques that are implemented to apply the crossover and mutation operators are explained. A simplified problem that is set to validate the genetic algorithm method is described.

The program *XCore*, which implements the various techniques used in this work, is introduced in chapter 10. The conclusions of the work are given in chapter 11. Here, the success of the method is discussed and some suggestions for further study are emphasized.

2. OPTIMUM LOADING PATTERN

In addition to the difficulty of solving the involved equations, the search space of an optimal fuel-loading pattern is another important problem, since the number of possible loading patterns is quite large. This case is illustrated in the following discussion.

At the most general level, loading pattern optimization problem depends on the number of fuel assembly types (N_F) and the number of core locations (N_L) that hold fuel assemblies, both of which are fixed for a given reactor core configuration.

For a single location there is no need for an optimization. However, for two locations and two assembly types there are four different possible loading patterns ($N_F=2$ and $N_L=2$).

The patterns may be listed as [1,1] [1,2] [2,1] [2,2], where [2,1] is the pattern corresponding to the one with assembly type 2 located at the first position and 1 at the second position.

Likewise, there are 9 different patterns for configuration of 2 locations and 3 assembly types.

[1,1] [1,2] [1,3] [2,1][3,2] [3,3]

Graphical representation of the above configurations with $N_L=2$ are given in Figure 2.1, where each point corresponds to a distinct loading pattern.

With this representation, the problem of finding an optimum loading pattern may be described as finding the point, or the loading pattern, on a hyper-cube with N_F points on one side in a N_L -dimensional space.

In actual reactor core configurations, there are about 150-200 assembly locations and 3-10 assembly types. A typical size of design space amounts to approximately

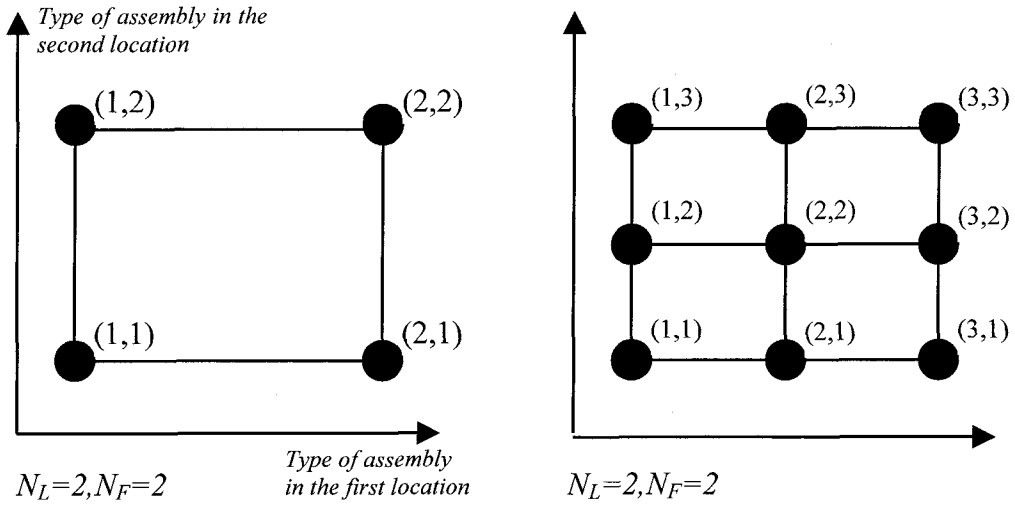


Figure 2.1 Loading pattern representation in two-dimensional configuration space

10^{20} - 10^{30} different loading patterns. With this consideration, calculating and comparing the fitness for all possible loading patterns one by one is impractical. For this purpose, optimization methods have been developed to overcome the problem. They try to find a near optimum from a large search space by calculating a limited set of them.

Optimum fuel management methods perform searches for the reloading pattern that exhibits a minimum power peaking factor, that is to say, the maximum NP_j , for any reloading pattern, must be as low as possible.

A minimum power peaking factor allows the reactor to operate at the maximum possible power density (Cullen et al, 1986). In case when the maximum power level is fixed, a minimum power peaking factor is the safest for operation, since the fuel assembly with the highest power remains well above the DNBR (*departure from nucleate boiling ratio*) or MCHFR (*minimum critical heat flux ratio*) limits.

There are two basic methods to reload cores: reloading to achieve maximum safety, and developing a low leakage core at the EOC (End of Cycle).

The first method is based on designing a core with the flattest possible power distribution. The loading scheme is also called the out-in type. The fresh fuel is first placed in the core periphery and in subsequent cycles moved into the core.

The Haling Principle is widely used for achieving a flat power distribution in the core. The method is briefly described in Appendix A.

The second method employs the in-out scheme to minimize fuel cost. First, the in-out scheme has the fresh fuel loaded into the central portion of the core and at the last cycle moved to the core periphery.

For a three-cycle configuration the reloading schemes are denoted as in-in-out, out-in-in, in-out-in, or out-in-out. Here, *out* refers to the peripheral locations and *in* to the central part of the core.

Fuel may be loaded either in annular zones or scattered with respect to their enrichments. In annular zone, loading each radial zone has a uniform enrichment.

Scatter-loading is the loading scheme that refers to where the fuel of high- k_{∞} 's are surrounded with low- k_{∞} fuel to prevent high-peak powers in the core.

In all of these strategies, the main problem is to determine the exact positions of the fuel assemblies. Although there are some guidelines that help refueling a reactor, there are no existing precise methods to construct an optimum-loading pattern. Presently, nuclear power plant cores are designed to achieve the flattest possible power distribution. To reach this goal, many studies are carried out to find optimum-loading patterns. The accumulated experience is used to find superior core configurations that meet the constraints better.

A typical PWR core initially has fuel assemblies with three different enrichments that are loaded in a mixed zone. The first reactor cycle is normally run for approximately one and one half year to receive the maximum possible burnup. Subsequent reactor cycles are designed to last at least one year. Approximately 1/3 of the core is removed after each reactor cycle, the exposed fuel moved to their new positions, and the new fuel loaded to the core periphery and other positions close to but not adjacent to the periphery.

There are some general rules for finding new core positions for the fuel assemblies. The guidelines enable the fuel management engineer to determine by iteration the accepted new loading configuration.

The first guessed core configuration rarely exhibits the appropriate NP_j 's, although the effective multiplication factor k_{eff} may be satisfactory.

The following is a sample from the list of rules of thumb gained by experience, for shuffling the fuel assemblies to improve the radial power distribution.

- New fuel assemblies should be loaded to the core periphery locations (out-in loading scheme).
- Scatter or checkerboard loadings should be preferred where possible (low- k_{∞} fuel adjacent to high- k_{∞} fuel).
- For places where both NP_j and k_{∞} are high, replacing the high- k_{∞} with a low- k_{∞} fuel assembly reduces the NP_j in this region of the core. The low- k_{∞} should be brought from a position a few assemblies distant. Likewise, a high- k_{∞} increases the NP_j 's in the region where it replaces the low- k_{∞} fuel assembly.
- A fuel assembly should never be exchanged with one having a lower- k_{∞} and a higher NP_j .
- The core may be configured to have one-quarter core symmetry. But, at a cost of higher NP_j 's, a 1/8 core symmetry may be also constructed.
- New fuel assemblies that are located in the inner regions should be moved one position at a time. Otherwise, their large k_{∞} produces large effects in the power distribution.

When a satisfactory core configuration is determined, a nuclear burnup code is used to deplete the core. The NP_j 's and k_{eff} must satisfy certain core performance criteria as a function of depletion during the reactor cycle. Calculated maximum NP_j must be less than a reference power fraction value not to exceed fuel temperature, DNBR and LOCA safety requirements. The effective multiplication factor of the core should be equal or greater than unity at EOC with the reactor operating at full power.

If the loading pattern does not comply with these conditions, it is modified until an acceptable loading is achieved. Continuing fuel assembly shuffling or lowering the enrichment of the new fuel may flatten the power distribution. In the case of lower

enrichment, it must be ensured that the enrichment is not too low for the subsequent cycles.

It is possible to find a loading pattern that complies with all the restrictions with a low enrichment fuel profile, but, this configuration may cause severe flux peaking in the subsequent cycles. To prevent this condition, several cycles beyond the subsequent cycle should be analyzed. Sometimes, the energy requirement of the core is not well defined. This is the case when the analyses are carried out many months prior to the reactor operation. Then, the analyses are performed for a range of powers for the core. This is called the exposure window. Performing safety analyses for a range of powers prevents any delay in the operating schedule.

2.1 Comparing Refueling Schemes

Refueling schemes are compared considering safety and economics of the resulting loading patterns. If two schemes are safe and reliable then the one that conserves U-235 is the most economical.

In general, the utility does not change the mechanical design of a power reactor. Poison control, enrichment and reloading patterns are all of the options available to them.

Once the energy requirement is met, the total number of fissions to produce this energy is determined. The initial mechanical design also fixes the plutonium buildup and the U-235 depletion for the cycle.

The U-235 amount to sustain the criticality can be minimized by using a core that has low-leakage at the EOC. Therefore, if a core design with reliable operation characteristics has a low leakage at the EOC, it is rated as the most economical one. However, if the safety is in doubt, flatter power distribution is chosen.

Using control rods to shape the power distribution is an alternative method to have a flat power distribution. But, this introduces extra complications. If the cost advantage in fuel cycle economics is small, a scheme that does not use inserted control rods should be chosen.

Usually, burnable poisons are used only in the first cycle. However, if EOC-low leakage cores are to be used in the future refueling operations, burnable poisons have to be used in these cycles to prevent excessive power peaking in the core.

2.2 Optimization

The nuclear engineer responsible for refueling a nuclear power plant uses established procedures to determine the loading pattern for a core at the end of reactor cycle. Experience and use of trial and error calculations allow him to determine the reloading pattern. The engineer attempts intuitively to optimize the reload pattern to minimize the energy costs while simultaneously assuring adequate margin to safety and operational limits. Analytical optimization methods are not presently used to automatically finalize the reload configuration, although several optimization methods have been developed over the past, which provide guidance.

The operation of a reactor over many core cycles occurs in a stage wise fashion. After each reactor cycle is complete, the fuel engineer makes some decisions. A decision contains components or variables, which are conditions that can be changed as desired.

During the optimization process, the fuel manager has the following variables, or components of decision vector, when choosing the loading pattern (the term decision vector is used to identify the decision and its various components assigned at each stage or cycle of the core history):

- the enrichment of fresh fuel,
- the quantity of fresh fuel (number of new fuel assemblies),
- the reuse of used fuel,
- the length of a reactor cycle (the residence time of each fuel batch in the reactor),
- the detailed reloading pattern,
- poison management and

- altering the mechanical design of the fuel assembly.

On the other hand, there are constraints imposed on the operation of the reactor that the fuel manager should take into account. The following is a list of some of the constraints:

1. The reactor must be capable of sustaining a chain reaction to meet the energy generation plan.
2. Fuel temperature must remain under melting point.
3. Surface heat flux should not exceed its cladding minimum critical heat flux ratio (MCHFR) in a BWR (Boiling-Water Reactor) or departure from nucleate boiling ratio (DNBR) in a PWR.
4. Reactor control system must always be capable of shutting the reactor down, even with the most reactive control rod stuck in the full out position (it is called stuck-rod criterion for a BWR to limit regional reactivity).
5. Local fuel burnup must not exceed the maximum permissible burnup to avoid cladding failure.

For full power operation, an analysis in two dimensions also defines the following parameters:

- axial nuclear hot channel factor,
- local nuclear peaking factor and
- engineering hot channel factor.

In the case of two dimensions, the constraints in item 2 and 3 can be replaced by a limit on the radial power peaking factor of a fuel assembly. The constraint given in item 5 can also be replaced by a limit on the average burnup of a fuel assembly.

3. REVIEW OF OPTIMIZATION METHODS

There are many methods used in in-core fuel management optimization to find an optimum loading pattern. Here, a few of the methods are summarized with their general outlines. The methods that are presented here are dynamic programming, heuristic search, simulated annealing, direct search, binary exchange, genetic algorithms and hybrid search.

3.1 Dynamic Programming

This method was developed by Bellman (1957) to optimize multistage processes such as refueling nuclear power plants. In the case of nuclear power plant refueling, one stage consists of a reloading followed by a period of reactor operation.

Dynamic programming is well suited to the discontinuous nature of fuel management (Stover and Sesonske, 1969). The fuel depletion equations and decision functions, both of which are discontinuous at the loading points, can readily be adapted to the transformation of the state variables from stage to stage by means of the search technique in dynamic programming. The optimal policy obtained by dynamic programming is also the global optimum, and constraints are easily handled since the optimum policy obtained automatically satisfies the constraints on the state variable.

Computer storage, however, is the most serious difficulty with dynamic programming since storage requirements increase exponentially with the dimension of the state variable. In order to store a very modest 10 values for each component of a n -component state vector, 10^n storage locations are needed.

In core management optimization, the stage, which is defined as the time during which the state variables are transformed, is the period of full power operation of the nuclear reactor (Stover, 1968). During this period, the state variable (U-235 mass) is transformed according to the reactor diffusion and nuclide depletion equations.

3.2 Heuristic Search

Heuristic search methods are part of a discipline known as the artificial intelligence (AI) (Galperin and Nissan, 1988). In this method, the heuristic knowledge of an expert, formulated in the form of rules, is applied to advise the user of recommended fuel shuffling moves and possible violations of power peaking constraints.

The heuristic knowledge of an expert is expressed in the form of “IF-THEN” production rules, where production is defined as a statement consisting of two parts: a condition and an action. The rules serve to reduce the search space, guiding the search to a set of acceptable solutions of the problem. At the final stage, one or more solutions are selected by an algorithmic or heuristic approach, depending on the number of patterns obtained. The heuristic selector may be an evaluator that assigns performance score values to each solution.

If properly designed, such a system can perform the task of a fuel manager expert, based on the best practice of a given expert.

The most general formulation of heuristic problem solving is defined as a state-space search. Here, state is a piece of information representing a certain stage of a solution, for example, a partially loaded core configuration.

The search technique is chosen depending on the factors associated with the specifics of the application domain. The following is a list to account for a search for the optimal fuel configuration.

- The possible configuration number is very large. It represents a large-scale problem.
- There exists more than one solution. Actually, a relatively large number of solutions is often feasible, with performance parameter values contained within a narrow range.
- Evaluation of the partial solutions or comparison of partially loaded cores is very difficult and has a high level of uncertainty.

- By its nature, generation of configurations is a forward oriented search. The construction starts from the initial state (empty core) and proceeds to the goal state through a number of partially loaded core states.

A core configuration is generated from scratch, instead of performing variations of a first-guess solution.

All of these considerations suggest a program conception and design with a depth-first search strategy. The search space is described by a tree structure in which each node represents a specific “fuel assembly-core position” pair. The tree is expanded by increasing its depth for each of the nodes at the last tree level.

The main idea of the method is to solve the problem of combinatorial explosion by limiting the search space by applying heuristic rules. During the progress, the efficiency and feasible execution times versus completeness of the final set of solutions are considered.

There are three conceptual components in the system designed to perform a heuristically guided search process: control strategy, set of heuristic rules, and one or more databases that contain necessary information on the physical system under consideration.

Heuristic rules are contained in a stable knowledge base. They eliminate forbidden partial solutions and prune the number of permitted solutions by recommending the near optimal choices.

The physical description of the system that describes the reactor core geometry and properties of the available fuel assemblies is contained in a separate database.

The tree, each node of which is occupied by a Fuel ID-Core Position pair, represents an expanding solution space. Collection of all permitted nodes for a specific fuel assembly group (ID) forms a single level of the tree. Permitted nodes are those that are not eliminated by the rule set.

The flow of the search procedure as implemented by **Galperin and Nissan (1988)** is given below:

1. The current fuel assembly that is to be loaded next is chosen from the list of assemblies.
2. Recursive descent on the existing tree of partial solutions is carried out until all nodes at the deepest level are visited. For each of these nodes, the current configuration, i.e. path to the node, is collected, which is necessary to apply the rules.
3. The level list for the current ID is created. The rules from rule set 1 are applied at this stage, excluding from the list all forbidden pairs.
4. The level list of pairs is linked to the current node, thus expanding the solution space tree. Such an expansion is performed for each node of the deepest level and fully expanded tree is returned with its depth increased.
5. Steps 1 through 4 are repeated until all available fuel assemblies are loaded. The final tree with a number of levels equal to the number of fuel assembly groups embodies the final solution tree. The set of final configurations is obtained as a set of paths to each of these nodes.

The rules used were:

1. Loading of fresh fuel into the innermost core positions is forbidden in order to avoid excessive power peaking at the core center.
2. Loading of fresh fuel into a core position adjacent to another fresh fuel is forbidden, unless one of the positions is at the core periphery, in order to avoid local power peaking.
3. Loading of twice-burned fuel into the core peripheral positions, which have the potential to result in too low a power density, is forbidden, in order to avoid very low local densities.
4. Loading of twice-burned fuel into the core position adjacent to another twice-burned fuel is forbidden, if the distance from the core center to be considered position exceeds a certain value, in order to avoid local power dips.

5. From the list of all permitted pairs, choose those that tend to place the fuel with the highest burnup toward the core periphery, in order to assign a higher preference value to configurations with potentially lower leakage.

The method assumes a prior knowledge of the properties of the desired solutions, in the form of encoded rules. Therefore, the success of the method heavily depends on the qualification of the expert.

3.3 Simulated Annealing

This optimization method is based on the crystal vibrations during the metal annealing process (Yamamoto, 1997). When molten metal is cooled slowly, the energy state of a crystal in the solid metal becomes lower than that of quickly cooled metal.

Simulated annealing is based on the hill climb method, which only accepts a higher fitness solution than the current one. But, a probabilistic state transition to a worse direction in the design space allows simulated annealing to escape from local optima.

This property is presented with the following example. If a solution X_{i+1} is found while searching the neighborhood of a solution X_i , and if the evaluated fitness value of solution X_{i+1} is higher than that of X_i , X_{i+1} is adopted and the search continues around the neighborhood of X_{i+1} . On the other hand, if evaluated fitness value of X_{i+1} is lower than that of X_i , solution X_{i+1} is eliminated in the simple hill climb method. However, the solution X_{i+1} has a chance to survive in simulated annealing. The probability of survival is defined as

$$p = e^{-\delta C/T} \quad (3.1)$$

where δC is the difference in evaluated fitness value between X_i and X_{i+1} , and T is the temperature of the system.

In the beginning of the simulated annealing process, T is set to a large value like in a real metal annealing. If T is large enough, the acceptance probability of solutions is almost unity, in other words, the method behaves like the random search. As the

simulated annealing method optimization search progresses, T gets lower and the acceptance probability of a poor solution becomes smaller.

This property prevents the simulated annealing method from getting trapped in local optima in the primary stage of optimization. As the process continues, the method starts a fine local search by rejecting solutions with low fitness values.

The speed of the cooling determines the level of success. For the case of a slowly decreasing T , the computing time increases, however, better solutions are obtained. A rapidly decreasing T results in short computing times and poorer solutions.

The method is implemented as follows.

1. An initial loading pattern is chosen. The fitness value of the loading pattern is calculated using a core calculation code.
2. Then, a new loading pattern is generated from the initial one by random shuffling of fuel assemblies, and the fitness value is calculated.
3. If the fitness value of the new loading pattern is higher than that of the base one, the new loading pattern is adopted as the new base-loading pattern. If the fitness value is less than that of the base loading pattern, the new loading pattern is accepted with a probability of $\exp(-\delta C/T)$, Eq. (3.1).
4. The system temperature is decreased according to $T^{n+1} = \alpha T^n$, where n is the number of generation, and α is the annealing factor, a number less than 1. The system temperature decreases more slowly with α getting closer to unity.
5. The process restarts from step 2 until the frequency of change in the base loading pattern or the temperature is less than a given number.

The method requires a relatively long time to find a solution. Besides, simulated annealing does not include any specifications on how to construct a loading pattern.

3.4 Direct Search

The direct search method is based on the same technique as the simulated annealing method, differing only in the poor loading pattern handling (**Suh and Levine, 1990**). The simulated annealing method accepts a loading pattern having a low fitness value than that of the base loading pattern. In the case of direct search, however, a loading pattern with a low fitness value is always rejected.

The direct search procedure is equivalent to simulated annealing with a lower limit system temperature T .

Direct search takes less computation time than simulated annealing. But, the procedure is unable to escape local optima, since it has no capability to treat the multi-modality in the design space.

3.5 Binary Exchange

Binary exchange method is one of the deterministic methods in loading pattern optimization (**Yamamoto, 1997**). The procedure is implemented as follows.

1. An initial (base) loading pattern is generated, randomly.
2. Perform all possible binary exchanges starting with the base loading pattern. Calculate the fitness values for all the generated patterns using a core calculation code.
3. Perform the best binary exchange that improves the fitness value. The generated loading pattern is the new base.
4. Steps 2 and 3 are repeated until a loading pattern complying with constraints is obtained.

To keep the number of loading patterns, which are generated by shuffling, as low as possible, the procedure is restricted to binary exchange only. For example, with ternary fuel shuffling the pattern number becomes quite large.

Binary exchange method guarantees a local optimum. In this case, the obtained loading pattern can not be improved by this method. On the other hand, the simulated annealing and direct search methods do not perform similar exhaustive searches, and a local or global optimum is not guaranteed. However, like binary exchange, direct search is unable to escape from local minima, due to its lack of capacity to treat the multi-modality in the design space.

Binary exchange performs the finest local search for the optimum loading patterns, when compared with simulated annealing and direct search methods.

3.6 Genetic Algorithms

Genetic algorithms are based on the governing principles of genetics and natural selection process of evolution (**Goldberg, 1989**). In this method, a candidate for the solution is represented by a digital chromosome, which has enough information to reproduce the original solution.

As an optimization method, genetic algorithm has an advantage in the global search capability, but does not perform well in fine local searching.

The method will be discussed in further detail in the following sections.

3.7 Hybrid Search

In the previous sections, some optimization methods were briefly described. The methods may be compared with respect to the computation time, the robustness against the multi-modality and global or local search capability.

A hybrid search combines the optimization methods to obtain a system that overcomes the weakness of each method alone, and improves the quality of the final solution without significant increase in computation time (**Yamamoto, 1997**).

Direct search and binary exchange may be used together. In this case, binary exchange follows direct search, using the best loading pattern generated by the method as the initial loading pattern.

Genetic algorithm and binary exchange may be grouped to obtain a second hybrid method. Here, binary exchange is executed using the best loading pattern obtained by genetic algorithm as an initial pattern.

Simulated annealing used together with binary exchange is another candidate system for the optimization problem.

Direct search and binary exchange are the candidates of the fine local search for the hybrid search strategy. The binary exchange can perform a finer local search than direct search, and generally, the computation time of binary exchange is shorter than direct search.

The hybrid search works in two steps. A global search method like genetic algorithm or simulated annealing generates a near optimum solution. Then, a local fine search method-like direct search or binary exchange uses the near optimum solution to obtain the final loading pattern.

There are many other methods applied to the problem with various success rates. However, all the methods must start by calculating the related core parameters to compare obtained loading patterns. In the next section, the core calculation methods are described, briefly.

4. CORE CALCULATIONS

In-core fuel management is concerned with

- the power and burnup distribution in the core that is used in the safety analysis to be performed later,
- the effective multiplication factor (k_{eff}),
- the isotopic inventory for the economic analysis (**Levine, 1982**).

Reactor designs differ extensively in their fuel assembly types, but, it is still possible to use similar procedures to analyze them. For the purpose of this work, thermal light-water reactors are considered namely PWR's and BWR's.

A nuclear power plant uses a fixed number of fuel assemblies N_{FA} to produce a thermal output of P_t . Most of the power ($\sim 95\%$) is generated by the slowing down of the fission products in the fuel rods. The slowing down of the fast neutrons in the water, gamma and beta radiation interactions in the core produce the rest of the power.

The total thermal power P_t produced by the core is the sum of the thermal power P generated in the fuel assemblies. P varies according to the position of the fuel assembly in the core and the fuel and poison content of the fuel rods.

The maximum power density q'''_{max} and maximum linear heat generation rate q'_{max} are limited so that the power production of any given fuel rod does not exceed any of the thermal safety limits. Departure from nucleate boiling (DNB) and fuel melting are two examples for safety constraints.

Power density distribution $q'''(r)$, which is the power produced per unit volume of the core at position r , is estimated to evaluate the performance of a given loading

pattern. Safety parameters like temperature distribution in all over the core, hot channel factors and isotopic inventory are calculated directly from $q'''(r)$.

The power produced by a fuel assembly with volume V_{FA} as

$$P = \int_{V_{FA}} q'''(\underline{r}) dV. \quad (4.1)$$

A fuel assembly is the minimum unit that is used in in-core fuel management activities. It has a unique identity and is moved as single component throughout the reactor core.

The isotopic content of a fuel assembly is the main consideration during the optimization operations. Starting with the known properties, the power history of the assembly is used to determine the isotopic content at any given time. The inventory information is the main input for economic evaluations.

The normalized power or power fraction for the fuel assembly in location j is given as

$$NP_j = \frac{P_j}{\bar{P}}, \quad (4.2)$$

where P_j is the power produced by the j th fuel-assembly and $\bar{P}$ is the average power produced per fuel assembly in the core,

$$\bar{P} = \frac{1}{N_{FA}} \sum_{j=1}^{N_{FA}} P_j. \quad (4.3)$$

$\bar{P}$ may also be expressed in terms of thermal output P_t :

$$\bar{P} = \frac{P_t}{N_{FA}}. \quad (4.4)$$

Then, P_j may be calculated as

$$P_j = NP_j \bar{P}. \quad (4.5)$$

These basic definitions are needed for the in-core fuel management procedures. The optimization analysis requires the power fractions NP_j be determined for every fuel assembly.

As a first approximation for the calculation of NP_j , the one group neutron theory may be used. It defines the power density for the fuel assembly j as

$$q_j''' = G \bar{\Sigma}_{fj} \bar{\Phi}_j, \quad (4.6)$$

where

G is the average energy released per fission that is absorbed in the fuel assembly including the moderator,

$\bar{\Sigma}_{fj}$ is the macroscopic thermal fission cross section averaged over the j th fuel assembly,

$\bar{\Phi}_j$ is the average thermal flux in the j th fuel assembly.

