2021 International Nuclear Atlantic Conference – INAC 2021
Virtual meeting, Brazil, November 29 – December 2, 2021

ISBN: 978-65-594-1256-3

Optimization of PWR Nuclear Reactor Reload Problem Utilizing Paralleled Structured PBIL

F. Castro[1], A. S. Nicolau[2], Schirru, R.[3]
M.M. de Lima[4]

[1]*flavioc@poli.ufrj.br*
[2]*andressa@lmp.ufrj.br*
[3]*schirru@lmp.ufrj.br*
[4]*alan@lmp.ufrj.br*
Federal University of Rio de Janeiro,
Brazil.

1. Introduction

The nuclear reactor reload optimization problem (POR) is a subject of great concern of nuclear engineering since each additional day of operation in full capacity means a revenue of about a million. The need of search in a vast sample space to optimize this kind of problem brought up the implementation of artificial intelligence (AI) algorithms with the purpose of reducing the necessary time, or number of evaluations in the sample space, for obtaining the best result possible based on these evaluations. The presented work make use of PBIL (Population-Based Incremental Learning), an optimization algorithm inspired on the evolution of the genotype (probability vector) within a population, in a parallel structure aiming, called PPBIL, in order to find a solution that maximizes the burning period while respecting the maximum power peaking factor permitted and reducing the mean evaluation time. This work uses the data of cycle 7 of Angra 1 PWR nuclear reactor, as it is commonly revisited in literature, using nodal reactor physics code RECNOD to feed the algorithm with the necessary parameters for evaluation of the samples. The results found with values obtained are quite like those found in literature; however, with a mean evaluation time 355ms utilizing four cores opposed to 715ms utilizing a single core in a 1.6GHz i5-8265 quad-core processor.

2. Methodology

The PBIL[1] is inspired on the evolution a genotype (probability vector) of a fixed sized population over the course of generations. Each individual is generated with each gene represented in binary with the probability of such gene being 0 determined by the probability in the same position of the probability vector. The population is initialized with each gene having 50% of probability of being 0 and as generations (iterations) pass by in accordance with Darwin's evolutionary theory the most fit individual is chosen and based on the genes of such individual the probability vector is updated for the next generation. The rate in which the most fit individual influences the probability vector of said population is determined by the algorithm learning rate and also determines how fast it will converge. However, at the cost of the quality of result due to the lowering of the number of evaluations. Each position of the probability vector is updated based on the Eq.1, [1]:

$$\mathrm{P}_K(\mathrm{i}) = \mathrm{P}_{K-1}(\mathrm{i}) \times (1 - \mathrm{LR}) + \mathrm{LR} \times \mathrm{Fittest}_K(\mathrm{i}) \quad (1)$$

Where P_K is the probability vector of the generation k, k is the number of the current generation, i is the position of the probability vector, $\mathrm{Fittest}_K$ is the binary sequence that determines the best individual found until the current generation and LR is the constant that determines the algorithm's learning rate.

The POR [2] consists in finding the best combination of fuel assemblies that maximizes the burning period of the nuclear reactor core. The core of Angra 1 is loaded with 121 fuel assemblies; however, due to the symmetric geometry of such reactor the problem consists of the permutation of 21 fuel assemblies composed

of 3 groups that can't be permutated between themselves: 1 "central" element, 10 "quartets" and 10 "octets" based on the number of symmetrical fuel assemblies are present on the core. Mathematically the problem now consists of the product of the permutation within each group of assemblies meaning the sample space consists of $1 \times 10! \times 10! = 1.317 \times 10^{13}$ possible combinations. Those conditions are translated in a single vector containing which fuel assembly, being 1 to 10 quartets and 11 to 20 octets, will fit in a labelled position in the core.

To fully utilize the PBIL its necessary to quantify the fitness of an individual. In this work it is done by considering individuals as binary strings of 100 digits which each 5 digits represent a real number between [0, 1] and its interpreted as a vector of the ranks using the random keys model [3], the first 10 numbers represent quartets and ranked from 1 to 10 and the last 10 numbers represent octets and ranked from 11 to 20. Such vector is used as input of the nodal reactor physics code RECNOD [1] which feeds the algorithm with the boron concentration (C_B) that is proportional do the length of the reactors cycle that is to be maximized and the maximum potency relative to the mean P_{rm} which is a quantity that compares to the maximum power peaking factor F_{XY} within a margin of error of approximately $\pm 2\%$. The security restriction $F_{XY} \leq 1{,}435$ can than be translated to the restriction $P_{RM} \leq 1{,}395$ without interfering in this study. The fitness of an individual can than be quantified by the equation below:

$$\boldsymbol{fit(x) = \begin{cases} \dfrac{1}{C_B} \; if \; P_{rm} \leq 1{,}395 \\ P_{rm} \; if \; P_{rm} > 1{,}395 \end{cases}} \tag{2}$$

The PBIL in this work, however, was developed in a parallel approach (PPBIL) and works as the following considering the processing core of rank 0 the brain (master) and cores 1 to 4 the workers (slave):

Program Start

If (Rank = 0):
    Initiate the probability vector with all positions with value 0.5
    Set Initial Parameters (number of generations, population size, learning rate);
    Set *globalBestFitness* as -1 to guarantee that any individual of the first generation could best it;
End If