Then the power produced by the j th fuel assembly may be rewritten as in Eqs (4.7) and (4.8).

$$P_j = q_j''' V_{FA} \quad (4.7)$$

$$P_j = G \bar{\Sigma}_{fj} \bar{\Phi}_j V_{FA} \quad (4.8)$$

It follows from Eqs. (4.8) and (4.4) that the absolute value of thermal flux in the assembly may be calculated as

$$\bar{\Phi}_j = \frac{NP_j \bar{P}}{G \bar{\Sigma}_{fj} V_{FA}}. \quad (4.9)$$

Substituting Eq. (4.8) into Eq. (4.3) yields

$$\bar{P} = \frac{\sum_{j=1}^{N_{FA}} G \bar{\Sigma}_{fj} \bar{\Phi}_j V_{FA}}{N_{FA}}. \quad (4.10)$$

Using Eqs. (4.2), (4.8) and (4.10) Eq. (4.11) is obtained.

$$NP_j = \frac{N_{FA} \bar{\Sigma}_{fj} \bar{\Phi}_j}{\sum_{j=1}^{N_{FA}} \bar{\Sigma}_{fj} \bar{\Phi}_j}. \quad (4.11)$$

This simple approximation states that NP_j is a function of nuclear parameters $\bar{\Sigma}_{fj}$ and $\bar{\Phi}_j$ only. It is also seen that the relative values of $\bar{\Phi}_j$ can be used. The neutron diffusion codes calculate this parameter relative to other fuel assemblies.

4.1 Cross Sections

The calculation of NP_j requires the macroscopic cross sections of the materials that compose the fuel assembly (like $\bar{\Sigma}_{fj}$ in Eq. 4.11). Diffusion method calculations that estimates the power distribution are based on the assumption of material homogeneity across the assembly. For the purpose of this work, the calculation of the cross sections and other group constants is performed by PSU-LEOPARD code (Petrovic, 1990).

LEOPARD assumes an infinite array of identical unit cells representing the fuel assembly. Each unit cell consists of a fuel pin and cladding surrounded by a moderator region obtained by homogenizing the actual fuel assembly. The non-lattice part of the fuel assembly is introduced into the unit cell as an “extra” region surrounding the other three regions.

Examples for non-lattice parts are the control rod thimbles, the instrumentation cell, and the fuel assembly interspatial water region. The unit cell that is composed of these four regions is called a supercell (Figure 4.1).

LEOPARD calculates flux weighted macroscopic cross sections for the fictitious supercell. The obtained cross section set consists of one macroscopic cross section for each of the cross section types. Actually, there are two or four sets of values depending on the number of energy groups the calculation is performed for.

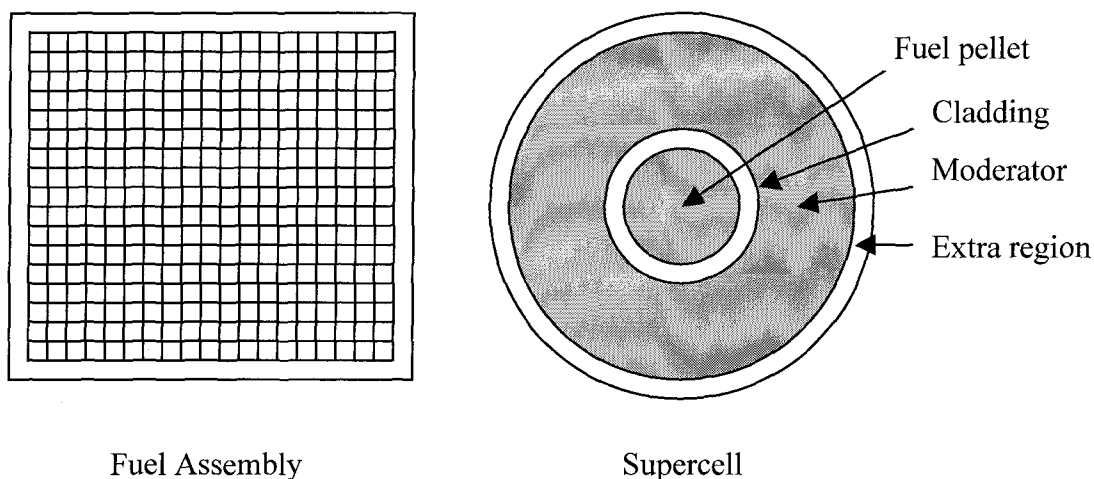


Figure 4.1 A supercell representation of a fuel assembly

Later, the macroscopic cross-sections generated by the LEOPARD code are used to calculate $\bar{\Phi}_j$ and, finally, NP_j .

4.2 Global Core Calculations

Global core analysis consists of calculating the actual reactor core power distribution, or NP_j , using the information obtained in the cross section generation step. For the purpose of this work, MCRAC diffusion code was used (Petrovic, 1990).

MCRAC is a fast, two-dimensional, two-group diffusion code, suitable for rough and preliminary core analysis. It uses cross sections in the form of polynomial functions of burnup and soluble boron concentration, as generated by the PSU-LEOPARD code.

The reactor core is modeled in either x - y or in r - z geometry. A rectangular mesh representation of the core is used for the calculations. The mesh points are located at the interval edges. In the x - y geometry, the core height is simulated through axial buckling.

One quarter or octant of the core is modeled. While zero current or reflective flux symmetry is assumed at the boundaries facing the rest of the core, zero flux is the boundary condition for the outer perimeter.

In the r - z geometry, mesh regions represent toroidal volumes. One-half of the core is modeled.

MCRAC solves the multi-group diffusion equation (Lamarsh, 1972),

$$\begin{aligned}
 & -\nabla \cdot D_j(\underline{r}) \nabla \Phi_j(\underline{r}) + \Sigma_{Rj}(\underline{r}) \Phi_j(\underline{r}) \\
 & = \sum_{\substack{i \\ i \neq j}}^{N_g} \Sigma_{s,ij}(\underline{r}) \Phi_i(\underline{r}) + \frac{\chi_g}{k_{eff}} \sum_i^{N_g} \nu_i \Sigma_{fi}(\underline{r}) \Phi_i(\underline{r}), \quad j = 1, 2, \dots, N_g
 \end{aligned} \tag{4.12}$$

where

N_g is the total number of energy groups,

j, i are the energy group indexes, $j, i \in \{1, 2, \dots, N_g\}$,

$\underline{r}$ is the space variable of (x, y) or (r, z) (the space dependence is omitted in the following text),

Φ_j is the neutron flux in group j ,

D_j is the diffusion coefficient in the group j ,

k_{eff} is the effective multiplication factor of the system,

χ_j is the neutron fission spectrum, or fraction of fission neutrons that are born in the group j ,

$\nu_i \Sigma_{fi}$ is the macroscopic fission cross section of group i times the average number of neutrons produced per fission (group fission constant),

Σ_{Rj} is the total macroscopic removal cross section from the group j , which is given by

$$\Sigma_{Rj} = \Sigma_{aj} + \delta D_j B_j^2 + \sum_{\substack{i \\ i \neq j}}^{N_g} \Sigma_{s,ji} \quad . \tag{4.13}$$

Here, Σ_{aj} is the macroscopic absorption cross section of the group j , δ is the geometry index (1 for x - y and 0 for r - z geometry), B_j^2 is the axial buckling for the group j , $\Sigma_{s,ji}$ is the macroscopic scattering cross section from group j to group i .

The two-group energy structure is used, where group one ($j=1$) is the fast-neutron energy group and group two ($j=2$) is the thermal neutron energy group. The cut-off energy between these two groups is defined in the cross sections information. In the case of LEOPARD, it is 0.625 eV.

The model assumes that there is no neutron upscattering from thermal group to the fast group. All neutrons are born in the fast energy group, or in terms of χ_j it is written as $\chi_1=1$ and $\chi_2=0$.

The multi-group diffusion equations (Eq. 4.12) reduces to the following for $N_g=2$ (two-group).

$$\nabla \cdot D_1 \nabla \Phi_1 - \Sigma_{R1} \Phi_1 + \frac{1}{k_{eff}} (\nu_1 \Sigma_{f1} \Phi_1 + \nu_2 \Sigma_{f2} \Phi_2) = 0 \quad (4.14)$$

$$\nabla \cdot D_2 \nabla \Phi_2 - \Sigma_{a2} \Phi_2 + \Sigma_{s,12} \Phi_1 = 0 \quad (4.15)$$

MCRAC solves Eqs. (4.14) and (4.15) in the finite difference approximation EQUIPOSE method (**Tobias and Fowler, 1962**). The implementation is based on the Exterminator-II code (**Fowler and Tobias, 1967**). MCRAC accelerates the calculation by the extrapolated Liebman overrelaxation.

The acceleration method uses expression

$$\Phi_j^{n-1}(\underline{r}) + \beta (\Phi_j^n(\underline{r}) - \Phi_j^{n-1}(\underline{r})) \quad (4.16)$$

instead of $\Phi_j^n(\underline{r})$, for iteration n .

Here, β is a constant usually given in the range $1.3 \leq \beta \leq 1.7$, and $\beta = 1$ corresponds to no overrelaxation.

The iteration process continues until the effective multiplication factor (k_{eff}) and pointwise flux convergence between the two successive iterations satisfy both of the following (ϵ is a small number).

$$\begin{aligned} \max_{(r)} \left| \frac{\Phi_j^n(r) - \Phi_j^{n-1}(r)}{\Phi_j^{n-1}(r)} \right| &< \epsilon, \\ \left| \frac{k_{eff}^n - k_{eff}^{n-1}}{k_{eff}^{n-1}} \right| &< \epsilon. \end{aligned} \quad (4.17)$$

Once cross sections and fluxes are determined, estimating NP_j is straightforward by using a relation like Eq. (4.11).

The 1.5 Group Model

MCRAC provides an alternative to the standard 2-group model. It is called the 1.5 group model, which assumes zero thermal neutron leakage between mesh volumes. The assumption is justified if the thermal neutron mean free path is several times smaller than the mesh interval dimension.

This approach corresponds to a coarse mesh configuration, it is to say, 1x1 or 2x2 mesh points per fuel assembly. In this case, the thermal diffusion Eq. (4.15) reduces to the following:

$$-\sum_{a2} \Phi_2 + \sum_{s,12} \Phi_1 = 0 \quad (4.18)$$

Eq. (4.18) may be solved for Φ_2 in terms of Φ_1 .

$$\Phi_2 = \frac{\sum_{s,12}}{\sum_{a2}} \Phi_1 \quad (4.19)$$

The result is substituted into Eq. (4.14) to obtain a equation expressed only in Φ_1 . The equation is solved by the EQUIPOISE method and Φ_2 is calculated by Eq. (4.19). No iteration is required between the two groups and the computing time is reduced remarkably. But, some additional error is introduced into the calculations.

Criticality Search

MCRAC determines the concentration of soluble boron to make the reactor critical, that is to say $k_{eff} = 1$, at each depletion step. The critical soluble boron search stops when $|k_{eff} - 1.0| < \epsilon_k$, which is a small number fixed at 0.0003. In most practical cases, this ensures that the soluble boron concentration is converged to the critical one within 5 ppm or less.

To speed up the computation for the critical boron search, the convergence criterion is increased by an order of magnitude.

If the criticality search is not activated, MCRAC calculates k_{eff} for each step for the given soluble boron concentration.

Power Dependent Xenon Calculations

LEOPARD calculates and adds the xenon contribution resulting from the equilibrium xenon concentration. However, equilibrium xenon concentration is flux dependent. Xenon has a significant thermal absorption cross-section, thus slight variations in xenon concentration causes remarkable deviation in flux distribution. LEOPARD calculates the cross sections using the core average power density. Fuel assemblies with different power level from the average require the correct xenon contribution in the cross sections.

This is solved by a feedback model that is used to link LEOPARD and MCRAC.

5. ARTIFICIAL NEURAL NETWORKS

5.1 Introduction

Artificial neural networks are physical cellular systems that can acquire, store, and process experiential knowledge. The knowledge is in the form of stable states embedded in the neurons of the network. Despite its application to practical tasks is quite recent, the idea has been around for over 50 years.

In the year 1943, McCulloch and Pitts described the first formal model of a computing neuron (Zurada, 1990). The model was suitable for logic and arithmetic operations. But, the implementation was not feasible with the bulky vacuum tubes.

Donald Hebb proposed the first learning scheme for updating neuron's connections in 1949. The scheme called Hebbian learning rule, states that information can be stored in connections.

First computers implementing neural networks were built during the 1950s. Frank Rosenblatt designed the Perceptron in 1958, which was a trainable machine that can classify patterns.

In the early 1960s a neural network architecture called ADALINE (for ADaptive LINEar combiner) was invented and Bernard Widrow and Marcian Hoff developed the Widrow-Hoff learning rule that minimizes the summed square error during training involving pattern classification. ADALINE was applied to pattern recognition, weather forecasting, and adaptive control.

Soon, after the mid-1960s, researchers realized the weaknesses of the available algorithms to solve more complex computational problems. Despite the existence of layered networks, the lack of efficient learning algorithms prevented any significant progress in the area. Neural network research entered a stagnation phase.

In 1986, James McClelland and David Rumelhart published a two-volume book on parallel distributed processing. The new learning rules introduced in this work initiated many new neural network research programs.

Today neural networks are applied to practical tasks. In addition to the computer implementations, large-scale integrated neural network chips have been fabricated.

5.2 Neural Networks

The neural network simulates the computational flexibility and power of the human brain (**Kosko, 1992**). This computation is performed by a set of computing nodes and connections between them. Data is fed as input to the nodes or neurons, the basic processing elements. The biological neuron and its artificial counterpart are shown in Figure 5.1.

The biological neuron consists mainly of 4 parts: cell body, dendrites, axons and synapses (**Al-Gutifan and Faisal, 1991**). The nucleus of the neuron receives its input through the dendrites from other neurons communicating by the synapses. The input is processed in the cell membrane and sent to other neurons over the axons. The learning is represented by the adjustment of the synapse properties. The synapses route incoming data to neurons that has established a connection during the learning process.

Similarly, the artificial neuron has 4 main parts: input paths, the transfer function, output paths and connection weights. A neuron sends a signal through its output paths (axon) to the input paths (dendrites) of other neurons. Then, the neuron sums the inputs and the result is modified with the transfer function and sent to other neurons in the succeeding layers.

There are many kinds of neural networks. For instance, they are classified with respect to the number of layers the neurons are arranged, or some networks provide instantaneous responses while others, called dynamic networks, need time to respond. Neural networks are also classified according to their learning methods.

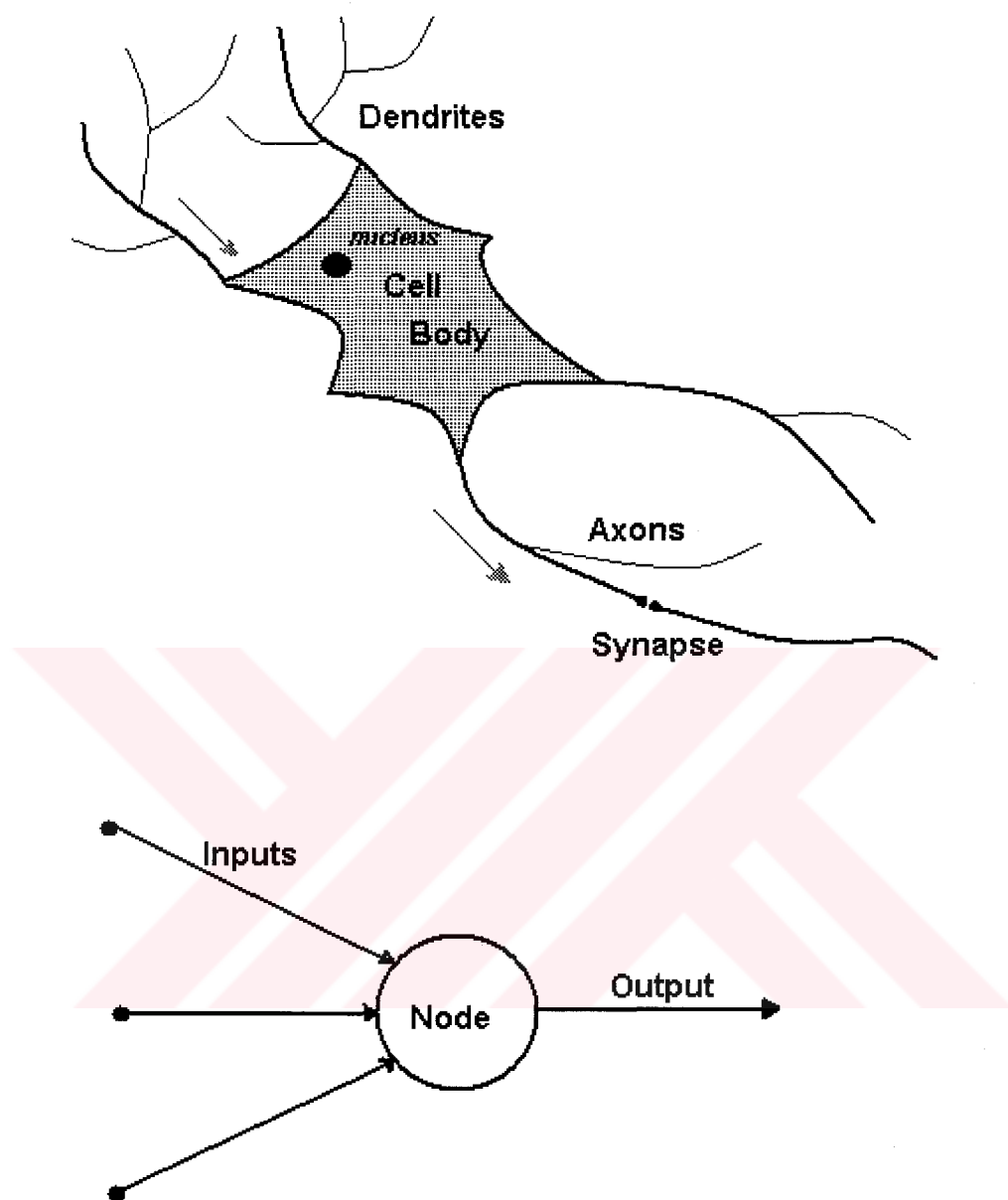


Figure 5.1 The biological neuron and its artificial model

5.3 The Back-propagation Algorithm

The multi-layered feed-forward neural networks waited for an appropriate learning algorithm for their application for more than a quarter of a century (**Hertz et al.**,

1991). The development of the back-propagation algorithm provided the means that was required.

This kind of networks can learn a mapping of any complexity by repeatedly presenting of the training samples. The tasks that can be performed by these networks include classification, function approximation, hand-written character recognition and expert systems.

The back-propagation network has at least three layers of neurons that connect the input to the output world (Figure 5.2). The input layer is connected to the hidden layer and the hidden layer to the output layer. Each layer is fully or partially connected to the succeeding layer. The input layer neuron has a single input connection from the outside world and transfers the signal without any modification. The neurons of the hidden and output layers perform a sum-up operation on its input signals and the transfer or activation function is applied. The transfer function simulates the biological threshold property of the neuron, which allows signals with specified magnitudes to pass from neuron to neuron. Each connection between neurons has a property called weight that is used to modify the transferred signal by multiplying it.

The network produces meaningful outputs for a specified problem only when the connection weights and threshold values are determined. A learning algorithm is required to estimate these values. Back-propagation learning algorithm determines these parameters by repeatedly applying sample patterns to the network input and output layers sequentially or randomly during the training. A pattern is submitted to the input and the weights and thresholds are adjusted so that the least mean square error is reduced. The error is a function of the difference of the expected output and the output obtained from the network. This process is continued until all mapping samples from the training set are learned within an acceptable overall error.

The neural network operates in a feed-forward manner when a set of signals is applied to the input. However, the weight adjustment propagates from the output layer through the hidden layers toward the input layer.

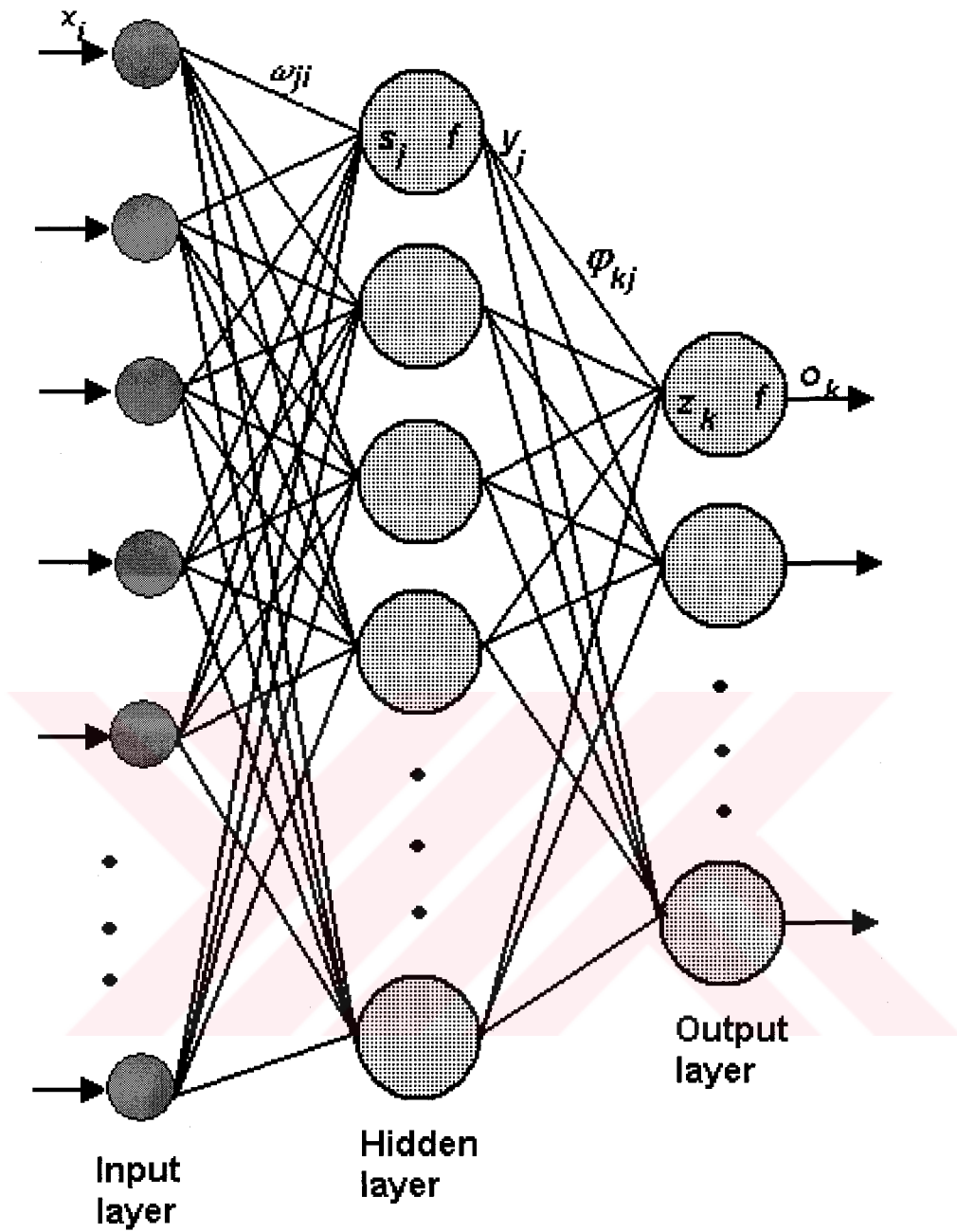


Figure 5.2 A Feed-forward neural network with one hidden layer.

5.3.1 Generating Output

Each of the hidden layer nodes accepts input from the input layer nodes. For a given input vector x the hidden layer node output y is calculated from Eq. (5.1a) and (5.1b) as follows (Figure 5.3):

$$s_j = \sum_i \omega_{ji} x_i \quad (5.1a)$$

$$y_j = f(s_j) \quad (5.1b)$$

where i is the i th node at the input layer, j is the j th node at the hidden layer, x_i is the i th component of the input vector, ω_{ji} is the weight between the i th and the j th nodes, s_j is the net weighted sum and f is the activation function (Figure 5.2).

The activation function is usually given as

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \quad (5.2)$$

and called a sigmoidal function. The function is shown in Figure 5.4. In Eq. (5.2), λ is proportional to the neuron gain determining the steepness of the function at the origin.

Similarly, the output from the output nodes is calculated from the hidden layer nodes. The non-linear f function is applied to all hidden layer outputs.

The output o of the output layer is given by

$$z_k = \sum_j \phi_{kj} y_j \quad (5.3a)$$

$$o_k = f(z_k) \quad (5.3b)$$

where k is the k th node of the output layer and z_k is the net weighted sum.

5.3.2 Training

For the network to produce a meaningful output the weights ω_{ji} and ϕ_{kj} should be specified for a given problem. The back-propagation algorithm, or also known as the generalized delta rule, is a gradient descent method that calculates the weights by modifying them to minimize the network error.

The error function E is given as

$$E = \frac{1}{2} \sum_k (t_k - o_k)^2 \tag{5.4}$$

Where t_k is the sample or target (desired) and o_k is the generated outputs at node k of the output layer.

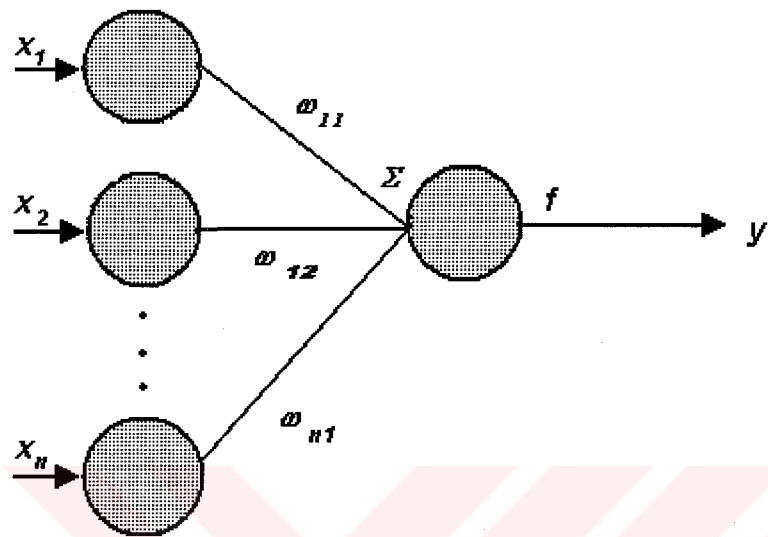


Figure 5.3 Output from a single node.

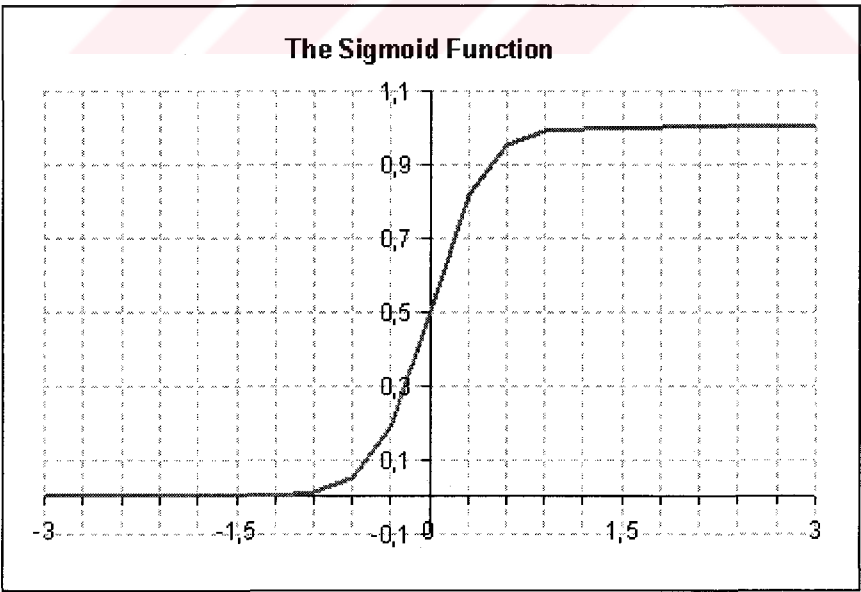


Figure 5.4 The sigmoid function.

The weight adjustment between the hidden and output layer is made with

$$\Delta\varphi_{kj} = -\eta \frac{\partial E}{\partial\varphi_{kj}} \quad (5.5)$$

where η is a positive constant. The error signal term δ_o for the k th neuron of the output layer is defined as

$$\delta_{ok} = -\frac{\partial E}{\partial z_k} \quad (5.6)$$

where z_k is given in Eq. (5.3a). Since, the component $\partial E/\partial\varphi_{kj}$ depends only on z_k by using the chain rule it may be written as

$$\frac{\partial E}{\partial\varphi_{kj}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial\varphi_{kj}} \quad (5.7)$$

From Eq. (1.3a) with constant y_j it follows that the second term is

$$\frac{\partial z_k}{\partial\varphi_{kj}} = y_j \quad (5.8)$$

Then, combining Eqs. (5.6) and (5.8) yields

$$\frac{\partial E}{\partial\varphi_{kj}} = -\delta_{ok} y_j \quad (5.9)$$

The weight adjustment Eq. (5.5) is written as

$$\Delta\varphi_{kj} = \eta \delta_{ok} y_j \quad (5.10)$$

To obtain an expression for the error signal term it is written as

$$\delta_{ok} = -\frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial z_k} \quad (5.11)$$

For the first term it is obvious that

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \quad (5.12)$$

From Eq. (5.3b) it follows that

$$f'_k(z_k) = \frac{\partial o_k}{\partial z_k} \quad (5.13)$$

Combining Eqs. (5.11), (5.12) and (5.13) one obtains for the error signal term

$$\delta_{ok} = (t_k - o_k) f'_k(z_k) \quad (5.14)$$

Finally, weight adjustment Eq. (5.10) may be rewritten as

$$\Delta \phi_{kj} = \eta (t_k - o_k) f'_k(z_k) y_j \quad (5.15)$$

A similar derivation leads to the weight adjustment for any layer of neurons that is not an output layer. For the three-layer network shown in Figure 5.2 the gradient descent formula is given as

$$\Delta \omega_{ji} = -\eta \frac{\partial E}{\partial \omega_{ji}} \quad (5.16)$$

Using the chain rule with s_j defined in Eq. (5.1a) one obtains Eq. (5.17).

$$\frac{\partial E}{\partial \omega_{ji}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial \omega_{ji}} \quad (5.17)$$

Since, the second term $\partial s_j / \partial \omega_{ji}$ is equal to x_i , the input vector, and with the error signal term

$$\delta_{yj} = -\frac{\partial E}{\partial s_j} \quad (5.18)$$

Eq. (5.16) may be rewritten as

$$\Delta \omega_{ji} = \eta \delta_{yj} x_i \quad (5.19)$$

The error signal term δ_{yj} can be computed by Eq. (5.20), obtained applying the chain rule.

$$\delta_{yj} = -\frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial s_j} \quad (5.20)$$

For the first term in Eq. (5.20) using

$$E = \frac{1}{2} \sum_k [t_k - f(z_k(y))]^2 \quad (5.21)$$

the following equation is derived.

$$\frac{\partial E}{\partial y_j} = -\sum_k (t_k - o_k) f'(z_k) \frac{\partial z_k}{\partial y_j} \quad (5.22)$$

Using Eq. (5.14) for δ_{ok} and Eq. (5.3a) for z_k , Eq. (5.22) becomes

$$\frac{\partial E}{\partial y_j} = -\sum_k \delta_{ok} \varphi_{kj} \quad (5.23)$$

The second term in Eq. (5.20) is calculated as

$$\frac{\partial y_j}{\partial s_j} = f'_j(s_j) \quad (5.24)$$

Combining Eqs. (5.23) and (5.24) one finds for δ_{yj}

$$\delta_{yj} = f'_j(s_j) \sum_k \delta_{ok} \varphi_{kj} \quad (5.25)$$

Finally, the weight adjustment Eq (5.16) becomes

$$\Delta \omega_{ji} = \eta f'_j(s_j) x_i \sum_k \delta_{ok} \varphi_{kj} \quad (5.26)$$

All sample input-output pairs are introduced one by one until the network error is reduced below a given value. The final φ and ω values are stored for later use.

The output values of the training samples should be normalized to be in the interval [0,1] or [-1,1], depending on the activation function.

5.3.3 Factors Affecting Learning

There are some important problems that affect back-propagation algorithm negatively. **Takeda and Goodman (1986)**, **Karnin (1990)** and **Brent (1991)** outline solutions. Local minima and network paralyse are two of them. The local minima problem is inherently contributed by the descent gradient method. The surface of the error function may have local minimums that prevent the algorithm to find the desired minimum. In this case, small weight changes are not sufficient to escape from the valley like false minimums.

A flat error function surface causes network paralysis due to a derivative that is close to zero. In this case, the derivative contributes little to weight changes, which slows down training.

The remedy for these problems requires considering the following factors:

- **Initial weights:** The initial values assigned to connection weights strongly affects the training, since this specifies the starting point for the solution. Typically, the weights are initialized with random small numbers. This is a simple solution for the network paralysis problem.
- **Learning constant:** The effectiveness of the back-propagation algorithm strongly depends on the learning constant η . Large η results in rapid convergence. But, in this case, there is the possibility of overshooting the solution. η value has to be determined experimentally for each problem.
- **Saturated nodes:** In the back-propagation, saturating nodes are treated individually to speed up learning.
- **Momentum method:** The momentum method helps the algorithm to accelerate the convergence and escape from local minima. This is achieved by adding a second term to Eq. (5.5), the weight adjustment equation. This second term is called the momentum term and consists of a fraction of the weight adjustment of the previous step. Eq. (5.5) is rewritten as

$$\Delta\varphi(t) = -\eta \nabla E(t) + \alpha \Delta\varphi(t-1) \quad (5.27)$$

where t and $t-1$ represents the current and most recent training steps and α is a constant usually chosen between 0.1 and 0.8.

- Number of hidden neurons: Despite of being one of the most important factor in the success of the algorithm the number of hidden neurons for a given problem has no conclusive answers yet. A complete analysis is quite difficult because of the complexity of the network mapping. However, there are codes, which optimize the network configuration.
- Adding a bias of the order of 0.1 to the derivative of the sigmoid activation function avoids the paralysis problem. Another approach is to use the derivative of the linear output instead of sigmoid output.



6. PREDICTING CORE POWER AND k_{eff} USING NEURAL NETWORK

6.1 The Neural Network Structure

In this work, the power distribution, which is required during the search for a suitable loading pattern, will be estimated more effectively and quickly by using a neural network. The neural network is trained to predict the generated power fraction by each fuel assembly. The fuel assembly locations and the effective multiplication factor are used as training input.

A fuel loading pattern is considered to be acceptable when its local power peaking factor is less than a given maximum power fraction value based on safety-limitations, and its effective multiplication factor k_{eff} is higher than a given minimum value. One of these loading patterns is selected as the optimum.

The Almaraz Nuclear Power Plant (NPP) core configuration and data has been used to demonstrate the technique (Ahnert et al., 1990).

Table 6.1 Fuel assemblies defined for Almaraz NPP

Fuel Assembly ID	Inventory (Quantity)	Enrichment (%)	Burnable Absorber Rods	k_{∞}
121	11	2.1	-	1.054
120	2	2.6	20	0.959
112	1	2.6	12	1.016
131	5	3.1	-	1.174
316	1	3.1	16	1.046
116	5	2.6	16	0.988
312	1	3.1	12	1.073

Almaraz NPP is a pressurized water reactor (PWR) that has 157 fuel assembly locations in its core. For this work, seven different assembly types were defined with respect to their uranium enrichment and burnable absorber content. A core-loading pattern was constructed by selecting a suitable configuration of these assemblies available from the stock. Table 6.1 lists the fuel assemblies that were defined for this particular case study.

Since, the NPP has a one-eighth symmetrical core, only 26 assembly locations have been used for the analysis. Figure 6.1 illustrates a sample core-loading pattern.

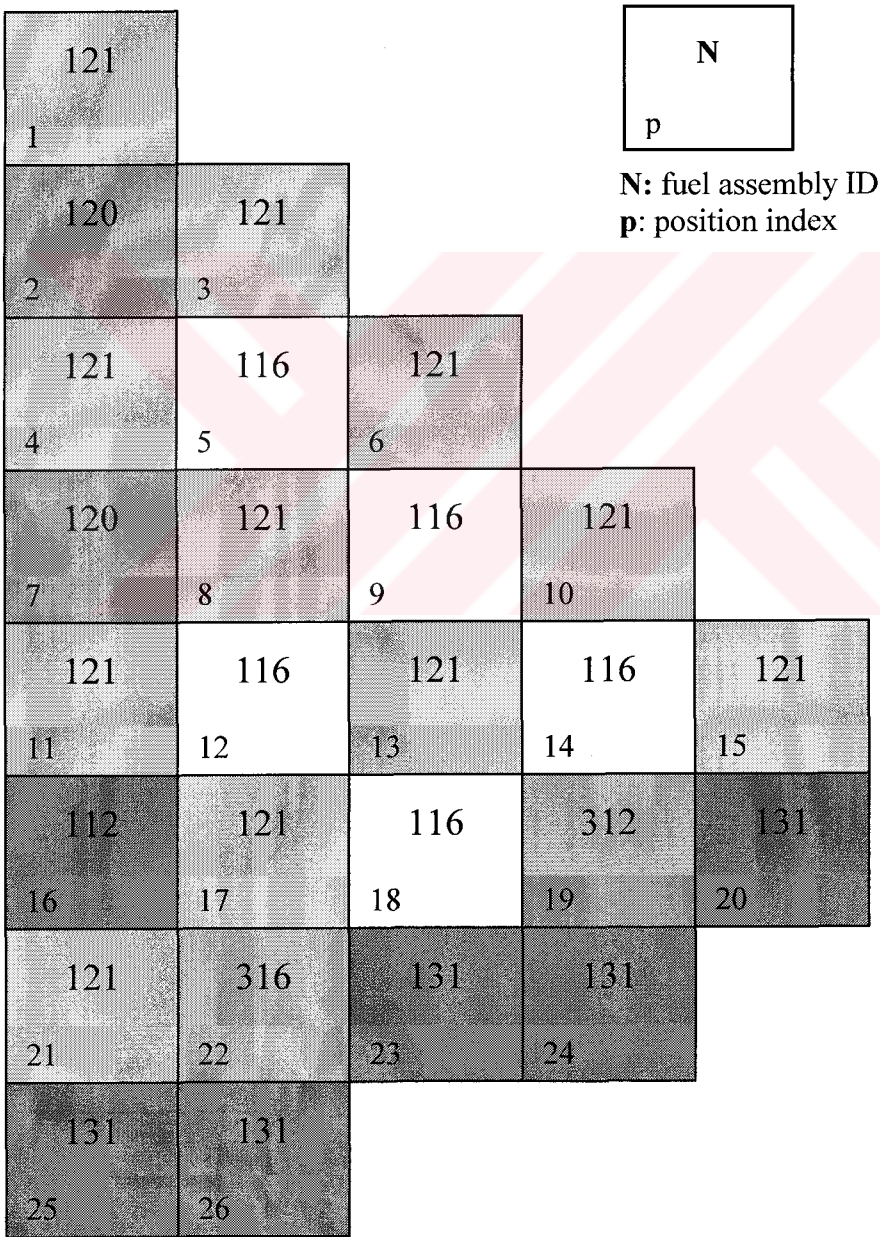


Figure 6.1 A sample of loading pattern for Almaraz NPP

All the calculations in this work were based on a two-dimensional geometry that is a radial cross-section of the core cylinder.

The neural network was designed to have 26 input nodes (plus a bias node), one for each assembly location, and 27 output nodes, one for each power fraction generated in the corresponding fuel assembly and one for the multiplication factor of the reactor (k_{eff}).

The number of hidden layer nodes was 250, which was determined by trial and error.

The reactor core was modeled for the MCRAC-LEOPARD in-core fuel management package to obtain the data required for the training of the neural network. Power distribution and effective multiplication factors were calculated for 2000 randomly generated loading patterns. 500 patterns were used for the training while the rest were stored for later usage to check the validity of the generated outputs of the neural network.

6.1.1 Training Patterns

Each one of the 2000 training pattern sets, which were generated by the in-core fuel management package, includes the fuel assembly types and their locations in the reactor core configuration, and the corresponding power fraction generated by the assembly and the effective multiplication factor for the loading pattern.

Table 6.2 Fuel assemblies and their encoded neural network values.

Fuel assembly ID	Neural network input value (β)
131	0.7
121	0.6
312	0.5
316	0.4
112	0.3
116	0.2
120	0.1

The elements of the input vector were indexed by the assembly location in the core as depicted in Figure 6.1, whereas the value of each element of the input vector associates with a particular fuel assembly composition as listed in Table 6.2. These input values may be assigned randomly, but in this implementation they were ordered with respect to their k_{∞} values.

For example, a loading pattern like the one given in Figure 6.1 may be mapped into a vector

$$[121 \ 120 \ 121 \ 121 \ 116 \ 121 \ 120 \ 121 \ 116 \ 121 \ 121 \ 116 \ 121 \ 116 \ 121 \ 112 \ 121 \ 116 \ 312 \ 131 \ 121 \ 316 \ 131 \ 131 \ 131 \ 131] \quad (6.1)$$

which with the help of Table 6.2 is encoded as:

$$x = [0.6, 0.1, 0.6, 0.6, 0.2, 0.6, 0.1, 0.6, 0.2, 0.6, 0.6, 0.2, 0.6, 0.2, 0.6, 0.3, 0.6, 0.2, 0.5, 0.7, 0.6, 0.4, 0.7, 0.7, 0.7, 0.7] \quad (6.2)$$

There are two requirements for the training patterns to be unique (Kim et al., 1993):

- Different core loading patterns must be represented with different vectors.
- The difference due to the location in the core must be taken into account.

Fulfilling these requirements guarantees a different input vector for every training pattern generated.

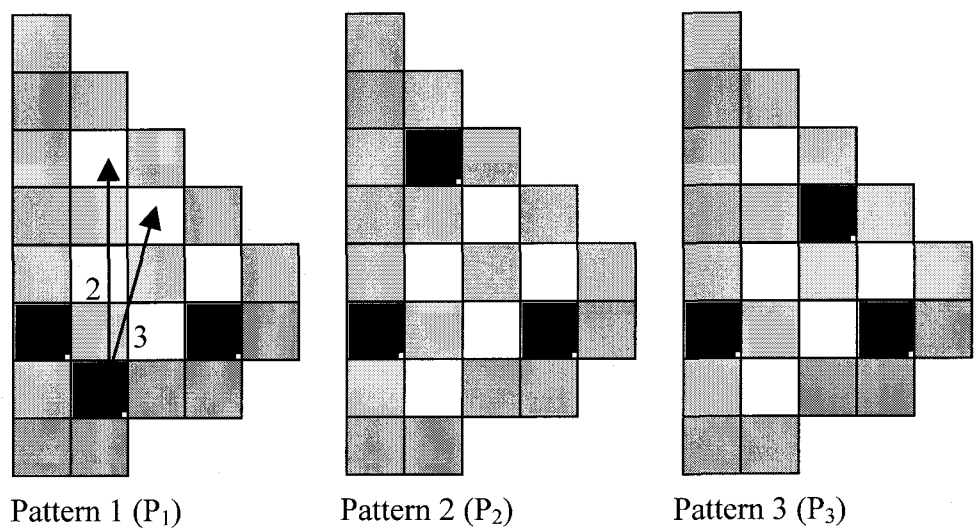


Figure 6.2 Similar loading patterns obtained by binary exchange operation

The input vector given in Eq. (6.2) complies with the first requirement. However, it does not include any remedies for the second condition. This situation is illustrated in Figure 6.2.

Here, P_2 and P_3 was obtained from P_1 by performing two distinct binary exchange operations between the fuel assembly with ID 316 located at position 22 and those located at position 5 and 9 (Figure 6.2).

It is easy to demonstrate the validity of the following relation:

$$\| P_1 - P_2 \|_1 = \| P_1 - P_3 \|_1 \quad (6.3)$$

where $\| \cdot \|_1$ stands for the Manhattan distance, which is a special case of Minkowski k -measure of a vector defined as:

for any $x \in R^n$

$$\| x \|_k = \left(\sum_{i=1}^n |x_i|^k \right)^{1/k} . \quad (6.4)$$

Therefore, the neural network will not be able to tell the difference between P_2 and P_3 with reference to the original pattern P_1 from which they were both generated.

Two different methods were applied to incorporate the location effect into the patterns.

1) *The neutron importance function method (Kim et al, 1993)*: The one-group adjoint flux is given as

$$i(n) = J_0 \left(\frac{2.405r}{R} \right) \quad (6.5)$$

where r is the distance of the n 'th fuel assembly from the center of the core, and R is the radius of the core. Then, for each pattern

$$I = \sum_n i(n) \beta(n) \quad (6.6)$$

was calculated and normalized to the interval [0.0,1.0] to obtain I_o . Here, β is the neural network value given in Table 6.2.

Then, the input vector given in Eq. (6.2) was multiplied to get the final input vector to the neural network

$$X = I_o x . \quad (6.7)$$

In this case, the sample-encoded vector given by Eq. 6.2 becomes

$$X = [0.585 \ 0.098 \ 0.585 \ 0.585 \ 0.195 \ 0.585 \ 0.098 \ 0.585 \ 0.195 \ 0.585 \ 0.585 \ 0.195 \ 0.585 \ 0.195 \ 0.585 \ 0.293 \ 0.585 \ 0.195 \ 0.488 \ 0.683 \ 0.585 \ 0.390 \ 0.683 \ 0.683 \ 0.683 \ 0.683].$$

2) *The k_{eff} method:* A second method has been developed for the position effect. A separate effective multiplication factor k_{eff} was calculated for different loading patterns constructed by locating a single fuel assembly and leaving other locations empty (filled by water). The single assembly was placed to each location in turn so that for a core with 26 assembly positions 26 k_{eff} 's were obtained. When the calculations were performed for 7 different fuel assemblies the number of k_{eff} was 182. The k_{eff} for the loading pattern that has one assembly FA_n located at position n is denoted by $k_{eff,n}^{FA_n}$.

Each fuel assembly member of the input vector x was multiplied by the corresponding k_{eff} value with respect to its position. This strategy included the position related effect to the input vector.

Input vector X becomes

$$X = [\beta_1^{FA_1} k_{eff,1}^{FA_1} \ \beta_2^{FA_2} k_{eff,2}^{FA_2} \ \dots \ \beta_{26}^{FA_{26}} k_{eff,26}^{FA_{26}}] \quad (6.8)$$

where $\beta_n^{FA_n}$ is the neural network value taken from Table 6.2 for assembly FA_n located at core position n .

The output vector (power fractions for the fuel assemblies and the effective multiplication factor for the core) was normalized to the interval [0.1,0.9], since, the

activation function produces results in $[0.0,1.0]$ and the effect of x going to infinity for $f(x)$ near 0 and 1 slows down the training.

6.1.2 The Results

500 out of the 2000 patterns generated by the in-core fuel management package were used to train the neural network. The k_{eff} method was used to add the position dependency effect. The weights were initialized with random numbers, so that, the network was prevented from converging to a weight set of identical values. These random values were chosen to be small numbers in the interval -0.1 and 0.1 .

The training iteration was assumed to be finished when the cumulative error $E=\frac{1}{2}\sum_k(t_k-o_k)^2$ was less than the convergence criterion, which was set to 0.005 for actual training sessions. A sample *error* versus *iteration number* data is given graphically in Figure 6.3.

Momentum parameter α and training rate η values, and the number of hidden layer nodes (n_h) are critical for the learning process. Unfortunately, no straightforward procedure exists to determine proper training parameters.

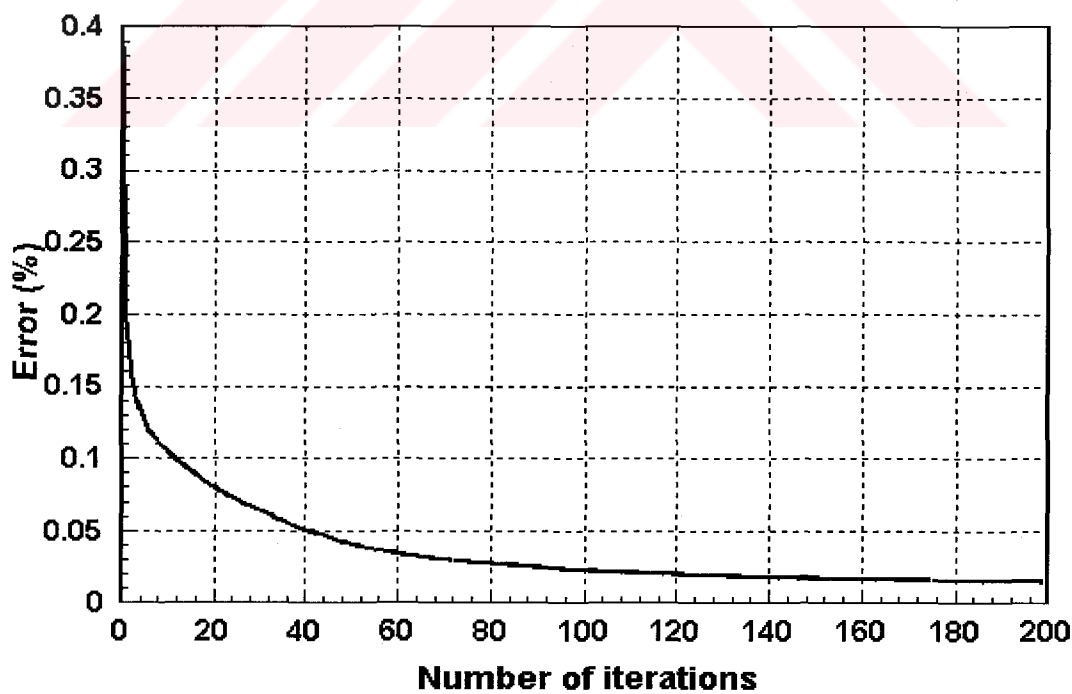


Figure 6.3 Network learning curve

The training was conducted for possible values and the near optimum parameters were specified from the graphs drawn from the results. These trial trainings were performed with a convergence criterion of 0.05 to obtain the results faster.

While α was changed from 0.1 to 1.0, as a starting point η was fixed to 0.6 and the number of hidden layer nodes was taken to be 150. The results are shown in Figure 6.4.

It can be easily seen that, for an α value of 0.9 the network is fastest and most stable. A value greater than about 0.94 made the network unstable. With a higher α value (>0.95) the network was unable to converge to the given criterion. An α value of 0.9 was used throughout all the network training.

Similarly, the training procedure was conducted for η values between 0.05 and 1.0. with a fixed α of 0.9, convergence criterion of 0.05 and 150 hidden layer nodes. The results are shown in Figure 6.5.

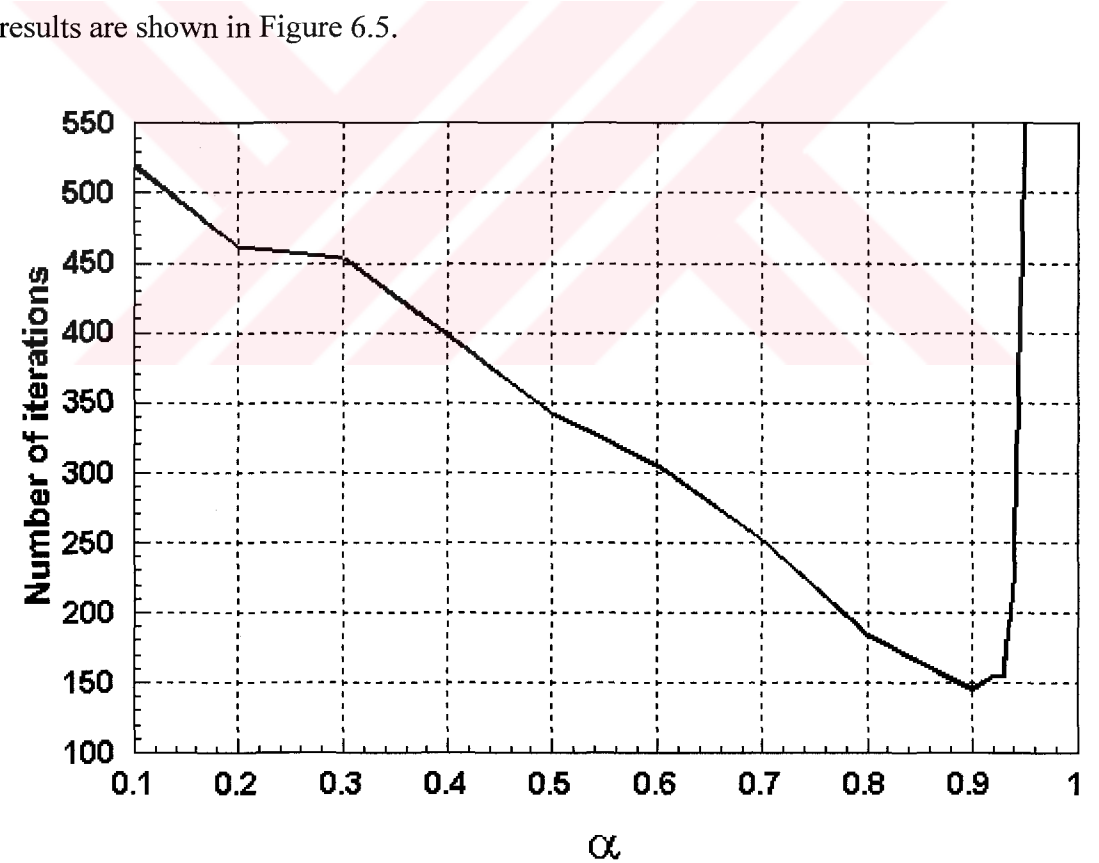


Figure 6.4 The number of iterations versus α (until $E<0.05$)

η values from 0.4 to 0.6 converged fastest while ~ 0.4 was the most stable and values greater than 0.5 caused some oscillation. η values above 1.0 prevented the network from convergence.

An η value of 0.4 was chosen to be the optimum parameter for the network training.