While (GenNumber < Maximum Number of Generations) or (Convergence Criteria yet to be reached):
    If (Rank = 0):
        Send probability vector **P** to all processing cores and the quantity **N** of individuals it will process per generation;
    End If
    Receive **P** and **N**;
    Create **N** individuals based on **P**;
    Calculate the fitness of each individual created;
    Select the individual that has the best *fitness* (lowest);
    Send the data about the individual to core rank 0;
    If (Rank = 0):
        Receive the best candidate of each processing unit;
        Select the best and set its *fitness* as *bestLocalFitness*;
        If (*bestLocalFitness* > *globalBestFitness*):
            globalBestFitness = bestLocalFitness;
        End If
    Update probability vector;

End If
End While
If (Rank = 0):
Post Processing and results display.
End If
Program End

3. Results and Discussion

At first glance as expected increasing the number of processing units utilized speed up the algorithm; however, if utilized more processing units than the number of cores available one the machine (4 in this scenario) the speed decreases as shown in the image below:

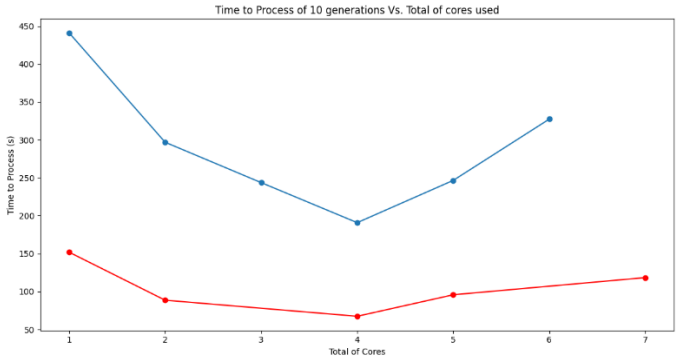

Figure 2: Time to Process 10 generations in function of the number of processes The red line presents tests utilizing a population of 20 per Generation while the blue line 60.

Additionally, observing the Figure 3 we can see that the mean time per sample is releated not only to the amount of processing units utilized, but also with the total number of evaluations (be it increasing the number of generations, population size, or both) to be done. Such fact is justified by the need of setting the initial variables and the enviroment preparation at the start of the process (copying files, moving files, etc.) being diluted as the number of evaluations increase.

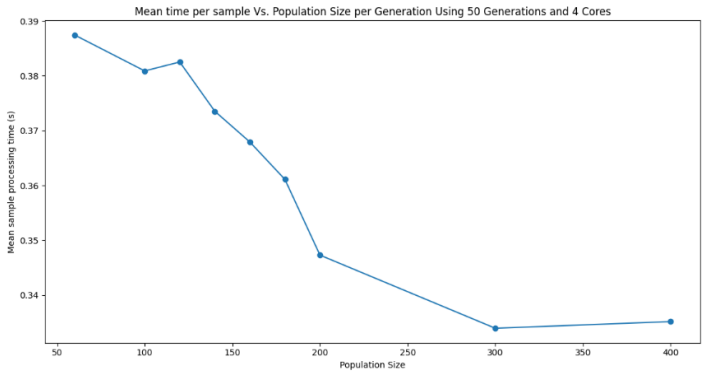

Figure 3: Mean evaluation time per sample in function of the size of the population fixed at 50 generations.

The convergence is evident and considering the algorithm still updated frequently the best individual found until the very end of the test it is safe to assume that by setting an environment capable of mean evaluation

time per sample even lower (by the usage of GPUs or/and Clusters) will achieve even best results and possibly in a lower total processing time.

The table I shows the results of PPBIL and the PBIL_MO[1] developed for the 7th cycle of Angra 1's nuclear reactor and utilize nodal reactor physics code RECNOD to feed the algorithm with the necessary parameters for evaluation of the samples.

Table I: Comparison between PPBIL and other algorithms in the literature.

Method	Best C_B (ppm)	Mean C_B (ppm)	Standard Deviation	Number of evaluations	Mean time per evaluation (ms)	Processor
PBIL_MO	1305	1004	71	10000	-	-
PPBIL	1331	1280	57	100000	355	1.6GHz i5-8265 quad-core processor

3. Conclusion

In this article was presented PPBIL a methodology based on PBIL (Population-Based Incremental Learning) within a population, in a parallel structure aiming in order to solve the POR. By analyzation the results presented in section 3, we can see that within the testing conditions PPBIL found best fitness results than standard PBIL, that is the parallel approach PPBIL has successfully increased the mean C_B lowered its standard deviation when compared with its predecessor (PBIL_MO) due to the capability of more evaluations of the sample space which shows the parallel approaches of the other methods might as well bring even better results than those already acquired. Additionally, PPBIL has shown a good mean time per evaluation which can still be lowered by the usage of multiple machines (CLUSTER) and/or by implementation of a GPU parallel approach making use of CUDA or other methods to achieve such approach.

Acknowledgements

We would like to acknowledge CNPq (National Research Council, Brazil), FAPERJ (Foundation for research of the state of Rio de Janeiro), for their support to this research.

References

[1] M.D., Machado. Algoritmo Evolucionário PBIL Multi-Objetivo Aplicado ao Problema da Recarga de Reatores Nucleares, D.Sc. thesis, COPPE/UFRJ, Brazil. (2005.)

[2] J.L.C., Chapot, F.C., Da Silva, R., Schirru. "A New approach to the use of Genetic algorithms to solve the Pressurized water Reactor's fuel Management optimization problem." *Annals of Nuclear Energy* , vol. 26, pp. 641- 655 (1999)

[3] J.C, Bean, "Genetic algorithms and random keys for sequencing and optimization". *ORSA Journal of Computing*, vol. 6, pp. 154 -160 (1994).