To determine the optimum number of hidden layer nodes, several training session were performed. The α and η parameters were set to 0.9 and 0.4, respectively, and the convergence criterion to 0.05. The number of hidden layer nodes for the network was changed from 50 to 500.

The relationship between the number of hidden layer nodes and the number of iterations is shown in Figure 6.6.

The network was unable to learn the relationship for a number of hidden layer nodes that was less than 50. When the value was between about 250 and 400, the number of iterations was the least.

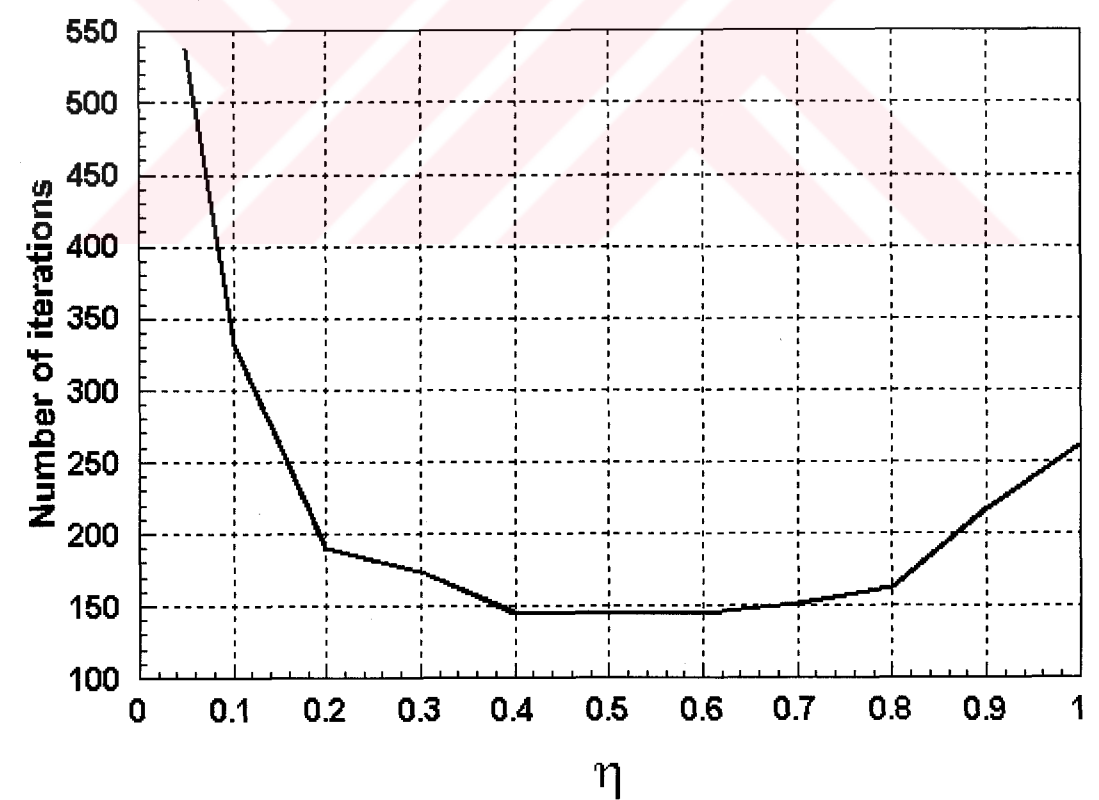


Figure 6.5 The number of iterations versus η (until $E < 0.05$)

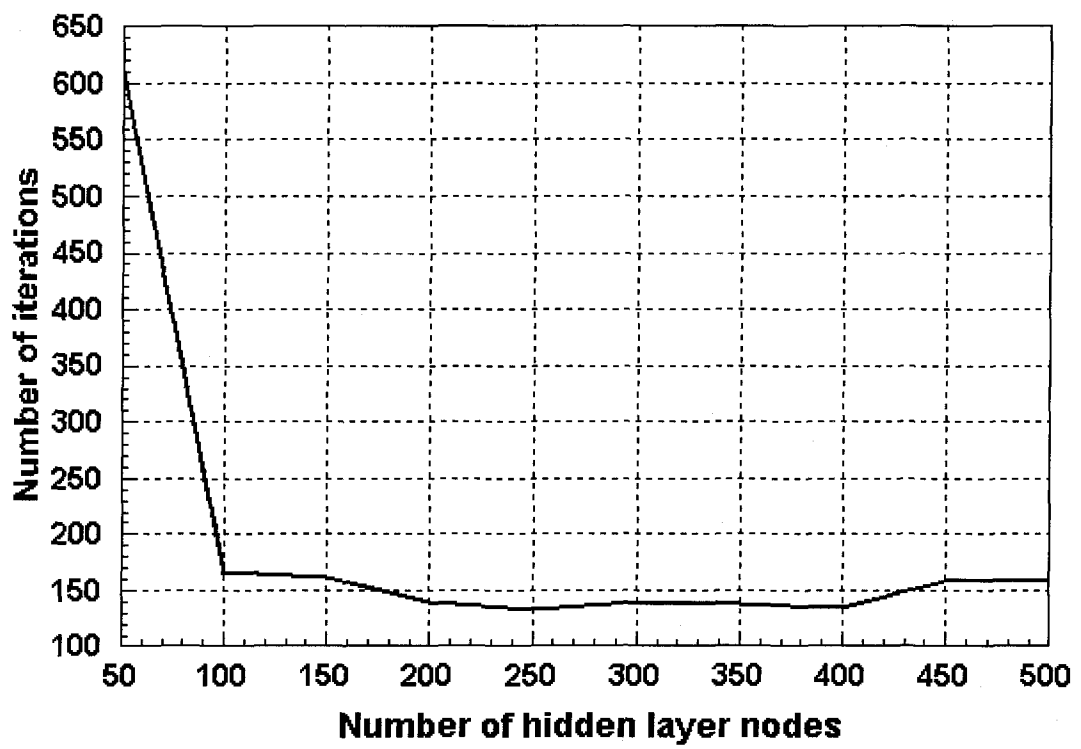


Figure 6.6 Number of iterations versus number of hidden layer nodes

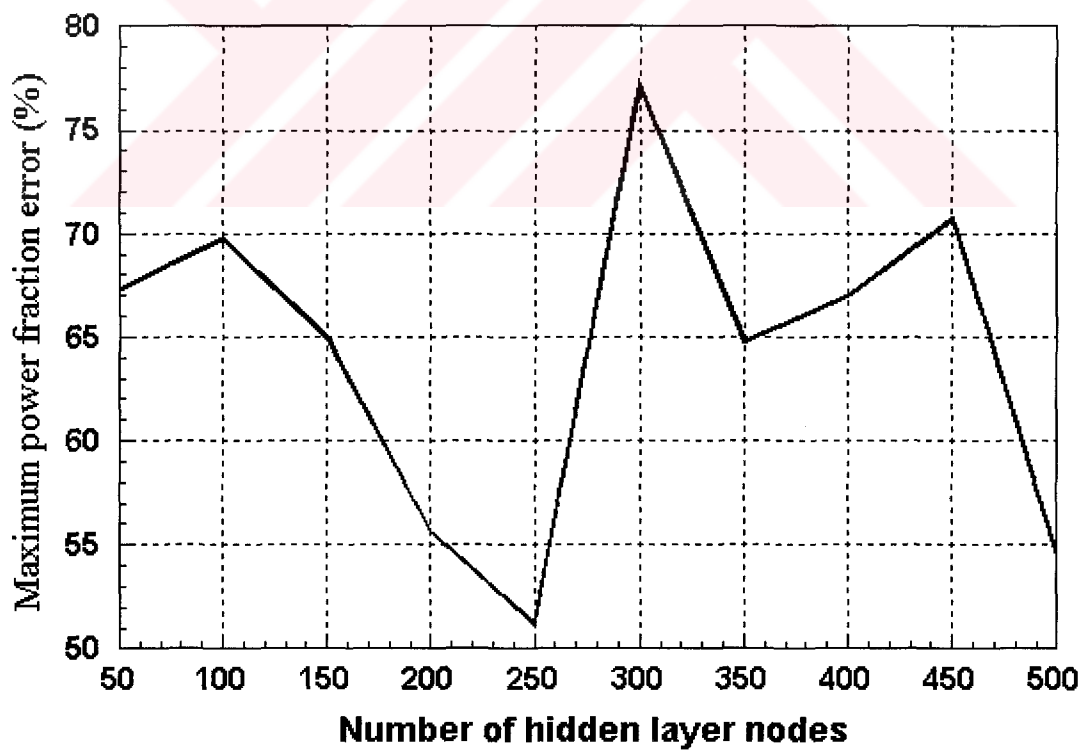


Figure 6.7 Maximum power fraction error versus number of hidden layer nodes

To estimate the success of the network with different number of hidden layer nodes, the maximum errors in percent corresponding to the difference of the target (desired) and generated outputs of power distribution are drawn in Figure 6.7.

It is clearly seen that, for a hidden layer node number of 250 the maximum error is the least. Hence, considering both of the results shown in Figure 6.6 and Figure 6.7, a hidden layer node number of 250 was set for this work.

To validate the found parameter values, the order of the experimentation has been changed. First, the number of hidden layer nodes was fixed while allowing α and η to change. Similar results were obtained.

The neural network was tested using the remaining 1500 randomly generated loading patterns. The network predicted the maximum power fraction with an average error of 6.14% and the maximum error was 14.5%. For the effective multiplication factor, k_{eff} , the results were 0.49% and 1.54% for the average and maximum errors, respectively.

The results for the maximum power fraction prediction and effective multiplication factor are shown in Figure 6.8 and Figure 6.9. It is seen that both for the power and k_{eff} data the linear fits approximately lies along the diagonal axis.

Finally, to show the effect of the position dependence of the assemblies, the network was trained for three different cases. In the first case, no effect was included. Then, the neutron importance function method was used. As a last step, the network was trained with the k_{eff} weighting method. Power distribution for the sample loading patterns were generated for all cases. The results are shown in Table 6.3.

Table 6.3 The results of the position effect methods

	Number of Iterations	Average Error in Maximum Power Fraction (%)
No effect	870	10.03
Importance function	879	9.57
k_{eff} method	1411	7.31

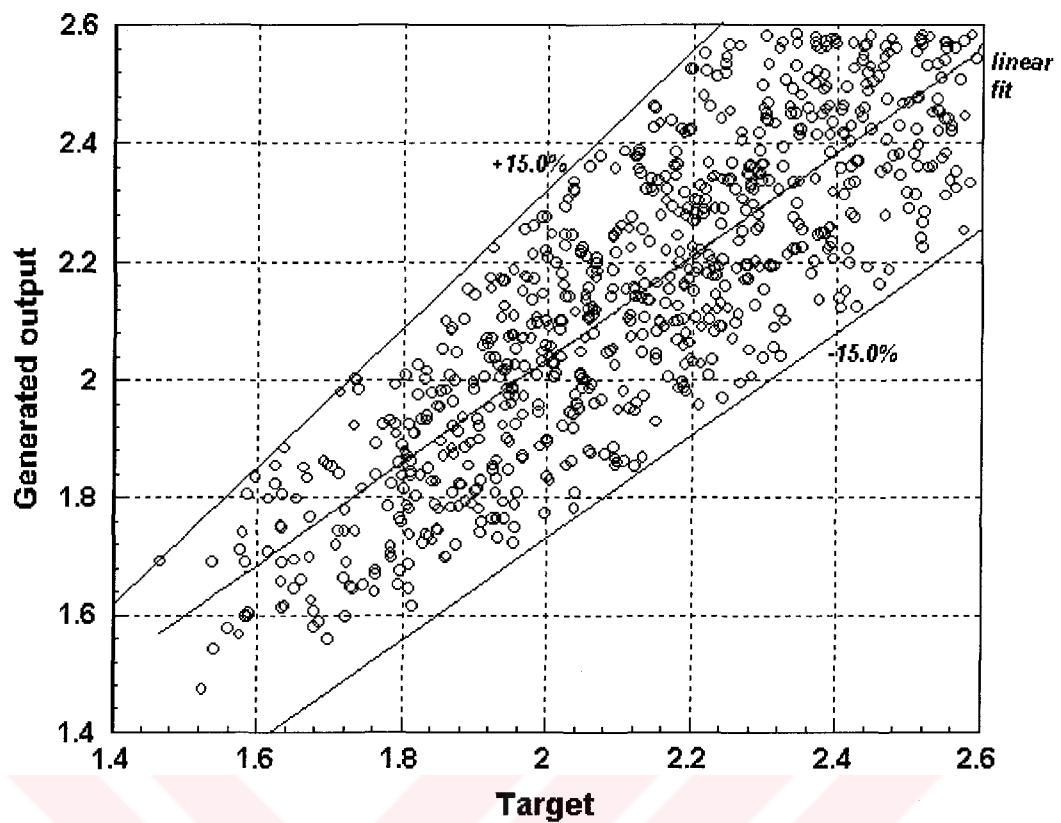


Figure 6.8 Maximum power fraction: generated output versus target values

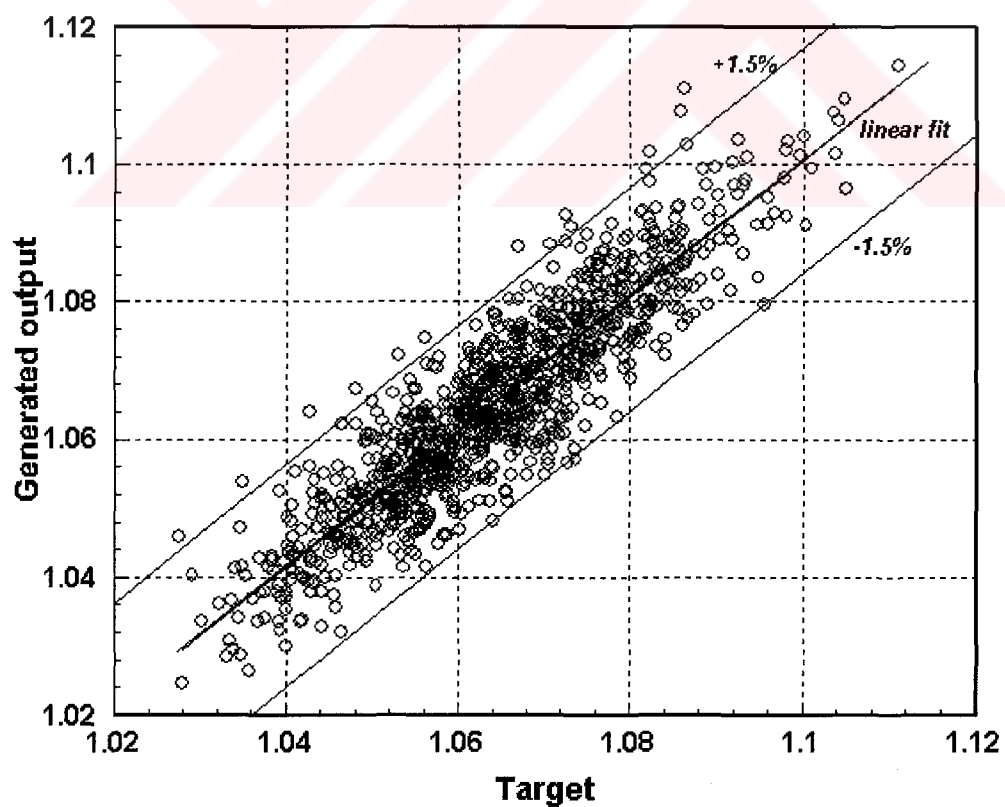


Figure 6.9 Effective multiplication factor: generated output versus target values

The k_{eff} method performed best with respect to the error in the maximum power fraction. The cost of the improvement was a larger number of iterations.

The calculation of the power distribution and k_{eff} takes about one minute on a personel computer using the conventional codes. On the other hand, for a single loading pattern, it takes about a fraction of a second with the neural network (Table 6.4).

Table 6.4 The time required for the calculation of a single loading pattern

	Time (seconds)
MCRAC-LEOPARD	~30
Artificial Neural Network	<1

Detailed sample outputs of the power distributions generated by the neural network are shown in figures of Appendix B.

Kim et al. (1993) used an artificial neural network to find optimal loading patterns. They predicted the power fraction and the k_{eff} with a maximum error of $\pm 6\%$ and $\pm 0.3\%$, respectively. Their model was based on a simple core with 3 different assembly types. There were two networks to estimate the power fraction and the k_{eff} separately.

Lysenko et al. (1999) used a dynamic node architecture network to predict the diffusion eigenvalue with lower errors. Similarly, their network estimated only k_{eff} .

The use of a single network for estimating the power fraction and k_{eff} , as in this work, results in relatively high errors, but spares the time required to train two networks. Actually, the higher errors do not compromise the output, since the main purpose is a fast and rough estimator that provides the genetic algorithm the parameters that is required to rank the solutions. The final loading patterns should pass a test with standard nuclear codes.

The following section describes a simple method to search for optimum patterns by using this neural network as the core parameter estimator.

7. OPTIMUM LOADING PATTERNS: SIMPLE RULE-BASED METHOD

At this stage of the work, the optimum loading patterns were searched by the simple method of generating random patterns, then, calculating and checking their fitness for the problem.

The number of fuel assemblies that were used to construct the patterns were fixed with a maximum to limit the occurrence of any given fuel assembly. A sample set of fuel assembly numbers is given in Table 6.1. These numbers were chosen to be comparable with the configuration of the Almaraz Power Plant.

At this point, it is interesting to note that using the fuel assemblies listed in Table 6.1 there are $\sim 3.5 \times 10^{14}$ different possible loading patterns for the current case (the number of different patterns is given by $N! / [n_1! n_2! \dots n_k!]$ where N is the number of locations and n_k is the number of assembly of type k). Even for a restricted search space the number of possible solutions are extremely large.

Then, a pattern was generated randomly with each assembly occurring exactly as many times as the quantity listed in Table 6.1. The total number of fuel assemblies given in the table was chosen to be equal to the number of the reactor core locations. Although for mismatching numbers the techniques are the same for all methods used in this work, the restriction simplifies the pattern construction algorithms.

To qualify as a candidate-loading pattern, it was expected to comply with certain predefined rules.

Three rules were defined for this work:

- 1) Assemblies with high-enrichment are loaded into core peripheral regions
- 2) Assemblies with burnable poisons are loaded into inner regions
- 3) Assemblies with low-enrichment or the ones that lack burnable poisons are loaded anywhere

One of the fuel assemblies shown in Table 6.1 was denoted as “high-enriched” and another one as “low-enriched” to apply the rules. Assembly 131 was chosen to be the high-enriched one, since; it has no burnable poisons and has the highest enrichment of 3.1%. Assembly 121 was used as the low enriched one. It has no burnable poison and has the lowest enrichment of 2.1%. Therefore, it may be placed anywhere in the loading pattern.

Rule 1 confines high-enriched fuel assembly to the outer regions of the reactor core. To avoid the unwanted situation of all loading patterns crowding the most outer locations of the reactor core, a flexible definition for the “outer region” is adapted. For this purpose, a *fuzzy variable outer region* is defined with a membership function $\mu(x)$, inspired from the concept of fuzzy logic. This function determines how far enriched fuel assemblies are allowed into the center of the core. Figure 7.1 shows the initial shape of $\mu(x)$, which is the possibility of a high-enriched fuel assembly to be loaded at a location along the radius of the core, where x is the distance from the reactor core center.

$\mu(x)$ is initialized with

$$\mu(x) = \begin{cases} \frac{x}{r}, & x < r \\ 1, & x \geq r, \quad r = 0.83 \end{cases} \quad (7.1)$$

where r is the distance of the outermost core locations from the core center.

Function $\mu(x)$ given in Figure 7.1 is modified for each loading pattern with respect to its success by increasing the possibility value for the most inner location where a high-enriched assembly resides. Similarly, failed loading patterns decrease the value for the location.

The fuzzy membership function evolves to a shape that is sufficiently loose to allow high-enriched assemblies to go to the inner regions of the core, but not loose enough to let them populate the innermost locations.

Initially, a high-enriched assembly was placed at the peripheral location of the core. This region is shaded in Figure 7.2.

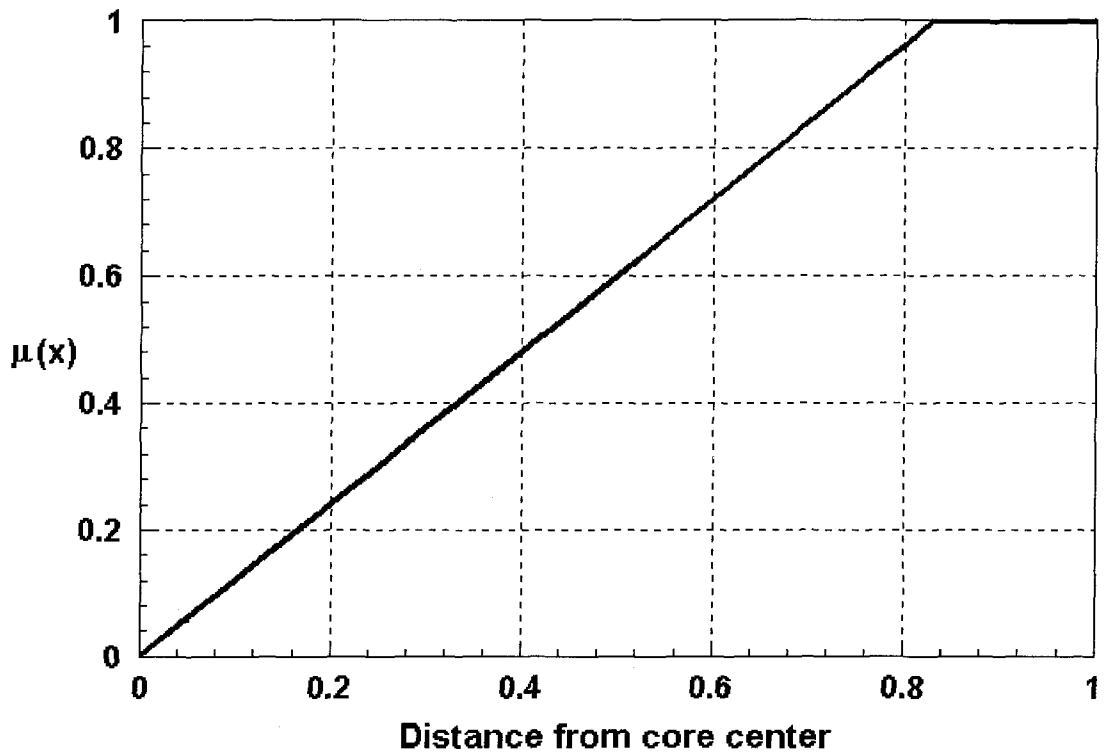


Figure 7.1 Initial fuzzy membership function for the outer region of the core

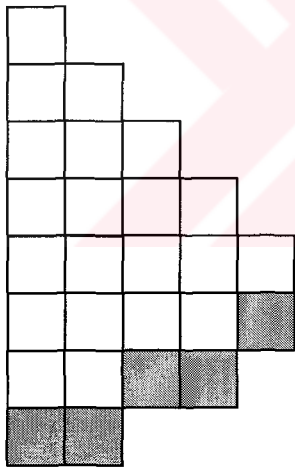


Figure 7.2 Reactor core inner/outer regions

For rule 2, “inner region” is defined as the core locations that do not face the core periphery. These locations are left blank in Figure 7.2.

Fuel assemblies including burnable poisons are placed anywhere in this inner region, randomly.

A candidate random loading pattern was constructed as follows (Figure 7.3):

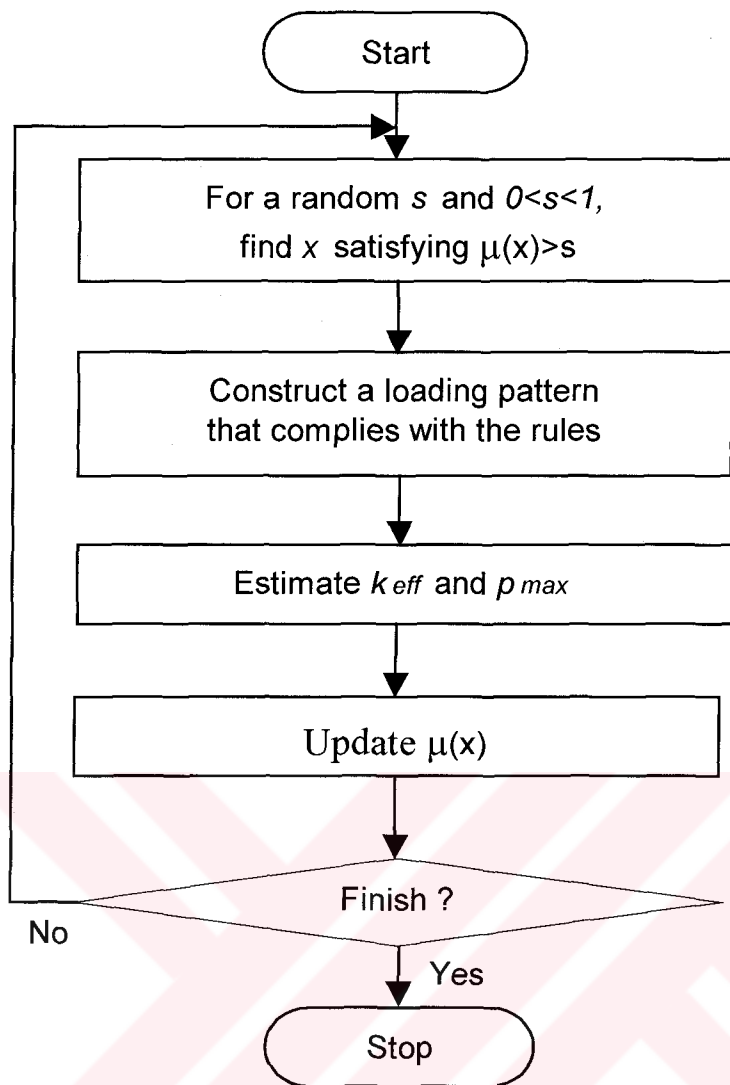


Figure 7.3 The flow chart of the simple rule-based method

A random number s is generated in the range 0.0 to 1.0. The minimum x value is determined that satisfies $\mu(x) > s$. This x specifies the minimum distance to the reactor core that a high-enriched fuel assembly can be located.

Then, a fuel assembly and its location are chosen randomly. If the number of fuel assemblies of the same type previously allocated is less than the given limit in Table 6.1 and the location complies with the three-rules, the actual placement occurs. This is repeated until all assembly locations are filled.

The power distribution and the effective multiplication factor (k_{eff}) are calculated using the neural network. The maximum power fraction p_{max} and k_{eff} are compared

with pre-determined reference values p_{ref} and k_{ref} , which are taken to be 1.4 and 1.0, respectively. Loading patterns that have a p_{max} , which is less than p_{ref} , and a k_{eff} , which is greater than k_{ref} are accepted as optimum.

$\mu(x)$ is modified with

$$\mu_{n+1}(x) = \begin{cases} 1, & \text{when } p_{max} \leq p_{ref} \\ \frac{p_{max} - p_{ref}}{1.6 - p_{ref}} (\mu_n(x) - 1) + 1, & \text{when } p_{ref} < p_{max} \leq 1.6 \\ 0.9\mu_n(x), & \text{when } p_{max} > 1.6 \end{cases} \quad (7.2)$$

where n is the step number and 1.6 is set as the minimum limit for an increase to occur. p_{max} values greater than 1.6 is assumed to be unacceptable causing $\mu(x)$ to be reduced. The case of $p_{max} \leq p_{ref}$ sets $\mu(x)$ to unity at the value of x where an optimum assembly is found.

The resulting function shape is modified so that one always gets a monotonous increasing function. It is to say, the right side of the modification point is never less than the left side.

For example, if the random number s is set to 0.6, x that satisfies $\mu(x) > s$ for the initial case given by Eq. 7.1 (for $r = 0.83$) is determined to be 0.5. For the case of loading patterns with p_{max} values of 1.8, 1.5 and 1.25, $\mu(0.5)$ is modified to 0.54, 0.73 and 1.0, respectively, using Eq. (7.2). The modified $\mu(x)$ for the three cases is shown in Figure 7.4.

The procedure restarts to create and check new loading patterns and continues until a required number of optimum patterns are stored.

7.1 Results

The simple rule-based random method was executed for two cases, to show the effect of having the dynamic mechanism of locating the high-enriched fuel assemblies. First, it was run with function $\mu^0(x)$,

$$\mu^0(x) = 1 \quad (7.3)$$

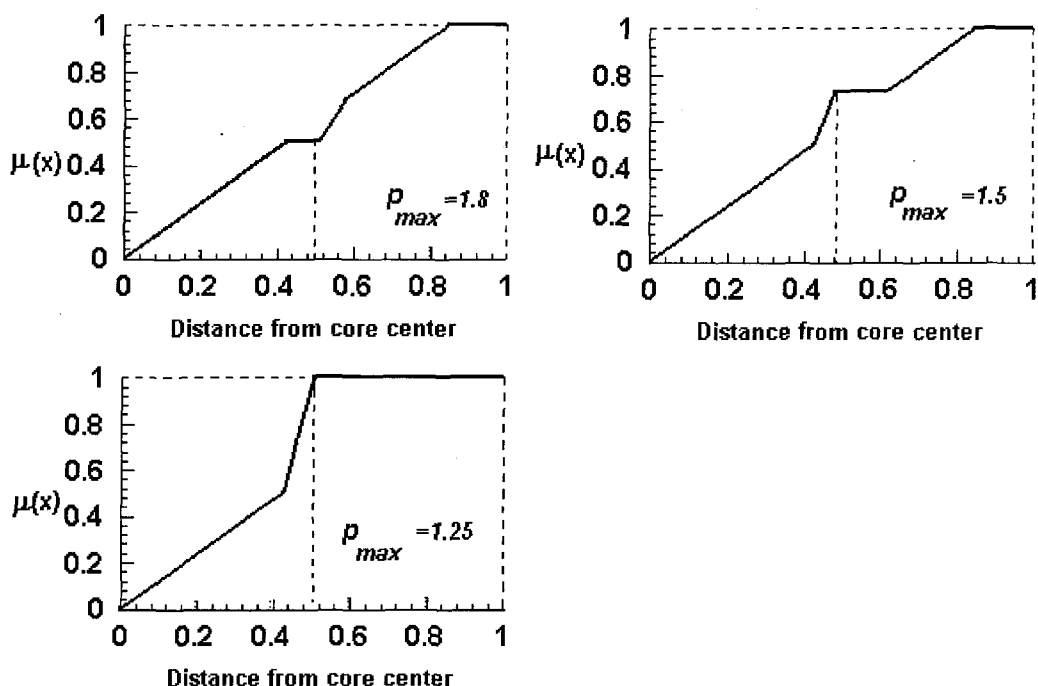


Figure 7.4 $\mu(x)$ modification for sample cases

It is to say, high-enriched assemblies were located anywhere. Then, the procedure was repeated with the dynamic limitation mechanism included.

For each case, one optimum loading pattern was found. This was repeated five times and the number of generated patterns that is required was compared. The results are listed in Table 7.1.

Table 7.1 Comparison of the results of the two simple rule-based random methods

Number of iterations per generated pattern	
$\mu^0(x)$	$\mu(x)$
5015	1137
4480	1665
23472	1028
6522	406
19246	1823
Averages	
11747	1211.8

It is clearly seen that, when dynamic location method is included, the number of pattern generation required to find the optimum is remarkably low compared with the straightforward random search method.

Table 7.2 lists the details of the results from the dynamic location method. One sample $\mu(x)$ and fuel assembly loading-pattern, obtained at the end of the execution, is shown in Figure 7.5. The fuel assembly distribution and the $\mu(x)$ functions for three solutions are displayed in the figures of Appendix C.

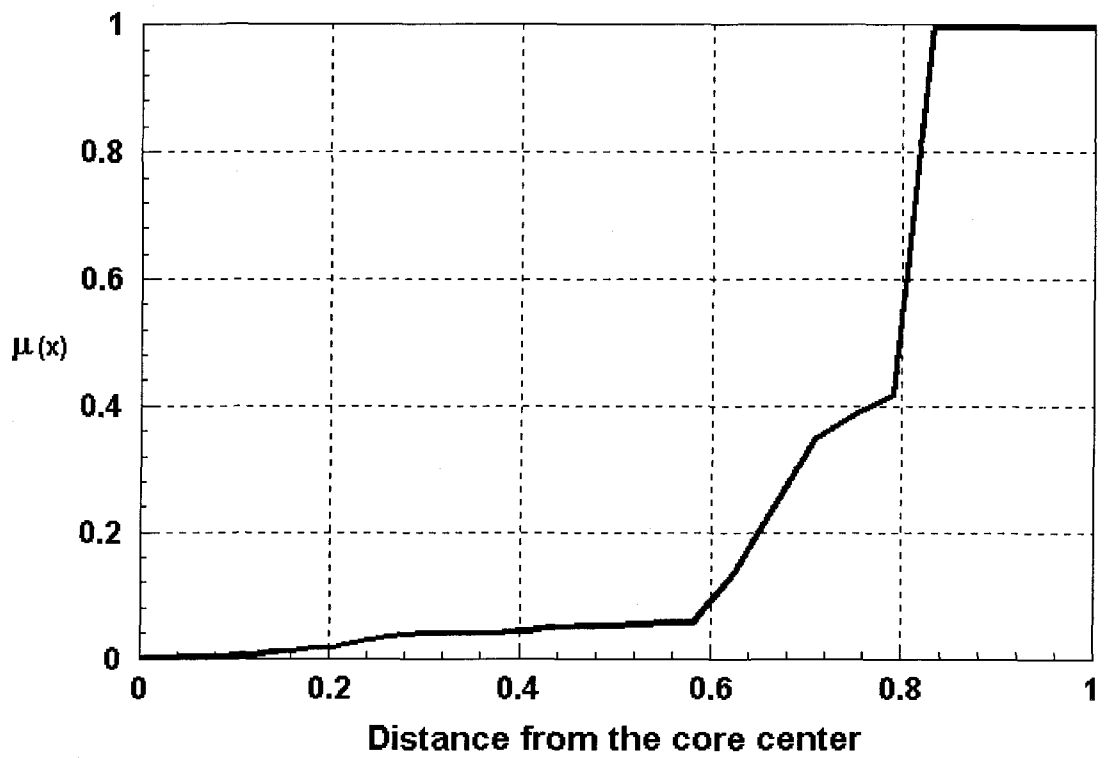
Table 7.2 Results of the simple rule-based random method

Sample Run	$P_{\max}$	K_{eff}
1	1.285	1.033
2	1.234	1.088
3	1.288	1.035

The results demonstrate that it is possible to find optimum loading patterns using this simple algorithm. Despite of the high number of patterns generated, a very fast neural network calculator makes it possible to find a desired solution within a reasonable time interval.

There is an important drawback of this approach. The successive patterns generated by the simple rule-based random method do not include any feedback from the previous experience. This technique produces completely independent solutions that cannot be fine-tuned to a better degree along the progress of the procedure.

The genetic algorithms remedy this shortcoming. Next section describes this method as optimizer.



116				
116	312			
116	121	120		
121	316	116	121	
121	120	121	121	112
121	116	121	121	131
121	121	131	131	
131	131			

Figure 7.5 $\mu(x)$ and fuel assembly distribution for an optimum loading pattern ($p_{\max}=1.285$, $k_{\text{eff}}=1.033$)

8. GENETIC ALGORITHMS

Genetic algorithms are used to solve computational problems, which require searching through a huge number of possibilities or simultaneous processing, by parallelism (**Mitchell, 1996**). The idea is to evolve a population of solutions to a given problem by using methods of genetic selection, crossover and mutation.

As it is the case in artificial neural networks, the idea of using evolution as an optimization tool for engineering problems emerged in the early stage of the computer era.

This method, which is categorized as a subset of evolutionary algorithms, was developed by John Holland. The work presented by Holland's 1975 book *Adaptation in Natural and Artificial Systems* was the first attempt to put computational evolution on a firm theoretical footing (**Holland, 1992**).

Genetic algorithms are based on the mechanics of natural selection and natural genetics (**Goldberg, 1989**). The concept of the survival of the fittest among string structures representing the problem is used together with a structured yet randomized information exchange. At each stage of the process, called a generation, a new set of strings (or a solution) is created using bits and pieces of the fittest of the old.

Genetic algorithms have the following attractive properties:

- A genetic algorithm is a massively parallel search method, which is able to evaluate many solutions simultaneously, instead of dealing one by one with the huge number of possible solutions.
- The method optimizes with continuous or discrete parameters (**Haupt and Haupt, 1998**).
- The genetic algorithms provide a list of optimum parameters, not just a single solution.

- They work with numerically generated data, experimental data, or analytical functions.
- Implementing evolution is remarkably simple. For example it does not require derivative information. Solutions are derived by means of variations, like mutation and crossover, followed by natural selection, which consists of choosing the fittest solutions. Nevertheless, these simple rules are thought to be the underlying mechanism of the bio-diversity seen in the nature.
- Genetic algorithms are adaptive. They provide the means to program for continuously changing environment. Typically, this condition is encountered in robot control. The algorithm may produce original procedures to adapt to new conditions.
- Problems that involve very complex rules to infer any reasonable solution may benefit from genetic algorithms. For instance, there are works investigating the possibility of constructing artificial intelligence systems by using few simple rules that interact by means of the genetic techniques. The massively parallel nature of the method is expected to approximate the highly complex underlying of the biological intelligence.

8.1 Fundamentals of The Algorithm

A particular solution to a problem is called a “chromosome”. Any element of the chromosome or the solution is named as the “gene”. Genes usually are strings of bits consisting of 1 and 0's, or sequence of adjacent bits, which encode the particular element of the chromosome. The complete collection of genetic material is called the “genome”. The term “genotype” refers to the particular set of genes contained in a genome. Two individuals that have identical genomes have the same genotype.

All possible solutions to a problem make up the search space. Each chromosome is represented as a point in the search space of candidate solutions. A genetic algorithm processes populations of chromosomes and replaces the ones which are unsuitable according to a given fitness function. This function estimates how well the processed chromosome solves the problem.

There are three main genetic operators:

Selection: This operator selects and reproduces chromosomes that satisfy or surpass the fitness function. This corresponds to natural selection: chromosomes with higher fitness values have a better chance of surviving and contributing, through crossover and mutation, to the next generation. The population size remains constant between generations, with new members replacing the previous population.

Crossover: This operator randomly chooses a location and exchanges a specified length of information between two chromosomes. The operator may combine the best part of each parent. In this case, the fitness is improved and the probability of survival is increased. The newly created two offspring are placed into the next generation of population. This represents the recombination of two single-chromosomes much like sexual reproduction in biological populations.

Mutation: This operator randomly chooses a location in a chromosome and changes some of the bits. It prevents the population from converging to a set of identical solutions by providing the possibility for the genetic algorithm to explore the entire search space. This operation has a very small probability.

Applying genetic algorithms starts by defining a string representation of a chromosome, or solution, of the given problem. In the first step, the number of chromosomes n in the population and the length l of each chromosome are specified.

The fitness function $f(x)$ is defined for the particular problem.

Next, a population of n chromosomes of length l is generated randomly. These chromosomes are the candidate solutions for the problem.

The fitness of each chromosome x is calculated with function $f(x)$.

The survival probability to the next generation of chromosomes, which have high fitness values, is increased by means of the selection and reproduction mechanism. It is to say, new copies of them are added to the population.

Parent chromosomes are selected according to their selected fitness, and the genetic operations crossover or random mutations are performed to make an offspring.

The current population is replaced with the new one and the process is restarted by calculating the fitness of the chromosomes and applying the operators. This iterative process is continued until there are highly fit chromosomes in the population.

The resulting population largely depends on the randomly created initial population. Usually, many solutions with different initial populations are generated to cover a wider range of the search space.

8.2 Applications

Genetic algorithms have been applied to a large number of scientific and engineering problems. Some examples:

- Optimization: genetic algorithms were applied to a variety of optimization tasks like numerical optimization, combinatorial optimization problems as circuit layout and job scheduling.
- Automatic programming: genetic algorithms were used to evolve computer programs for specific tasks, and to design other computational structures such as cellular automata and sorting networks.
- Machine learning: genetic algorithms were applied to machine learning applications such as classification and prediction tasks. This includes predicting weather and protein structures. They were also used to evolve certain machine aspects as weights for neural networks.
- Economics: genetic algorithms were applied to the modeling of economical processes; for example, the emergence of economical markets.
- Immune systems: genetic algorithms were used to model various aspects of natural immune systems; for example, somatic mutation during an individual's lifetime.
- Ecology: genetic algorithms were applied to model ecological phenomena like biological arms races, host-parasite co-evolution.

- Population genetics: genetic algorithms were used to study questions in population genetics, such as “Under what conditions will a gene for recombination be evolutionarily viable?”.
- Evolution and learning: genetic algorithms were applied to study how individual learning and species evolution affect one another.
- Social systems: genetic algorithms were used to study the evolutionary aspects of social systems, such as the evolution of social behavior in insect colonies, and the evolution and communication in multi-agent systems.

Here is an application with some detail.

8.2.1 Traveling Salesman

The traveling salesman problem represents a classical optimization problem, which cannot be solved by the traditional techniques. The problem may be outlined as follows: find the shortest route for a salesman to take in visiting N cities.

In this case, the genetic algorithm is supposed to solve a reordering or permutation problem. The crossover and mutation operators are required to handle the chromosomes in such a way that the solution contains only one type of the individual items of the parameter. For the traveling salesman problem, this is the city.

The fitness function f is given as

$$f = \sum_{n=0}^N \sqrt{(x_n - x_{n+1})^2 + (y_n - y_{n+1})^2} \quad (8.1)$$

which is the total distance traveled by the salesman between the cities at coordinates (x_n, y_n) for the ordering $n=1, \dots, N$.

The problem string (chromosome) has a length of N , where each item is an integer representing a city. The order of the items defines the route of the salesman.

The genetic operators are applied to the strings while ensuring that a city exists only once in a given solution. The mutation operator readily complies with this when two random positions in the chromosome are selected and exchanged. However,

crossover operation requires further processing. As an example, two strings representing a problem of 6 cities are given as follows

Parent 1: [4 2 3 6 1 5]

Parent 2: [3 2 5 1 6 4].

A crossover operation at the third position exchanges 2 integers resulting in the following offspring

Offspring 1: [4 2 **5** 1 *XX*]

Offspring 2: [*XX* **3** 6 *XX* 4].

The repeated integers were replaced with the *X* (place holder). Then, *X* positions are pushed to the beginning of the offspring and all integers that were in those positions are pushed off to the left of the chromosome and wrap around to the end of the offspring.

Offspring 1: [*XX* **5** 1 4 2]

Offspring 2: [*XX* **3** 6 2 4]

Finally, *X*'s are replaced with the original integers.

Offspring 1: [3 6 **5** 1 4 2]

Offspring 2: [5 1 **3** 6 2 4].

This process is called the order crossover. There are other methods to do the task (partially matched crossover and cycle crossover).

8.3 Theory

In genetic algorithm, crossover causes genotypes to be cut and spliced (Louis, 1993). Because of this property, it is necessary to consider the substrings created and manipulated by crossover, instead of analyzing the fate of individual strings. These substrings define regions of the search space and are called schemas. In other words, a schema is a template that identifies a subset of strings with similarities at certain

string positions. Holland's schema theorem is fundamental to the theory of genetic algorithms.

The templates in the schema theorem are made up of ones, zeros and asterisks representing wild cards. For example consider binary strings of length 6. The schema 1^*0^*1 describes the set of all strings of length 6 with 1s at positions 1 and 6, and a 0 at position 4. The "*" (asterisk) denotes a "don't care" symbol which means that positions 2, 3 and 5 can be either a 1 or a 0. The strings that fit this template are called the instances of the schema, for example, 101011 and 100001.

A given bit-string of length l is an instance of 2^l different schemas. For example, the string 11 is an instance of schemas **, *1, 1*, and 11. It can be shown that any given population of n individual strings may represent by 2^l to $n \cdot 2^l$ different schemas. With these definitions it may be stated that, at a given generation, while a genetic algorithm is explicitly evaluating the fitness of the n strings in the population, it is actually implicitly estimating the average fitness of a much larger number of schemas. In this context, the average fitness of a schema is defined to be the average fitness of all possible instances of that schema. The evaluation of the instances gives an estimate of the average fitness of that schema.

The schemas are not explicitly represented in a genetic algorithm calculation and the results are not stored. However, it can be shown that, the effect is contributed in terms of the increase and decrease in numbers of instances of given schemas in the population.

The approximate behavior of the instances of schemas may be derived as follows. Let H be a schema with at least one instance present in the population at time t . Here, H stands for hyperplane (since, schemas define hyperplanes of various dimensions). $m(H,t)$ is defined as the number of instances of H at time t , and $u(H,t)$ is the observed average fitness of instances of H in the population at time t .

$E(m(H,t+1))$, the expected number of instances of H at time $t+1$, is the quantity that specifies the behavior.

With the assumption that the expected number of offspring of a string x is equal to $f(x)/\bar{f}(t)$, where $f(x)$ is the fitness of x and $\bar{f}(t)$ is the average fitness of the population at time t , and x is in the population at time t , one may write

$$E(m(H, t+1)) = \sum_{x \in H} f(x) / \bar{f}(t) \quad (8.2)$$

where $x \in H$ denote “ x is an instance of H ” and with

$$u(H, t) = \frac{\sum_{x \in H} f(x)}{m(H, t)} \quad (8.3)$$

Eq. (8.2) is rewritten as

$$E(m(H, t+1)) = \frac{u(H, t)}{\bar{f}(t)} m(H, t) \quad (8.4)$$

The increases or decreases of schema instances in the population depends on the quantity $u(H, t)$.

The genetic operators crossover and mutation may destroy and create instances of schema H . At this step, if only the destructive effect is considered, one may specify a lower bound on $E(m(H, t+1))$ by modifying the right side of Eq. (8.4) as follows.

A schema H survives to the next generation under a single-point crossover if one of the offspring is an instance of H . Then, the probability $S_c(H)$ that H will survive single-point crossover may be given as

$$S_c(H) \geq 1 - p_c \left(\frac{d(H)}{l-1} \right) \quad (8.5)$$

where p_c is the probability whether a single-point crossover will be applied to a string, $d(H)$ is the defining length of H and l is the length of bit strings in the search space. Eq. (8.5) states that the probability of survival under crossover is higher for shorter schemas.

The probability $S_m(H)$ that a schema H will survive under mutation of an instance of H may be given as

$$S_m(H) = (1 - p_m)^{o(H)} \quad (8.6)$$

where p_m is the probability of any bit mutated and $o(H)$ is the order of H or the number of defined bits in H . It is seen that the probability of survival under mutation is higher for lower order schemas.

Using these disruptive effects Eq. (8.4) may be rewritten as

$$E(m(H, t+1)) \geq \frac{u(H, t)}{\bar{f}(t)} m(H, t) \left(1 - p_c \frac{d(H)}{l-1} \right) \left((1 - p_m)^{o(H)} \right). \quad (8.7)$$

Eq. (8.7) describes the growth of a schema from one generation to the next and may be interpreted as implying that short, low-order schemas that have an average fitness above the mean will receive exponentially increasing numbers of samples over time. This equation is known as the Schema Theorem.

This theorem includes only the destructive effects of the genetic operators, leading to a lower bound. However, the Building Block Hypothesis (**Goldberg, 1986**) states that crossover is the main power of genetic algorithms. It recombines instances of good schemas to form instances of better and higher-order schemas.

On the other hand, Holland proposed that mutation is the tool what prevents the loss of diversity at a given bit position. It provides the genetic algorithms to cover a wider search space by preventing fixations.

John Holland's original motivation for developing genetic algorithms was to construct a theoretical framework for adaptation as seen in nature, and to apply it to the design of artificial adaptive systems. According to Holland, an adaptive system must persistently identify, test, and incorporate structural properties hypothesized to give better performance in some environment. Schemas are meant to be a formalization of such structural properties. In the context of genetics, schemas correspond to constellations of genes that work together to effect some adaptation in an organism. Adaptation is possible only in a world in which there is structure in the environment to be discovered and exploited. Adaptation is impossible in a sufficiently random environment.

The schema analysis shows that the calculation of the fitnesses of the N members of a population is only one part of the process of the genetic algorithm. It also implicitly estimates the average fitnesses of a large number of schemas by calculating the observed average fitnesses of schemas with instances in population. Since, this secondary process does not need any computational time, Holland calls as implicit parallelism. This analysis also shows that those schemas whose fitness estimates remain above average receive increasing numbers of instances in the population.

It may be further stated that, with respect to this theorem, selection operator increasingly focuses the search space with estimated above-average fitnesses. The crossover operator merges high-fitness schemas or building blocks together on the same chromosome to create higher fitness chromosomes, and the mutation operator ensures that the genetic diversity in the population is not lost.

Holland describes adaptation as a tension between exploration and exploitation. Here, exploration is defined as the search for new, useful adaptations, and exploitation is the use and propagation of these adaptations.

There exists a tension since any move toward exploration, in other words, testing previously unseen schemas or schemas whose instances seen so far have low fitness, takes away from the exploitation of tried and true schemas. In any adapting system, a balance between exploration and exploitation should be established. It is to say, the system has to keep trying out new possibilities not to overadapt and be inflexible. By the way, it also has to continually incorporate and use past experience as a guide for future behavior.

In addition to the previous discussion, the theory of schemas does not make predictions concerning the population composition, the speed of population convergence, or the distribution of fitnesses in the population over time. Several researchers proposed more detailed mathematical models to explain the behavior of the genetic algorithms. These models reveal some interesting properties of the genetic operators using simplified algorithms and play important roles in understanding the overall behavior (Vose, 1998).

9. OPTIMUM LOADING PATTERNS: GENETIC ALGORITHM

At this stage, genetic algorithms were used to find optimum loading patterns. For this purpose, a genetic representation or encoding of a pattern, called a chromosome, was constructed.

As a permutation problem the fuel management optimization resembles the traveling salesman problem. In this case, each fuel assembly corresponds to one city and the order of the cities to be visited is related to the order, which the fuel assemblies and burnable poisons are located in the reactor core. It means that the crossover operator must be modified so that it produces chromosomes that do not violate the contents of the fuel inventory (or one city does not exist more than once in the route of the traveling salesman).

Using one of the *partially mapped-crossover*, *order crossover* or *cycle crossover* implementations solves this difficulty (**Haupt and Haupt, 1998**). In the previous section (Genetic Algorithms), *order crossover* for the traveling salesman problem was described. Similarly, for this purpose a re-ranking technique was designed that is based on ordering fuel assembly ID's in the chromosome and renumbering them with respect to their orders.

Initially, this was achieved by building a one-dimensional array of ID numbers of the fuel assemblies based on the technique by Parks (1996). This is illustrated in

Figure 9.1 with a sample-loading pattern.

To have a different chromosome for each loading pattern, fuel assembly ID numbers were exchanged with unique rank values. In this work, the fuel assembly types were given rank numbers with respect to the order of their k_{∞} values. Fuel assemblies with the same ID number were given rank numbers with respect to the processing order. The number of fuel assemblies for each type was taken from Table 6.1. Fuel assemblies and their rank numbers are listed in Table 9.1.

The loading pattern

116													
116	312												
116	121	120											
121	316	116	121										
121	120	121	121	112									
121	116	121	121	131									
121	121	131	131										
131	131												

The one-dimensional representation

116	116	312	116	121	120	121	316	116	121	121	120	121	→
→	121	112	121	116	121	121	131	121	121	131	131	131	131

Figure 9.1 The loading pattern and the one-dimensional representation

By replacing all assembly ID numbers of the one-dimensional representation in Figure 9.1 with the values given in Table 9.1, the chromosome for the loading pattern was obtained.

Finally, for the loading pattern in Figure 9.1 the chromosome is given as in Figure 9.2.

Table 9.1 Fuel assembly types and their rank numbers

Assembly	131	131	131	131	131	121	121	121	121	121	121	121	121
Rank	26	25	24	23	22	21	20	19	18	17	16	15	14
Assembly	121	121	121	312	316	112	116	116	116	116	116	120	120
Rank	13	12	11	10	9	8	7	6	5	4	3	2	1

3	4	10	5	11	1	12	9	6	13	14	2	15	→
→	16	8	17	7	18	19	22	20	21	23	24	25	26

Figure 9.2 A sample chromosome representing the loading pattern of Figure 9.1

The next step was to generate a set of loading patterns as the initial population to apply genetic algorithms to the optimization. Later, this set of loading patterns was used to breed the desired fuel assembly distribution.

200 random loading patterns were generated with the technique that was used in the simple rule-based random method. To have a more random initial pattern population, the dynamic location method for the high-enriched fuel assemblies was disabled.

A chromosome for each of the patterns was constructed and stored. Core calculations were performed using the neural network for each pattern to obtain the power distribution and the effective multiplication constant.

Fitness values, which are measures of qualification for the respective loading patterns, were estimated for each chromosome in the population. For a chromosome with higher fitness value there is a higher probability of survival to the next generation. Fitness f was defined to be

$$f = c_k (k_{eff} - k_{ref}) - c_p (p_{max} - p_{ref}) \quad (9.1)$$

where k_{eff} and p_{max} are the results of the neural network calculation and k_{ref} and p_{ref} are the reference effective multiplication factor and maximum power fraction, which were taken to be 1.0 and 1.5, respectively; and c_k and c_p are the corresponding weights. c_k and c_p were set to 0.3 and 0.7, respectively. A higher power fraction weight ensures a higher contribution from the power fraction value. This was preferred, since a lower maximum power peak was more desirable in an optimum-loading pattern.

f , given with Eq. (9.1), results in a higher fitness when k_{eff} gets higher and p_{max} gets lower. The optimum loading pattern is the one having the largest k_{eff} and the smallest possible p_{max} . In this case, f produces the maximum result.

Loading patterns complying with the restrictions that specify optimum, it is to say, patterns that have p_{max} values less than p_{ref} and k_{eff} values greater than k_{ref} , were archived. For this work, the archive had a maximum size of 10, which is the maximum number of optimum loading patterns to be archived before terminating the execution. The places left by the archived chromosomes were filled by new random loading patterns. So, the size of the population was kept fixed.

At the next step, pairs of loading patterns, which are called “parents”, were selected from the population and archived groups. One third of the population that have the

For each crossover operation, the block was selected randomly. First, a random number that specifies the block size was generated. This number was between the minimum and the maximum. For the pattern given in Figure 9.3, this number is 5.

Then, a random fuel assembly location was selected. Starting from the selected location the rest of the fuel assemblies to be included into the block was determined according to a random walk technique that was designed to construct 2-dimensional random blocks.

Having determined the size and location of the block to be used in the crossover operation, the corresponding fuel assemblies were exchanged between the two loading patterns of the parent.

The resulting loading patterns were the offspring of the crossover operation. However, the rank information for the patterns may include duplicates, now. To re-rank both of them, one further operation was required.

Small random numbers were added to all rank values of both patterns (**DeChaine and Feltus, 1996**). Then, they were re-ranked to obtain the final offspring for the selected parent.

The crossover operation procedure is illustrated in Figure 9.4.

9.2 Mutation

The mutation operator is applied separately to each of the parent chromosomes. Small changes made by the operator ensure the availability of all the search space by preventing the solution from getting stuck to a limited region.

The number of mutation that is to be applied is determined randomly. This number is 1, 2 or 3. Then, two fuel assembly locations are selected and the assemblies are exchanged.

The resulting two chromosomes are the offspring. A mutation operation is shown in Figure 9.5.

3				
4	10			
5	11	1		
12	9	6	13	
14	2	15	16	8
17	7	18	19	22
20	21	23	24	
25	26			

4				
18	12			
13	17	23		
5	1	8	25	
22	15	24	16	26
3	10	21	2	9
19	11	6	14	
7	20			

A block of size 4 is selected from the parent chromosomes to initiate a operation

3				
4	10			
5	11	1		
12	1	6	13	
14	15	15	16	8
3	10	18	19	22
20	21	23	24	
25	26			

4				
18	12			
13	17	23		
5	9	8	25	
22	2	24	16	26
17	7	21	2	9
19	11	6	14	
7	20			

Fuel assemblies located in the blocks are exchanged

3.6				
4.5	10.5			
5.1	11.9	1.2		
12.6	1.3	6.5	13.7	
14.7	15.0	15.5	16.8	8.7
3.5	10.4	18.0	19.0	22.4
20.4	21.4	23.6	24.3	
25.1	26.6			

4.1				
18.1	12.8			
13.1	17.1	23.5		
5.5	9.8	8.1	25.7	
22.0	2.5	24.1	16.7	26.2
17.7	7.5	21.7	2.6	9.6
19.6	11.2	6.2	14.7	
7.4	20.2			

Small random numbers are added to the rank values

4				
5	10			
6	11	1		
12	2	7	13	
14	15	16	17	8
3	9	18	19	22
20	21	23	24	
25	26			

3				
18	12			
13	16	23		
4	10	8	25	
22	1	24	15	26
17	7	21	2	9
19	11	5	14	
6	20			

Chromosomes are re-ranked to obtain the offspring

Figure 9.4 A cycle of crossover operation

When the application of the crossover or mutation operators are successfully executed, each of the parent chromosomes were processed and modified. The last step of the new generation creation process was to re-calculate the fitness values for the parent chromosomes.

The last one third of the loading patterns in the global population list was replaced with the newly generated chromosomes. This action corresponds to the elimination of the failed offspring that fit poorly to the environment.

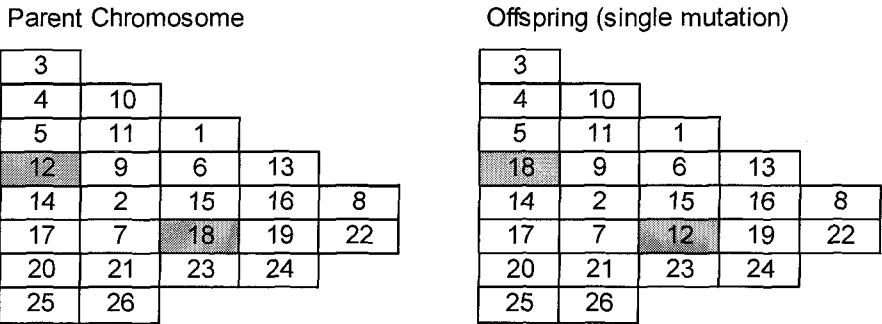


Figure 9.5 A sample mutation operation

The chromosomes in the population were re-sorted with respect to their fitness values and the process restarts. Chromosomes complying with the optimum loading pattern criteria were archived and a new generation was created from selected parents.

The iteration continued until the number of optimum loading patterns exceeded the maximum patterns that were allowed to be in the archive.

At the end of the execution, the chromosomes were converted back to the form shown in

Figure 9.1, it is to say, to the ID-number representation. This was done by looking up each rank value from Table 9.1 and replacing it with its ID number.

The possibility of archiving identical loading patterns was another issue. This problem was solved by checking each candidate pattern against archived ones and eliminating unwanted duplicates.

The flow chart for the genetic algorithm is given in Figure 9.6.

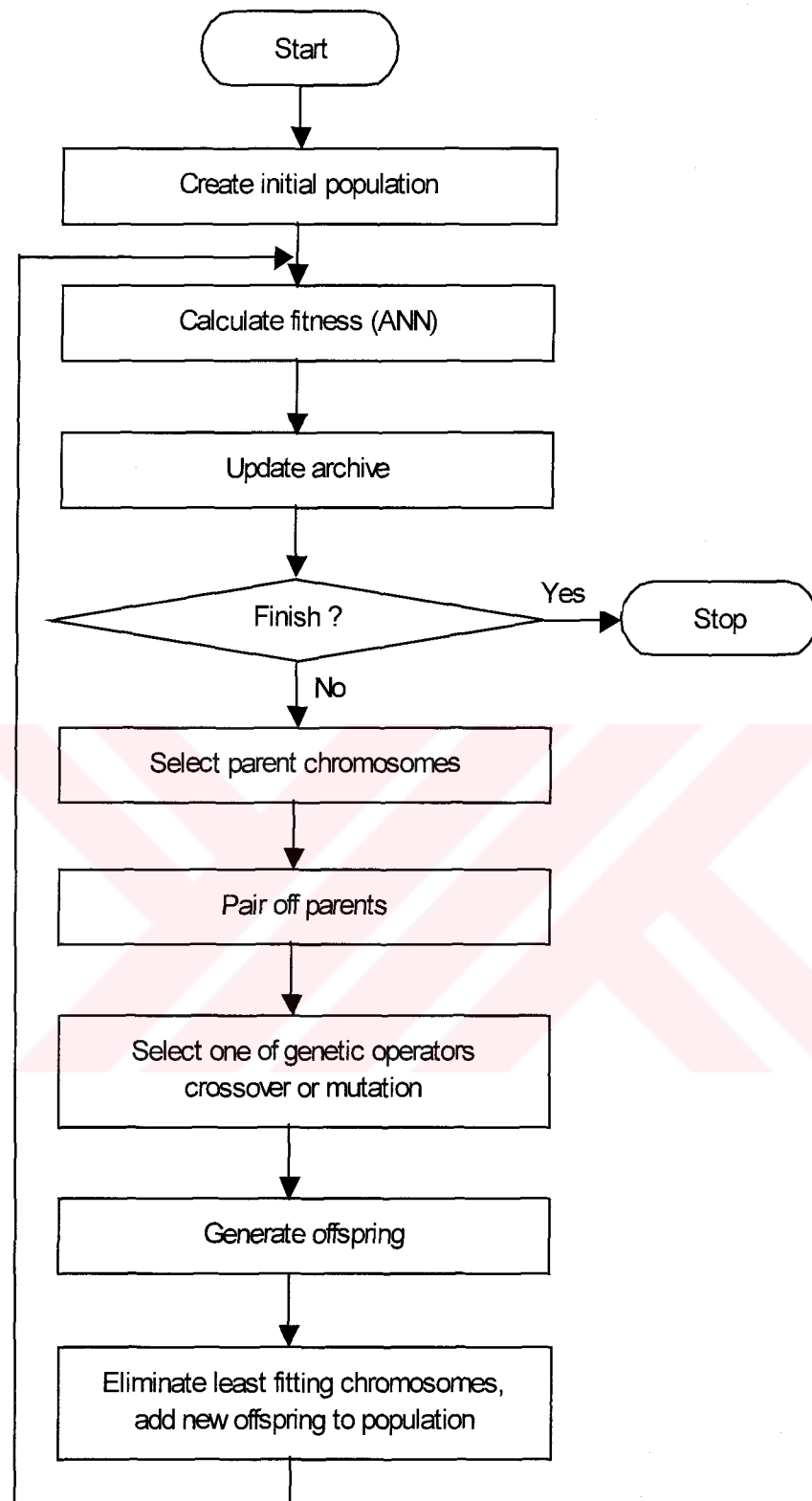


Figure 9.6 The flow chart of GA in fuel pattern selection

9.3 Results

9.3.1 Verification of the Search Procedure

To demonstrate and verify the technique with a simple case, the number of fuel assembly types was reduced to 2 from 7. 24 fuel assemblies of type 121 and 2 fuel assemblies of type 116 were used to generate the patterns. In this case, the number of possible loading patterns was 325.

This simplified case provided a search space where the calculation of all the parameters of the 325 possible loading patterns was feasible. Once the search results were obtained, the success of the genetic algorithm was validated.

The two fuel assembly types (121 and 116) were chosen to satisfy the criticality requirement of the core. Since, many different possibilities obtained by the combination of the fuel assembly types did not produce reasonable loading patterns.

All the 325 different loading patterns were calculated using the neural network to obtain the core parameters.

The genetic algorithm was used to generate 5 loading patterns with $p_{\max}$ less than 1.6 and k_{eff} larger than 1.0. The value of the reference power fraction of 1.6 was determined so that there were about 10 loading patterns satisfying the criterion in the 325 different possibilities. The initial population size was selected to be 20.

The number of generation to produce the desired number of loading patterns was 7.

The results obtained at the end of a sample search together with the top 10 (sorted with respect to their fitness values) out of 325 loading patterns are listed in

Table 9.2. It is seen that, the most optimum-loading pattern is in the optimum list with other four of the top 8 out of the 325 possibilities. The genetic algorithm seems to have missed some higher fitness patterns. The following discussion provides a possible reason for this situation.

Table 9.2 The best 10 of all loading patterns and a sample GA result

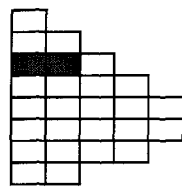
All Loading Patterns (top 10 out of 325)			Optimums selected by GA		
P _{max}	K _{eff}	Fitness	P _{max}	K _{eff}	Fitness
1.493712	1.056560	0.02137	1.577696	1.062341	-0.03568
1.504018	1.053977	0.01338	1.493712	1.056560	0.02137
1.514983	1.058759	0.00714	1.542664	1.054219	-0.01421
1.542664	1.054219	-0.01421	1.596004	1.053885	-0.05104
1.550375	1.054102	-0.019	1.514983	1.058759	0.00714
1.577696	1.062341	-0.03568			
1.591533	1.059088	-0.04635			
1.596004	1.053885	-0.05104			
1.601356	1.052807	-0.05511			
1.608914	1.060672	-0.05804			

Figure 9.7 illustrates the top 10 patterns, where dark and light colors represent 116 and 121 ID fuel assemblies, respectively. Additionally, the Manhattan distance between the two 116 ID assemblies are given. The Manhattan distance is defined as distance between two points measured along axes at right angles. In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is $|x1 - x2| + |y1 - y2|$.

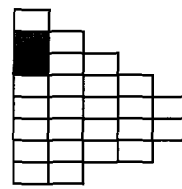
The distance between the two high-enriched fuel assemblies play an important role during the block-wise crossover operation. A pattern with smaller Manhattan distance is described with a shorter, low-order schema and has a higher probability of survival to the next generation.

100 search sessions were executed for the simplified case. The results are listed in Table 9.3.

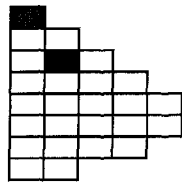
It is clearly seen that, loading patterns with small Manhattan distance was selected more frequently as predicted by the Schema Theorem. 46.4% of all selected patterns has a Manhattan distance of 1 and 30.8% is 2. Only 22.8% of patterns has a distance of 3.



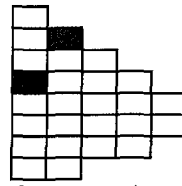
Pmax = 1.493712 (GA)
MD = 1



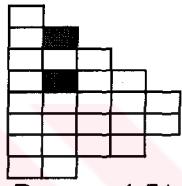
Pmax = 1.577696 (GA)
MD = 1



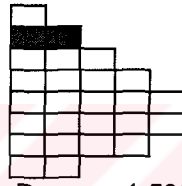
Pmax = 1.504018
MD = 3



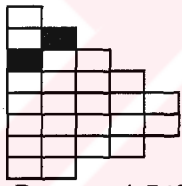
Pmax = 1.591533
MD = 3



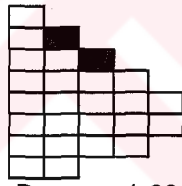
Pmax = 1.514983 (GA)
MD = 2



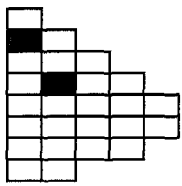
Pmax = 1.596004 (GA)
MD = 1



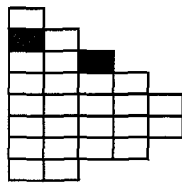
Pmax = 1.542664 (GA)
MD = 2



Pmax = 1.601356
MD = 2



Pmax = 1.550375
MD = 3



Pmax = 1.608914
MD = 3

MD = Manhattan distance

Figure 9.7 Patterns of the genetic algorithm results

This result is a consequence of the nature of genetic algorithms. In other words, it performs a coarse search that produces near optimum results.

Table 9.3 GA search results for the simplified case

p_{max}	k_{eff}	Fitness	Number of times being selected by GA	Manhattan distance
1.577696	1.062341	-0.03568	97	1
1.542664	1.054219	-0.01421	93	2
1.596004	1.053885	-0.05104	68	1
1.493712	1.056560	0.02137	67	1
1.514983	1.058759	0.00714	61	2
1.504018	1.053977	0.01338	52	3
1.550375	1.054102	-0.019	38	3
1.591533	1.059088	-0.04635	24	3

9.3.2 Calculations Using 7-Assembly Types

The procedure was repeated with the full 7 assembly types to demonstrate the technique for a realistic case. Genetic algorithm was used to generate 25 loading patterns that comply with the p_{max} and k_{eff} restrictions, which were set to 1.4 and 1.0, respectively.

The results that were archived at the end of the calculation are given in Table 9.4.

The number of generations required to accumulate the desired number of patterns was 24. It takes a few seconds to generate all the optimum population, since the calculations were carried out by the fast neural network algorithm. Because of this acceleration, most of the patterns were generated at the last several iterations.

Figure 9.8 shows the progress of average fitness value of the archived population along the generations. The change of the maximum fitness value in the population is illustrated in Figure 9.9.

Along the calculation process, the fitness values gradually increased and fitter patterns were accumulated in the archive.

The time that is required to generate one pattern gets shorter during the execution.

Table 9.4 Archived results of the sample calculations with the genetic algorithm

p_{max}	k_{eff}
1,383089	1,052585
1,323069	1,054277
1,394021	1,03145
1,360441	1,04375
1,395731	1,056448
1,397488	1,047677
1,273009	1,05247
1,326133	1,044959
1,388094	1,034512
1,393167	1,04824
1,370616	1,032716
1,385096	1,036185
1,392338	1,044205
1,383126	1,029877
1,397182	1,035324
1,344216	1,053761
1,360402	1,05681
1,362785	1,033751
1,307644	1,05218
1,251989	1,054144
1,352418	1,031406
1,381362	1,038221
1,331956	1,051878
1,356203	1,061095
1,382474	1,04554

This trend may be seen more clearly in Figure 9.10. The average of power fractions in the population decreased as the iteration number increased. Similarly, Figure 9.11 shows the decrease of the best maximum power fraction values of loading patterns with respect to the iteration number.

The genetic algorithm method gradually generated loading patterns that were fitter than the previous generation. In other words, the experience accumulated during the process is used to build better solutions, while the successive patterns generated by the simple rule-based method do not include any feedback from the previous experience.

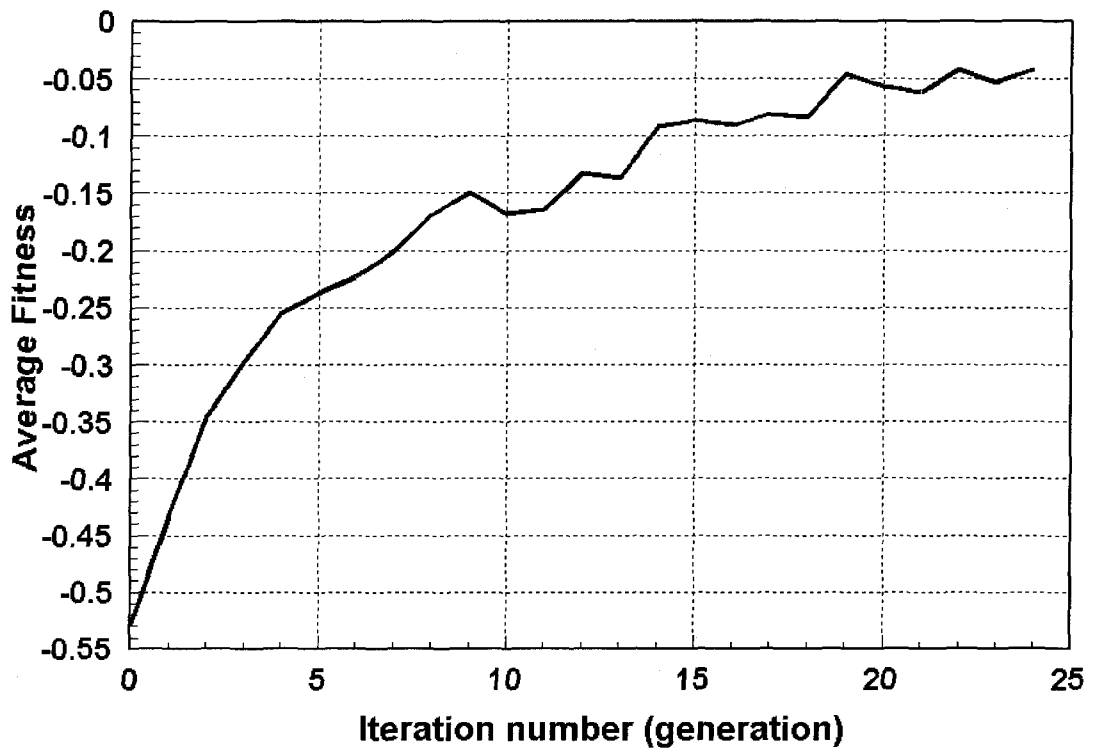


Figure 9.8 Average fitness value of the population along generations

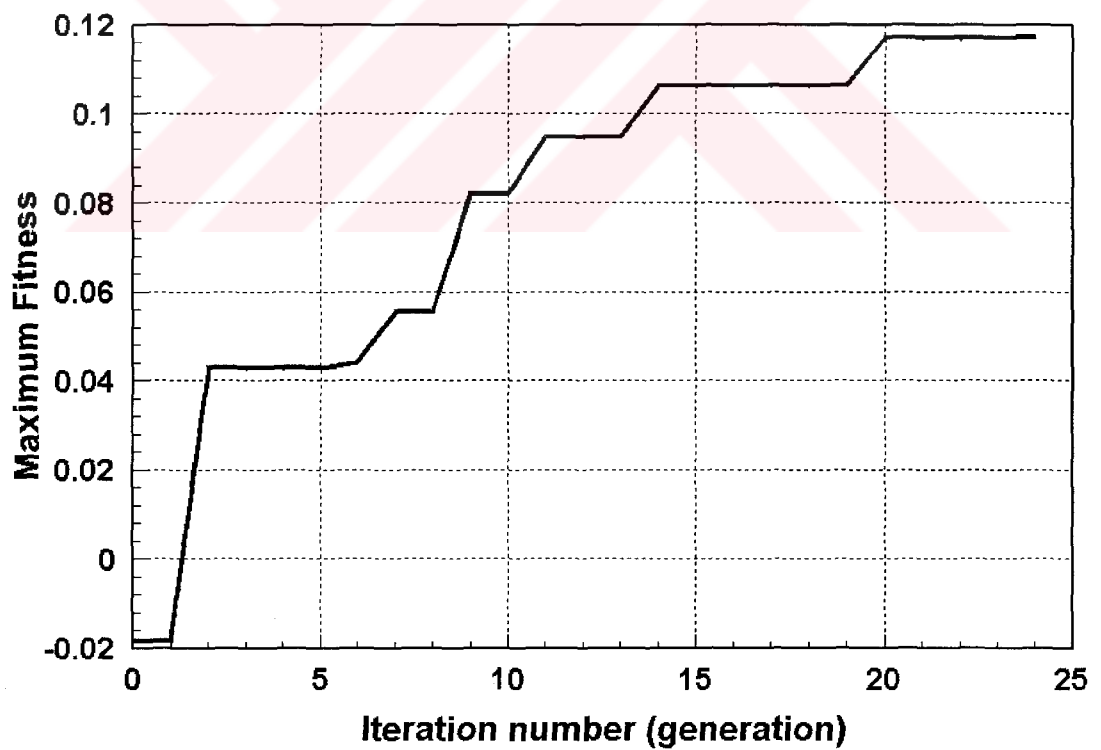


Figure 9.9 Maximum fitness value of the population along generations

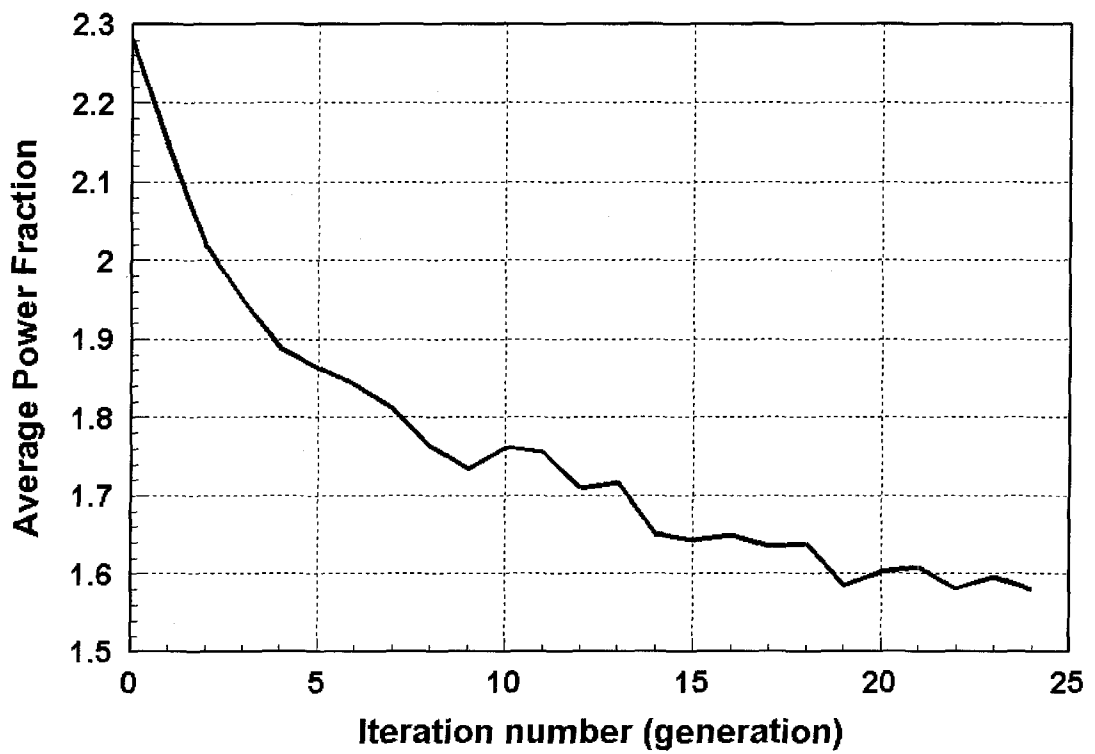


Figure 9.10 Average power fraction of the population

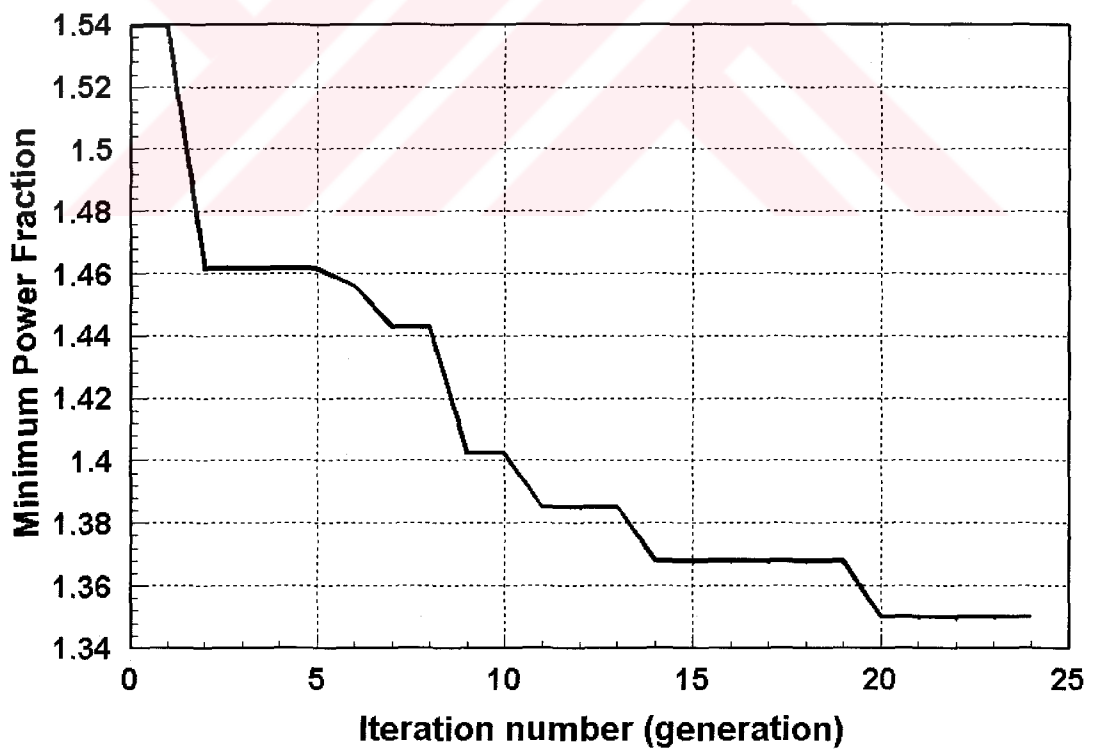


Figure 9.11 Minimum power fraction in the population

Thus, the simple rule based method produces completely independent solutions that cannot be fine-tuned to a better degree along the progress of the procedure. This is the superior property of the genetic algorithm over the simple rule-based one.

Ahnert et. Al (1990) presents a PWR benchmark for the Almaraz Nuclear Power Plant, whose core configuration was used in this work. It is interesting to compare the loading pattern used in the benchmark and some of the results obtained by the genetic algorithm (Table 9.5). The genetic algorithm generates some loading patterns that have higher fitness values.

Table 9.5 A PWR benchmark vs. GA results

	p_{max}	k_{eff}	Fitness
PWR Benchmark	1.281	1.013	0.157
GA			
Sample 1	1.273	1.052	0.175
Sample 2	1.252	1.054	0.189

10. THE *XCORE* PROGRAM

Computer program package *XCore* has been developed to generate and recommend optimum loading patterns complying with the restrictions imposed by the physical limitations of the reactor core.

It is a complete package including a graphical user interface to handle all the cumbersome details of in-core fuel management activities. The input data required by the calculations are fed through graphical forms to reduce user errors. Output is displayed by means of various graphics techniques to increase comprehension.

The program was written with the programming language C++. It runs under the operating system Linux on personal computers. *XCore* requires the X-Windows system.

The main window of the program is shown in Figure 10.1.

The *XCore* system consists of the following components: a graphical user interface, LEOPARD cell calculation code, MCRAC global core calculation code, a neural network algorithm implementation, simple rule-based random-search module, a genetic algorithm implementation (Figure 10.2).

All modules run in a separate thread to enable the graphical user interface to process the events received from the X-server while the system is calculating in the background.

The graphical user interface is responsible for displaying the core information in a visual format. A quarter of the core is sketched on the main program window. The fuel assemblies are displayed as colored boxes, which include the relevant numerical data. To represent three main distributions; namely 1) neutronics, 2) thermal-hydraulics, and 3) isotope information, there are different colors, which are graded to display the variations.

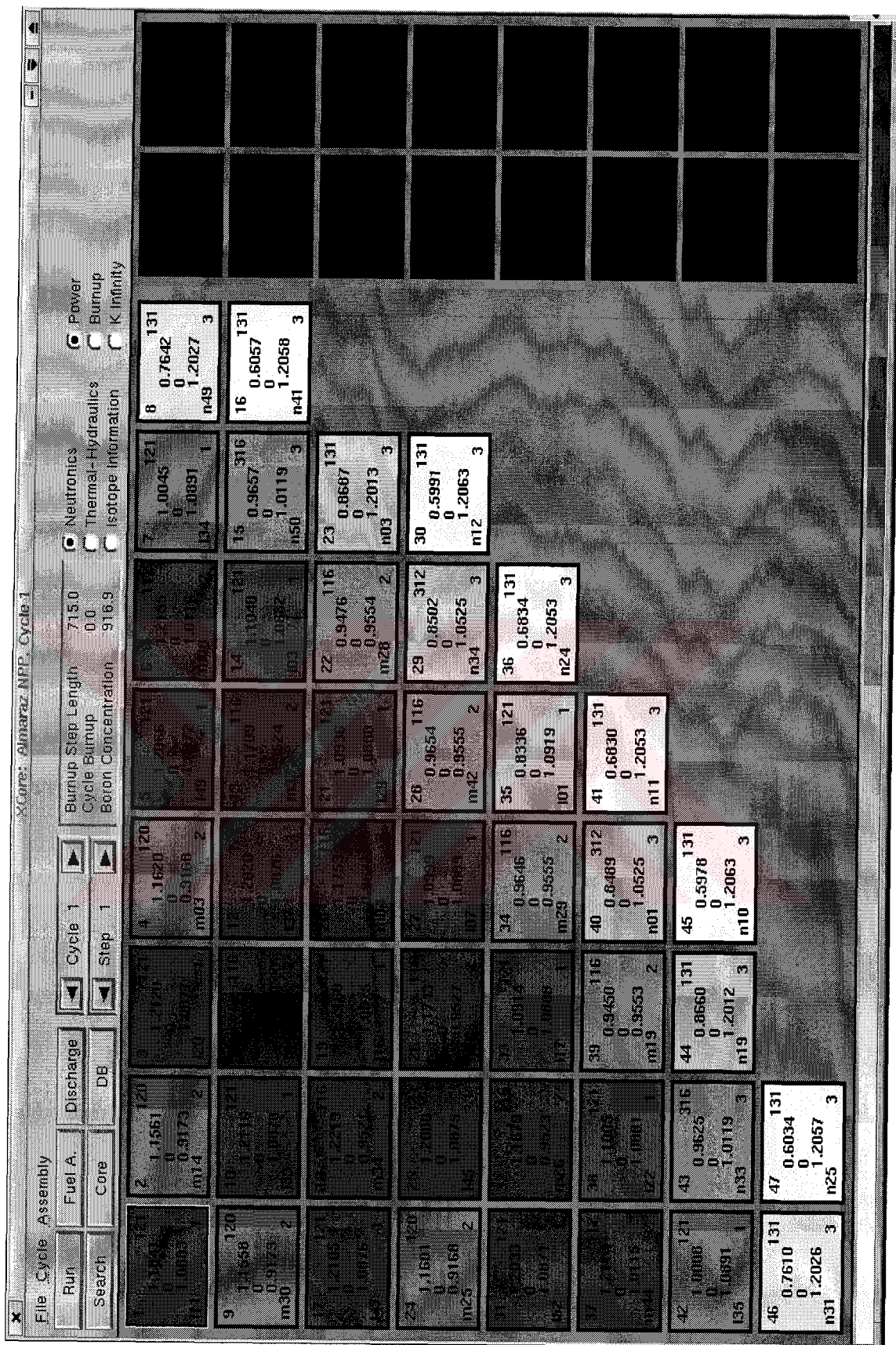


Figure 10.1 The main window of XCore

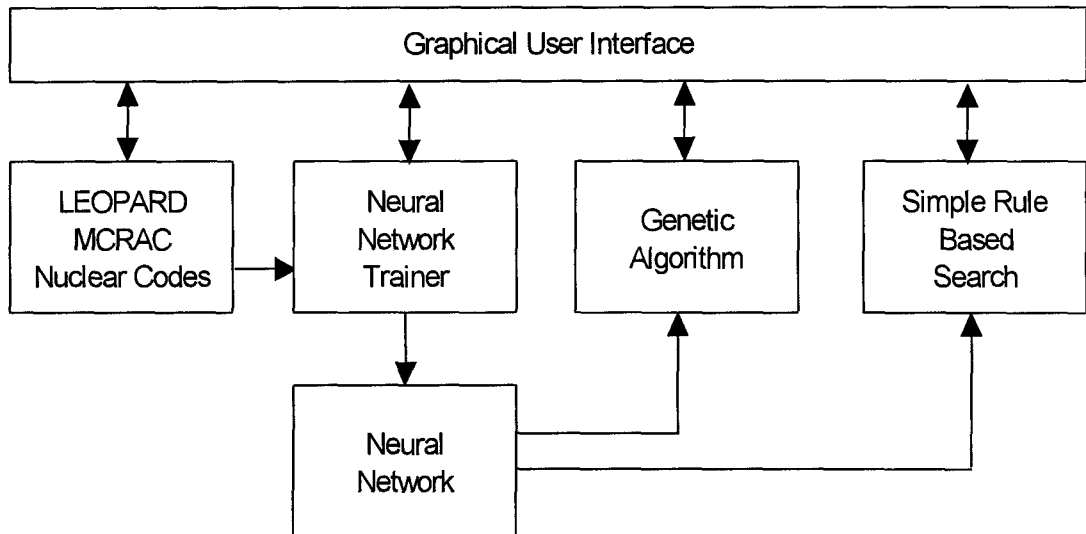


Figure 10.2 Program modules

These main titles are further subdivided into the following subtitles.

- 1) Neutronics: a) Power, b) Burnup, c) K infinity.
- 2) Thermal-Hydraulics: a) Fuel, b) Cladding, and c) Moderator temperatures.
- 3) Isotope Information: a) Uranium loss, b) Plutonium gain, c) Enrichment.

In addition, the color maps extends into cycle life represented with total core burnup. Color maps are refreshed at each time step.

The color map system also supports the multi-cycle design of the reactor core.

The reactor core may be designed from scratch using the tools of the graphical user interface. Fuel assembly types are defined and assigned to any fuel assembly. Clicking the right button of the mouse on the box representing it, accesses the information about a fuel assembly. In this case, a menu is displayed where several options are presented.

A manual fuel shuffling mechanism has been implemented. Pressing the left button of the mouse on a fuel assembly box may drag any fuel assembly. The fuel assembly may be dropped at any empty location, or if released on a location holding a fuel assembly they exchange places, the latter going to the assembly park. The loading

pattern obtained by shuffling fuel assemblies is re-calculated and a new color map is displayed automatically.

The codes LEOPARD and MCRAC are used for the cell and power calculations for the loading patterns given by the user. For the case of an optimal pattern search, the neural network estimates the core parameters.

Sample-loading patterns are generated by the neural network module and calculated by LEOPARD/MCRAC codes. Then, the neural network is trained using the obtained data. The simple rule-based random search and genetic algorithm modules predicts optimum loading patterns using information calculated by the neural network.

The simple rule-based random-search algorithm generates fitter loading patterns by using randomly generated patterns and applying some rules to restrict the search space. The user has to specify the fuel assembly types and their numbers that are to be used in the calculations. The resulting optimum loading patterns are presented to the user as in Figure 10.3. Any one of the patterns may be set for the reactor core.

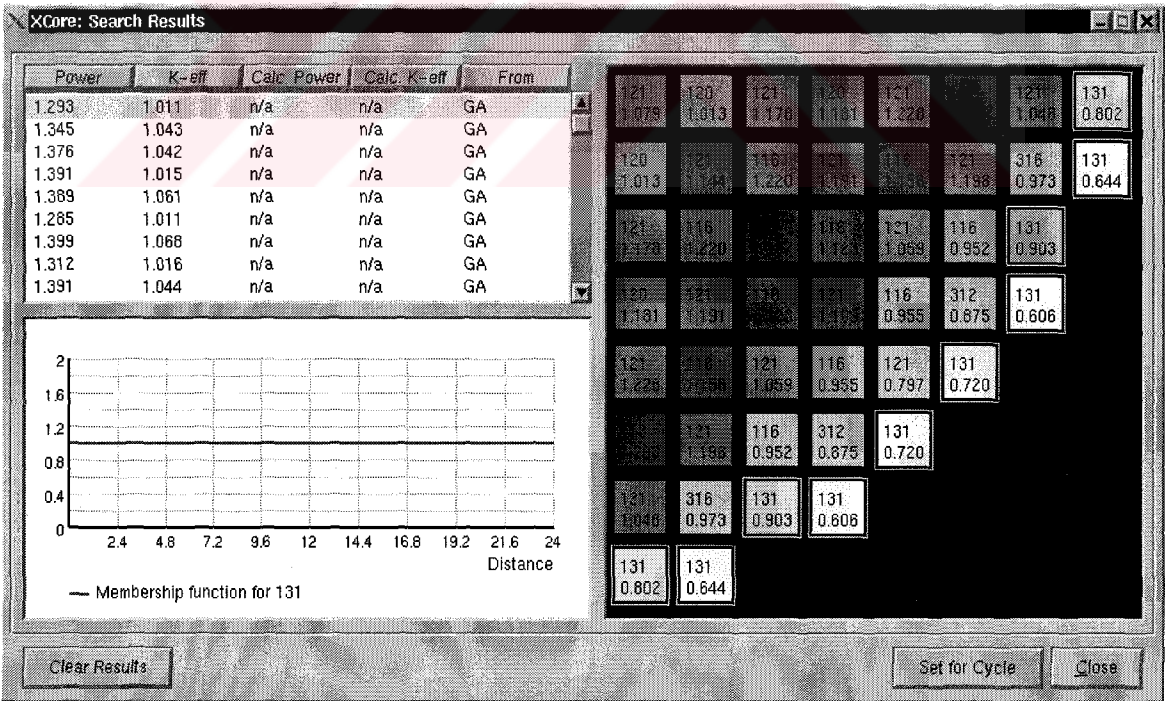


Figure 10.3 A sample *XCore* window that lists GA selected loading patterns

Similarly, the genetic algorithm produces loading patterns that are presented as more optimal. The user may decide to replace the current loading pattern with one of the generated ones.



11. CONCLUSIONS AND DISCUSSIONS

In this study, a program package, related to the activities of in-core fuel management for pressurized light water reactors, has been developed that generates and recommends optimum fuel loading patterns.

For this purpose, in the first stage of the work, the core of the Almaraz Nuclear Power Plant was modeled with the nuclear codes LEOPARD and MCRAC. Seven different fuel assembly types were defined and the two-group constants were generated by LEOPARD. The global core configuration calculations were carried out by the multi-group diffusion code MCRAC.

In this work, it was attempted to overcome one of the main problems of the power distribution calculations, namely, the slow and large core calculation codes. The power of neural networks was exploited for fast calculation in finding the appropriate loading patterns.

An artificial neural network was implemented to be used as a fast calculator of the core parameters. 2000 randomly generated loading patterns were evaluated as the data necessary for the training and validation of the neural network. MCRAC estimated the power distribution and effective multiplication factor k_{eff} for the sample loading patterns. A method has been developed called *the k_{eff} method* to construct the input vectors that are fed to the neural network.

At the next step, the neural network was trained with 500 sample loading patterns. The obtained weight set was tested with the remaining 1500 loading patterns. This weight set, which is valid for the seven-fuel assembly types defined in the LEOPARD model, was stored for later usage.

On an average personal computer, the global core calculation to obtain the power distribution using MCRAC takes about 15-20 seconds, while about a fraction of a second is required for the neural network to produce the result.

This property provides a user with the means to analyze a large number of loading patterns in relatively short time interval. In other words, a much larger search space is investigated compared with the conventional methods.

However, neural networks introduce an additional amount of error to the solution depending on the sufficiency of the training. During the calculations, the average error was estimated to be approximately 6% and the maximum error 15%.

Considering that the main purpose of this procedure is to estimate the core parameters of as many loading patterns as possible in a short time interval, it is acceptable to make use of a method with relatively rough estimates. Actually, this condition does not compromise the quality of final result, since the loading patterns obtained by *XCore* is accepted as the optimum loading pattern for the problem, then they are tested with a licensed nuclear code.

At the next stage of the study, optimum loading patterns were searched with a simple rule-based random-search and the genetic algorithm methods. The generated candidate loading patterns were rated according to their fitness values to find the best solutions. The core parameters were calculated by the neural network.

The successes of the two methods were measured and compared with respect to the number of iterations required to obtain the same number of candidate loading patterns. It was seen that genetic algorithm is a more powerful optimizer. The number of iterations is fewer and the computation time required to generate the desired number of near-optimum loading patterns are less in the case of the genetic algorithm.

The random search method does not guarantee any solution. Some executions were not successful to provide adequate results. However, genetic algorithm uses the experience gained during the iterations in converging to the right pattern. Its ability to converge is the main power of the genetic algorithms.

The membership function $\mu(x)$ considerably enhances the simple rule-based algorithms search capability. The computation time is reduced and more solutions are obtained in a given time interval. However, it was necessary to include previous expert knowledge to implement the enhancement. The high and low enriched fuel

assemblies that enable the construction of better loading patterns should be known before the calculation starts.

The genetic algorithm does not need any expert knowledge to carry out the procedure. This provides the user with a less restricted method. It is to say the genetic algorithm is more general and searches a broader search space. For instance, in the case of an improper selection of high and low enriched fuel assemblies it is unlikely that the simple rule-based algorithm will be able to find desired loading patterns.

The computer program package *XCore* was developed to implement this procedure for the optimization of loading patterns. As a user-friendly program, it provides the user with an easy to use graphical user interface that helps with the tasks of in-core fuel management.

As a final word, it can be stated that, the implementation in *XCore* is able to advise loading patterns complying with the restrictions by using the simple rule-based random-search method, or more successfully, by the genetic algorithm.

If properly configured, a neural network is a fast and relatively accurate tool to calculate nuclear power core parameters.

It appears that evolutionary approaches will be more commonplace in the coming years. Emerging new computational paradigms will gradually free the domain experts from regular daily chores and spare them extra brain power to spend on their specialty, which in turn will accelerate the advancement of the science and technology.

Further Study

The implemented neural network algorithm requires the network parameters and the number of nodes to be external input data from the user. This is an extra burden on the user. Implementing an adaptive network may solve this inconvenience. For instance, an automatic pruning algorithm that reduces the number of hidden nodes with respect to the problem, which improves generalization and shortens the time it takes to train the network may be developed and incorporated into *XCore*.

Another improvement to the algorithm may be based on implementing a procedure to adapt the values of network parameters α and η with respect to the network error change. This may speed up the network training.

Use of genetic algorithms as a tool to train the neural network architecture is possible. There have been several efforts to estimate the network parameters by genetic algorithms. One is based on encoding the network architecture directly into a genetic algorithm chromosome. Another one evolves grammars that can be used to develop network architectures.

On the other hand, currently, the implemented program searches for an optimum-loading pattern considering only the beginning of cycle (BOC) power distribution and effective multiplication factor. A more realistic approach would be to optimize for more variables of the system under consideration, it is to say, the reactor core.

A few examples may be given as follows:

- The maximization of the cycle length by increasing the discharge burnup of the batches. For this purpose, an artificial neural network may be trained to account for the burnup during a cycle of the reactor operation. The evolution may be incorporated into the network by simply applying the *backpropagation through time* algorithm. A better approach would be to employ a recurrent architecture, which needs the implementation of a new training algorithm.
- The maximization of the end of cycle (EOC) power distribution and k_{eff} , or the reactivity. This requires the burnup of the fuel assemblies to be calculated. Once the burnup is known, it will be possible to extend the analysis to multiple cycles.
- The minimization of the feed enrichment. Using fresh fuel assemblies with lower enrichment fuel reduces the overall cost. Decreasing the number of fresh fuel assemblies result to the same effect.
- The minimization of the burnable poison inventory.

Another improvement to the system would be to minimize the fast neutron leakage from the core. This is an important consideration to limit the damage to the pressure

vessel caused by the fast neutron flux. A loading pattern that reduces the leakage would be more preferable.

In the calculations, a more flexible burnable poison handling may be implemented. Currently, the burnable poison information is contributed to the solution only through the cross sections generated by LEOPARD. A robust system would also recommend burnable poison rod number and their positions for the fuel assemblies including them.

Finally a word of caution is due: in the safety critical applications, all the results based on the emerging computational intelligence techniques are subject to rigorous scrutiny by the domain experts with the well-established traditional or licensed methods, before being implemented.



REFERENCES

- Ahn,D.H. and Levine,S.H.**, 1985. Automatic Optimized Reload and Depletion Method for a Pressurized Water Reactor, *Nuclear Technology*, **71**, 535-546.
- Ahnert,C., Aragones,J.M. and Merino,F.**, 1990. PWR Benchmark Parameters, Almaraz Nuclear Power Plant, IAEA, Vienna.
- Al-Gutifan and Faisal Husni**, 1991. An Application of Artificial Intelligence and Neural Networks to In-Core Fuel Management, *M.Sc. Thesis*, The University of Tennessee, Knoxville.
- Aragones,J.M., Martinez-Val,J.M. and Corella,M.R.**, 1977. Core-state Models for Fuel Management of Equilibrium and Transition Cycles in Pressurized Water Reactors, *Nuclear Technology*, **34**, 398-411.
- Barlett,E.B. and Uhrig,R.E.**, 1992. Nuclear Power Plant Status Diagnostics Using an Artificial Neural Network, *Nuclear Technology*, **97**, 272-280.
- Brent, R.**, 1991, Fast Training Algorithms for Multilayer Neural Nets, *IEEE Transactions on Neural Networks*, **2**, 346-354.
- Chapot, J.L.C., Silva, F.C.D. and Schirru, R.**, 1999. A New Approach to the Use of Genetic Algorithms to Solve The Pressurized Water Reactor's Fuel Management Optimization Problem, *Annals of Nuclear Energy*, **26**, 641-655.
- Cheon,S.W. and Chang,S.H.**, 1993. Application of Neural Networks to a Connectionist Expert System for Transient Identification in Nuclear Power Plants, *Nuclear Technology*, **102**, 177-191.

- Cullen, D.E., Muranaka, R. and Schmidh, J., 1986.** Applications in Nuclear Data and Reactor Physics, World Scientific Publishing, PA, USA.
- DeChaine, M.D. and Feltus, M.A., 1995.** Nuclear Fuel Management Optimization Using Genetic Algorithms, *Nuclear Technology*, **111**, 109-114.
- DeChaine, M.D. and Feltus, M.A., 1996.** Fuel Management Optimization Using Genetic Algorithms and Expert Knowledge, *Nuclear Science and Engineering*, **124**, 188-196.
- Driscoll, M.J., Downar, T.J. and Pilat, E.E., 1990.** The Linear Reactivity Model for Nuclear Fuel Management, American Nuclear Society, La Grange Park, Illinois, USA.
- Fowler, T.B. and Tobias, T.B., 1967.** EXTERMINATOR-II, A Fortran Code for Solving Multi-group Diffusion Equation in Two-Dimensions, ORNL-4078, Oak Ridge National Laboratory.
- Francois, J.L. and Lopez, H.A., 1999.** SOPRAG: A System for BWR's Reload Pattern Optimization Using Genetic Algorithms, *Annals of Nuclear energy*, **26**, 1052-1063.
- Galperin, A. and Nissan, E., 1988.** Application of a Heuristic Search Method for Generation of Fuel Reload Configuration, *Nuclear Science and Engineering*, **99**, 343-352.
- Galperin, A., Kimhi, S. and Segev, M., 1989.** A Knowledge-Based System for Optimization of Fuel Reload Configurations, *Nuclear Science and Engineering*, **102**, 43-53.
- Galperin, A. and Kimhy, Y., 1991.** Application of Knowledge-Based Methods to In-Core Fuel Management, *Nuclear Science and Engineering*, **109**, 103-110.
- Goldberg, D.E., 1989.** Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company Inc., Massachusetts.

- Guo,Z. and Uhrig,R.E.,** 1992. Use of Artificial Neural Networks to Analyze Nuclear Power Plant Performance, *Nuclear Technology*, **99**, 36-42.
- Haupt,R.L. and Haupt,S.E.,** 1998. Practical Genetic Algorithms, Wiley-Interscience Publication, New York.
- Hertz,J., Krogh,A. and Palmer,R.G.,** 1991. Introduction to the Theory of Neural Computation, Addison Wesley Publishing Company, The Advanced Book Program, Redwood City, CA.
- Hobson,G.H. and Turinsky,P.J.,** 1986. Automatic Determination of Pressurized Water Reactor Core Loading Patterns that Maximize Beginning-of-Cycle Reactivity within Power-peaking and Burnup Constraints, *Nuclear Technology*, **74**, 5-13.
- Holland, John H.,** 1992. Adaptation in Natural and Artificial Systems, MIT Press, Massachusetts.
- Hoshino,T.,** 1972. In-core Fuel Management Optimization by Heuristic Learning Technique, *Nuclear Science and Engineering*, **49**, 59-71.
- Hoshino,T.,** 1978. Optimum Fuel Loading and Operation Planning for Light Water Reactor Power Stations. Part I: Pressurized Water Reactor Case Study, *Nuclear Technology*, **39**, 46-61.
- Jouse,W.C. and Williams,J.G.,** 1993. Safety Control of Nuclear Power Operations Using Self-Programming Neural Networks, *Nuclear Science and Engineering*, **114**, 42-54.
- Karnin,E.,** 1990. A Simple Procedure for Pruning Back-Propagation Trained Neural Networks, *IEEE Transactions on Neural Networks*, **1**, 239-242.
- Kavaklıoğlu,K. and Upadhyaya,B.R.,** 1994. Monitoring Feedwater Flow Rate and Component Thermal Performance of Pressurized Water Reactors by Means of Artificial Neural Networks, *Nuclear Technology*, **107**, 112-123.

- Kim, H.G., Chang, S.H. and Lee, B.H.,** 1993. Pressurised Water Reactor Core Parameter Prediction Using an Artificial Neural Network, *Nuclear Science and Engineering*, **113**, 70-76.
- Kim, H.G., Chang, S.H. and Lee, B.H.,** 1993. Optimal Fuel Loading Pattern Design Using an Artificial Neural Network and a Fuzzy Rule-Based System, *Nuclear Science and Engineering*, **115**, 152-163.
- Kim, H.K. and Lee, S.H.,** 1993. Neural Network Model for Estimating Departure from Nucleate Boiling Performance of a Pressurized Water Reactor Core, *Nuclear Technology*, **101**, 111-121.
- Kim, K. and Barlett, E.B.,** 1994. Error Prediction for a Nuclear Power Plant Fault-Diagnostic Advisor Using Neural Networks, *Nuclear Technology*, **108**, 283-296.
- Kim, Y.J., Downar, T.J. and Sesonske, A.,** 1987. Optimization of Core Reload Design for Low-Leakage Fuel Management in Pressurized Water Reactors, *Nuclear Science and Engineering*, **96**, 85-101.
- Kosko, Bart,** 1992, *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ.
- Kropaczek, D.J. and Turinsky, P.J.,** 1991. In-Core Nuclear Fuel Management Optimization for Pressurized Water Reactors Utilizing Simulated Annealing, *Nuclear Technology*, **95**, 9-31.
- Lamarsh, J.R.,** 1972. *Introduction to Nuclear Reactor Theory*, Addison-Wesley Publishing, Massachusetts.
- Levine, S.H.,** 1982. In-Core Fuel Management for the Course on Operational Physics of Power Reactors, in *Proceedings on Course on Operational Physics of Power Reactors*, pp. 351-450, International Centre for Theoretical Physics, Trieste.

- Levine, S.H.**, 1992. History and Status of Reloading Techniques for Light Water Reactors, *IAEA Research Coordination Meeting on In-core Fuel Management*, Vienna.
- Louis, Sushil J.**, 1993. Genetic Algorithms As a Computational Tool for Design, *Ph.D. Thesis*, Indiana University.
- Lysenko, M.G., Wong, H. and Maldonado, G.I.**, 1999. Predicting Neutron Diffusion Eigenvalues with a Query-Based Adaptive Neural Architecture, *IEEE Transactions on Neural Networks*, **10**, 790-800.
- Maldonado, G.I. and Turinsky, P.J.**, 1995. Application of Non-linear Nodal Perturbation Theory to Nuclear Fuel Reload Optimization, *Nuclear Technology*, **110**, 198-219.
- Mitchell, Melanie**, 1996. An Introduction to Genetic Algorithms, MIT Press, Massachusetts.
- Moon, H.**, 1988. Development of a Multi-cycle Scoping Code Package for the Fuel Management in Pressurized Water Reactors, *Master of Science Thesis*, Department of Nuclear Engineering, Pennsylvania State University.
- Ohga, Y. and Seki, H.**, 1993. Abnormal Event Identification in Nuclear Power Plants Using a Neural Network and Knowledge Processing, *Nuclear Technology*, **101**, 159-167.
- Parks, G. T.**, 1996. Multiobjective Pressurized Water Reactor Reload Core Design by Nondominated Genetic Algorithm Search, *Nuclear Science and Engineering*, **124**, 178-187.
- Pereira, C.M.D.N., Schirru, R. and Martinez, A.S.**, 1999. Basic Investigations Related to Genetic Algorithms in Core Designs, *Annals of Nuclear Energy*, **26**, 173-193.
- Petrovic, B.G.**, 1990. In-Core Fuel Management: PWR Core Calculations Using MCRAC, *Workshop On Reactor Calculations For Applications In Nuclear Technology*, Zagreb, Yugoslavia.

- Quist,A.J., Vangeement,R. and Hoogenboom,J.E.,** 1999. Application of Non-Linear Optimization to Reactor Core Fuel Reloading, *Annals of Nuclear Energy*, **26**, 423-448.
- Roh,M., Cheon,S. and Chang,S.,** 1991. Power Prediction in Nuclear Power Plants Using a Back-Propagation Learning Neural Network, *Nuclear Technology*, **94**, 270-278.
- Sauar,T.O.,** 1971. Application of Linear Programming to In-Core Fuel Management Optimization in Light Water Reactors, *Nuclear Science and Engineering*, **46**, 274-283.
- Sauer,I.L. and Driscoll,M.J.,** 1985. A Core Reload Pattern and Composition Optimization Methodology for Pressurized Water Reactors, Department of Nuclear Energy internal report, MIT.
- Shatilla,Y.A., Little,D.C., Penkrot,J.A. and Holland,R.A.,** 2000. In-core Fuel Management With Biased Multiobjective Function Optimization, *Nuclear Technology*, **130**, 282-295.
- Stover, R.L. and Sesonke, A.,** 1969. Optimization of BWR Fuel Management Using an Accelerated Exhaustive Search Algorithm, *J. Nuclear Energy*, **23**, 673.
- Stover,R.L.,** 1968. Optimized Scatter-Loading of A Nuclear Power Reactor, *PhD Thesis*, Purdue University.
- Suh,J.S. and Levine,S.H.,** 1990. Optimized Automatic Reload Program for Pressurized Water Reactors Using Simple Direct Optimization Techniques, *Nuclear Science and Engineering*, **105**, 371-382.
- Suzuki,A. and Kiyose,R.,** 1971. Application of Linear Programming to Refueling Optimization for Light Water Moderated Power Reactors, *Nuclear Science and Engineering*, **46**, 11-130.

- Takeda,M. and Goodman,J.W.,** 1986. Neural Networks for Computation: Number Representations and Programming Complexity, *Applied Optics*, **25**, 3033-3046.
- Tambouratsiz,T. and Antonopoulos-Domis,M.,** 1999. On-Line Stability Monitoring of BWR's Using Artificial Neural Networks, *Annals of Nuclear Energy*, **26**, 1287-1302.
- Tobias, M.L. and Fowler, T.B.,** 1962. The EQUIPOSE Method, A simple Procedure for Group Diffusion Calculations in Two and Three-Dimensions, *Nuclear Science and Engineering*, **12**, 513.
- Toshinsky, V.G., Sekimoto, H. and Toshinsky, G.I.,** 1999. Multiobjective Fuel Management Optimization for Self-Fuel-Providing LMFBR Using Genetic Algorithms, *Annals of Nuclear Energy*, **26**, 783-802.
- Toshinsky,V.G., Sekimoto,H. and Toshinsky,G.I.,** 2000. A Method to Improve Multiobjective Genetic Algorithm Optimization of a Self-Fuel-Providing LMFBR by Niche Induction Among Non-dominated Solutions, *Annals of Nuclear Energy*, **27**, 397-410.
- Turinsky,P.J.,** 1999. Mathematical Optimization of Incore Nuclear Fuel Management Decisions, Status and Trends, *ATW-Internationale Zeitschrift fur Kernenergie*, **44**, 454.
- Uhrig,R.E.,** 1991. Potential Application of Neural Networks to the Operation of Nuclear Power Plants, *Nuclear Safety*, **32**, 68-79.
- Van Geemert,R., Quist,A.J., Hoogenboom,J.E. and Gibcus,H.P.M.,** 1998. Research Reactor In-Core Fuel Management Optimization by Application of Multiple Cyclic Interchange Algorithms, *Nuclear Engineering and Design*, **186**, 369-377.
- Vose,M.D.,** 1998. The Simple Genetic Algorithm, Foundation and Theory, Cambridge, Mass. MIT Press.

- Wade,D.C. and Terney,W.B.,** 1971. Optimal Control of Nuclear Reactor Depletion, *Nuclear Science and Engineering*, **45**, 199-217.
- Yamamoto, Akio,** 1997. A Quantitative Comparison of Loading Pattern Optimization Methods for In-Core Fuel Management of PWR, *Journal of Nuclear Science and Technology*, **34**, 339-347.
- Yung-An,C., Chuan-Wen,H. and Chang-An,S.,** 1986. A Theory of Fuel Management via Backward Diffusion Calculation, *Nuclear Science and Engineering*, **93**, 78-87.
- Zurada, Jacek M.,** 1990. Introduction to Artificial Neural Systems, West Publishing Compony, New York.



APPENDIX A

The Haling Principle

The Haling Principle is based on an iterative procedure to find a refueling pattern and a dynamic control pattern that achieve a constant power distribution during the core depletion (Levine, 1982). The process starts with the end of cycle (EOC) core.

The main target of the principle is to maintain a power peaking factor as low as possible. For this purpose, the use of burnable poisons, or any other means, are necessary to match the burnup of the fuel.

Actually, the depletion trajectory specified by the principle can never be attained precisely, but it establishes a goal for the refueling engineer to meet. The burnup distribution change that takes place during the reactor cycle is referred to as the depletion trajectory.

The goal of the Haling Principle is sketched in Figure A.1 where the maximum normalized power (power fraction p_{max}) is plotted as a function of core burnup.

Here, curve A, B and C represent different cores with maximum power fraction as displayed. The curve A follows the Haling principle while B or C does not.

The methods for computing the reload pattern and depletion trajectory to follow curve A is described below.

For a power reactor, the power distribution $P(\underline{r})$, where $\underline{r}$ is the space vector, may be defined as

$$P(\underline{r}) = \bar{P}A(\underline{r}) \tag{A.1}$$

and the burnup distribution $Bu(\underline{r})$ as

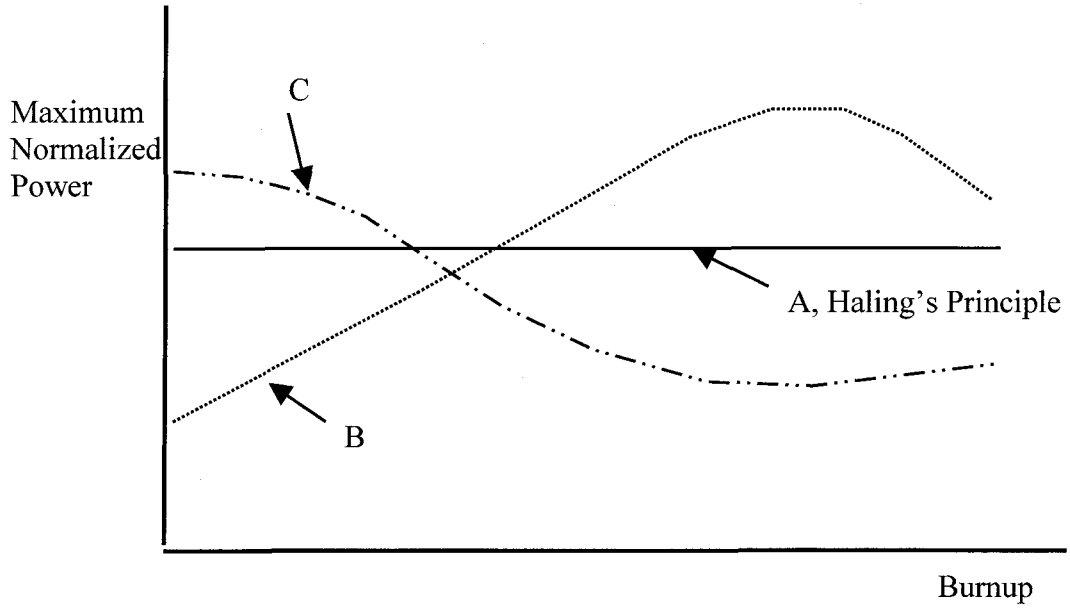


Figure A.1 Maximum normalized power as a function of burnup

$$Bu(\underline{r}) = \overline{Bu} A(\underline{r}) = \frac{\overline{Bu}}{\overline{P}} P(\underline{r}) \quad (\text{A.2})$$

where $\overline{P}$ is the average power density and $\overline{Bu}$ is the average burnup in the core, and $A(\underline{r})$ is a normalized shape function.

The Haling principle attempts to maintain $A(\underline{r})$ constant during the reactor cycle. During this process, $A(\underline{r})_{\max}$ is minimized.

The steady state two-group diffusion equation for the fast neutron group is given as

$$\nabla \cdot D_1(\underline{r}) \nabla \Phi_1(\underline{r}) - \Sigma_1(\underline{r}) \Phi_1(\underline{r}) + \frac{\nu(\underline{r}) P(\underline{r})}{\lambda G(\underline{r})} = 0 \quad (\text{A.3})$$

where $G(\underline{r})$ is the energy generated per fission. For a critical reactor $\lambda=1$, and using Eqs. (A.2) and (A.3) the following equation is obtained.

$$\nabla \cdot D_1(\underline{r}) \nabla \Phi_1(\underline{r}) - \Sigma_1(\underline{r}) \Phi_1(\underline{r}) + \nu(\underline{r}) \frac{\overline{P} Bu(\underline{r})}{\overline{Bu} G(\underline{r})} = 0 \quad (\text{A.4})$$

Then a guess is made for $A(\underline{r})$ and $\overline{Bu}$ through which $P(\underline{r})$ and $Bu(\underline{r})$ are evaluated. The spatial temperatures, moderator density, xenon, and the nuclear properties are all evaluated to correspond to the correct $P(\underline{r})$. The next step is to solve Eq. (A.4) for the flux distribution $\Phi_1(\underline{r})$ by either inverting the matrix or using an iteration method.

For the large 3-D calculations the iteration method is most efficient. Once the $\Phi_1(\underline{r})$ distribution is determined, the thermal flux can be calculated in a similar manner using the thermal neutron group diffusion equation

$$\nabla \cdot D_2(\underline{r}) \nabla \Phi_2(\underline{r}) - \Sigma_2 \Phi_2(\underline{r}) + P(\underline{r}) \Sigma_1(\underline{r}) \Phi_1(\underline{r}) = 0 \quad (\text{A.5})$$

With both $\Phi_1(\underline{r})$ and $\Phi_2(\underline{r})$ determined, a new more correct power distribution $P^{(l)}(\underline{r})$ is computed using

$$P^{(l)}(\underline{r}) = \overline{P} [\Sigma_{f1}(\underline{r}) \Phi_1(\underline{r}) + \Sigma_{f2}(\underline{r}) \Phi_2(\underline{r})] G(\underline{r}) \quad (\text{A.6})$$

where the superscript identifies the iteration number. The process is repeated in an iterative fashion calculating a new improved $P^n(\underline{r})$ after each n th iteration. Because the equations are non-linear, the use of acceleration techniques to speed up convergence helps to shorten the time to compute the reload pattern.

The procedures used to assign individual fuel assemblies their new position, determine the number and enrichment of new fuel assemblies, and define other needs for the reload configuration depends on the implementation. However, there are some standard rules to follow.

Usually, scatter-loading is employed to reduce power peaking and new fuel is loaded near the center of the core.

APPENDIX B

Sample power distribution calculated by the neural network.

121 2,312 2,238 3,45					ID Calculated Target Error (%)
121 2,05 2,049 0,071	121 1,818 1,777 2,278				
121 1,68 1,66 1,298	116 1,373 1,372 0,202	120 1,246 1,204 3,539			
131 1,691 1,705 0,829	316 1,303 1,337 2,557	131 1,553 1,589 2,277	112 1,284 1,35 4,872		
116 0,969 0,941 3,012	116 0,939 0,934 0,669	116 1,035 1,029 0,521	121 1,149 1,16 0,94	131 1,288 1,322 2,536	
312 0,854 0,897 4,748	121 0,881 0,847 3,995	121 0,881 0,855 2,992	121 0,811 0,799 1,466	131 0,733 0,762 3,828	
121 0,667 0,597 11,593	121 0,634 0,605 4,722	131 0,612 0,639 4,305	120 0,312 0,314 0,712		
116 0,253 0,244 3,617	121 0,238 0,238 0,274				

Figure B.1 Power distribution calculated by the neural network

Maximum and average neural network errors in percent for all core locations. These results were obtained by the 1500 randomly generated loading patterns that were not used in the training session.

17,55 4,41					Max. Error Avg. Error (%)
18,57 5,73	18,97 5,37				
18,57 6,46	18,19 2,87	12,32 3,15			
15,75 3,09	14,14 3,1	8,17 2,47	12,75 3,39		
10,15 2,46	7,52 1,84	12,71 3,89	14,52 4,59	18,16 4,45	
14,77 2,99	10,95 2,41	13,64 3,85	17,43 4,76	18,8 5,69	
18,54 4,31	18,75 4,01	17,29 4,86	18,72 5,97		
18,98 6,14	18,92 6,67				

Figure B.2 Neural network errors for power distribution

APPENDIX C

The following 3 figures display sample loading patterns obtained by the simple rule-based method. The graph illustrates the final $\mu(x)$ evolved at the end of a single session. The pattern representation is the last loading pattern generated during this session.

$P_{\max} = 1.285, K_{\text{eff}} = 1.033$

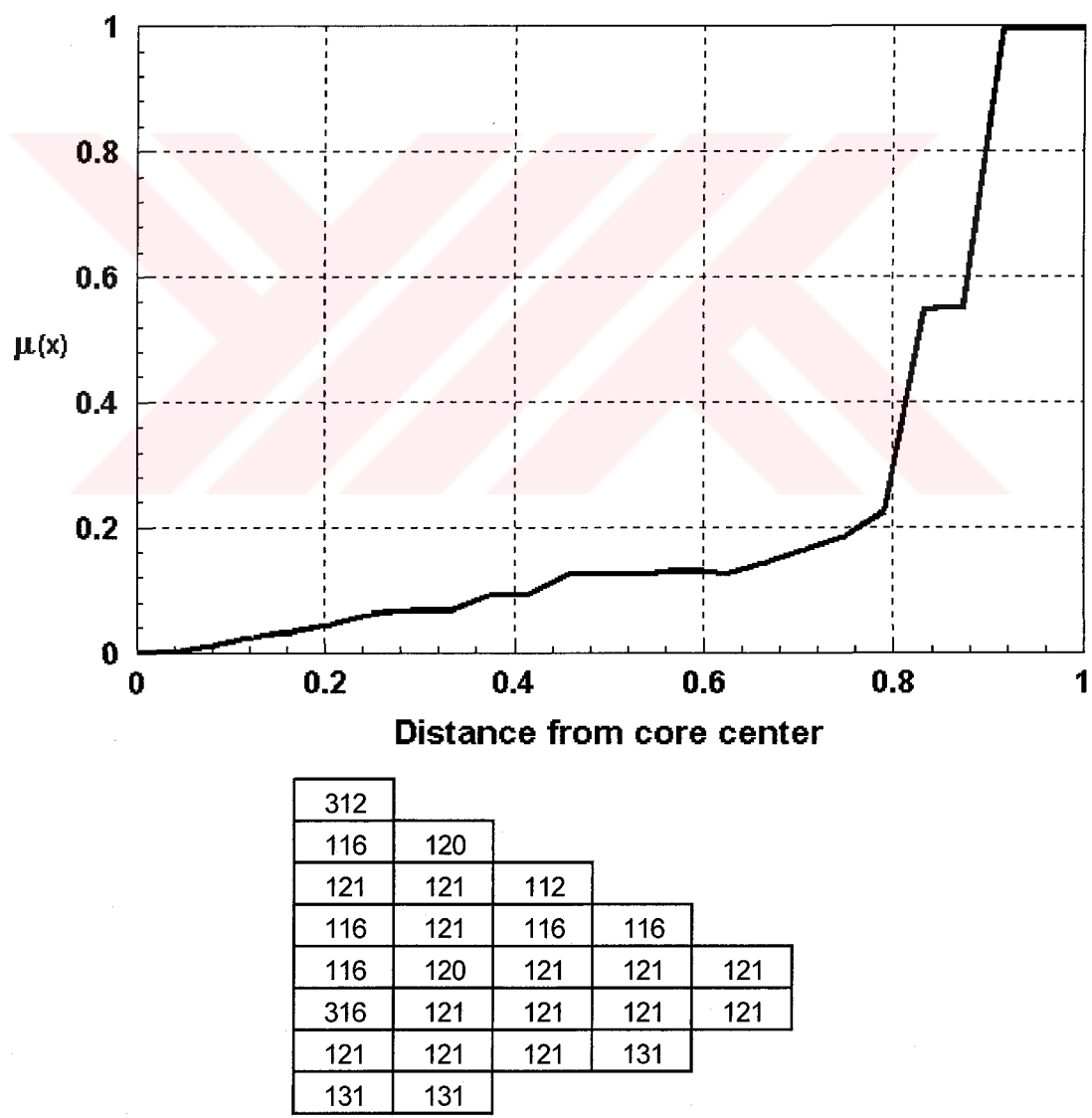


Figure C.1 $\mu(x)$ and fuel assembly distribution for an optimum loading pattern

Sample Problem Execution: Simple rule-based random method results 2

$P_{\max} = 1.234, K_{\text{eff}} = 1.038$

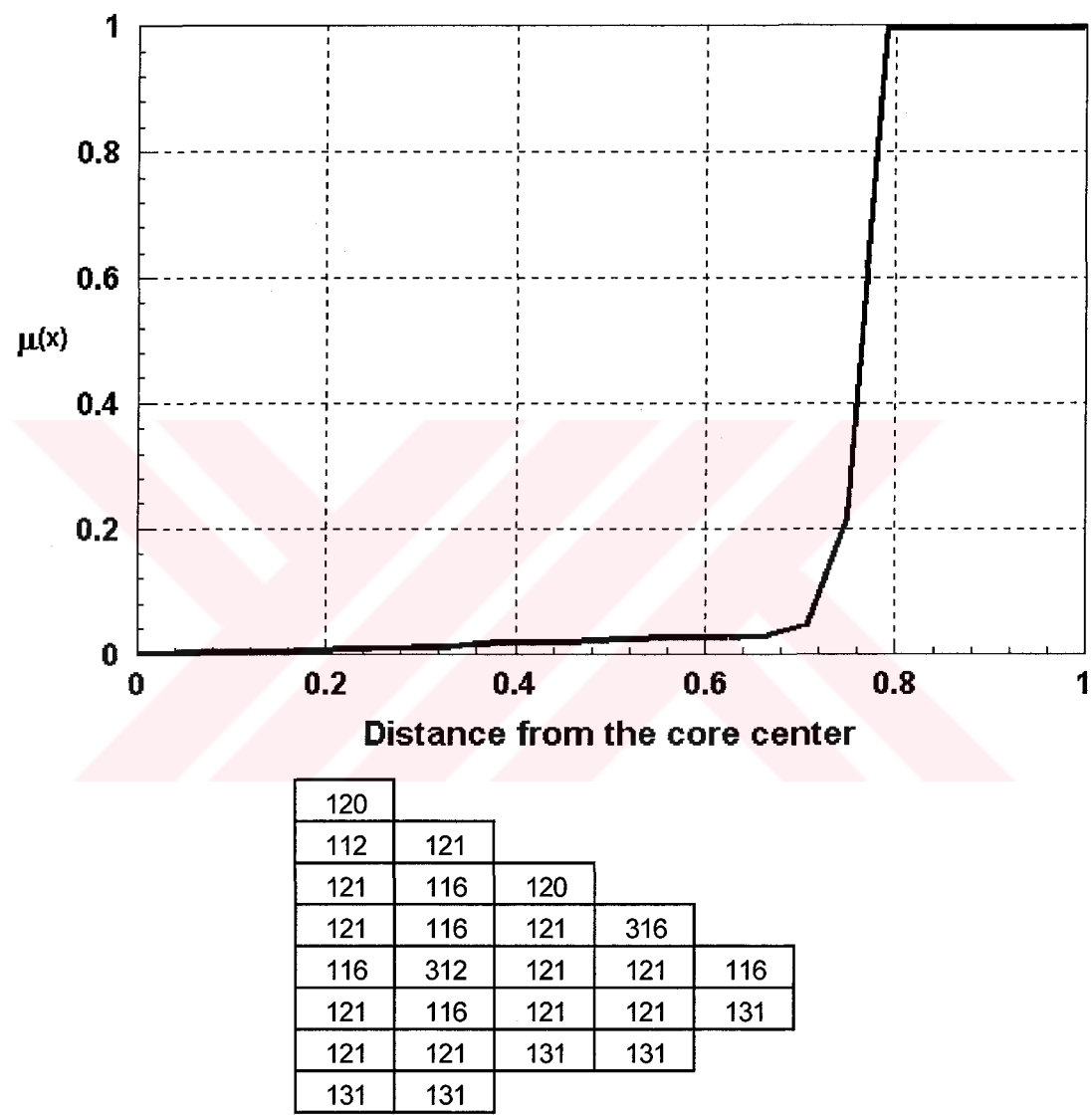


Figure C.2 $\mu(x)$ and fuel assembly distribution for an optimum loading pattern

Sample Problem Execution: Simple rule-based random method results 3

$P_{\max} = 1.288, K_{\text{eff}} = 1.035$

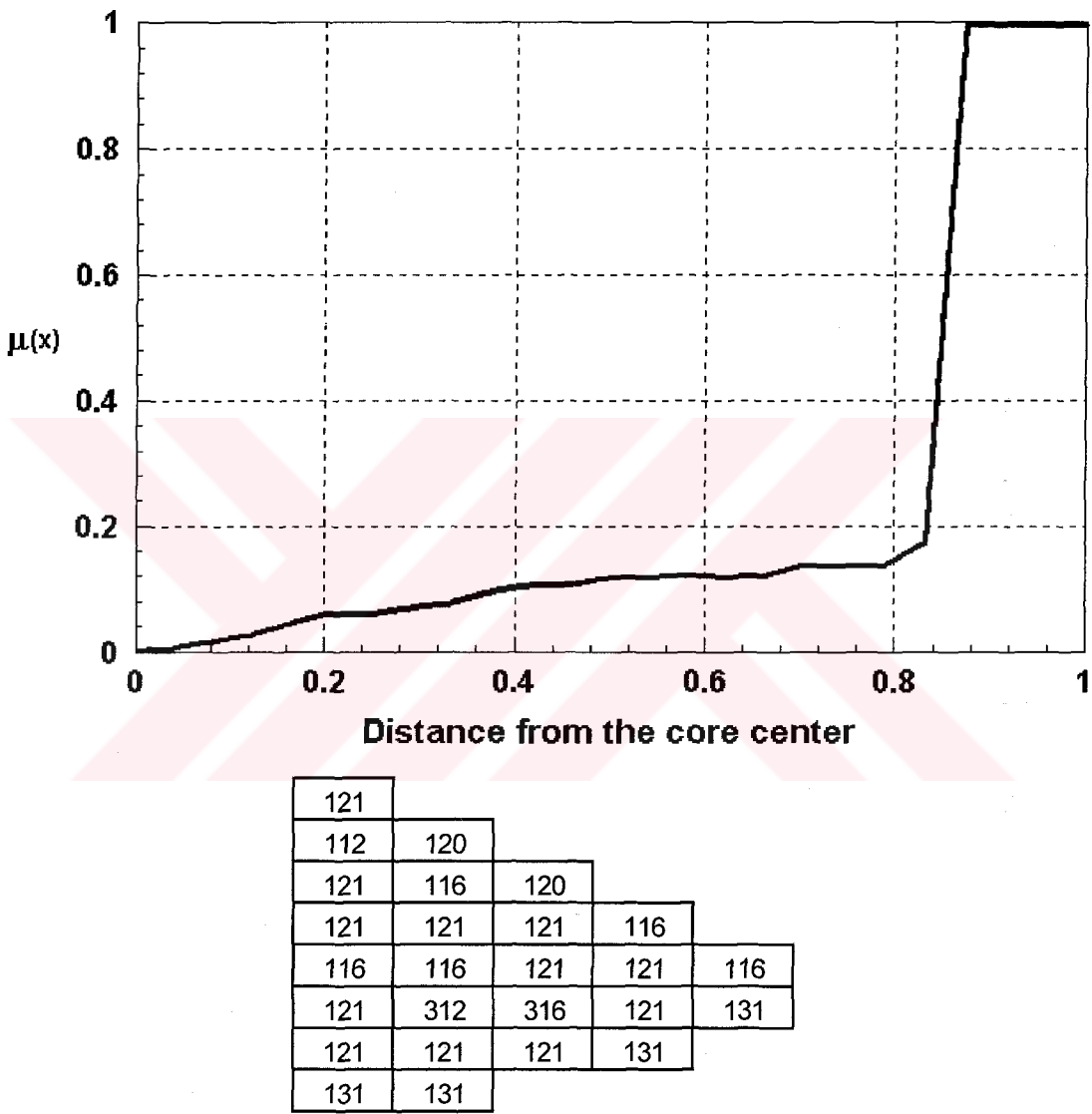


Figure C.3 $\mu(x)$ and fuel assembly distribution for an optimum loading pattern

RESUME

Adem Erdoğan was born in Alaca, Turkey on February 7, 1968. He graduated from Alaca Nedim Tuğaltay High School in 1985. The same year, he entered the University of Hacettepe at Ankara. He received his Bachelor of Science Degree in Nuclear Engineering in 1990 and Master of Science in Nuclear Engineering in 1993.

He works for the Turkish Atomic Energy Authority as a researcher since 1993.

His main interests are computer programming, computational intelligence and its application to nuclear engineering.

